Offloading Delay Constrained Transparent Computing Tasks With Energy-Efficient Transmission Power Scheduling in Wireless IoT Environment

Feng Shan, *Member, IEEE*, Junzhou Luo, *Member, IEEE*, Jiahui Jin, *Member, IEEE*, and Weiwei Wu, *Member, IEEE*

***Abstract*—Billions of lightweight Internet of Things (IoT) devices have been deployed for various applications nowadays. Most of them first collect interested data and then process them in some degree according to application requirements. Transparent computing (TC) is a promising technique that makes such lightweight devices suitable to process even large-size applications. The advantage of TC is to separate code storage from its execution, allowing IoT devices to load code blocks from nearby TC storage server on demand. Distinct from existing work, this paper allows the TC IoT devices to offload some tasks to servers, since wireless IoT devices are usually powered by batteries, having limited energy resources. If a task is offloaded, a challenging problem is that its input data collected by the IoT device must be transferred as well, which incurs additional transmission time and energy. This paper proposes a two-step approach aiming at minimizing the energy consumption of the IoT device while satisfies the delay constraint. This approach first studies the offloading decision problem that determines for each task whether to offload task data or load task code blocks, while loading code indicates code receiving and executing energy cost. Second, the transmission power scheduling problem is investigated to further reduce offloading energy for a given delay constrained offloading task set. Heuristic decision making algorithms and optimal power scheduling algorithm are proposed, respectively. Such two-step approach is shown by extensive simulation to be near optimal for the original problem thanks to the optimal design of the power scheduling algorithm.**

***Index Terms*—Delay, energy-efficient, Internet of Things (IoT), offloading, scheduling algorithms, wireless transmission.**

Manuscript received May 15, 2018; revised August 3, 2018 and October 26, 2018; accepted November 16, 2018. Date of publication November 29, 2018; date of current version June 19, 2019. This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFB1003000, in part by the National Natural Science Foundation of China under Grant 61702097, Grant 61320106007, Grant 61632008, Grant 61702096, Grant 61602112, and Grant 61672154, in part by the Jiangsu Provincial Natural Science Foundation under Grant BK20170689 and Grant BK20160695, in part by the Aeronautical Science Foundation of China under Grant 2017ZC69011, in part by the Jiangsu Provincial Key Laboratory of Network and Information Security under Grant BM2003201, in part by the Fundamental Research Funds for the Central Universities under Grant 2242018K41047, and in part by the Key Laboratory of Computer Network and Information Integration of the Ministry of Education of China under Grant 93K-9. *(Corresponding author: Junzhou Luo.)*

The authors are with the School of Computer Science and Engineering, Southeast University, Nanjing 211189, China (e-mail: shanfeng@seu.edu.cn; jluo@seu.edu.cn; jjin@seu.edu.cn; weiweiwu@seu.edu.cn).

Digital Object Identifier 10.1109/JIOT.2018.2883903

I. Introduction

With the popularity of the Internet of Things (IoT), billions of lightweight IoT devices such as smart city monitors, smart home devices, and industrial sensors, are deployed. Most of these IoT devices collect interested data from deployed environment and process them in some degree according to application requirements. Transparent computing (TC) is an efficient way to make such lightweight devices more powerful, suitable, and secure for processing and executing even large-size applications. The core of TC is to separate code storage from its execution, allowing IoT devices to load code blocks from nearby TC storage servers on demand [1]–[6].

A typical TC IoT device loads application code blocks and performs local execution on demand in order to guarantee low response time for delay sensitive applications. Since most IoT devices are battery powered, they are usually limited in resources, such as low energy supply. We hence allow one more option for IoT devices, i.e., to select and offload a portion of tasks to TC servers for remote execution, because these TC storage servers usually have more computation and energy resources. However, if a task is offloaded to the servers, its collected data as input must be delivered to the servers as well since these tasks are to process these data collected by IoT devices. This will incur additional transmission time and energy. Such scenario is illustrated in Fig. 1.

Assume each task data transmission has its own completion delay constraint. Then, an IoT device has two options in executing a task—it either: 1) receives task code blocks from the TC server to execute locally, referred as loading (task) code blocks or 2) sends the collected data to the TC server to execute remotely, referred as offloading (task) data. In order to minimize the energy consumption and satisfy the delay constraint, this paper proposes a two-step approach. We first study the offloading decision problem and then investigate the transmission power scheduling problem.

The offloading decision problem balances energy consumptions between offloading data and loading code. An intuition behind is, some tasks may have large data amount and tight deadlines to offload but small code block to load and execute, while other tasks may have small data amount and loose deadlines to offload but large code block to load and execute.

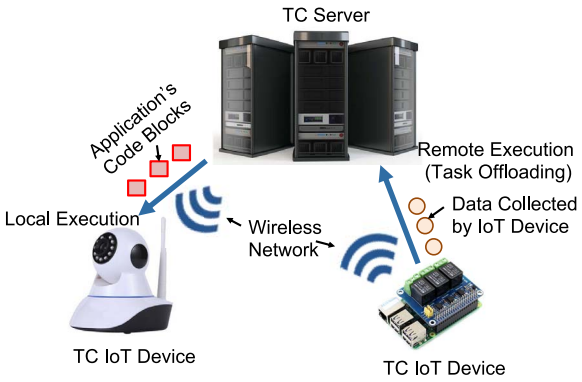


Fig. 1. Typical TC IoT device loads application code blocks from a nearby TC storage server and performs local execution on demand. Since IoT devices are usually powered by a battery, having limited energy resources. We hence consider one more option: offloading tasks to the server for remote execution. However, if a task is offloaded, its input data collected by the device must be transferred as well, which incurs additional transmission time and energy.

Hence, choose the right set of task set to offload/load is important to minimize total energy consumption for an IoT device to transmit data, receive, and execute code.

Once the loading task set is determined, the local execution code blocks are determined. We hence assume the local execution energy of each task can be estimated by code analysis [7]. According to previous research [8], for a task, the receiving (code loading) energy is proportional to the code block size, while the transmitting (data offloading) energy is related to not only the data size but also its transmission delay and the channel status.

The transmission power scheduling problem focuses on minimizing transmission energy for a given set of tasks. By Shannon–Hartley theorem on wireless channel capacity, a slower transmission rate is preferred to save energy, while a higher rate is preferred to shorten transmission delay. Moreover the worse channel quality/state, the higher transmission power is needed to reach a fixed data transmission rate; while the better channel state, the lower power to reach the fixed rate. Hence, an optimal transmission rate considers not only the data amount, deadline of each offloading task, but also the wireless channel states.

Such two problems form a two-step approach toward minimizing IoT device energy consumption. We can see that the offloading decision made in the first step will directly affect the energy consumption in the second step. Inappropriate offloading task set (e.g., with tight data transmission deadlines when channel quality is low) will increase the transmission energy even if optimal power scheduling can be found for this task set. Such connection must carefully managed in algorithm designs.

The contributions of this paper are summarized as follows.

1) We propose to offload delay constrained TC tasks from the IoT device to the TC storage server to save IoT device energy. We formulate this problem and introduce a two-step approach via two subproblems: a) the offloading decision problem and b) the transmission power scheduling problem to reduce energy consumption and satisfy delay constraint.
2) We analysis the relationship between the two subproblems. For the first offloading decision problem, we present two heuristic algorithms to iteratively solve it from two different directions, i.e., the break-down approach and the building-up approach.
3) For the second transmission power schedule problem with a given offload task set, it is discovered to be a longstanding open question. We design the highest-water-level interval first (HIF) algorithm to solve it optimally in which novel concepts, namely *data interval* and its *water level*, are introduced. By iteratively locating the data interval with the highest water level, the HIF policy computes the optimal power schedule.
4) The proposed two-step approach is shown by extensive simulation to be near optimal for the original problem. These proposed algorithms are also compared to a simple baseline method.

This paper is organized as follows. Section II introduces the system model and defines the problem. Section III proposes to decompose this problem and discusses two heuristic algorithms for making offloading decision. Section IV presents the HIF algorithm that optimally solves the transmission power scheduling problem. The simulation results are discussed in Section V. Related works are given in Section VI. Section VII concludes this paper.

II. System Model and Problem Formulation

We consider a TC system consisting of a lightweight wireless IoT device, a storage cache server, and a single user point-to-point wireless fading channel between the device and the server. Let $\mathbf{T} = \{T_1, T_2, \ldots, T_n\}$ be a set of n tasks generated at the IoT device. For any task T_i, $i = 1, 2, \ldots, n$, we use arrival time (event) a_i to denote the time that it is generated and ready for processing. Since the IoT device may generate a task at any time, we sort all the tasks by their arrival times such that $a_1 \leq a_2 \leq \cdots \leq a_n$. The task code block size is assumed to be c_i. We assume the execution energy on an IoT device can be estimated by code analysis, and is denoted as e_i. Let the input data size be b_i and assume its data transmission deadline (event) is d_i ($> a_i$). Note that such deadline can be easily obtained from the task completion deadline since its execution time on a powerful server is predictable. We assume arbitrary deadline for task T_i, which is the most generate model in the literature. If we sort the deadlines such that $d_{q_1} \leq d_{q_2} \leq \cdots \leq d_{q_n}$, then sequence $q_1, q_2, \ldots, q_n$ may be any permutation of $1, 2, \ldots, n$. Hence, a task can be represented by a quintuple, $T_i = (a_i, b_i, d_i, c_i, e_i)$, where a_i is the task generate (arrive) time, b_i and d_i is the input data size and its transmission deadline, respectively, c_i and e_i is the code block size and its local execution energy, respectively.

The execution of a task T_i requires both code blocks originally located at the server and the input data collected and stored at the IoT device. A task T_i has two execution options, either: 1) locally at the IoT device which requires its code block with size c_i to be delivered from the server and consumes e_i execution energy, referred as loading (task) code blocks or 2) remotely at the server which means a total of b_i collected

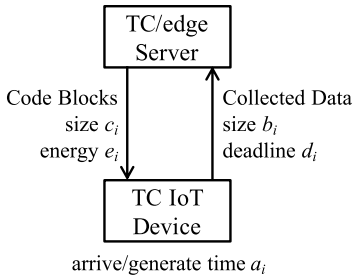


Fig. 2. There is an IoT device and a TC server in our system. Each task $T_i = (a_i, b_i, d_i, c_i, e_i)$ executes either locally at the IoT device or remotely at the server. Task T_i is generated (arrives) at time a_i, and local execution requires its code block with size c_i to be loaded (received) and the local execution energy is e_i; while remote execution requires a total of b_i collected data must be offloaded (transmitted) to the server before deadline d_i.

data must be transferred from the device, referred as offloading (task) data. Such system model is illustrated in Fig. 2. For a data offloading task T_i, its data transmission can start only after its arrival time a_i and must finish before its deadline d_i. Such constraints are called the causality constraint [9].

Define the offloading decision vector $\mathbf{x}$, whose elements are binary variables

$$x_i = \{0, 1\} \qquad \forall i \in [1, n] \tag{1}$$

where $x_i = 0$ indicates that task T_i is executed locally at the IoT device (code loading task) and $x_i = 1$ means that it is executed remotely at the server (data offloading task).

Our focus is the energy consumption of the IoT device, which consists of three major parts, i.e., the data transmitting energy for data offloading tasks, the code block receiving and executing energy. According to previous works [8], the receiving energy E_i for task T_i is proportional to the code block size c_i. Hence, we have

$$E_i = \alpha c_i, \quad i : x_i = 0 \tag{2}$$

where α is a constant factor. In the following, we introduce the transmitting energy.

Following previous works [9]–[18], we assume the wireless channel for data transmission has a dynamic channel quality, represented by a time-vary channel states, or equivalently, the fading factor. Although the channel quality varies as time goes by, we assume it is constant in a small time interval. Assume the channel quality changes only at these m time points, $f_1, f_2, \ldots, f_m$ in the interested time duration $[a_1, d_{q_n}]$. Such time points are also called fading changing event points. Now, we have n arrivals, n deadlines and m fading changes in consideration. We say there is an event sequence consisting of $2n + m$ events, including arrival events, deadline events, and fading changing events, see the time axis of Fig. 4 to find an example sequence. For presentation clarity, we assume no two events occur at the same time. If it indeed happens, we can treat them as occur in sequence with very small interval, and then our proposed algorithm applies.

The time interval between two adjacent event points is called an epoch, named as epoch $t, t = 1, 2, \ldots, \tau$, where $\tau = 2n + m - 1$. Denote the length of the epoch t as l_t. Obvious, the fading factor is constant in any epoch. Denote time-varying fading factor as $\mathbf{h}$, where element h_t is the fading factor in epoch t.

The IoT device is assumed to be able to adaptively change its transmission power and its corresponding transmission rate. Following previous works [9]–[19], we assume in epoch t, the transmission power p_t, the transmission rate r_t and the fading factor h_t is related through the following function:

$$r_t = \log(1 + h_t p_t) \qquad \forall t \in [1, \tau]. \tag{3}$$

Definition 1 (Task Data Transmission Rate): The *task data transmission rate* $\mathbf{r}_i$ is vector whose element $r_{i,t}, t = 1, 2, \ldots, \tau$ is the transmission rate for task i in epoch t.

Then, in epoch t, the transmitted data amount from task T_i is $l_t r_{i,t}$, where $i = 1, 2, \ldots, n$, $t = 1, 2, \ldots, \tau$. Note, it is assumed that the data of a task can be transmitted in any segments.

By the causality constraint, task data transmission rate must satisfy the following equation:

$$\sum_{t=1}^{\tau} l_t r_{i,t} = \sum_{t=\xi(a_i)}^{\xi(d_i)-1} l_t r_{i,t} = b_i, \quad i : x_i = 1 \tag{4}$$

where function $\xi(\cdot)$ is used to map each event point to its rank in this event sequence. For example, $\xi(a_k)$ is the rank of arrival event a_k in the sequence. Function $\xi(\cdot)$ is easy to obtain and known before scheduling.

Then, the overall data transmission rate in epoch t can be calculated as

$$r_t = \sum_{i:x_i=1} r_{i,t} \qquad \forall t \in [1, \tau]. \tag{5}$$

By connecting the transmission rate r_t to the transmission power p_t in epoch t through (3), we present the total energy consumption E of the IoT device as

$$E = \underbrace{\sum_{i:x_i=0} (E_i + e_i)}_{\text{rece. \& exec. code}} + \underbrace{\sum_{t=1}^{\tau} l_t p_t}_{\text{send data}}. \tag{6}$$

Our goal is to minimize E. We define our *offloading min-E* problem as follows.

Definition 2 (Offloading Min-E Problem): Given a set of tasks $\mathbf{T}$ and the time-varying fading factor $\mathbf{h}$, the *minimum energy* TC *offloading problem* is to determine the offloading decision variable $\mathbf{x}$ and the task data transmission rate $\mathbf{r}_i, i = 1, 2, \ldots, n$ such that the total communication energy consumption E in (6) is minimized while constraints (1)–(5) are satisfied.

In this paper, we consider the offline problem, where $\mathbf{T}$ and $\mathbf{h}$ are known in advantage before schedule. The indepth study of the offline problem provides important insights and may help in solving online problem. This is because, the study of the offline problem reveals basic properties and important essentials of the problem, which may guide the design of the online algorithm.

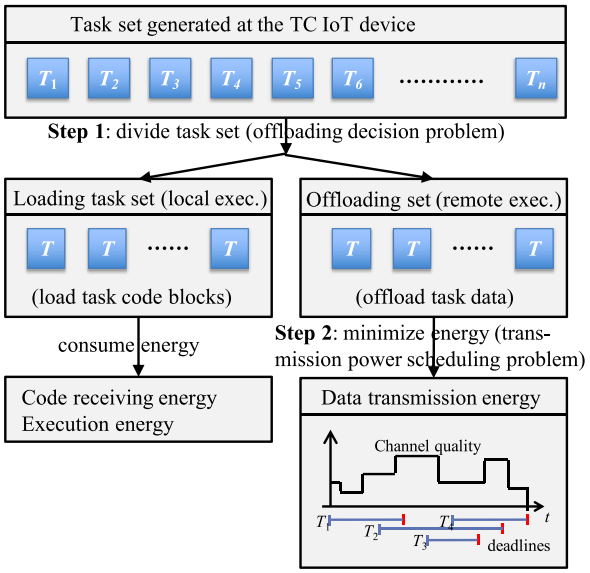


Fig. 3. Two-step approach to minimize the energy consumption of the IoT device, which includes the code receiving and execution energy and the data transmission energy. Note, for a task, the code loading (receiving) energy is proportional to the code block size, while the data offloading (transmitting) energy depends on not only the data offloading task set itself, including data size, arrival time, and deadline but also depends on the channel quality and how many tasks to share this channel. Hence, we propose the transmission power scheduling problem (step 2) to minimize the data transmission considering the transmission delay constraints.

III. Problem Decomposition and Offloading Decision Making

It is quite challenging to solve the *offloading min-E* problem directly. This is because this problem aims at minimizing the sum energy consumption of the IoT device, which consists of: 1) the code receive and 2) execution energy and 3) the data transmission energy. The code receive and execution energy, i.e., the first two energy, can be directly determined once the code loading task set is determined. However, the data transmission energy, i.e., the third energy, is quite complicate to minimize.

We hence propose a two-step approach to decompose the *offloading min-E* problem, as illustrated in Fig. 3. The offloading decision problem is studied in the first step to divide the task set generated at the TC IoT device into code loading task set and data offloading task set. According to previous analysis [8], for any task in the loading task set, its code loading energy by wireless receiving is proportional to the code block size and the code execution energy can be estimated by code analysis, so the energy consumption to load and execute tasks is determined once the loading tasks are determined. However, data transmission energy for the offloading task set is related to not only the data size but also its transmission delay and the channel quality. Then, the transmission power scheduling problem is investigated in the second step to minimize the transmission energy by adaptively change the transmission power while satisfy individual task deadlines and consider channel quality. This problem is a longstanding open question.

The two problems are highly coupled since the offloading task set output by step 1 is the input for step 2 to minimize the transmission energy. To decouple the two problems and reduce the complexity, we assume there is an algorithm that optimally solve the transmission power scheduling problem for a given offloading task set. Let us call such algorithm HIF. Our basic idea is to iteratively refine the offloading task set and repeatedly invoke HIF. In such way, we make the offloading decision for each tasks.

Algorithm 1: Break-Down

```
p = 0.5 // control parameter for task switching ratio
T = {T_1, T_2, ..., T_n};
while true do
    Initialize vector E^improve = 0;
    foreach T_i ∈ T do
        E_offload = HIF(T) − HIF(T/T_i);
        E_load = α c_i + e_i;
        E_i^improve = E_offload − E_load
    end
    if E_i^improve ≤ 0 for ∀i then break;
    Let n_>0 be the number of E_i^improve > 0;
    Select top p * n_>0 tasks from E^improve;
    Update T by subtracting these tasks;
end
return T;
```

More specifically, our core idea to decide a specific task to be executed remotely or locally is to estimate the energy consumption for both options and choose the more energy efficient one. The basic idea on estimating is to invoke the optimal HIF algorithm twice, once include this specific task in input offloading task set and once without it but including it in the loading task set. This energy difference is used to estimate the energy consumption of this specific task. Based on this basic idea, we propose two symmetrical algorithms to make the offloading decision from two different directions, i.e., the break-down approach and the building-up approach.

In the break-down approach, every task is initially included in the offloading task set and the loading task set is empty. We invoke the HIF algorithm to compute the minimum total energy in transmitting these data to the server. Then we subtract each task from the offloading task set, one at a time, and invoke again the HIF to compute new energy consumption. The energy difference is the energy saved by not offloading that task to the server. Then that task must execute locally, where we can easily compute its energy consumption on receiving code and execution it. As a result, the data transmission energy is reduced but the code receiving energy is increased, the IoT energy consumption can be reduced if the decrease is greater than the increase. We compute the improvement for every task and select these tasks with the greatest energy improvements to switch they from offloading to loading. After updating the offloading task set, the same problem repeats. A formal algorithm of these steps is give in Algorithm 1.

Algorithm 1 works in iteration. In each iteration, a number of tasks are switched from offloading task set to loading task set. After task sets are updated, the same procedure repeats in the next iteration. At the begin of an iteration, suppose the current task set is a subset of the entire task set, i.e., $\mathbf{T} \subseteq \{T_1, T_2, \ldots, T_n\}$, and $T_i = (a_i, b_i, d_i, c_i, e_i)$. Then $E_{\text{offload}} = \text{HIF}(\mathbf{T}) - \text{HIF}(\mathbf{T}/T_i)$ is the energy saved in not offloading the collected data of T_i to the server. If the task T_i execute locally at the device, then energy used to receive code blocks from the server and the execution these code blocks are $E_{\text{load}} = \alpha c_i + e_i$. Hence, the possible IoT device energy improvement can be calculated as $E_{\text{offload}} - E_{\text{load}}$, note it is possible that this improvement value is negative. We compute such improvement for every task $T_i \in T$, and store it into an array $\mathbf{E}^{\text{improve}}$ as in the **foreach** loop. In $\mathbf{E}^{\text{improve}}$, only element with positive value indicts a real energy saving improvement. Let $n_{>0}$ be the number of positive elements, and we select a portion p of the such tasks to switch from offloading to loading, i.e., top $p{*}n_{>0}$ task from $\mathbf{E}^{\text{improve}}$. The iteration stops when array $\mathbf{E}^{\text{improve}}$ contains no positive element, i.e., no task can be switched to make a IoT device energy consumption improvement.

Note that, parameter p in line 1 is the control parameter for task switching ratio, i.e., how many task can be switched in a single iteration. It also controls the tradeoff between the speed and accuracy of this algorithm. We will discuss such tradeoff in detail in simulations.

Another approach is the build-up approach, which initially assume all tasks are in code loading task set and execute locally, hence incurring the code block receiving and executing energy. By testing every task $T_i, i = 1, 2, \ldots, n$, we can select the top $pn_{>0}$ tasks that improve the IoT device energy consumption by switching from code loading to data offloading. After updating the offloading task set $\mathbf{T}$ by including these tasks, the same problem repeats. The iteration stops when there is no task whose switch can reduce the IoT device energy consumption. The details of algorithm build-up is quite similar to Algorithm 1, we omit it for space limitation.

IV. Transmission Power Scheduling

This section studies the transmission energy consumption minimization problem for a given set of offloaded tasks. This problem is first formally defined and the novel notion *water level* is introduced. Some basic properties for the optimal water levels are presented; and then the optimal HIF policy is introduced; lastly, the correctness of HIF policy is proved.

Definition 3 (Transmission Power Scheduling Problem): Given a set of tasks $\mathbf{T}$ and the time-varying fading factor $\mathbf{h}$, the *minimum energy problem* is to determine the task data transmission rate $\mathbf{r}_i, i = 1, 2, \ldots, n$ and the transmission schedule for each task such that the total transmission energy consumption is minimized

$$\begin{aligned} \min.\quad & \sum_{t=1}^{\tau} l_t p_t \\ \text{s.t.}\quad & r_t = \log(1 + h_t p_t) \qquad \forall t \in [1, \tau] \end{aligned}$$

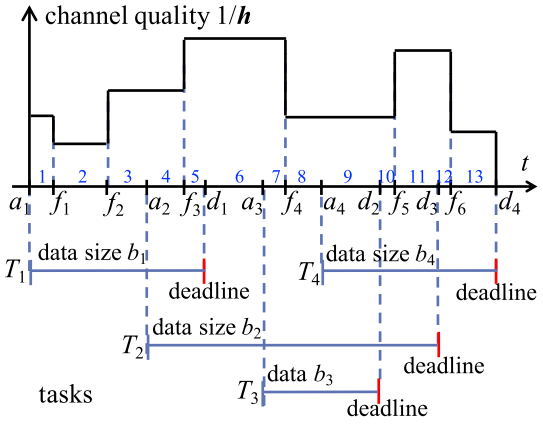


Fig. 4. Transmission power scheduling problem. It is a longstanding open question, which we solve optimally. Tasks are allowed to have arbitrary arrival times, arbitrary deadlines and arbitrary data sizes. The time-varying fading factor h is allowed to be arbitrary. The goal is to determine a transmission power schedule such that the transmission energy consumption is minimized. In this example, there are four tasks and six fading changes, and 13 epochs in blue.

$$\sum_{t=1}^{\tau} l_t r_{i,t} = \sum_{t=\xi(a_i)}^{\xi(d_i)-1} l_t r_{i,t} = b_i \qquad \forall i \in [1, n]$$

$$r_t = \sum_{i=1}^{n} r_{i,t} \quad \forall t \in [1, \tau].$$

The *transmission power scheduling problem* is an open question for many years since Prabhakar *et al.* [10] and Uysal-Biyikoglu *et al.* [11] formulated a simple version of this problem more than a decade ago. Many researchers have been working to analytically solve this problem but only partially progress has been made [9], [12]–[16]. Fig. 4 is an example of this problem in which there are four tasks and six fading changes.

It is not hard to find that to determine the task data transmission rate $\mathbf{r}_i, i = 1, 2, \ldots, n$, it is enough to determine the determined the overall transmission rate $r_t, t = 1, 2, \ldots, \tau$. This is because we can always schedule the task data transmission according to the earliest deadline first (EDF) policy to fulfill the overall rate r_t. In this way, the transmission rate $\mathbf{r}_i$ for each task can be determined. By (3), transmission rate r_t and transmission power p_t uniquely determine each other, therefore, solving the *transmission power scheduling problem* is equivalent to compute the power schedule p_t for each epoch $t = 1, 2, \ldots, \tau$.

We propose a novel notation *water level* related to p_t as follows.

Definition 4 (Water Level): For any given power schedule, $p_i, i = 1, 2, \ldots, \tau$, the water level w_t is defined as

$$w_t = \begin{cases} p_t + \frac{1}{h_t}, & \text{if } p_t \neq 0 \\ \text{undefined}, & \text{otherwise} \end{cases}$$

where h_t is the fading factor.

The notation *water level* w_t is introduced and inspired by the analysis in Appendix A. Then, given water level w_t, we

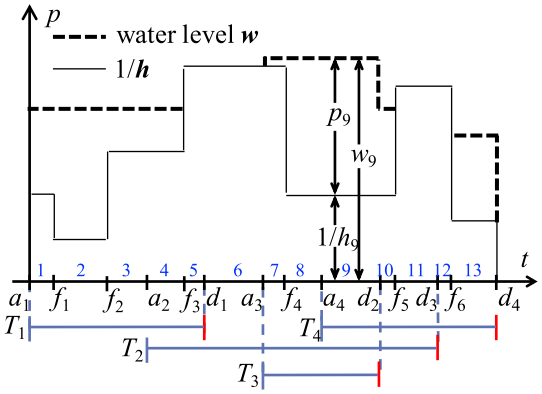


Fig. 5. Illustration of the optimal water level $\mathbf{w}^{\text{opt}}$ for the example given in Fig. 4. From this figure, we can see that $\mathbf{w}^{\text{opt}}$ and $\mathbf{p}^{\text{opt}}$ uniquely determine each other since $\mathbf{h}$ is known in advance. Note that the *water level* is undefined in epochs 5, 6, 7, 11, and 12, where transmission power is zero.

can compute p_t using formula

$$p_t = \left(w_t - \frac{1}{h_t}\right)^+.$$

Obviously, the optimal power schedule $\mathbf{p}^{\text{opt}}$ and the optimal *water level* $\mathbf{w}^{\text{opt}}$ uniquely determine each other. We will therefore focus on the optimal water level $\mathbf{w}^{\text{opt}}$. An illustration of the optimal water level is given in Fig. 5.

In the rest of this section, we focus on solving the *transmission power scheduling problem* by computing the optimal water level $\mathbf{w}^{\text{opt}}$.

A. Basic Properties of Optimal Power Policy

We first present some basic properties that any optimal water level $\mathbf{w}^{\text{opt}}$ must have.

Lemma 1 (Water Level Equalization): For any power schedule, if in epochs i and j, the water levels do not equal, assuming $w_i < w_j$, then a shared water level w, $w_i < w < w_j$, can be used in both epochs to improve the schedule unless the causality constraints do not allow.

Proof: See Appendix B. ∎

As an immediate result of Lemma 1, if there is only one task to transmit data, then the optimal water level is constant over epochs regardless of the fading level. This result is consistent with the famous *water-filling* method [22]. When multiple packets are in consideration, the water level changes because of the causality constraints. The changes have the following properties.

Lemma 2: In $\mathbf{w}^{\text{opt}}$, if two water level w_i^{opt} and w_j^{opt} are adjacent, i.e., there is no water level (defined) between epochs i and j, and if $w_i < w_j$ ($w_i > w_j$), then there must be $a(n)$ arrival (deadline) point in between them.

Proof: See Appendix C. ∎

Note, two adjacently defined water levels may or may not be in two adjacent epochs, since in some epochs, the water level may be undefined. We explain this lemma using the example $\mathbf{w}^{\text{opt}}$ in Fig. 5. In $\mathbf{w}^{\text{opt}}$, two water levels $w_4^{\text{opt}} < w_7^{\text{opt}}$ are adjacent since no water level is defined in epochs 5 and 6 by this lemma, there must be an arrival point in epochs 5 and 6; two levels $w_9^{\text{opt}} > w_{10}^{\text{opt}}$, then the border of the two epochs must be a deadline; since $w_{10}^{\text{opt}} > w_{13}^{\text{opt}}$, a deadline point must be in between them according to this lemma.

Lemma 3: Let $T_k = (a_k, b_k, d_k, c_k, e_k)$ be any task whose data is transmitted according to the optimal water level $\mathbf{w}^{\text{opt}}$. Let H be the set of all epochs contained in time interval $[a_k, d_k)$, and $H' \subseteq H$ be the subset of H which is not used to transmit T_i. The following two statements are true.

1) The water level $\mathbf{w}^{\text{opt}}$ used for any epoch of $H - H'$ must be the same water level w.
2) The water level $\mathbf{w}^{\text{opt}}$ used for any epoch of H' must be higher or equal to the water level w.

Proof: See Appendix D. ∎

We use $\mathbf{w}^{\text{opt}}$ in Fig. 5 to explain this lemma. Take T_3 as an example. Its H set contains epochs 7–9 which transmit no other task data, so $H' = \varnothing$. According to this lemma, water level in $H - H'$, i.e., epochs 7–9, must be equal. Take T_2 as another example. Its H set contains epochs 4–11, while H' contains epochs 7–9 since they are used to transmit T_3. According to this lemma, water levels in $H - H'$, i.e., epochs 4 and 10, must be equal to a value w, and water level in H' must be equal to or higher then w.

B. Water Level and Its Computation

With the optimal properties prepared, this section introduces how to compute the water level. But before that, we first introduce one related definition.

Definition 5 (Data Interval [19]): Given a task set $T_k = (a_k, b_k, d_k, c_k, e_k)$, $1 \le k \le n$, the data interval $I[i, j]$ is defined as the time interval from the arrival time a_i to the deadline d_{q_j}, $1 \le i, j \le n$, e.g., $I[i, j] = [a_i, d_{q_j})$ when $a_i \le d_{q_j}$. $I[i, j]$ is undefined, when $a_i > d_{q_j}$.

Note that $d_{q_1} \le d_{q_2} \le \cdots \le d_{q_n}$ and sequence $q_1, q_2, \ldots, q_n$ is a permutation of $1, 2, \ldots, n$.

Our proposed HIF policy works in iteration. We define the following four parameters for each data interval $I[i, j]$, $1 \le i, j \le n$, which will be modified in each iteration.

1) The task set $S[i, j]$ is the set of tasks whose arrival time and deadline are both contained inside $I[i, j]$ and have not been assigned epochs yet. Initially, $S[i, j] = \{P_k | [a_k, b_k) \subseteq I[i, j]\}$.
2) The data load $B[i, j]$ is the sum amount of data contained in $S[i, j]$, i.e., $B[i, j] = \sum_{T_k \in S[i,j]} b_k$.
3) The available epochs $T[i, j]$ is the set of all available epochs contained in interval $I[i, j]$. Initially, $T[i, j]$ contains all epochs in interval $[a_i, d_{q_j}]$.
4) The water level $W[i, j]$, as a constant value, transmits $B[i, j]$ data in $T[i, j]$, e.g., $B[i, j] = \sum_{t \in T[i,j]} l_t \log(h_t W[i, j])$.

Major notations used in this paper are summarized in Table I for the reader's convenience.

For the example illustrated in Figs. 4 and 5, $I[2, 3] = [a_2, d_{q_3}) = [a_2, d_2)$; the task set $S[2, 3] = \{T_2, T_3\}$, the data load $B[2, 3] = b_2 + b_3$; the available epochs $T[2, 3]$ includes epochs 4–11; $W[2, 3]$ is a shared water level for epochs in $T[2, 3]$ to transmit $B[2, 3]$ data. Although the $W[2, 3]$ is not shown in the figure, its exact value can be computed by the classic water-filling technique: water can be gradually filled

TABLE I
MAJOR NOTATIONS AND THEIR EXPLANATIONS

Notation	Explanation
T_k	The k-th task $T_k = (a_k, b_k, d_k, c_k, e_k)$, with arrival time a_k, data size b_i, transmission deadline d_i, code block size c_i, execution energy e_i
$\mathbf{h}_t$	fading factor (channel quality) in Epoch t
x_k	$x_k = 0$ indicates that task T_k is executed locally at the IoT device (code loading task) and $x_k = 1$ means that it is executed remotely at the server (data offloading task)
$I[i,j]$	$= [a_i, d_{q_j})$, the time interval from the arrival time a_i to the deadline d_{q_j}
$S[i,j]$	the set of tasks whose arrival time and deadline are both contained inside $I[i,j]$ and have not been assigned epochs yet
$B[i,j]$	the sum amount of data contained in $S[i,j]$
$T[i,j]$	the set of all available epochs contained in interval $I[i,j]$
$W[i,j]$	a constant value transmits $B[i,j]$ data in $T[i,j]$. *e.g.,* $B[i,j] = \sum_{t\in T[i,j]} l_t \log(h_t W[i,j])$.

Algorithm 2: IntervalWaterLevel($I[i, j]$)

```
H=sort(h) // sort epochs such that the
    channel fading factors are in
    non-increasing order
for q ← 1 to K do
    w = 1/H_q, B = 0;
    for k = 1 to K do
        if w > 1/h_k then B = B + l_k log(w h_k);
    end
    if B ≥ B[i,j] then break;
end
Solve B[i,j] = Σ_{k=1}^{q-1} l_k log(w_x h_k);
return w_x
```

into interval epochs 4–11, and stops filling as soon as the corresponding transmission power can support transmitting $b_2 + b_3$ data. Such water level is $W[2, 3]$.

Since computing $W[i, j]$ is one of the most important steps of HIF policy, we specifically design an efficient Algorithm 2 which directly computes $W[i, j]$. We assume the number of available epochs in $T[i, j]$ is K. We sort the K epochs in non-increasing order of the channel factors, i.e., $1/h_1 \le 1/h_2 \le \cdots \le 1/h_k$. As the water is gradually filled in, the water level reaches values in $\{1/h_i, i = 1, 2, \ldots, K\}$, one by one. When the water level w equals $1/h_q$, the total data transmitted with current water level is $B = \sum_{k<q} l_k \log(wh_k)$. If $B < B[i, j]$, then we must have the water level $W[i, j] > w$, and otherwise, $W[i, j] \le w$. By computing B for each $(1/h_q)$, $q = 1, 2, \ldots, K$, we can determine a q such that $1/h_q \le W[i, j] \le 1/h_{q+1}$. So, by solving $B[i, j] = \sum_{k=1}^{q-1} l_k \log(w_x h_k)$, we can compute $w_x = W[i, j]$ directly.

C. Highest-Water-Level Interval First Policy

To compute the optimal water level $\mathbf{w}^{\text{opt}}$, the HIF policy works in iteration. In each iteration, the water level is computed for every data interval. Amongst all the data intervals, we locate the one with the highest water level. Let it be $I[i, j]$ and its water level be $W[i, j]$. Then, the HIF policy transmits all packets from $S[i, j]$ in the epochs $T[i, j]$. It will be proved

Algorithm 3: HIF

```
while exist some tasks not yet assigned epochs do
    foreach data interval I[i,j], 1 ≤ i,j ≤ n do
        Identify the task set S[i,j];
        Compute the data load B[i,j];
        Determine the available epochs T[i,j];
        Compute the water level W[i,j] by
        Algorithm INTERVALWATERLEVEL;
    end
    Locate the highest water level interval I[i,j];
    Assign water level W[i,j] to epochs in T[i,j];
    Mark all the epochs in T[i,j] as unavailable;
    Mark all the tasks of S[i,j] as assigned epoch;
end
```

that any optimal water level should use $W[i, j]$ in data interval $I[i, j]$, and exactly $B[i, j]$ data from $S[i, j]$ can be delivered in $I[i, j]$. We then update the available packet set by subtracting $S[i, j]$ and update the available epoch set by subtracting $T[i, j]$. After such update, the same problem appears and we again locate the highest water level interval by the same procedure. Details are presented in Algorithm 3.

We use the example in Fig. 5 to illustrate the execution of Algorithm 3. In the first while loop, after the foreach loop computes the water level for every possible interval, $I[3, 2]$ is located as the highest water level interval, and task T_3 is assigned with epochs 7–9, which use the same water level $W[3, 2]$. After updates, the remain task set is $\{T_1, T_2, T_4\}$ and the available epochs are $\{1-6, 10-13\}$. In the second while loop, interval $I[1, 3]$ is located as the highest water level, and task T_1 and T_2 are assigned with epochs 1–6, 10, and 11, where the water level $W[1, 3]$ is used in epochs 1–4 and 10. After updates, the remain task set is $\{T_4\}$ and the available epochs are $\{12, 13\}$. In the third while loop, interval $I[4, 4]$ is located as the highest water level, and task T_4 is assigned with epochs 12 and 13, where the water level is defined in epoch 13.

The correctness of the HIF policy depends on whether using $W[i, j]$ as the water level in data interval $I[i, j]$ is optimal. If this is true for the first iteration, then, by the recursive native of HIF policy, we can conclude that it is optimal in every iteration. The following theorem states that this is true for the first iteration.

Theorem 1: Given a set of tasks $T = \{T_i | 1 \le i \le n\}$, $T_i = (a_i, b_i, d_i, c_i, e_i)$, among all the data intervals, if the highest water level interval is $I[i, j]$, then the following statements must be true.

1) Any optimal transmission policy must assign water level $W[i, j]$ to every epoch of $I[i, j]$.
2) Any optimal transmission policy must transmit exactly the packets $S[i, j]$ in $I[i, j]$.

Proof: See Appendix E. ∎

All the water levels computed by the HIF policy are optimal by the recursive native of HIF policy.

After the optimal water levels are calculated, the transmission power is determined, hence the packet transmission

schedule can be uniquely determined by applying EDF rule to select packets from the data queue to transmit.

V. Simulations

In our simulation, we investigate the performance of the two proposed algorithms involving break-down, build-up, and HIF. Since no other existing work studies the same problem, we compare them against the optimal results when the input task set size is small and against a simple intuitive baseline algorithm when the input set is large.

Since the optimal offloading task set must be one of the subset of the entire task set. When the entire task set is small, we can use brute force to enumerate every possible combination subset and compute the total IoT device energy consumption. However, when the task set size becomes larger, the number of combinations grows exponentially, hence the brute force method fails to compute the optimal solution within a reasonable time. We therefore propose a baseline method, which randomly selects n_{baseline} combinations from all possible subsets, and output the best solution amongst them.

A. Simulation Settings

We assume a total of n tasks are generated whose input data size follows an uniform distribution $\mathsf{U}(1, 50)$ and code block size follows an uniform distribution $\mathsf{U}(1, 4)$. The code block receiving energy consumption parameter α is set to 1. The local execution energy of the code is assumed to follow an uniform distribution $\mathsf{U}(0, 2)$. The task arrivals are assumed to following Poisson process. The average interarrival time is set to be 1. The task data transmission delay is assumed to follow an uniform distribution $\mathsf{U}(0, 2 * d)$, where d is the average delay constraint. We assume the wireless channel for transmit data from the device to the server has a dynamic random channel quality, whose channel state (fading factor) follows an uniform distribution $\mathsf{U}(0.05, 2 * h - 0.05)$, where h is the average channel fading value.

In our simulations, the default setting is the fading factor $h = 0.25$, the average delay constraint $d = 4$ and the packet number $n = 5$ for brute force method. When the task number is small, i.e., $n = 5$, break-down and build-up algorithms are compared against the brute force optimal solution, and parameter h and d are changed one at a time to evaluate their impact on algorithm performance. When the task number is large, i.e., n changes from 8 to 50, the two algorithms are compared to the baseline method to evaluate the algorithm performance. The execution time is measured on an Apple iMac computer with a 4-core Intel i5 processor working at 3.2 GHz, and the system memory is 16 GB. We set $n_{\text{baseline}} = 100$ for the baseline algorithm.

Each point shown in figures of this section is the mean value of simulation results from 50 random instances. In each instance, a total of n packets are randomly generated according to the above settings.

B. Simulation Results

The impact of offloading decisions are shown in Table II, where three types of energy consumption are compared, i.e., when all task executed remote, all task executed locally and when tasks are optimally selected to offload. We generate 50 random instances, and the energy consumptions for the first six instances are presented in the first six columns, while the average energy consumptions of all instances are presented in the last column. We can see from Table II that in every instance, the energy for both remote and local execution is higher than that for selective offloading. On average, the proposed offloading method reduces over 40.0% energy off the traditional TC block streaming method, and reduces over 53.9% energy off the remotely executing method.

TABLE II
Energy Consumptions Comparison

index	1	2	3	4	5	6	avg.
remote ex.	172.1	234.9	119.4	137.9	183.1	87.5	175.6
local ex.	114.0	115.0	119.0	143.0	192.0	175.0	134.8
offloading	74.5	84.0	76.1	64.3	125.6	50.6	80.9

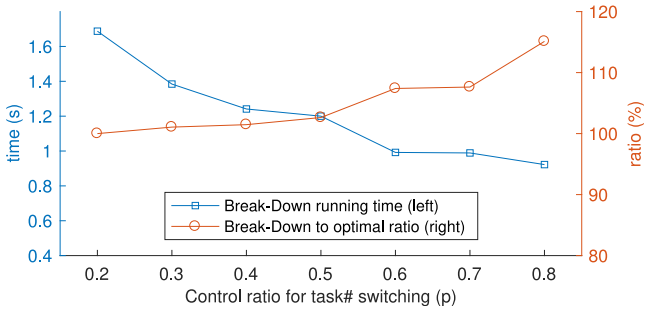


Fig. 6. Tradeoff between algorithm execution time and output accuracy. The more time sent (iteration), the more accurate the results.

In Fig. 6, we consider the break-down heuristic, in which every task is initially included in the offloading task set and the loading task set is empty. Task-switching ratio control parameter p controls how many tasks can be switched in a single iteration. We can see that the running time curve declines as p grows greater, while the accuracy curve (represented by the break-down to optimal ratio in terms of energy consumption) rises as p grows. This is because, when p is small, less tasks are selected and switched in each iteration, resulting in more iterations to finish the algorithm which slows down the algorithm. By the HIF algorithm designed in Section IV, the less tasks switched in one iteration, the more accurate its energy is estimated by line 6 of Algorithm 1. When p is large, more tasks can be switched in one single iteration, which leads to a fewer loops before algorithm ends. However, the energy estimate may be inaccurate since mistakes occur and the accuracy of the algorithm drops. Therefore, in simulations, we set $p = 0.5$ as default value to find a good tradeoff.

In Fig. 7, the energy consumptions of our two proposed algorithms are compared to either the optimal solution or the baseline method.

From Fig. 7(a), it can be observed that under various data transmission delay constraints, both our proposed algorithms, i.e., the break-down and build-up, performance well. Their curves are close to the optimal solution, indicting they are efficient in making offloading decision and schedule

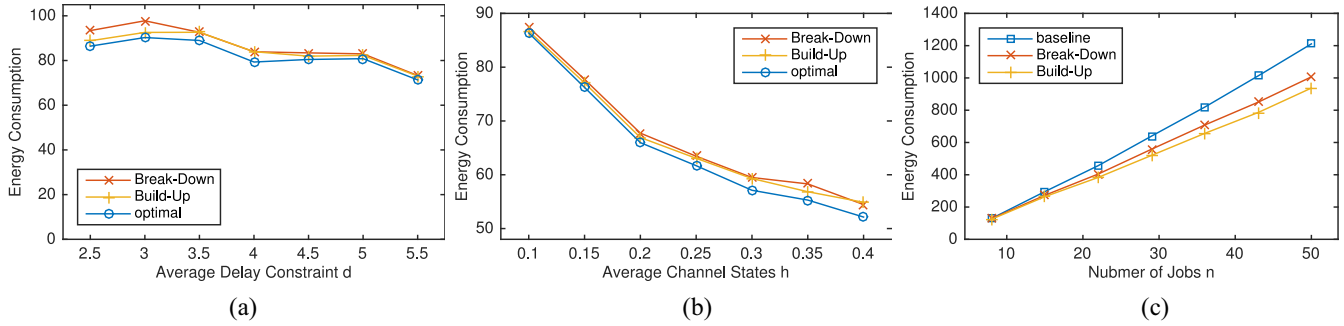


Fig. 7. Evaluation of the energy consumption by our two proposed algorithm. In (a) and (b), the optimal value is computed by brute force search. In (c), a baseline value is depicted.

energy-efficient data transmission. Meanwhile, we notice that energy consumption decreases as the average delay constraint increases for all three curves. This is because the longer a delay constraint is, the less urgent the task data transmission is, which implies that lower transmission rate can be used to deliver it and therefore consumes less energy.

It can be conclude from Fig. 7(b) that both break-down and build-up are efficient in performance under various channel state conditions. Note that the three curves descend as the fading factor enlarges. This is because, generally, higher fading factor means better channel quality, so lower transmission power and less energy consumption.

In Fig. 7(c), we change the task number, from 8 to 50 with step 7, to evaluate its impact on algorithm performance. Since the brute force method no longer computes the optimal solution at such input size, we instead use the baseline method for comparison. We can see that under different task numbers, both algorithms performance better than the baseline method. And the gap grows as the task increases, indicting our proposed algorithms have advantage when large number of packets are in consideration. The curves show rising trends with the increase of packet number, since more packets means more data to transfer, more energy needs to be consumed.

VI. Related Work

As a new paradigm of modern computing, TC has been proposed and studied in recent years. Researchers study how to improve TC from various aspects, e.g., code execution schemes [1]–[3], cache storage frameworks [4]–[6], and resource managements [20], [21]. Peng *et al.* [1] proposed a block-streaming APP execution scheme, which splits the codes of a whole service into numerous functional blocks and loads them on demand. Ren *et al.* [2] proposed a scalable IoT architecture that combines edge computing and TC, in which service provisioning flow is from edge servers to IoT devices, and data processing flow is in reverse direction. Zhang *et al.* [3] further reduced the service delay by proactively streaming blocks before the requests. Besides the above TC improvement from code execution, other improvements are from cache storage. Zhang *et al.* [4] designed a multilevel cache scheme to speed up the code block access. Liu *et al.* [5] proposed a simulation framework to evaluate a particular cache scheme. Jin *et al.* [6] used cooperative storage and D2D data sharing to reduce the code accessing delay. Other works focus on resource management which can be incorporated into TC. Zhang *et al.* [20] investigated a new resource allocation algorithm to manage both energy and spectrum resource. Zhang *et al.* [21] studied a utility-optimal resource management and allocation algorithm for energy harvesting IoT devices.

Tremendous research efforts have been made to design delay-constrained energy-efficient power scheduling algorithms with or without the consideration of dynamic channel states, namely, time-varying fading factors. Prabhakar *et al.* [10] and Uysal-Biyikoglu *et al.* [11] are among the first group of researchers who formulated the delay-constrained energy efficient packet transmission problem. They considered the case where all packets have a common deadline and the arrival time and size of each packet are known in prior to the scheduling. An optimal scheduling algorithm is presented to guarantee to deliver all packets before the deadline with minimum energy consumption. Zafer and Modiano [14], [15] thus presented an optimal algorithm that allows each packet to have an individual deadline. They proposed the cumulative curves to track packet arrivals and packet departures. The key observation is that a feasible departure curve always lies between the arrival curve and minimum departure curve. However, they still need to make an undesirable assumption that a packet arriving earlier carries an earlier deadline, which will be referred to as aligned deadlines in this paper. Shan *et al.* [19] solved the most general case in which arbitrary deadlines are allowed. The common deadline model and the aligned deadline model are both special cases of this more general model. They arose the concept of data interval and propose the densest interval first policy to control the transmission power and schedule the transmission rate that minimize the energy consumption.

These above papers investigate over a static channel. In the real world, a wireless communication channel is usually a time-varying fading channel. El Gamal *et al.* [9] and Uysal-Biyikoglu and El Gamal [12] proposed the MoveRight and FlowRight algorithms that solve this problem when packets have aligned deadlines. The main idea of the MoveRight/FlowRight algorithm is to iteratively calculate the local optimal solution for every two adjacent time-slots, and this iterative local optimization is proved to lead to the globally optimum solution. However, such an iteration-based algorithm

has a high computational complexity, where hundreds of seconds may be required in actual computation [9], [12]. Moreover, it can not handle the more general arbitrary deadline model.

From the above discuss, we can conclude that the energy efficient power scheduling problem with arbitrary individual deadline guarantee over a fading channel is still open.

VII. Conclusion

This paper has formulated the *offloading min-E* problem, and decomposed it into the offloading decision problem and the transmission power scheduling problem. Two heuristic approaches, namely break-down and build-up were presented to make the offloading decision for the first problem. Although the transmission power scheduling problem is a longstanding open question, we proposed the HIF policy to optimally solved this problem after introducing some optimality properties for it. The HIF policy was designed based on novel notations such as *date interval* and *water level*. Simulations have shown these proposed algorithms are efficient through extensive simulations.

In the future, we plan to extend this paper that considers only one IoT device and one TC server to a more complex scenario where multiple IoT devices are connected to a TC server.

Appendix A
Introduction of Water Level

We use a toy example to introduce the *water level* that minimizes the energy consumption with delay constraint. This analysis is inspired by the well-known water-filling power allocation method which aims to maximize the throughput [22].

Given one task $T = \{c, b, 0, 2\}$ with arrival time $a = 0$, deadline $d = 2$. Assume the fading factor is h_1 in $[0, 1)$ and h_2 in $[1, 2)$. Hence, there are three events and two epochs. We want to compute the optimal transmission power p_1 and p_2 for the two epochs, such that the total consumed energy is minimized

$$\text{min.} \quad p_1 + p_2 \tag{7}$$

$$\text{s.t.} \quad \log(1 + h_1 p_1) + \log(1 + h_2 p_2) = b \tag{8}$$

$$p_i \geq 0, i = 1, 2. \tag{9}$$

By the KKT conditions for convex program [23], we associate Lagrangian multiplier w with constraint function (8) and multiplier μ_i with (9). Then we have the following Lagrangian function [23]:

$$L(\mathbf{p}, w, \mu) = p_1 + p_2 - w(\log(1 + h_1 p_1) + \log(1 + h_2 p_2) - b) - \mu_1 p_1 - \mu_2 p_2.$$

By the necessary and sufficient KKT conditions, we have $\mu_i p_i = 0$ and $(\partial L / \partial p_i) = 0$ for $i = 1, 2$. Thus,

$$p_i = \frac{w}{1 - \mu_i} - \frac{1}{h_i} = \left(w - \frac{1}{h_i}\right)^+, \qquad i = 1, 2$$

where the second equation is because $\mu_i p_i = 0$ which means at least one of μ_i and p_i must be 0. This implies that $p_1 + 1/h_1 = p_2 + 1/h_2$ when $p_1 > 0$ and $p_2 > 0$. We therefore define $p_t + 1/h_t$ as the *water level*.

Appendix B
Proof of Lemma 1

Assume $w_i = p_i + (1/h_i)$ and $w_j = p_j + (1/h_j)$, where h_i and h_j are channel fading factors, p_i and p_j are transmission powers, so $p_i = w_i - (1/h_i)$ and $p_j = w_j - (1/h_j)$. Therefore, the energy consumption E of the two epochs is calculated as

$$E = p_i l_i + p_j l_j = w_i 1_i + w_j l_j - l_i/h_i - l_j/h_j. \tag{10}$$

The data transmission B is

$$B = r_i l_i + r_j l_j = l_i \log h_i w_i + l_j \log h_j w_j. \tag{11}$$

There are two cases to improve the schedule, i.e., consume E while transmit more than B, or transmit B while consume less than E. We consider the formal case by finding a common water level w for both epochs and show the data transmission is increased. The other case is similar and left to the readers.

The common w can be computed as follows:

$$w = \frac{w_i l_i + w_j l_j}{l_i + l_j}. \tag{12}$$

Now, replace both w_i and w_j with w in two epochs. It is easy to check that the consumed energy does not change after the replacement. But the new transmitted data B' is changed to

$$B' = l_i \log h_i w + l_j \log h_j w.$$

The difference is

$$\begin{aligned}\Delta B &= B - B' \\ &= l_i \log w_i + l_j \log w_j - \left(l_i + l_j\right) \log w \\ &= (l_1 + l_2)\left(\frac{l_1}{l_1 + l_2} \log w_1 + \frac{l_2}{l_1 + l_2} \log w_2 \right. \\ &\qquad \left. - \log\left(\frac{l_1}{l_1 + l_2} w_1 + \frac{l_2}{l_1 + l_2} w_2\right)\right) \\ &\leq 0.\end{aligned}$$

The last inequality follows from the fact that the function log is a concave function.

As a conclusion, the water levels of any two epochs can be equalized to transmit more with the same amount of energy consumption as long as the casuality constraints allow.

Appendix C
Proof of Lemma 2

We prove the first half, i.e., if $w_i < w_j$ then there is an arrival point in between, and the second half is symamtctic.

In the seek of contradiction, we assume there is no arrival point between epochs i and j. Then, the water levels of the two epochs can be equalized by moving a certain amount of data transmitted in epochs j to i. According to Lemma 1, such equalization improves on the original schedule. We now show such equalization satisfies the *casuality constraints*. First, every packet is finished before its deadline, because more data is transmitted in an earlier epoch. Second, no packet will be transmitted before its arrival time, because no packet is moved forward across an arrival point. This conflicts the optimality of the policy.

APPENDIX D
PROOF OF LEMMA 3

We prove (1) by contradiction. Assume, in $\mathbf{w}^{\text{opt}}$, two water levels $w_i^{\text{opt}} < w_j^{\text{opt}}$ are used in epochs i and j which are contained inside set $H - H'$ of task T_k. Then, the two water levels can be equalized by moving some amount of packets from epochs j to i, since between epochs transmits only data from T_k. By doing this, the optimal water level $\mathbf{w}^{\text{opt}}$ is improved according to Lemma 1, which is a contradiction. Then, we prove (2). By contradiction, we assume $\mathbf{w}^{\text{opt}}$ is lower than w in the epoch x, where epoch x is contained inside in H'. Now consider epoch y in $H-H'$, since both $x \in H$ and $y \in H$, we can always move some amount of data transmission from epoch y to x without violet the causality constraints. Since the equalization between x and y improves $\mathbf{w}^{\text{opt}}$, contradicting its optimality, therefore (2) is true as well.

APPENDIX E
PROOF OF THEOREM 1

We prove (1) by contradiction. Assume $\mathbf{w}^{\text{opt}}$ is the optimal water level used in $I[i,j]$ and $\mathbf{w}^{\text{opt}} \neq W[i,j]$ in some epochs inside $[a_i, d_{q_j})$. There must be an epoch $[e_k, e_{k+1}) \subseteq [a_i, d_{q_j})$ where the optimal water level $\mathbf{w}^{\text{opt}} > W[i,j]$. Because if $\mathbf{w}^{\text{opt}} \leq W[i,j]$ for entire $I[i,j]$, then $\int_{t\in[a_i,d_{q_j})} \log(w_t^{\text{opt}} h_t)\,\mathrm{d}t < \int_{t\in[a_i,d_{q_j})} \log(W[i,j]h_t)\,\mathrm{d}t$, which implies some packets must miss their deadlines. Therefore, $\mathbf{w}^{\text{opt}} > W[i,j]$ holds in epoch $[e_k, e_{k+1}) \subseteq [a_i, d_{q_j})$. We then extend this epoch $[e_k, e_{k+1})$ to the longest time interval $[e_u, e_v)$ where every epoch has their $\mathbf{w}^{\text{opt}} > W[i,j]$. Note, $[a_i, d_{q_j})$ may not contain the time interval $[e_u, e_v)$ or vice versa. Thus, the water level increases/decreases at e_u/e_v. Otherwise, we can extent the $[e_u, e_v)$ to be a larger time interval. Note, it is possible that water level is undefined in $[e_{u'}, e_u)$ or in $[e_v, e_{v'})$, for some $u' < u$ and $v' > v$.

By Lemma 2, e_u is an arrival point, or an arrival point is inside $[e_{u'}, e_u)$, which is assumed to be a_p. Similarly, a deadline point d_{q_q} is at e_v or inside $[e_v, e_{v'})$. Thus, there must exist a data interval $[a_p, d_{q_q})$, and its water level is no higher than $W[i,j]$, because the $I[i,j]$ is the highest water level interval. However, we have $\mathbf{w}^{\text{opt}} \geq W[i,j]$ for every epoch in $[e_u, e_v)$ and $\mathbf{w}^{\text{opt}} > W[i,j]$ for epoch $[e_k, e_{k+1}) \subseteq [e_u, e_v)$

$$\begin{aligned}
&\int_{t\in\left[a_p,d_{q_q}\right)} \log\left(\left(w_t^{\text{opt}} h_t\right)\mathrm{d}t = \int_{t\in[e_u,e_v)} \log w_t^{\text{opt}} h_t\right)\mathrm{d}t \\
&> \int_{t\in[e_u,e_v)} \log\left(W\left[i,j\right]h_t\right)\mathrm{d} \\
&= \int_{t\in\left[a_p,d_{q_q}\right)} \log\left(W\left[i,j\right]h_t\right)\mathrm{d} \\
&\quad t > B\left[p,q\right].
\end{aligned}$$

Thus, the optimal water level $\mathbf{w}^{\text{opt}}$ transmits more data than the $B[p,q]$ in the data interval $I[p,q]$. We therefore conclude that there must be a packet P_x not belonging to $S[i,j]$ is transmitted in the data interval $I[p,q]$. Packet P_x either arrives before a_p or has a deadline after d_{q_q}. This contradicts Lemma 3. Therefore, $W[i,j]$ is the optimal water level for every epoch in $I[i,j]$. The statement (1) is proved.

According to Algorithm 2, the water level $W[i,j]$ transmits exactly $B[i,j]$ data in $I[i,j]$, and all packets in $S[i,j]$ have arrival times and deadlines inside $I[i,j]$. Hence, packets in $S[i,j]$ must be transmitted in $I[i,j]$.

REFERENCES

[1] X. Peng *et al.*, "BOAT: A block-streaming app execution scheme for lightweight IoT devices," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 1816–1829, Jun. 2018.

[2] J. Ren, H. Guo, C. Xu, and Y. Zhang, "Serving at the edge: A scalable IoT architecture based on transparent computing," *IEEE Netw.*, vol. 31, no. 5, pp. 96–105, Aug. 2017.

[3] D. Zhang, R. Shen, J. Ren, and Y. Zhang, "Delay-optimal proactive service framework for block-stream as a service," *IEEE Wireless Commun. Lett.*, vol. 7, no. 4, pp. 598–601, Aug. 2018, doi: 10.1109/LWC.2018.2799935.

[4] D. Zhang, Y. Zhou, and Y. Zhang, "A multi-level cache framework for remote resource access in transparent computing," *IEEE Netw.*, vol. 32, no. 1, pp. 140–145, Jan./Feb. 2018.

[5] J. Liu, Y. Zhou, and D. Zhang, "TranSim: A simulation framework for cache-enabled transparent computing systems," *IEEE Trans. Comput.*, vol. 65, no. 10, pp. 3171–3183, Oct. 2016.

[6] J. Jin, J. Luo, Y. Li, and R. Xiong, "COAST: A cooperative storage framework for mobile transparent computing using device-to-device data sharing," *IEEE Netw.*, vol. 32, no. 1, pp. 133–139, Jan./Feb. 2018.

[7] R. W. Ahmad *et al.*, "Enhancement and assessment of a code-analysis-based energy estimation framework," *IEEE Syst. J.*, to be published, doi: 10.1109/JSYST.2018.2823733.

[8] P. Serrano, A. Garcia-Saavedra, G. Bianchi, A. Banchs, and A. Azcorra, "Per-frame energy consumption in 802.11 devices and its implication on modeling and design," *IEEE/ACM Trans. Netw.*, vol. 23, no. 4, pp. 1243–1256, Aug. 2015.

[9] A. El Gamal, C. Nair, B. Prabhakar, E. Uysal-Biyikoglu, and S. Zahedi, "Energy-efficient scheduling of packet transmissions over wireless networks," in *Proc. IEEE INFOCOM*, Jun. 2002, pp. 1773–1782.

[10] B. Prabhakar, E. U. Biyikoglu, and A. El Gamal, "Energy-efficient transmission over a wireless link via lazy packet scheduling," in *Proc. IEEE INFOCOM*, vol. 1, Apr. 2001, pp. 386–394.

[11] E. Uysal-Biyikoglu, B. Prabhakar, and A. El Gamal, "Energy-efficient packet transmission over a wireless link," *IEEE/ACM Trans. Netw.*, vol. 10, no. 4, pp. 478–499, Aug. 2002.

[12] E. Uysal-Biyikoglu and A. El Gamal, "On adaptive transmission for energy efficiency in wireless data networks," *IEEE Trans. Inf. Theory*, vol. 50, no. 12, pp. 3081–3094, Dec. 2004.

[13] W. Chen, M. J. Neely, and U. Mitra, "Energy efficient scheduling with individual packet delay constraints," in *Proc. IEEE INFOCOM*, May 2007, pp. 1136–1144.

[14] M. A. Zafer and E. Modiano, "A Calculus approach to minimum energy transmission policies with quality of service guarantees," in *Proc. IEEE INFOCOM*, vol. 1, Mar. 2005, pp. 548–559.

[15] M. A. Zafer and E. Modiano, "A calculus approach to energy-efficient data transmission with quality-of-service constraints," *IEEE/ACM Trans. Netw.*, vol. 17, no. 3, pp. 898–911, Jun. 2009.

[16] A. Fu, E. Modiano, and J. N. Tsitsiklis, "Optimal transmission scheduling over a fading channel with energy and deadline constraints," *IEEE Trans. Wireless Commun.*, vol. 5, no. 3, pp. 630–641, Mar. 2006.

[17] M. Zafer and E. Modiano, "Minimum energy transmission over a wireless channel with deadline and power constraints," *IEEE Trans. Autom. Control*, vol. 54, no. 12, pp. 2841–2852, Dec. 2009.

[18] M. Zafer and E. Modiano, "Delay-constrained energy efficient data transmission over a wireless fading channel," in *Proc. Inf. Theory Appl. Workshop*, 2007, pp. 289–298.

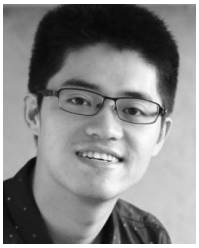
[19] F. Shan, J. Luo, and X. Shen, "Optimal energy efficient packet scheduling with arbitrary individual deadline guarantee," *Comput. Netw.*, vol. 75, no. 2014, pp. 351–366. Dec. 2014

[20] D. Zhang *et al.*, "Two time-scale resource management for green Internet of Things networks," *IEEE Internet Things J.*, to be published, doi: 10.1109/JIOT.2018.2842766.

[21] D. Zhang *et al.*, "Utility-optimal resource management and allocation algorithm for energy harvesting cognitive radio sensor networks," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3552–3565, Dec. 2016.

[22] D. Tse and P. Viswanath, *Fundamentals of Wireless Communication*. Cambridge, U.K.: Cambridge Univ. Press, 2005.

[23] S. P. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.



Feng Shan (M'17) received the Ph.D. degree in computer science from Southeast University, Nanjing, China, in 2015.

He is currently an Assistant Professor with the School of Computer Science and Engineering, Southeast University. He was a Visiting Scholar with the School of Computing and Engineering, University of Missouri–Kansas City, Kansas City, MO, USA, from 2010 to 2012. His current research interests include energy harvesting, wireless power transfer, swarm intelligence, and algorithm design and analysis.

Junzhou Luo (M'07) received the B.S. degree in applied mathematics and the M.S. and Ph.D. degrees in computer network from Southeast University, Nanjing, China, in 1982, 1992, and 2000, respectively.

He is a Full Professor with the School of Computer Science and Engineering, Southeast University. His current research interests include next generation network architecture, network security, cloud computing, and wireless LANs.

Prof. Luo is the Chair of the ACM Special Interest Group on Data Communication China, the Co-Chair of the Technical Committee on Computer Supported Cooperative Work in Design of the IEEE Systems, Man, and Cybernetics Society, and a member of the IEEE Computer Society and ACM.

Jiahui Jin (M'17) received the Ph.D. degree in computer science from Southeast University, Nanjing, China, in 2015.

He is an Assistant Professor with the School of Computer Science and Engineering, Southeast University. He had been a Visiting Ph.D. Student with the University of Massachusetts at Amherst, Amherst, MA, USA, from 2012 to 2014. His current research interests include large-scale data processing, distributed systems, and parallel task scheduling.

Weiwei Wu (M'14) received the B.Sc. degree in computer science from the South China University of Technology, Guangzhou, China, in 2006, and the Ph.D. degree in computer science from the City University of Hong Kong, Hong Kong, and the University of Science and Technology of China, Hefei, China, in 2011.

He was the Post-Doctoral Researcher with the Mathematical Division, Nanyang Technological University, Singapore, in 2012. He is currently an Associate Professor with Southeast University, Nanjing, China. His current research interests include optimizations and algorithm analysis, wireless communications, crowdsourcing, cloud computing, reinforcement learning, game theory, and network economics.