

Optimal On-Off Control for a Class of Discrete Event Systems with Real-Time Constraints

Lei Miao · Lijian Xu · Dingde Jiang

Received: date / Accepted: date

Abstract This paper studies an optimal ON-OFF control problem for a class of discrete event systems with real-time constraints. Our goal is to minimize the overall costs, including the operating cost and the wake-up cost, while still guaranteeing the deadline of each individual task. In particular, we consider the homogeneous case in which it takes the same amount of time to serve each task and each task needs to be served by d seconds upon arrival. The problem involves two subproblems: (i) finding the best time to wake up the system and (ii) finding the best time to let the system go to sleep. We study the two subproblems in both off-line and on-line settings. In the off-line case that all task information is known a priori, we combine sample path analysis and dynamic programming to come up with the optimal solution. In the on-line scenario where future task information is completely unknown, we show that the optimal time to wake up the system can be obtained without relying on fu-

The authors' work is supported in part by a start-up funding provided by Middle Tennessee State University.

Lei Miao
Mechatronics Engineering
1301 East Main Street, Box 19
Murfreesboro, TN 37132-0001, USA
Tel.: +1-615-898-2256
Fax: +1-615-898-5697
E-mail: lei.miao@mtsu.edu

Lijian Xu
Department of Electrical and Computer Engineering Technology
Farmingdale State College, Farmingdale, NY 11735, USA
E-mail: xul@farmingdale.edu

Dingde Jiang
School of Astronautics and Aeronautic
University of Electronic Science and Technology of China, Chengdu 611731, China
E-mail: jiangdd@uestc.edu.cn

ture task arrivals. We also perform competitive analysis for on-line control and derive the competitive ratios for both deterministic and random controllers.

Keywords discrete event systems · real-time systems · quality-of-service · optimization · dynamic programming · competitive ratio

1 Introduction

There exists a large amount of Discrete Event Systems (DESs) that involve allocation of resources to satisfy real-time constraints. One commonality of these DESs is that certain tasks must be completed by their deadlines in order to guarantee Quality-of-Service (QoS). Examples arise in wireless networks and computing systems, where communication and computing tasks must be transmitted/processed before the information they contain becomes obsolete [1] [2], and in manufacturing systems, where manufacturing tasks must be completed before the specified time in the production schedule [3]. Another commonality of these DESs is that they all require the minimization of cost (e.g., energy). An interesting question then arises naturally: *how can we allocate resources to such DESs so that the cost is minimized and the real-time constraints are also satisfied?* To answer this question, one often has to study the trade-off between minimizing the cost and satisfying the real-time constraints: processing the tasks at a higher speed makes it easier to satisfy the real-time constraints but harder to reduce the cost; conversely, processing the tasks at a lower speed makes it harder to satisfy the real-time constraints but easier to reduce the cost. This trade-off is often referred to as the energy-latency trade-off and has been widely studied in the literature [1] [4] [5].

In this paper, our objective is to utilize the energy-latency trade-off to minimize the cost while guaranteeing the real-time constraint for each task. Different from most existing papers that assume the system's service rate (the control variable) is a continuous function of time, we assume that the DES only operates at one of the two states: ON and OFF. One motivating example of such DES is wireless sensor networks, in which operation simplicity must be maintained. For example, the radio of a ZigBee wireless device can be either completely off or transmitting at a fixed-rate, e.g., 250kb/s in the 2.4GHz band. Another difference between this paper and others is that we assume that a wake-up cost is incurred whenever the system transits from the OFF state to the ON state.

In this paper, we solve both off-line and on-line optimal ON-OFF control problems. Our main contributions are two-fold: (i) We combine *sample path analysis* and *Dynamic Programming* (DP) to obtain the optimal off-line solution and (ii) We perform competitive analysis and derive the competitive ratios of both deterministic and random on-line controllers. Some results of this paper are previously shown in two conference papers: [6] and [7], which primarily focus on off-line control. One new contribution of this paper is the competitive analysis for on-line control. Another new contribution is that we introduce an idling cost in the system model. We point out that the addition

of this idling cost makes our problem formulation more realistic because it often exists in real-world applications. For example, energy is consumed when a motor is spinning without any load attached and when a sensor is turned on, but not actively processing information.

In this journal version, we improve some proofs to incorporate the idling cost; we also move all the proofs and tables to the appendix in order to enhance the continuity of the analysis in the paper. The organization of the rest of the paper is as follows: in Section 2, we discuss related work; we introduce the system model and formulate our optimization problem in Section 3; the off-line and on-line results are presented in Sections 4 and 5, respectively; finally, we conclude in Section 6.

2 Related Work

There are two lines of work that are closely related to this paper. One is transmission scheduling for wireless networks, in which the transmission rate of a wireless device is adjusted so as to minimize the transmission cost and satisfy real-time constraints. This line of work is initially studied in [8] with follow-up work in [4] where a homogeneous case is considered, assuming all packets have the same deadline and number of bits. By identifying some properties of this convex optimization problem, Gamal et al. propose the “MoveRight” algorithm in [4] to solve it iteratively. However, the rate of convergence of the MoveRight algorithm is only obtainable for a special case of the problem when all packets have identical energy functions; in general the MoveRight algorithm may converge slowly. Zafer et al. [9] study an optimal rate control problem over a time-varying wireless channel, in which the channel state was modeled as a Markov process. In particular, they consider the scenario that B units of data must be transmitted by a common deadline T , and they obtain an optimal rate-control policy that minimizes the total energy expenditure subject to short-term average power constraints. In [10] and [11], the case of identical arrival time and individual deadline is studied by Zafer et. al. In [12], the case of identical packet size and identical delay constraint is studied by Neely et. al. They extend the result for the case of individual packet size and identical delay constraint in [13]. In [5], Zafer et. al. use a graphical approach to analyze the case that each packet has its own arrival time and deadline. However, there are certain restrictions in their setting; for example, the packet that arrives later must have later deadlines. Wang and Li [14] analyze scheduling problems for bursty packets with strict deadlines over a single time-varying wireless channel. Assuming slotted transmission and changeable packet transmission order, they are able to exploit structural properties of the problem to come up with an algorithm that solves the off-line problem. In [15], Poulakis et. al. also study energy efficient scheduling problems for a single time-varying wireless channel. They consider a finite-horizon problem where each packet must be transmitted before D_{\max} . Optimal stopping theory is used to find the optimal start transmission time between $[0, D_{\max}]$ so as to minimize the

expected energy consumption and the average energy consumption per unit of time. Zhong and Xu [16] formulated optimization problems that minimize the energy consumption of a set of tasks with task-dependent energy functions and packet lengths. In their problem formulation, the energy functions include both transmission energy and circuit power consumption. To obtain the optimal solution for the off-line case with backlogged tasks only, they develop an iterative algorithm RADB whose complexity is $O(n^2)$ (n is the number of tasks). The authors show via simulation that the RADB algorithm achieves good performance when used in on-line scheduling. [1] studies a transmission control problem for task-dependent cost functions and arbitrary task arrival time, deadline, and number of bits. They propose a GCTDA algorithm that solves the off-line problem efficiently by identifying certain critical tasks. The GCTDA algorithm is an extension to the CTDA algorithm [17] designed by Mao and Cassandras for dynamic voltage scaling related applications. They extend the CTDA algorithm to multilayer scenarios in [18]. Our model is different from all the above works by letting the system operate in one of the discrete modes and also including a wake-up cost at each time instant that the system transitions from OFF to ON state.

The other line of research studies On-OFF scheduling in Wireless Sensor Networks (WSNs). Solutions in the Medium Access Control (MAC) layer, such as the S-MAC protocol [19], have been developed to coordinate neighboring sensors' ON-OFF schedule in order to reduce both energy consumption and packet delay. These approaches do not provide specific end-to-end latency guarantee. In [20], routing problems are considered in WSNs where each sensor switches between ON and OFF states. The authors formulate an optimization problem to pick the best path that minimizes the weighted sum of the expected energy cost and the exponent of the latency probability. In another work in [21], Ning and Cassandras formulate a dynamic sleep control problem in order to reduce the energy consumed in listening to an idle channel. The idea is to sample the channel more frequently when it is likely to have traffic and less frequently when it is not. The authors extend their work in [22], by formulating an optimization problem with the goal of minimizing the expected total energy consumption at the transmitter and the receiver. Dynamic programming is used to come up with an optimal policy that is shown to be more effective in cost saving than the fixed sleep time. [23] studies the ON-OFF scheduling in wireless mesh networks. By assuming a fixed routing tree topology used for task transmission, each child in the tree knows exactly when its parents will wake up, and the traffic is only generated by the leaves of the tree, the authors formulate and solve an optimization problem that minimizes the total transmission energy cost while satisfying the latency and maximum energy constraints on each individual node. The major difference between this paper and the existing ones in this line of research is that we study a system with a real-time constraint for each individual task. To the best of our knowledge, ON-OFF scheduling with a real-time constraint for each individual task has not been studied extensively.

It is worth noting that there also exists papers related to the service rate control problem, in which the optimal service rate policy of either single-server or multi-server queueing systems are derived in order to minimize an average cost. A recent representative work along this line can be found in [24] where Xia et al. study the service rate problem for tandem queues with power constraints. They formulate the model as a Markov decision process with constrained action space and use sensitivity-based optimization techniques to derive the conditions of optimal service rates, the optimality of the vertexes of the feasible domain for linear and concave operating cost, and an iterative algorithm that provides the optimal solution. Our problem formulation is different from these works in two aspects: (i) We consider tasks with real-time constraints and (ii) We include system wake-up cost on top of the service cost.

3 System Model and Problem Formulation

We consider a finite horizon scenario that a DES processes N tasks with real-time constraints. In particular, task i , $i = 1, \dots, N$, has arrival time a_i (generally random), deadline $d_i = a_i + d$, and B number of operations. Both d and B are constants. In the *off-line* setting, we assume that the task arrival time a_i is known to the controller a priori. The DES can only operate in one of the two modes: ON and OFF. When it is in the OFF mode, there is no operating cost associated. When it is in the ON or active mode, the system can either be *busy* or *idling*. When the system is busy, it processes the tasks at a constant rate R with fixed operating cost C_B per unit time. When the system is idling, no tasks are waiting to be served, and the system cost is C_I ($C_I \leq C_B$) per unit time. Furthermore, we assume that whenever a transition from the OFF mode to the ON mode occurs, a fixed wake-up cost C_W is incurred; examples of such costs include: the large amount of current (known as inrush current) required when a motor is turned on, the energy needed to initialize electric circuits when RF radio is turned on in a wireless device, and so on. Note that the wake-up cost may also include system wearout cost, if the system can only be turned on for certain number of times during its lifetime. In our previous work in [6] and [7], $C_I = C_B$. As we will show later, when C_I is different from C_B , it does not make the analysis significantly harder, and the off-line optimal solution can still be obtained by DP.

Our system model above is quite generic and is applicable to a wide range of engineering applications; for example, one can use ultra-low power wake-up receivers [25] to conserve energy in WSNs. Next, we formulate the off-line optimization problem.

As we mentioned earlier, the task information is known to the controller a priori in the off-line setting. Our objective is to find the optimal ON and OFF time periods so as to (i) finish all the tasks by their deadlines and (ii) minimize the cost.

Definition 1 Suppose the system is woken up at t_1 , put to sleep at t_2 ($t_1 < t_2$), and kept active from t_1 to t_2 . Then, we call the time interval $[t_1, t_2]$ an **Active Period (AP)**.

Definition 2 In any AP, the periods during which the system is actively serving tasks are known as **Busy Periods (BPs)**. The rest of the time periods in that AP are known as **Idle Periods (IPs)**.

Let $r(t)$ be the rate that the system is capable of serving tasks at time t . It is piecewise constant and at any given time t , it can only be either 0 (when the system is OFF) or R (when the system is ON). See Fig. 1 for an illustration of how $r(t)$ looks like and how the APs are formed. Note that $r(t)$ is not the actual service rate since the system is only serving tasks during the **BPs**, not the **IPs**.

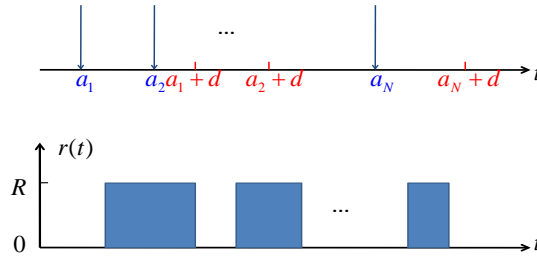


Fig. 1: Off-line control illustration.

We now introduce the control variables. Our first control variable is α , the number of APs. The second control variable is a $\alpha \times 2$ array \mathbf{t} that contains 2α time instants. These time instants satisfy:

$$t_{i,1} < t_{i,2} < t_{j,1} < t_{j,2}, \quad \forall i, j \in \{1, \dots, \alpha\}, \quad i < j$$

and define α number of APs. See Fig. 1 for illustration. The off-line problem $Q(1, N)$ can then be formulated:

$$\begin{aligned} \min_{\alpha, \mathbf{t}} \quad & J = \alpha C_W + \sum_{i=1}^{\alpha} [C_I(t_{i,2} - t_{i,1} - \tau_{i,B}) + C_B \tau_{i,B}] \\ \text{s.t.} \quad & \int_{\max(a_j, x_{j-1})}^{x_j} r(t) dt = B, \\ & x_j \leq d_j, \quad x_0 = 0, \quad j = 1, \dots, N \\ & r(t) = R \sum_{i=1}^{\alpha} [u(t - t_{i,1}) - u(t - t_{i,2})] \end{aligned}$$

where x_j is the departure time of task j , $u(t)$ is the unit step function, and $\tau_{i,B}$ is the length of the busy periods in the i -th AP. The first constraint ensures that exactly B number of operations are executed for each task. The second one is the real-time constraint. The third one makes sure that the processing rate is R only during each AP. Note that $\tau_{i,B}$ is dependent on the number of tasks served in AP_i . To represent $\tau_{i,B}$, we use N_i^S and N_i^E to denote the first (starting) task and the last (ending) task in AP_i , respectively:

$$\begin{aligned} N_i^E &= \arg \max_{j \in \{1, \dots, N\}} (d_j \leq t_{i,2}) \\ N_i^S &= \arg \min_{j \in \{1, \dots, N\}} (a_j \geq t_{i,1}) \\ \tau_{i,B} &= \max((N_i^E - N_i^S + 1) \frac{B}{R}, 0) \end{aligned}$$

Notice that $Q(1, N)$ above may not always be feasible. Consider the case that N tasks arrive at the same time and need to be served in d seconds. In order to meet the deadlines of all the tasks, we must have $R \geq \frac{NB}{d}$. Since R is a constant, the condition above obviously is not true when N is large. In this paper, we only consider the case that $Q(1, N)$ is indeed feasible, and we have the following assumption on the task arrival rate.

Assumption 1 *Within any time interval of d seconds, the number of task arrivals must not exceed $\lfloor \frac{d}{\beta} \rfloor$, where $\beta = B/R$ is the time it takes to process a single task.*

We emphasize that d in Assumption 1 is the deadline of each task upon arrival. To make the problem more interesting, we also assume that $\lfloor \frac{d}{\beta} \rfloor > 1$.

Lemma 1 *Under Assumption 1, $P1$ is always feasible.*

$Q(1, N)$ is a hard optimization problem, due to the nondifferentiable terms in the constraints and the objective function. It cannot be easily solved by standard optimization software. In what follows, we will first discuss optimal off-line control, using which we will then establish the results for on-line control.

4 Off-line Control

In this section, we focus on the off-line control problem, in which all task arrivals are known to us a priori. We need to find out when the system should wake up and start to serve the first task in an AP. Similar to the “just-in-time” idea exploited in [4] for adaptive modulation, the system should wake up as late as possible so that it may potentially reduce the active time. The question is how late the system should wake up. This is answered by the following results.

Lemma 2 Suppose that tasks $\{k, \dots, n\}$ are all the tasks served in an AP on the optimal sample path of $Q(1, N)$ and starting the AP at time either t_A or t_B , $a_k \leq t_A < t_B \leq d_k - \beta$, is feasible. Then,

$$C_{k, \dots, n}^A \geq C_{k, \dots, n}^B$$

where $C_{k, \dots, n}^A$ and $C_{k, \dots, n}^B$ are the corresponding costs of serving tasks $\{k, \dots, n\}$ in the AP for the two different starting time t_A and t_B , respectively.

Lemma 2 indicates that an AP on the optimal sample path of $Q(1, N)$ should be started as late as possible. We now utilize this result to figure out when exactly the first task k should be served.

Lemma 3 If tasks $\{k, \dots, n\}$ are all the tasks served in an AP on the optimal sample path of $Q(1, N)$ and the number of task arrivals in $[a_k, d_k - \beta)$ is 0, then the optimal starting time to transmit task k is $d_k - \beta$, i.e.,

$$x_k^* = d_k,$$

where x_k^* is the optimal departure time of task k .

Lemma 3 shows that we can delay the transmission of the first task in an AP to β seconds before its deadline, provided that there are no other arrivals before that time. Next, we discuss the case that there exists other task arrivals before $d_k - \beta$.

Lemma 4 Suppose task k is the first task in an AP on the optimal sample path of $Q(1, N)$ and the number of task arrivals in $[a_k, d_k - \beta)$ is m , $0 < m \leq \lfloor \frac{d}{\beta} \rfloor - 1$. Let

$$\begin{aligned} \delta_j &= \beta(j - k) - (a_j - a_k) \\ z &= \arg \max_{j=k+1, \dots, k+m} \{\delta_j\} \end{aligned} \quad (1)$$

The optimal starting time to serve task k is:

$$\begin{cases} d_k - \beta, & \text{if } \delta_z \leq 0 \\ d_k - \beta - \delta_z, & \text{if } \delta_z > 0 \end{cases}$$

Having discussed when to wake up the system, we now find out when the system should go to sleep. Apparently, the optimal time to end an AP depends on future task information. In what follows, we first establish some results that identify the end of an AP based on future task arrival information.

Lemma 5 If $d_j + C_W/C_I < a_{j+1}$, $j \in \{1, \dots, N - 1\}$, then task j ends an AP on the optimal sample path of $Q(1, N)$.

Lemma 5 basically indicates that if the deadline of task j is at least C_W/C_I seconds apart from the next task arrival, then task j ends an AP on the optimal sample path. Note that this is just a sufficient, but not necessary condition of an AP ending on the optimal sample path. In some cases, whether a task should end an AP is determined by not only the next arrival, but also all subsequent ones. Let $d_0 = -\infty$ and $a_{N+1} = \infty$. We introduce the following definition.

Definition 3 Consecutive tasks $\{k, \dots, n\}$, $1 \leq k \leq n \leq N$, belong to a *super active period* (SAP) in problem $Q(1, N)$ if $d_{k-1} + C_W/C_I < a_k$, $d_n + C_W/C_I < a_{n+1}$, and $d_j + C_W/C_I \geq a_{j+1}$, $\forall j \in \{k+1, \dots, n-1\}$.

Each SAP contains one or more APs. SAPs can be easily identified by simply examining all the task deadlines and arrival times and applying Lemma 5. It implies that instead of working on the original problem $Q(1, N)$, we now only need to focus on each SAP, which is essentially a subproblem $Q(k, n)$.

We now define our decision points in each SAP. A decision point x_t , $t \in \{k, \dots, n-1\}$, is the departure time of task t satisfies $x_t < a_{t+1}$. If $x_t \geq a_{t+1}$, then x_t is not a decision point because the system should stay active at x_t and process task $t+1$. At each decision point, the control is letting the system either go to sleep or stay awake. Let us take a look at some examples, in which $d = 10$, $C_W = 10$, and $C_B = C_I = C = 1$. Note that C_B and C_I could be different in general; for simplicity, we let them equal to each other in the examples. We also assume that $B = R$, i.e., it takes a unit of time to complete a task. Fig. 2 and Fig. 3 show two different sample paths for a simple two-task scenario: $a_1 = 0$ and $a_2 = 19$. In both sample paths, task 1's optimal wake up time is determined by Lemmas 4.2 and 4.3. The only decision point is x_1 , at which the system needs to decide if it should go to sleep or stay awake. In particular, the system in Fig. 2 wakes up at $t_1 = 9$, finishes task 1 at its deadline $d_1 = 10$, stays awake, and finishes task 2 at $t_2 = 20$. The total cost is: $C_W + C(t_2 - t_1) = 21$. In Fig. 3, the system wakes up at $t_1 = 9$, finishes task 1 at its deadline $d_1 = t_2 = 10$, and goes to sleep. Then, it wakes up at $t_3 = 28$ (once again determined by Lemmas 4.2 and 4.3 and finishes task 2 at $t_4 = 29$. The total cost of this case is: $2C_W + C[(t_2 - t_1) + (t_4 - t_3)] = 22$. It is evident that at decision point $x_1 = 10$, the optimal control is to let the system stay awake (shown in Fig. 2).

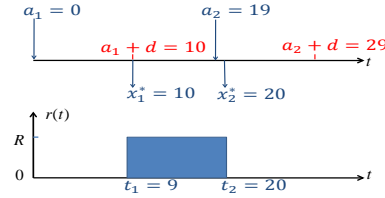


Fig. 2: Sample path #1 of scenario #1.

Now, let us consider another scenario (Fig. 4 and Fig. 5), in which we keep the previous tasks 1 and 2 unchanged and add task 3. Our first decision point is again at $x_1 = 10$. In Fig. 4, the system wakes up at $t_1 = 9$, finishes task 1 at its deadline $d_1 = 10$, stays awake, finishes task 2 at time 20, stays awake, and finally finishes task 3 at time $t_2 = 30$. The total cost is: $C_W + C(t_2 - t_1) = 31$. In Fig. 5, the system wakes up at $t_1 = 9$, finishes task 1 at its deadline $d_1 =$

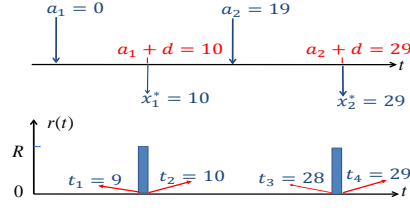


Fig. 3: Sample path #2 of scenario #1.

$t_2 = 10$, and goes to sleep. Then, it wakes up at $t_3 = 28$ and finishes tasks 2 and 3 at $t_4 = 30$. The total cost of this case is: $2C_W + C[(t_2 - t_1) + (t_4 - t_3)] = 23$. It is evident that at decision point $x_1 = 10$, the optimal control is to let the system go to sleep (shown in Fig. 5).

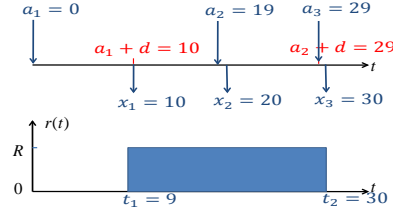


Fig. 4: Sample path #1 of scenario #2.

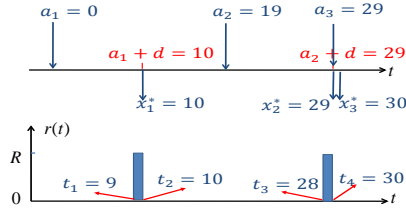


Fig. 5: Sample path #2 of scenario #2.

From the above examples, we can conclude that the optimal decision on if the system should stay awake or go to sleep when it finishes all on-hand tasks depends on future task arrivals (task 3 in the examples above). A first look at the problem seems to suggest that in the worst case, the system may have to make a decision about if it should go to sleep or stay awake after each task departure; the total number of possible sample paths could be as high as 2^N , which makes the problem intractable when N is large. However, a closer look

at the problem indicates that the off-line optimal ON-OFF control problem can be solved by DP, which has been widely used to solve a large class of problems with special structural properties. In the context of DES, however, its usage has been very limited to date. For example, in [17] and [1] where the problem formulation is similar to the one in this paper, both CTDA and GCTDA algorithms are not DP-based. We will show next that for the DES studied by this paper, DP and sample path analysis can be used together to obtain the optimal solution. In particular, it is done by introducing two types of tasks: starting and following.

Definition 4 In problem $Q(k, n)$, where tasks $\{k, \dots, n\}$ form an SAP, the first task of any AP is called a *starting* task. Tasks that are not starting tasks are known as *following* tasks.

Since the case that $k = n$ is trivial, we assume that $k < n$ in our analysis in order to make the problem more interesting. Note that APs contain one task only do not have following tasks. For any task $i \in \{k, \dots, n\}$, it must either be a starting task or a following one. We are interested in finding out the optimal cost of serving tasks $\{i, \dots, n\}$, and we use $Q^S(i, n)$ and $Q^F(i, n)$ to denote the optimization problems of serving tasks $\{i, \dots, n\}$ when task i is a starting and following task, respective. Note that in these two problems, only tasks $\{i, \dots, n\}$ are served and all other tasks in $\{k, \dots, n\}$ are not considered. In problem $Q^F(i, n)$, the system is active when task i arrives; therefore, task i will be served right after its arrival. Let J_i^S and J_i^F be the minimum cost of $Q^S(i, n)$ and $Q^F(i, n)$, respectively. When $i = n$, we can easily calculate J_n^S and J_n^F : $J_n^S = C_W + C_B\beta$, $J_n^F = C_B\beta$. Note that J_n^F does not include the wake-up cost C_W , since by assumption, task n is a following task. The operating cost, $C_B\beta$, is identical in both cases. Suppose that J_i^S and J_i^F , $i \in \{k+1, \dots, n\}$ are both known, the next step is to find J_{i-1}^S and J_{i-1}^F .

We first focus on J_{i-1}^S . By assumption, task $i-1$ is a starting task. We use Lemmas 3 and 4 to find out the optimal starting time of task $i-1$ in problem $Q^S(i-1, n)$. Let the optimal starting time be $s_{i-1,n}^{i-1}$. For tasks in $\{i, \dots, n\}$, find task l that satisfies the following:

$$\begin{aligned} s_{i-1,n}^{i-1} + (j-i+1)\beta &\geq a_j, \forall j \in \{i-1, \dots, l-1\}, \\ \text{and } s_{i-1,n}^{i-1} + (l-i+1)\beta &< a_l \end{aligned} \quad (2)$$

If task l does not exist, then it is a trivial case that the system is always busy serving tasks $\{i-1, \dots, n\}$, and there is a single AP that starts from $s_{i-1,n}^{i-1}$ and ends at $s_{i-1,n}^{i-1} + (n-i+2)\beta$. In this case, $J_{i-1}^S = C_W + (n-i+2)\beta C_B$. We now consider the more interesting case that task l does exist. In particular,

$$J_{i-1}^S = \min(V_{i-1,l}^{SS} + J_l^S, V_{i-1,l}^{SF} + J_l^F) \quad (3)$$

where $V_{i-1,l}^{SS}$ is the cost of serving tasks $\{i-1, \dots, l-1\}$ when task l is a starting task:

$$V_{i-1,l}^{SS} = C_W + (l-i+1)\beta C_B \quad (4)$$

$V_{i-1,l}^{SF}$ is the cost of serving tasks $\{i-1, \dots, l-1\}$ when task l is a following task:

$$V_{i-1,l}^{SF} = C_W + (l-i+1)\beta C_B + [a_l - s_{i-1,n}^{i-1} - (l-i+1)\beta]C_I \quad (5)$$

We now focus on J_{i-1}^F . We emphasize again that in this case, task $i-1$ sees an active system upon its arrival; it will be served right away since it is the first task in $Q^F(i-1, n)$. For tasks in $\{i, \dots, n\}$, find task l that satisfies the following:

$$\begin{aligned} a_{i-1} + (j-i+1)\beta &\geq a_j, \forall j \in \{i-1, \dots, l-1\}, \\ \text{and } a_{i-1} + (l-i+1)\beta &< a_l \end{aligned} \quad (6)$$

Once gain, task l may not exist, and it corresponds to the case that the system is always busy serving tasks $\{i-1, \dots, n\}$. In this case, there is a single AP that starts from a_{i-1} and ends at $a_{i-1} + (n-i+2)\beta$. We have $J_{i-1}^F = (n-i+2)\beta C_B$. We now consider the more interesting case that task l does exist. We have:

$$J_{i-1}^F = \min(V_{i-1,l}^{FS} + J_l^S, V_{i-1,l}^{FF} + J_l^F) \quad (7)$$

where $V_{i-1,l}^{FS}$ is the cost of serving tasks $\{i-1, \dots, l-1\}$ when task l is a starting task:

$$V_{i-1,l}^{FS} = (l-i+1)\beta C_B \quad (8)$$

$V_{i-1,l}^{FF}$ is the cost of serving tasks $\{i-1, \dots, l-1\}$ when task l is a following task:

$$V_{i-1,l}^{FF} = (l-i+1)\beta C_B + [a_l - a_{i-1} - (l-i+1)\beta]C_I \quad (9)$$

In Table 1, we show the algorithm that returns the optimal cost of $Q(k, n)$. This algorithm involves two more algorithms that return the optimal costs of $Q^S(i-1, n)$ (Table 2) and $Q^F(i-1, n)$ (Table 3), respectively.

Theorem 1 J_k^S is the optimal cost of problem $Q(k, n)$.

We have proved that when the algorithm in Table 1 stops, J_k^S is the optimal cost of problem $Q(k, n)$. The corresponding optimal control, i.e., the starting time and ending time of each AP, can be traced back iteratively by identifying the J_l^S or J_l^F that each J_{i-1}^S or J_{i-1}^F points to. The procedure is provided in Table 4.

Next, we use the example in Fig. 4 and Fig. 5 to show how the above algorithms work. We have three tasks 1, 2, and 3 belong to a SAP ($k=1$ and $n=3$). Initially, $J_n^S = J_3^S = C_B + C\beta = 11$, and $J_n^F = J_3^F = C\beta = 1$. In the first iteration ($i=n=3$), we calculate J_{i-1}^S and J_{i-1}^F . To calculate $J_{i-1}^S = J_2^S$, we first figure out $s_{2,3}^2 = 28$. Then, we find out that no task l satisfies (2). Therefore, tasks 2 and 3 form a single AP in problem $Q^S(2, 3)$, and $J_2^S = 12$. To calculate $J_{i-1}^F = J_2^F$, we identify that task $l=3$ satisfies (6). We then use (7) to obtain $J_2^F = \min(V_{2,3}^{FS} + J_3^S, V_{2,3}^{FF} + J_3^F) = \min(1 + J_3^S, 10 + J_3^F) = 11$. In the final iteration ($i=n-1=2$), we only need to calculate $J_{i-1}^S = J_1^S$. Because $s_{1,3}^1 = 9$ and task $l=2$ satisfies (2), we use (3) to calculate J_1^S :

$J_1^S = \min(V_{1,2}^{SS} + J_2^S, V_{1,2}^{SF} + J_2^F) = \min(11 + J_2^S, 20 + J_2^F) = 23$. This is the optimal cost obtained in Fig. 5. If we follow the procedure in Table 4, we will get the exact same optimal solution as shown in Fig. 5. The details are omitted.

Next, we use simulation results to show how the optimal solution performs compared with a naive approach, in which the controller simply goes to sleep when there is no backlog and wakes up when a new task arrives. Let optimal to naive ratio be the ratio between the optimal cost and the cost of the naive controller. Fig. 6 shows how the optimal to naive ratio varies when the task arrival process and the wake-up cost C_W change. In the simulation, we have 100 runs that correspond to 100 maximum interarrival time from $1ms$ to $100ms$ with step size $1ms$. 1000 tasks and various C_W values are used in each run. The interarrival time between two adjacent tasks is uniformly distributed between 0 and the maximum interarrival time in each run. The values of the other parameters are as follows: $d = 20ms$, $C_B = 30mW$, $C_I = 100\mu W$, and $\beta = 1ms$.

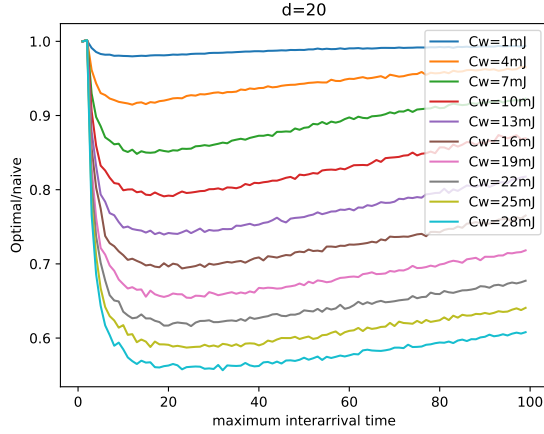


Fig. 6: Optimal to naive ratio under various wake-up cost and interarrival time

We have a couple of observations. First, the cost saving of the optimal solution is greater when C_W is larger. Second, the maximum cost saving occurs when the interarrival time is not too small or too large: when it is too small, a single AP will be sufficient to complete all the tasks, and the optimal and the naive solutions are essentially the same; when it is too large, many APs are needed, and the advantage of the optimal controller gets smaller. As we can see from the result, the cost saving of the DP algorithm in the $C_W = 28mJ$ case is as large as 50%, and it will be ever greater when C_W is higher.

5 On-line Control

In the previous section, we combine structural properties of the optimal sample path and dynamic programming to find the optimal solution to the off-line control problem. In this section, we study on-line control where future task arrival information is unknown to the controller. Essentially, the controller needs to decide the starting time and ending time for each AP.

5.1 Starting an AP

We first focus on the following questions: how can we determine the best time to start an AP in on-line control and how different is it from the optimal time in off-line control?

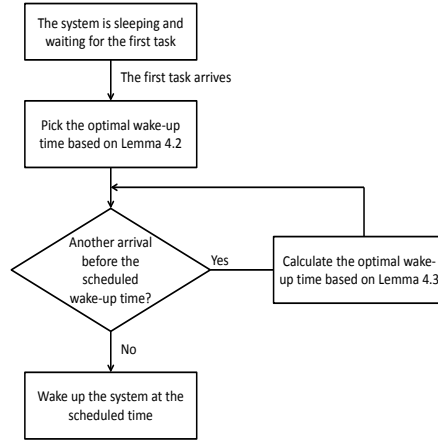


Fig. 7: On-line control: starting an AP

Fig. 7 shows the proposed on-line control mechanism for determining the wake-up time. It is an iterative algorithm that dynamically adjusts the wake-up time based upon the backlog and the newly available task information. Initially, right after the first task arrives, the scheduled wake-up time is $a_1 + d - \beta$ (determined by Lemma 3). If there are other task arrivals before the scheduled wake-up time, the controller will recalculate the wake-up time using the results in Lemma 4; otherwise, the system will be woken up at the scheduled time.

Lemma 6 *Suppose that tasks $\{k, \dots, n\}$ form an AP on the optimal sample path of $Q(1, N)$. If the system is OFF before task k arrives in on-line control, then the wake-up time returned by the on-line control mechanism in Fig. 7 is optimal.*

Lemma 6 indicates that for on-line control, the lack of future task information does not incur any penalty when starting an AP: the optimal time to start an AP can be determined iteratively using the backlog and the newly available task information.

We now turn our attention to ending an AP in on-line control.

5.2 Ending an AP

When all backlogged tasks have been served in an on-line setting, the controller needs to decide when to end an AP and put the system to sleep. This decision depends on future task information and the values of idling cost C_I and wake-up cost C_W . For example, if the next task $t + 1$ arrives very soon, the optimal control at decision point x_t might be letting the system stay active; conversely, if the next task $t + 1$ arrives after a long time, then the system perhaps should go to sleep at decision point x_t .

When some future task information is known, techniques such as Receding Horizon Control (RHC) can be utilized to make decisions. In this paper, we focus on the scenario that no future task information is available at all.

In general, the control at each decision point is the following: let the system stay awake for another θ_t seconds. If no task arrives within the θ_t seconds, then put the system to sleep after θ_t seconds; o.w., serve the newly arrived tasks and wait for the next decision point. Note that the subscript t indicates θ_t could be different at each decision point.

Let J^* be the optimal cost of the off-line problem $Q(1, N)$ and \tilde{J} be the cost of the on-line controller. Our objective is to develop competitive on-line controllers which can quantify their worst-case performance deviation from the optimal off-line solution.

One challenge of competitive analysis is to find out the worst-case scenario. In our problem, the unnecessary cost in on-line control occurs when the system is idling: the controller must decide if and when to sleep. Therefore, the worst case occurs when each AP contains only one task so that the decision has to be made over and over again for every single task. This property actually simplifies our analysis, and in particular, we tackle the competitive ratio problem from two different aspects: a deterministic controller and a randomized one.

5.2.1 Deterministic Controller

We first consider a deterministic controller in which θ_t is a fixed constant value θ . The on-line controller is *c-competitive* if $\tilde{J}(I, \theta) \leq cJ^*$, $\forall I \in \mathcal{I}$, where \mathcal{I} is the set of all possible task arrival instances and I is one task arrival instance. c is called the competitive ratio of the deterministic on-line controller and is essentially the *upper-bound (i.e., worst case)* of the ratio between the on-line cost \tilde{J} and the off-line optimal cost J^* .

Lemma 7 *The best competitive ratio c^* of a deterministic controller is obtained when $\theta = C_W/C_I$, and $\lim_{N \rightarrow \infty} c^* = (2 + \gamma)/(1 + \gamma)$, where N is the number of tasks and $\gamma = C_B\beta/C_W$.*

Lemma 7 shows that the competition ratio of a deterministic algorithm depends on the ratio between $C_B\beta$, the cost of serving one task, and C_W , the cost of waking up the system. If this ratio is very small, then the competitive ratio is close to 2; if the ratio is very large, then the competitive ratio is close to 1.

5.2.2 Randomized Controller

In a different methodology, we assume that θ_t is determined by a randomized algorithm that returns a value based on certain probability distribution P . During on-line control, the controller essentially is playing a game with an adversary (i.e., the task arrival process). Our job is to find the optimal probability distribution and the corresponding competitive ratio. We point out that the competitive ratio of a randomized on-line algorithm A is defined with respect to a specific type of adversary. In this paper, we assume an oblivious adversary [26], in which the worst instance for the randomized algorithm A is chosen without the knowledge of the realization of the random variable used by A . We say randomized algorithm A is c -competitive if $E_P[\tilde{J}(A, I)] \leq cJ^*(I), \forall I \in \mathcal{I}$, where $\tilde{J}(A, I)$ is the cost of algorithm A under task arrival instance I in on-line control and $J^*(I)$ is the corresponding off-line optimal cost. Note that the task arrival instance I must be fixed before the expectation is taken. The competition ratio of randomized algorithm A_P (algorithm A using probability distribution P) is:

$$c(A_P) = \sup_{I \in \mathcal{I}} \frac{E_P[\tilde{J}(A_P, I)]}{J^*(I)}$$

Our goal is to find the best possible probability distribution that yields the best competitive ratio c^* :

$$c^* = \inf_P \sup_{I \in \mathcal{I}} \frac{E_P[\tilde{J}(A_P, I)]}{J^*(I)}$$

This is essentially a minimax problem, and one way of solving it is to use Yao's minimax principle [27], which states: a randomized algorithm may be viewed as a random choice between deterministic algorithms; in particular, the competitive ratio of a randomized algorithm against any oblivious adversary is the same as that of the best deterministic algorithm under the worst-case distribution of the adversary's input. In our case, the adversary's input is the task arrival instance after each AP. Let its probability distribution be G . Using Yao's principle and von Neumann minimax theorem, we get:

$$c^* = \sup_G \inf_{A \in \mathcal{A}} \frac{E_G[\tilde{J}(A, I_G)]}{J^*(I_G)} \quad (10)$$

where \mathcal{A} is the set of all randomized algorithms, I_G is a specific task arrival instance under probability distribution G , and the expectation is now performed with respect to G . We now use the following lemma to find c^* .

Lemma 8 *The best competitive ratio c^* of a randomized controller is obtained when θ_t is a random variable X , whose probability density function is*

$$f_X(x) = \begin{cases} \frac{1}{\frac{C_W}{C_I}(e-1)} e^{x/(C_W/C_I)}, & \text{if } x \leq C_W/C_I \\ 0, & \text{if } x > C_W/C_I \end{cases} \quad (11)$$

When this controller is used, $\lim_{N \rightarrow \infty} c^* = (\gamma + 1.58)/(\gamma + 1)$, where $\gamma = C_B\beta/C_W$.

Lemma 8 shows that the competition ratio of a random controller also depends on the ratio between $C_B\beta$ and C_W . If this ratio is very small, then the competitive ratio is close to 1.58; if the ratio is very large, then the competitive ratio is close to 1.

6 Conclusions

In this paper, we study the optimal ON-OFF control problem for a class of DESs with real-time constraints. The DESs have operating costs C_B and C_I per unit time and wake-up cost C_W . Our goal is to switch the system between the ON and the OFF states so as to minimize cost and satisfy real-time constraints. In particular, we consider a homogeneous case that all tasks have the same number of operations and each one's deadline is d seconds after the arrival time. For the off-line scenario that all task information is known to the controller a priori, we show that the optimal solution can be obtained via a two-fold decomposition: (i) super active periods that contain one or more active periods can be identified easily using the task arrival times and deadlines and (ii) the optimal solution to each super active period can be solved using dynamic programming. Simulation results show that compared with a simple heuristic, the cost saving of the DP algorithm can be 50% or more.

In on-line control, we show that the best time to start an AP can be obtained via an iterative algorithm and is guaranteed to be the same as the off-line problem. To decide the best time to end an AP in the on-line setting where no future task arrival information is available, we evaluate both deterministic and random controllers and derive their competitive ratios; these results quantify the worst-case on-line performance deviation from the optimal off-line solution.

References

1. L. Miao, J. Mao, and C. G. Cassandras, "Optimal energy-efficient downlink transmission scheduling for real-time wireless networks," *IEEE Transactions on Control of Network Systems*, in print, DOI 10.1109/TCNS.2016.2545099.

2. J. W. S. Liu, *Real - Time Systems*. NJ: Prentice Hall Inc., 2000.
3. D. L. Pepyne and C. G. Cassandras, "Optimal control of hybrid systems in manufacturing," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 1108–1123, 2000.
4. A. E. Gamal, C. Nair, B. Prabhakar, E. Uysal-Biyikoglu, and S. Zahedi, "Energy-efficient scheduling of packet transmissions over wireless networks," in *Proceedings of IEEE INFOCOM*, vol. 3, 23-27, New York City, USA, 2002, pp. 1773–1782.
5. M. Zafer and E. Modiano, "A calculus approach to energy-efficient data transmission with quality-of-service constraints," *IEEE/ACM Trans. on networking*, vol. 17, pp. 898–911, 2009.
6. L. Miao and L. Xu, "Optimal wake-up scheduling for energy efficient fixed-rate wireless transmissions with real-time constraints," in *14th Wireless Telecommunications Symposium*, New York, NY, 2015.
7. L. Miao, "Optimal on-off scheduling for a class of discrete event systems with real-time constraints," in *American Control Conference*, Seattle, WA, USA, May, 2017.
8. E. Uysal-Biyikoglu, B. Prabhakar, and A. E. Gamal, "Energy-efficient packet transmission over a wireless link," *IEEE/ACM Transactions on Networking*, vol. 10, pp. 487–499, Aug. 2002.
9. M. Zafer and E. Modiano, "Minimum energy transmission over a wireless channel with deadline and power constraints," *IEEE Trans. on Automatic Control*, vol. 54, pp. 2841 – 2852, 2009.
10. —, "Delay-constrained energy efficient data transmission over a wireless fading channel," in *IEEE Information Theory and Applications Workshop*, San Diego, CA, USA, Jan-Feb 2007.
11. —, "Optimal rate control for delay-constrained data transmission over a wireless channel," *IEEE Trans. on Information Theory*, vol. 54, pp. 4020 – 4039, 2008.
12. W. Chen, M. Neely, and U. Mitra, "Energy efficient scheduling with individual packet delay constraints: offline and online results," in *IEEE Infocom*, Anchorage, Alaska, USA, May 2007.
13. W. Chen, U. Mitra, and M. Neely, "Energy-efficient scheduling with individual packet delay constraints over a fading channel," *ACM Wireless Networks*, vol. 15, pp. 601–618, 2009.
14. X. Wang and Z. Li, "Energy-efficient transmissions of bursty data packets with strict deadlines over time-varying wireless channels," *IEEE Trans. on Wireless Communications*, vol. 12, pp. 2533–2543, 2013.
15. M. I. Poulakis, A. D. Panagopoulos, and P. Constantinou, "Channel-aware opportunistic transmission scheduling for energy-efficient wireless links," *IEEE Trans. on Vehicular Technology*, vol. 62, pp. 192–204, 2013.
16. X. Zhong and C. Xu, "Online energy efficient packet scheduling with delay constraints in wireless networks," in *IEEE Infocom*, Phoenix, AZ, April 2008.
17. J. Mao, C. G. Cassandras, and Q. Zhao, "Optimal dynamic voltage scaling in energy-limited nonpreemptive systems with real-time constraints," *IEEE Trans. on Mobile Computing*, vol. 6, no. 6, pp. 678–688, 2007.
18. J. Mao and C. G. Cassandras, "Optimal control of multilayer discrete event systems with real-time constraint guarantees," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 10, pp. 1425–1434, 2014.
19. W. Ye, J. S. Heidemann, and D. Estrin, "Medium access control with coordinated adaptive sleeping for wireless sensor networks," *IEEE/ACM Trans. on Networking*, vol. 12, pp. 493–506, 2004.
20. W. Lai and I. C. Paschalidis, "Routing through noise and sleeping nodes in sensor networks: latency vs. energy trade-offs," in *the 45th IEEE Conference on Decision and Control*, San Diego, CA, USA, 2006, pp. 2716–2721.
21. X. Ning and C. G. Cassandras, "Dynamic sleep time control in event-driven wireless sensor networks," in *the 45th IEEE Conference on Decision and Control*, San Diego, CA, USA, 2006, pp. 2722–2727.
22. —, "Optimal dynamic sleep time control in wireless sensor networks," in *the 47th IEEE Conference on Decision and Control*, Cancun, Mexico, 2008, pp. 2332–2337.
23. R. Cohen and B. Kapchits, "An optimal wake-up scheduling algorithm for minimizing energy consumption while limiting maximum delay in a mesh sensor network," *IEEE/ACM Transactions on Networking*, vol. 17, pp. 570–581, 2009.

24. L. Xia, D. Miller, Z. Zhou, and N. Bambos, "Service rate control of tandem queues with power constraints," *IEEE Transactions on Automatic Control*, vol. 62, no. 10, pp. 5111–5123, 2017.
25. N. M. Pletcher, S. Gambini, and J. Rabaey, "A 52 w wake-up receiver with 72 dbm sensitivity using an uncertain-if architecture," *IEEE Journal of solid-state circuits*, vol. 44, no. 1, pp. 269–280, 2009.
26. S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson, "On the power of randomization in on-line algorithms," *Algorithmica*, vol. 11, no. 1, pp. 2–14, 1994.
27. A. C.-C. Yao, "Probabilistic computations: Toward a unified measure of complexity," in *Foundations of Computer Science, 1977., 18th Annual Symposium on.* IEEE, 1977, pp. 222–227.
28. A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki, "Competitive randomized algorithms for nonuniform problems," *Algorithmica*, vol. 11, no. 6, pp. 542–571, 1994.

APPENDIX

Proof of Lemma 1: To see this, consider the solution that the system is woken up at a_1 and stays active until d_N . Because in any d seconds,

$$Rd \geq \lfloor \frac{d}{\beta} \rfloor B,$$

that is, the number of task departures is not less than the number of task arrivals, the backlog is always zero at the integer multiples of d seconds. This means that all task arrivals can be served within d seconds. Hence, the proposed solution is always feasible under Assumption 1. ■

Proof of Lemma 2: Because

$$a_k \leq t_A < t_B \leq d_k - \beta$$

we have

$$x_j^A \leq x_j^B, \quad j = k, \dots, n \quad (12)$$

where x_k^A, \dots, x_n^A and x_k^B, \dots, x_n^B are the task departure times in the two sample paths, respectively. It means that the departure time of task j in sample path A is no later than that in sample path B . Note that (12) holds because the sensor stays on in the AP and $t_A < t_B$.

Let t_{EA} and t_{EB} be the ending time of the AP when the starting time is t_A and t_B , respectively. We have:

$$C_{k, \dots, n}^A = C_w + C_B(n - k + 1)\beta + C_I[t_{EA} - t_A - (n - k + 1)\beta]$$

$$C_{k, \dots, n}^B = C_w + C_B(n - k + 1)\beta + C_I[t_{EB} - t_B - (n - k + 1)\beta]$$

where the three terms in each equation correspond to the wake-up cost, the cost of serving the $n - k + 1$ tasks, and the idling cost, respectively.

By assumption that $t_A < t_B$, we know for sure that the idling time of sample path A is not less than that of sample path B , i.e., $t_{EA} - t_A \geq t_{EB} - t_B$. Therefore,

$$C_{k, \dots, n}^A \geq C_{k, \dots, n}^B \quad \blacksquare$$

Proof of Lemma 3: Time $d_k - \beta$ is the latest time to serve task k , in order to meet its hard deadline requirement. Invoking Lemma 2, we only need to show that $Q(1, N)$ is still feasible when we delay the transmission of task k until $d_k - \beta$. Under Assumption 1 and in the worst case, there could be $\lfloor \frac{d}{\beta} \rfloor - 1$ tasks $\{k + 1, \dots, k + \lfloor \frac{d}{\beta} \rfloor - 1\}$ arriving at $d_k - \beta$. It means that at time $d_k - \beta$, we have $\lfloor \frac{d}{\beta} \rfloor$ tasks in the backlog. If we start transmitting all these tasks $\{k, \dots, k + \lfloor \frac{d}{\beta} \rfloor - 1\}$ at $d_k - \beta$, then it takes maximum d seconds to send all of them, and each task's deadline is met. Again, under Assumption 1, the earliest time that task $k + \lfloor \frac{d}{\beta} \rfloor$ can arrive is d_k . If the system stays active at $d_k + d - \beta$, then task $a_{k + \lfloor \frac{d}{\beta} \rfloor}$

can be served by its deadline $d_k + d$. Similarly, all subsequent tasks can be transmitted by their deadlines. Therefore, $Q(1, N)$ is still feasible after we postpone task k 's transmission time to $d_k - \beta$. ■

Proof of Lemma 4: Invoking Lemma 2 again, we need to show that $d_k - \beta$ and $d_k - \beta - \delta$ are the latest feasible starting times for the two cases, respectively.

In the worse case, there could be $\lfloor \frac{d}{\beta} \rfloor - m - 1$ tasks arriving at $d_k - \beta$. As we have shown in Lemma 3, these tasks and all subsequent ones can be served before their deadlines as long as we start the AP no later than $d_k - \beta$. Therefore, we only need to focus on the tasks that arrive before $d_k - \beta$.

Case 1: $\delta_z \leq 0$. This implies that

$$\frac{a_j - a_k}{j - k} \geq \beta, \text{ for } j \in \{k + 1, \dots, k + m\} \quad (13)$$

$d_k - \beta$ is the latest possible starting time for task k . We need to show that $Q(1, N)$ is still feasible when we start serving task k at $d_k - \beta$, i.e., starting serving tasks at this time will satisfy the real-time constraints for tasks $\{k + 1, \dots, k + m\}$. If task k is done at $d_k - \beta$, then the departure time x_j of task j , $j \in \{k + 1, \dots, k + m\}$, is

$$x_j = d_k + \beta(j - k)$$

From (13), we have

$$x_j \leq d_k + a_j - a_k = a_j + d = d_j$$

Thus, the deadlines of all the tasks $\{k + 1, \dots, k + m\}$ are met, and $d_k - \beta$ is the optimal starting time.

Case 2: $\delta_z > 0$. We need to show $d_k - \beta - \delta_z$ is a feasible starting time for all tasks $\{k, \dots, k + m\}$. We first show causality. The starting time s_j of task $j \in \{k + 1, \dots, k + m\}$ is:

$$s_j = d_k - \beta - \delta_z + \beta(j - k) \quad (14)$$

Using (1),

$$\begin{aligned} s_j &= d_k - \beta - [\beta(z - k) - (a_z - a_k)] + \beta(j - k) \\ &= d_k - \beta + \beta(j - z) + (a_z - a_k) \\ &= d - \beta + \beta(j - z) + a_z \end{aligned} \quad (15)$$

We use contradiction to prove it. Suppose $s_j < a_j$, we have

$$\begin{aligned} d - \beta + \beta(j - z) + a_z &< a_j, \text{ i.e.,} \\ d - \beta + \beta(j - z) &< a_j - a_z \end{aligned} \quad (16)$$

1) When $k < j \leq z \leq k + m$, we have

$$a_j - a_z \leq 0 \quad (17)$$

By Assumption 1,

$$\beta(j - z) = -\beta(z - j) \geq -(d - \beta),$$

i.e.,

$$d - \beta + \beta(j - z) \geq 0 \quad (18)$$

Combining (17) and (18), (16) is not true.

2) When $k < z < j \leq k + m$, we have

$$d - \beta > a_j - a_z > 0$$

and

$$d - \beta + \beta(j - z) > d$$

Combining the two inequalities, we conclude that (16) is not true either.

We can now assert

$$s_j \geq a_j$$

which satisfies causality.

Next, we show the departure time of each task $j \in \{k+1, \dots, k+m\}$ is before the task's deadline. Again, we use x_j to denote the departure time of task j , and

$$x_j = s_j + \beta$$

Invoking (15),

$$x_j = d + \beta(j - z) + a_z$$

We need to show

$$x_j = d + \beta(j - z) + a_z \leq a_j + d,$$

i.e.,

$$\beta(j - z) + a_z \leq a_j \quad (19)$$

From (1), we have

$$\begin{aligned} \delta_j &= \beta(j - k) - (a_j - a_k) \leq \\ \delta_z &= \beta(z - k) - (a_z - a_k), \\ j &= k + 1, \dots, k + m \end{aligned}$$

Rearranging the terms above, we obtain (19).

Finally, the departure time of task z is exactly $a_z + d$, indicating that $d_k - \beta - \delta_z$ is the latest possible time to start serving task z . ■

Proof of Lemma 5: We use x_j^* and s_{j+1}^* to denote the departure time of task j and the starting time of task $j + 1$, respectively, on the optimal sample path of $Q(1, N)$. Using Lemma 1, we have

$$x_j^* \leq d_j \quad (20)$$

From causality,

$$s_{j+1}^* \geq a_{j+1} \quad (21)$$

By assumption, we have

$$a_{j+1} - d_j > C_W / C_I \quad (22)$$

Combining (20), (21), and (22), we get

$$s_{j+1}^* - x_j^* > C_W / C_I \quad (23)$$

Next, we use a contradiction argument to prove the lemma. Let the optimal sample path of $Q(1, N)$ be sp^* and the corresponding cost is J^* . Suppose that task j does not end an **AP** on sp^* . It means that the system stays active from x_j^* to s_{j+1}^* . The optimal cost is then $J^* = (s_{j+1}^* - x_j^*)C_I + J_R$, where J_R is the rest of the cost beyond time interval $[x_j^*, s_{j+1}^*]$. Consider another sample path sp' , which is identical to sp^* , except that the system goes to sleep at x_j^* and wakes up at s_{j+1}^* . The system cost is now $J' = C_W + J_R$. Using (23), we obtain $J' < J^*$, which contradicts the assumption that sp^* is the optimal sample path. ■

Proof of Theorem 1: We use induction to prove it.

Step 1: Task n can either be a starting task or a following task. When it is a starting task, it is obvious that J_n^S is the optimal cost of $Q^S(n, n)$. When it is a following task, it is also obvious that J_n^F is the optimal cost of $Q^F(n, n)$.

Step 2: Suppose that J_j^S is the optimal cost of problem $Q_j^S(j, n)$, and J_j^F is the optimal cost of problem $Q_j^F(j, n)$, $j \in \{i, \dots, n\}$. We need to show that J_{i-1}^S and J_{i-1}^F are the optimal cost of problem $Q_{i-1}^S(i-1, n)$ and $Q_{i-1}^F(i-1, n)$, respectively. Since the proofs are similar, we only show that J_{i-1}^S is the optimal cost of problem $Q_{i-1}^S(i-1, n)$. By assumption, task $i-1$ is a starting task. We can use Lemmas 3 and 4 to find $s_{i-1,n}^{i-1}$, the optimal starting time of task $i-1$. We now discuss two cases:

Case 1: Task l that satisfies (2) does not exist.

It implies that $s_{i-1,n}^{i-1} + (j-i+1)\beta > a_j, \forall j \in \{i-1, \dots, n\}$, i.e., the system is busy serving tasks whenever a task $j \in \{i-1, \dots, n\}$ arrives. Therefore, there is no reason to go to sleep, and tasks $\{i-1, \dots, n\}$ form a single AP. From Line 14 of Table 2, $J_{i-1}^S = C_W + (n-i+2)\beta C_B$ is the optimal cost of problem $Q^S(i-1, n)$.

Case 2: Task l that satisfies (2) does exist.

In this case, task l has not arrived when task $l-1$ departs the system. It has two subcases: the system should either go to sleep when task $l-1$ departs or stay awake (and serve task l when it arrives). The subcase that yields a smaller cost is the optimal solution, and this is calculated in (3). ■

Proof of Lemma 6: We consider two cases.

Case 1: The optimal wake-up time is $d_k - \beta$. This happens when either Lemma 3 or the $\delta_z \leq 0$ case of Lemma 4 applies. The on-line control mechanism picks the same wake-up time upon the arrival of task k , and it does not change. Therefore, the wake-up time in on-line control is the same as the optimal wake-up time on the optimal sample path.

Case 2: The optimal wake-up time is $d_k - \beta - \delta_z$. This happens when the $\delta_z < 0$ case of Lemma 4 applies. In on-line control, the initial wake-up time is set to $d_k - \beta$. With the arrival of tasks between a_k and $d_k - \beta$, this scheduled time is adjusted to $d_k - \beta - \delta_j$, for some $j \in \{k+1, \dots, k+m\}$. By definition of δ_z , we have

$$\begin{aligned} d_k - \beta - \delta_j &\geq d_k - \beta - \delta_z \\ &= d_k - \beta - [\beta(z-k) - (a_z - a_k)] \\ &= d_k - \beta - \beta(z-k) + a_z \\ &\geq a_z \end{aligned}$$

The above implies that all intermediate wake-up times and the optimal wake-up time are after the arrival of task z . Therefore, the on-line control policy is able to wake up the system at the optimal time $d_k - \beta - \delta_z$ after task z arrives. ■

Proof of Lemma 7: The worst-case happens when each AP only contains a single task. After each task is served, the system stays active for θ seconds and then goes to sleep; it wakes up again after the next task arrives. For any θ , we have the ratio between the on-line cost and the optimal cost:

$$c(\theta) = \frac{C_W + NC_B\beta + (N-1)(C_I\theta + C_W)}{C_W + NC_B\beta + (N-1)\min(C_I\theta, C_W)} \quad (24)$$

where the numerator is the on-line cost and the denominator is the off-line cost. Note that both costs have three terms: the first term C_W is the wake-up cost for serving the very first task; the second term $NC_B\beta$ is the actual cost of serving the N tasks; and the last term is the cost between two adjacent tasks. We can rewrite (24) into:

$$c(\theta) = \frac{C_W/N + C_B\beta + (N-1)(C_I\theta + C_W)/N}{C_W/N + C_B\beta + (N-1)\min(C_I\theta, C_W)/N}$$

It follows that

$$\lim_{N \rightarrow \infty} c(\theta) = \frac{C_B\beta + (C_I\theta + C_W)}{C_B\beta + \min(C_I\theta, C_W)}.$$

Because

$$\frac{C_I\theta + C_W}{\min(C_I\theta, C_W)} \geq 2$$

and the equality holds when $\theta = C_W/C_I$, we have

$$c^* = \lim_{N \rightarrow \infty} c(C_W/C_I) = \frac{C_B\beta + 2C_W}{C_B\beta + C_W} = \frac{2+\gamma}{1+\gamma} \quad \blacksquare$$

Proof of Lemma 8: Similar to the deterministic algorithm case, the worst case also occurs when each AP only contains a single task. At the i -th decision point, the system stays active for $\theta_t = X$ seconds, where X is a random variable returned by algorithm A ,

and then goes to sleep if no tasks arrive during this period. For serving N tasks, the ratio between the on-line cost and the optimal-cost is:

$$c(\theta_t) = \frac{C_W + NC_B\beta + (N-1)E_G[\tilde{J}_b(A, I_G)]}{C_W + NC_B\beta + (N-1)J_b^*(I_G)} \quad (25)$$

where the numerator is the on-line cost and the denominator is the off-line cost. Note that both costs have three terms: the first term C_W is the wake-up cost for serving the very first task; the second term $NC_B\beta$ is the actual cost of serving the N tasks; and the last term is the cost between two adjacent tasks. Note that the reason why the expectation is taken with respect to G is due to the insight provided by equation (10). We can rewrite (25) into:

$$c(\theta_t) = \frac{C_W/N + C_B\beta + (N-1)E_G[\tilde{J}_b(A, I_G)]/N}{C_W/N + C_B\beta + (N-1)J_b^*(I_G)/N}.$$

It follows that

$$\lim_{N \rightarrow \infty} c(\theta_t) = \frac{C_B\beta + E_G[\tilde{J}_b(A, I_G)]}{C_B\beta + J_b^*(I_G)}. \quad (26)$$

Let y be the time it takes for the next task to arrive after the system finishes serving the previous task. Similar to other on-line scheduling scenarios such as the ski rental and the snoopy caching problems [28], it can be seen via variational analysis that $E_G[\tilde{J}_b(A, I_G)]/J_b^*(I_G)$ is uniform with respect to y , i.e., it is independent from y . Letting $\tilde{c} = E_G[\tilde{J}_b(A, I_G)]/J_b^*(I_G)$, our goal is to come up the best algorithm A that minimizes \tilde{c} . It has been shown in [28] that \tilde{c}^* is $e/(e-1) \approx 1.58$, and the probability distribution P that achieves this ratio is in (11). Note that $J_b^*(I_G) = \min(C_I \times y, C_W)$. The impact of \tilde{c} to (26) is the greatest when $J_b^*(I_G)$ takes the maximum value C_W . In this case, $E_G[\tilde{J}_b(A, I_G)]$ takes its value $1.58C_W$. Therefore,

$$\lim_{N \rightarrow \infty} c^* = \frac{C_B\beta + 1.58C_W}{C_B\beta + C_W} = \frac{\gamma + 1.58}{\gamma + 1} \blacksquare$$

1. $J_n^S = C_W + C_B\beta$, $J_n^F = C_B\beta$, and
set both $J_n^S \rightarrow next$ and $J_n^F \rightarrow next$ to NULL.
2. for $(i = n; i - k \geq 1; i - -)$ {
3. Initialize $J_{i-1}^S \rightarrow next$ and $J_{i-1}^F \rightarrow next$ to NULL
4. Solve $Q^S(i-1, n)$
5. Solve $Q^F(i-1, n)$
6. }

Table 1: The algorithm that returns the optimal cost of $Q(k, n)$

-
1. Use Lemmas 3 and 4 to find $s_{i-1,n}^{i-1}$, the optimal starting time of task $i-1$
 2. If (there exists l that satisfies (2)) {
 3. Calculate $V_{i-1,l}^{SS}$ and $V_{i-1,l}^{SF}$ using (4) and (5), respectively
 4. If $(V_{i-1,l}^{SS} + J_l^S \leq V_{i-1,l}^{SF} + J_l^F)$ {
 5. $J_{i-1}^S = V_{i-1,l}^{SS} + J_l^S$
 6. $J_{i-1}^S \rightarrow next = J_l^S$
 7. }
 8. else {
 9. $J_{i-1}^S = V_{i-1,l}^{SF} + J_l^F$
 10. $J_{i-1}^S \rightarrow next = J_l^F$
 11. }
 12. }
 13. else { // single AP case
 14. $J_{i-1}^S = C_W + (n-i+2)\beta C_B$
 15. }

Table 2: The algorithm that returns the optimal cost of $Q^S(i-1, n)$

1. If (there exists task l that satisfies (6)) {
2. Calculate $V_{i-1,l}^{FS}$ and $V_{i-1,l}^{FF}$ using (8) and (9), respectively
3. If $(V_{i-1,l}^{FS} + J_l^S \leq V_{i-1,l}^{FF} + J_l^F)$ {
4. $J_{i-1}^F = V_{i-1,l}^{FS} + J_l^S$
5. $J_{i-1}^F \rightarrow next = J_l^S$
6. }
7. else {
8. $J_{i-1}^F = V_{i-1,l}^{FF} + J_l^F$
9. $J_{i-1}^F \rightarrow next = J_l^F$
10. }
11. }
12. else { //single AP case
13. $J_{i-1}^F = (n-i+2)\beta C_B$
14. }

Table 3: The algorithm that returns the optimal cost of $Q^F(i-1, n)$

```

1.   $J = J_k^S$ ,  $i = J.task = J's$  subscript, and
     $J.type = J's$  superscript
2.  while (  $J \rightarrow next$  is not NULL){
3.     $J' = J -> next$ 
4.     $next\_task = J'.task$ 
5.     $next\_type = J'.type$ 
6.    If ( $J.type = "S"$ ){
7.      AP starts at  $s_{i,n}^i$ ;
8.    }
9.    If ( $next\_type = "S"$ ) {
10.     AP ends after task  $next\_task - 1$  is served;
11.      $J = J'$  and  $i = J.task$ ; continue;
12.   }
13.   If ( $next\_type = "F"$ ) {
14.     Keep the system active through  $a_{next\_task}$ 
15.   }
16.    $J = J'$  and  $i = J.task$ 
17. }

```

Table 4: The procedure that returns the optimal control to $Q(k, n)$