On Minimum Delay Duty-Cycling Protocol in Sustainable Sensor Network

Shaojie Tang*† Jie Wu† Guihai Chen‡ Cheng Wang§ Xuefeng Liu¶ Tao Li¶ Xiang-Yang Li*
*Illinois Institute of Technology † Department of CIS, Temple University § Tongji University
‡ State Key Lab of Novel Software Technology, Nanjing University ‡ Hong Kong Polytechnic University

***Abstract*—To ensure sustainable operations of wireless sensor networks, environmental energy harvesting has been well recognized as one promising solution for long-term applications. Unlike in battery-powered sensor networks, we are targeting a duty-cycle adjustment to optimize the network performance, *e.g.*, delay minimization, with full harvested energy utilization. In this paper, we introduce a set of duty-cycle adjustment schemes that will minimize cross traffic delay (CTD) in energy-harvesting sensor networks. We first present an offline solution by assuming that the link reliability and traffic distribution are known a priori. Based on the submodular property of the CTD function, we theoretically prove that a simple greedy algorithm can achieve constant approximation. We next propose a class of online algorithms that do not require the knowledge of link reliability and traffic distribution. For each of these algorithms, we give a theoretical bound on the performance. We have evaluated our design with a TelosB-based implementation and experimental results corroborate our theoretical analysis.**

***Index Terms*—Wireless sensor networks, solar powered, duty cycle, submodular.**

I. Introduction

The wireless sensor network (WSN) is demonstrated to be a promising technology due to the rapid development in micro-electronics and sensor technology, and WSNs are widely used in field monitoring, military surveillance, assisted living, and scientific research. Among those applications, many of them require the system to remain stable over time. To support those long-term applications, sustainable sensor networks, which harvest energy from their surrounding environment, have been recognized as a promising solution. It is possible to extract energy from several sources, and some of the available harvesting technologies include solar, thermal, optical, and kinetic energies. Among these, solar energy harvesting through photo-voltaic conversion and vibrational energy through piezoelectric elements are able to provide relatively higher power densities, which make them more suitable in WSNs that consume power on the order of mW.

In most previous works, when adjusting the duty-cycle in wireless sensor networks in order to minimize the delivery delay, researchers have to take into account the very limited resources of the nodes. While many researchers assume that all nodes in a sensor network are battery-driven, as discussed in this work, nodes can also be powered by other energy sources such as solar power. Due to the varying environment conditions, the energy supply to individual nodes may change dramatically over time [10]. This makes the design of a duty-cycle adjustment scheme more complicated by the fact that the energy source is not permanent. For nodes in sustainable sensor networks, their duty-cycle has to be adjusted continuously with a varying energy supply, and we would like to make full use of harvested energy while maintaining the sustainability of the whole network. As is pointed out in [8], [13], [19], the harvested energy from solar panels is often not able to power the sensors continually. Thus, nodes have to work in low duty-cycle, *e.g.*, each node has to stay in a dormant state for most of the time.

978-1-4673-2447-2/12/$31.00

Recently, several important works [8], [10], [9] put their focus on dynamically adjusting the duty-cycle for sustainable or low duty-cycle sensor networks in order to minimize the communication delay. In this work, we study the problem that is initially proposed and studied in [10]. Basically, we aim to dynamically adjust each node's activity pattern with available energy budgets, in order to minimize the communication delay at individual nodes. The major contributions of this paper are summarized as follows:

► We first show that the cross traffic delay function is submodular, which has several very nice tractability properties;

► Under the assumption that the link reliability and traffic distribution are known, we theoretically prove that a simple greedy algorithm provides constant approximation in terms of cross traffic delay minimization. To the best of our knowledge, this is the first theoretical bound on this greedy algorithm under a general problem setting;

► Without knowing the link reliability and traffic distribution, we propose a class of online algorithms. The key idea of these algorithms is learning/exploiting. On one hand, these algorithms try to learn the performance of different activity patterns, and on the other hand, they are able to stick to a "good" activity pattern after a sufficiently large number of rounds;

► We have extensively evaluated our design on a real testbed consisting of 40 sensors, spanning 30 days, to verify the effectiveness of our design in practice.

The rest of this paper is organized as follows: Section II discusses the related work. Section III articulates the network model and related assumptions and formalizes the problem studied in this paper. We propose an offline greedy algorithm in Section IV and a set of online algorithms in Section V. We conduct extensive experiments in Section VI and conclude this paper in Section VII.

II. Related Work

A number of approaches have been proposed to provide timely data communications in sensor networks. Gu *et al.* [7] introduce a dynamic switch-based forwarding scheme to minimize the impact of sleep latency in low duty-cycle networks. Gu *et al.* [8] present an adjustment protocol to consume a minimum amount of energy while satisfying a E2E delay bound specified by application requirements. Su *et al.* propose both on-demand and proactive routing algorithms for intermittently connected networks due to duty-cycling [17]. Gu *et al.* propose a centralized spatiotemporal delay control scheme for battery powered low duty-cycle networks [9].

Energy harvesting for sensor networks has been widely studied recently. However, most previous works put their focus mainly on the hardware design element [13], [12], [19]. Few prior works study how to adjust node duty-cycles to minimize the communication delay in energy harvesting networks. Most recently, ESC protocol introduces transparent middleware for minimizing network-wide delays with a varying energy budget over time in energy-harvesting sensor networks [10]. Unlike previous works, in this work, we theoretically prove that a simple greedy algorithm provides constant approximation in a general problem setting, *e.g.*, a general number of active instances could be available at given time slot. Most importantly, we take a first try on implementing a class of online algorithms on duty-cycle adjustment. At the heart of our online approach is a modified algorithm for multi-armed bandit (MAB) problems [6], [16].

III. System Model

We first briefly introduce the network model, and the basic assumptions made in this work; we then formulate our problem.

A. Network Model

As in most existing works [10] [11], we assume that (i) neighboring nodes are synchronized in the unit of a time instance, and (ii) there is at most one packet transmission within such a time instance. At a given time instance t, a sensor node is either in an *active* or a *dormant* state. When a node is in the active state, it can sense the environment and receive packets from neighboring nodes. A dormant node turns all of its function modules off except for a timer to wake itself up. Each node in the network has its own working schedule, and all neighboring nodes know each other's schedule [3], [10], [8]. A dormant node will wake up either if it is scheduled to wake up, or if it has some packets to send to a neighboring node that is active at that time. *Therefore, a node can wake-up and transmit a packet when its receiver is in the active state, but it can only receive a packet when itself is in the active state.*

B. Working Schedule and Sleep Latency

The working schedule of a sensor node is defined as the active/dormant pattern of the sensor node during its lifetime. It contains a set of active instances, during which a node can receive packets. The node working schedules usually are periodic [18], [11]. Consequently, let the duration of a period be T time instances. The working schedule of any node therefore can be represented by a set of time instances at which the node is in the active state. Let Γ denote the working schedule; we can have $\Gamma = \{t_1, t_2, t_3, \cdots, t_n\}$, where t_i is the time instance at which the node is active, and we call t_i its active instance.

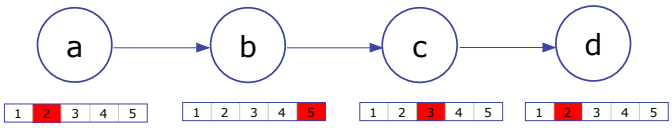

Fig. 1. A Linear Network.

For transmission in low duty-cycle sensor networks, the sender may have to wait for its receiver to become active. For any pair of transceivers, we define the sleep latency as the time duration from the moment when a packet is ready at the sender to the moment when the receiver becomes active. For example, in a linear network, shown in Figure 1, let the working schedule for four nodes a, b, c, and d be $\{2\}$, $\{5\}$, $\{3\}$, and $\{2\}$. Assume node that a has a packet that is destined for node b and is ready at time 2, then the sleep latency is 3. Sleep latency is usually in the order of seconds, which is orders of magnitude longer than other delivery latencies such as processing delay and transmission delay. We thus only consider sleep latency when measuring the cross traffic delay in the rest of this work.

C. Cross Traffic Delay

The detailed description for cross traffic delay (CTD) can be found in [10]. For completeness of presentation, we give a brief introduction to some key definitions. In low duty-cycle networks, a packet can be transmitted only when the receiver is active; thus, nodes in such networks will stay in the dormant state for the majority of time. Here, we assume that data traffic/congestion is low, which is shown to be true for existing low duty-cycle sensor networks.

For a node b in the network, there may be a set of nodes that forward packets to node b, which are called predecessors of node b. Similarly, node b would forward its packets to a certain set of nodes, which are successors of node b. To model the CTD related to node b, we consider the expected delays for packets from all predecessor nodes through node b, and then to corresponding successor nodes. Let the bi-directional link quality p_{ab} denote the success ratio of a round-trip transmission (DATA and ACK) between a and b. Denote the maximum retransmission times as K; the probability that the packets (arriving at node b from a) arrive at the k-th attempt, under the condition that the packet is delivered within K retransmissions, can be expressed as $P_{ab}(k) = \frac{(1-p_{ab})^{k-1} p_{ab}}{1-(1-p_{ab})^K}$.

Delay Over a Single Link: For a packet ready at time t at node a, the expected transmission delay to reach node b can be formulated as: $D_{ab}(t) = \sum_{k=1}^{K} P_{ab}(k) L_t^b(k)$ where $L_t^b(k)$ denotes the sleep latency for k-th attempt.

Delay from One Predecessor to One Successors: The expected delay from predecessor p_i to successor s_j of the packet ready at time t at predecessor p_i, is the product of the probability that the packet arrives node b at its k-th attempted transmission and corresponding cross-traffic delay, which can be expressed as: $D_{p_i s_j}(t) = \sum_{k=1}^{K} P^{p_i b}(k)(L_t^b(k) + D_{bs_j}(t + L_t^b(k)))$.

Delay from Multiple Predecessors to Multiple Successors: We next calculate the expected CTD at node b from multiple predecessor nodes to multiple successor nodes. Assume that the number of active instances of a predecessor node p_i is N_{p_i}; during each active instance $t_k^{p_i}$, we use $W_k^{p_i s_j}$ to represent the percentage of arriving traffic that is originated at p_i, and destined to a successor node s_j. Let the number of predecessor nodes and successor nodes at node b be N^p and N^s, respectively. We can express the expected delay of cross traffic at node b under schedule Γ_i as:

$$D_{\Gamma_i} = \sum_{i=1}^{N^p} \sum_{j=1}^{N^s} \sum_{k=1}^{N_{p_i}} W_k^{p_i s_j} D_{p_i s_j}(t_k^{p_i}) \tag{1}$$

D. Problem Formulation

The harvested energy may vary significantly over time at a sensor node. In order to make full use of the available energy supply, we need to decrease duty-cycles with a minimum increased delay when there is a shortage in the power supply, and we need to increase the duty-cycle with a maximum decreased delay when there is abundant energy. As in [10], we assume that the working schedules of the predecessors and successors of a node are known and up-to-date. For a given node b, assume that its previous working schedules $\Gamma = \{t_1, t_2, \cdots, t_m\}$, and its current energy supply can afford $m + k$ active instances to guarantee the sustainability of the node. In order to utilize those k additional active instances, in this work, node b can add those active instances on top of its previous working schedules.

For ease of analysis, assume that D_{max} is the largest possible CTD, *e.g.*, D_{max} can be chosen as $T \times K$. We define the reduced CTD (RCTD) of working schedule Γ_i as:

$$g(\Gamma_i) = D_{max} - D_{\Gamma_i}$$

Basically, $g(\Gamma_i)$ describes the reduced delay under working schedule Γ_i. Our goal is then to find a working schedule Γ_i to maximize the RCTD. Formally, given the number of extra active instances k, we aim to find a schedule Γ with maximum RCTD:

$$\Gamma = \arg\max_{\Gamma_i} \{g(\Gamma_i)\}$$

Notice that k could be negative when the current energy supply cannot afford m active instances, in this case, our problem lies in removing k active instances from the previous schedule in order to maximize the resulting RCTD. As we will discuss later, this can also be solved easily based on the solution for the first case.

IV. Offline Duty-cycle Adjustment

In this section, we study the offline case under the assumption that (1) the link reliability and (2) the traffic distribution are pre-known. For ease of explanation, we first introduce a basic operation in our duty-cycle adjustment.

Augmenting Single Active Instance: The operation "Augmenting Single Active Instance" basically aims at augmenting exactly one active instance of a previous schedule in order to maximize the resulting RCTD. A simple and effective algorithm for this case has been proposed in [10]. To make this paper self-containing, we briefly introduce the algorithm proposed in [10]: For a given node b, we can divide its period time into multiple intervals according to active instances of its predecessors and successors. For example, assume that the predecessor and successor of node b are node p and node s, and the working schedule of node p and node s are $\{1, 5\}, \{3, 7\}$. According to their locations on the cyclic working schedules, we can easily obtain four intervals: $(1, 3), (3, 5), (5, 7), (7, 1)$. Then, instead of performing an exhaustive search on all time instances within a period time T and choosing the time instance that yields a maximum RCTD, we only need to try those four intervals due to the fact that the resulting RCTD is the same among the same interval.

In a real implementation, the change in harvested energy could be bursty, and therefore a node may need to increase (or decrease) **multiple** active instances simultaneously. We next put our focus on finding an efficient and effective schedule scheme for the bursty augmentation and decrement case. Essentially, we propose a simple Hill-Climbing algorithm for augmenting and decreasing multiple active instances. We theoretically prove that the work schedule generated from our algorithm can achieve constant approximation of the optimal schedule in terms of maximizing the RCTD, *e.g.*, the RCTD resulting from our schedule is no smaller than a constant multiplied that of an optimal schedule.

For simplicity of analysis, we next use bursty augmentation as an example to show how the algorithm works and how to derive the performance guarantee for that algorithm. Both the algorithm and proof can work in a similar way for the bursty decrement case.

The Hill-Climbing Algorithm works as follows: Assume at time t, node b can afford k additional active instances. Then, we recursively apply the "Augmenting Single Active Instance" operation to schedule those k active instances *one by one*.

To provide a performance bound on the generated schedule, we begin by introducing a concept called *submodular*. Consider an arbitrary function $f(A)$ that maps subsets of a finite ground set U to non-negative real numbers. We say that f is submodular if it satisfies a natural "diminishing returns" property: the marginal gain from adding an element to a set S is at least as high as the marginal gain from adding the same element to a superset of S. Formally, a submodular function satisfies: $f(S \cup \{v\}) - f(S) \geq f(T \cup \{v\}) - f(T)$ for all elements v and all pairs of sets $S \subseteq T$. Submodular functions have several very nice tractability properties; the one that may

help us here is described in the following. Suppose we have a function f that is submodular, takes only nonnegative values, and is monotone in the sense that adding an element to a set cannot cause f to decrease: $f(S \cup \{v\}) \geq f(S)$ for all elements v and sets S. We wish to find a k-element set S for which $f(S)$ is maximized. This is an NP-hard optimization problem (it can be shown to contain the Hitting Set problem as a simple special case), but a result of Nemhauser, Wolsey, and Fisher [4], [14] shows that the following greedy Hill-Climbing algorithm approximates the optimum to within a factor of $(1-1/e)$, where e is the base of the natural logarithm: start with the empty set, and repeatedly add an element that gives the maximum marginal gain.

Theorem 1: ([4], [14]) For a non-negative, monotone submodular function f, let S be a set of size k obtained by selecting elements one at a time, each time choosing an element that provides the largest marginal increase in the function value. Let S^* be a set that maximizes the value of f over all k-element sets. Then, $f(S) \geq (1-1/e)f(S^*)$; in other words, S provides a $(1-1/e)$-approximation.

In order to utilize the result from the previous theorem, we next prove that the RCTD function $g()$ is indeed submodular.

Lemma 2: Given node b, assume that the work schedule of all its predecessors and successors is fixed, and the function $g(\Gamma_i)$ is non-negative, monotone, and submodular.

Proof: Please refer to the Appendix for the detailed proof. ■

Theorem 1 and Lemma 2 together imply the following theorem.

Theorem 3: For node b, assume that Γ_i is the work schedule resulting from Hill-Climbing algorithm, we have:

$$g(\Gamma_i) \geq (1 - \frac{1}{e}) g(\Gamma_i{}^*)$$

where $\Gamma_i{}^*$ denotes the optimal schedule. In other words, it provides a $(1-1/e)$-approximation.

When k is negative where we need to remove k active instances from previous schedule, we can simply remove those k active instances which are last added to the previous schedule by the Hill-Climbing algorithm. Based on the fact that the remaining schedule is still obtained from the Hill-Climbing algorithm, we have the same performance guarantee shown in Theorem 3.

V. Online Duty-cycle Adjustment

In the previous section, we present a duty-cycle adjustment under the assumption that both the link quality and the traffic distribution is pre-known. In real scenarios, since the link quality and data arrival rate are not always known a priori, we next combine several learning techniques with the previous scheme in order to relax those two constraints. We seek to develop a protocol for selecting the schedule Γ_i at each *round* r, such that, after a small number of rounds, the average performance of our online algorithm converges to the same performance of the offline strategy (which knows the objective functions, link reliability, and traffic distribution). In this work, we set the duration of one round r as one period, *e.g.*, T time instances. We define the *stable time* of one duty-cycle as the duration for which the same duty-cycle can last, *e.g.*, if the energy supply can support 20% of the duty-cycle from 10:00 AM to 11:00 AM, we say that the stable time for this duty-cycle is 1 hour. As verified by our experiment, it is reasonable to assume that the stable time is usually much longer than one transmission period. In our experiment, we set $T = 1$ minute, and the average stable time of one duty-cycle is around 40 minutes, that is, 40 times of T, and it is even longer on cloudy or rainy day. This ensures that the online algorithm has enough time to find a good schedule when searching space, which is the decided total number of possible schedules, is bounded.

The main idea of our algorithm is *learning/exploit*, which is initially proposed for the Multi-Armed Bandits problem [2]. Inspired by their ideas, we present three methods by extending their results to fit our problem setting. The online algorithm is parameterized by a learning rate γ, an exploration probability $1-\gamma$, and the number of runs R. With exploitation probability $1-\gamma$, we adopt the schedule used in the previous round. Here, γ is a relatively small parameter that depends on the number of total rounds R. This enables our schedule to get close to the static optimal. With probability γ, our algorithm explores and estimates the RCTD of each possible work schedule with the same probability. Since we never know whether our current schedule is already the *best*, our algorithm will also consider more exploitation on other schedules to find other, possibly better schedule. The number of runs R denotes the total number of rounds the learning/exploit operation needs to take, *e.g.*, after learning/exploit for R rounds, our algorithm will stick to the best schedule in the rest of stable time. Intuitively, the larger R is, the closer our final schedule will be to the optimal schedule. As a tradeoff, more explorations lead to more frequent adjustments, which will increase the message complexity, *e.g.*, share the current schedule with all neighbors. Thus, we must carefully decide the value of R in order to well balance the control overhead and performance.

In the offline case, based on the pre-known link reliability and traffic distribution, $g(\Gamma_i)$ can be calculated directly. When this information is unknown, we can only get the RCTD of Γ_i used in the previous round r, denoted by $g(\Gamma_i)(r)$: $g(\Gamma_i)(r) = D_{max} - D_{\Gamma_i}(r)$, where $D_{\Gamma_i}(r)$ is the CTD of Γ_i at round r; this can be obtained after transmission. For ease of analysis, we normalize $g(\Gamma_i)(r)$ into range $[0,1]$ by defining a scaled RCTD $x_i(r)$ as: $x_i(r) = g(\Gamma_i)(r)/D_{max}$.

For any algorithm A and for any $R > 0$, let $X_A(R) = \sum_{r=1}^{R} x_{i_r}(r)$ be the *return* at time horizon R of algorithm A choosing schedule $\Gamma_{i_1}, \Gamma_{i_2}, \cdots, \Gamma_{i_R}$. In what follows, we will write X_A instead of $X_A(R)$ whenever the value of R is clear from the context.

Our measure of performance for an algorithm A is the *weak regret*; given any time horizon R, the weak regret of algorithm A is defined by $X_{max}(R) - X_A(R)$, where $X_{max}(R) = \max_i \sum_{r=1}^{R} x_i(r)$.

We thus compare our protocol against all strategies that can select a fixed schedule for use over the entirety of stable

time, *e.g.*, the previous offline Hill-Climbing algorithm is one such strategy, the best such strategy obtains $X_{max}(R)$. The difference between this quantity and what our protocol obtains is known as its weak regret, and an algorithm is said to be *no-regret* if its average regret tends to be zero (or less) as R goes to infinity.

Algorithm 1 Online Algorithm 1(OL-1)

Parameters: real number $\gamma < 1/2$, number of extra active instances k, rounds R.

Initialization: Set $w_i(0) = 1$ for all $1 \leq i \leq n^k$.

1: At the r-th round, randomly select a schedule $\Gamma(r) = \Gamma_i$ according to the following distribution $\mathbf{p}_i(r)$ $\forall i \in [1, n^k]$:

$$\mathbf{p}_i(r) = (1-\gamma)\frac{w_i(r)}{\sum_i^{n^k} w_i(r)} + \frac{\gamma}{n^k}$$

2: Assume that Γ_i is the schedule selected in round r, get the scaled RCTD $x_i(r)$

$$x_i(r) = g(\Gamma_i)(r)/D_{max}$$

3: Calculate the virtual RCTD $x'_j(t)$ for all possible schedules:

$$x'_j(r) = \begin{cases} \frac{x_j(r)}{\mathbf{p}_i(r)} & \text{if } j = i \\ 0 & \text{otherwise.} \end{cases}$$

4: Update the weights:

$$w_i(r+1) = w_i(r) \cdot \exp(\frac{\gamma x_i(r)}{\mathbf{p}_i(r) n^k})$$

5: After R rounds, select the schedule with the highest $\mathbf{p}_i$ as the schedule for the rest of stable time;

In our study, we propose three versions of online algorithms:

A. Method 1: Try k Active Instances Together At The Same Time

Given k additional active instances, assume that there are n possible intervals that can be augmented, then the total number of possible assignments(schedule) of those k active instances is n^k. Our algorithm maintains a set of weights w_i, one for each schedule Γ_i, initialized to 1. At every round r, it will select each assignment Γ_i with probability:

$$\mathbf{p}_i(r) = (1-\gamma)\frac{w_i(r)}{\sum_i^{n^k} w_i(r)} + \frac{\gamma}{n^k}$$

It is with probability γ that it learns, picking schedule uniformly at random $\frac{1}{n^k}$. In addition, with probability $1-\gamma$ it exploits, picking schedule Γ_i with a probability proportional to its weight $w_i(r)$. Once a schedule has been selected and implemented, feedback $x_i(r)$, denoting the scaled RCTD from the previous round r by using Γ_i, is obtained, and the weight $w_i(r+1)$ is updated to:

$$w_i(r+1) = w_i(r) \cdot \exp(\frac{\gamma x_i(r)}{\mathbf{p}_i(r) n^k})$$

Then, we have the following performance guarantee.

Theorem 4: [1] For any $R > 0$ and $\gamma = \min\{1, \sqrt{\frac{n^k \ln n^k}{(e-1)R}}\}$,

$$\frac{1}{R}(X_{max} - X_{OL-1}) \leq \Theta(\sqrt{\frac{n^k \ln n^k}{R}}),$$

i.e., the average regret $\frac{1}{R}(X_{max} - X_{OL-1})$ converges to zero.

Algorithm 2 Online Algorithm 2(OL-2)

Parameters: real number $\gamma < 1/2$, number of extra active instances k, rounds R, .

1: $l \leftarrow k$;
2: **while** $l > 0$ **do**
3: Reset $\mathbf{p}_i$ and w_i;
4: Apply algorithm OL-1 for R/k rounds for one active instance based on the current schedule;
5: Assign the active instance to the interval with the highest $\mathbf{p}_i$ (we choose the second highest one if occupied);
6: $l \leftarrow l - 1$;
7: **end while**

When k is negative, we can simply remove those active instances that have the lowest probability from the current schedule.

B. Method 2: Online version of Hill-Climbing Algorithm

One problem with Method 1 is that the searching space grows exponentially with k, which makes it not scale, in contrast to the traditional MAB problem, where the arms are assumed to have independent feedback. In the duty-cycle adjustment problem, the utility function is submodular, and thus the feedback is correlated across different schedule. The key idea behind our second algorithm is to turn the offline greedy algorithm into an online algorithm by replacing the greedy assignment of each active instance that maximizes the RCTD by a bandit algorithm. Basically, we spend R/k rounds (by assuming that R/k is an integer, otherwise we spend $\lfloor R/k \rfloor$ rounds) for each of those k active instances; after every R/k rounds, we assign one active instance to the interval with the highest probability. Clearly, since we only need to consider schedule one active instance at each round, the search space is reduced from $O(n^k)$ to $O(nk)$.

We next provide a theoretical bound on the performance of Method 2. In Section IV, we show that for the models we are considering, the resulting RCTD function $g(\Gamma_i)$ is submodular. A subtle difficulty lies in the fact that the result of Nemhauser *et al.* [4] assumes that the greedy algorithm can evaluate the underlying function exactly, which may not be the case for the current model. However, by applying algorithm OL-1 for a sufficiently large number of rounds, we are able to find the interval with the maximum RCTD with high probability. Furthermore, one can extend the result of Nemhauser *et al.* to show that for any $c > 0$, there is a $\epsilon > 0$, such that by using

$(1+c)$-approximate values for the function to be optimized, we obtain a $(1-1/e-\epsilon)$-approximation.

Theorem 5: When $R \to \infty$ and $\gamma = \min\{1, \sqrt{\frac{n \ln n}{(e-1)R}}\}$, we obtain a $(1-1/e-\epsilon)$-approximation.

When k is negative, we can simply remove those k active instances that were last added to the previous schedule.

C. Method 3: Mixed Online Strategy

We first give a brief review of the previous two methods: Method 1 does not scale since the number of possible schedules grows exponentially with k. For Method 2, in order to test single active instances one by one, it sacrifices the utilization of current resources. Therefore compared with Method 1, the convergence time of Method 2 is dramatically reduced due to its significantly smaller searching space, *e.g.*, linearly grows with k. However, Method 2 experiences the risk of wasting a large number of active instances at the end. For instance, assume that the current energy supply supports $k = 6$ additional active instances, and the stable time is 100 rounds. If Method 2 takes $R/k = 20$ rounds for each active instance (in total $R = 120$ rounds), at the end of 100 rounds, we never have the chance to utilize all 6 active instances together. Especially when the harvested energy changes frequently, Method 2 may not provide a satisfying solution.

To tackle those issues, we propose a mixed online strategy by balancing the trade-off between Method 1 and Method 2. On one hand, we try to utilize the fact that the resulting RCTDs are correlated across different schedules to reduce the searching space. On the other hand, we intend to reduce the risk of wasting too many available resources. The key idea is to treat k rounds as one *super* round. At each super round, our algorithm assigns all k active instances one by one according to the choices of the k bandit algorithms, much like Method 2 does. Once the i-th active instance has been scheduled, the i-th bandit algorithm receives the incremental RCTD as feedback based on already assigned instances. This method is more like a combination of Method 1 and Method 2. Essentially, we try different assignments of k instances together at each super round, which is similar to Method 1, we assign k instances one by one during each super round which works like Method 2. Notice that after every super round, we will keep the updated $\mathbf{p}_i$ and w_i from the previous super round. By applying Method 3, the searching space is still as low as $O(nk)$. More importantly, we are able to fully utilize all k active instances every super round.

The proof for the following theorem is omitted here to save space:

Theorem 6: When $R \to \infty$ and $\gamma = \min\{1, \sqrt{\frac{n \ln n}{(e-1)R}}\}$, then OL-3 obtains a $(1-1/e-\epsilon)$-approximation on average RCTD.

Similar as for Method 2, we remove those active instances which have the lowest probability from current schedule when k is negative.

Algorithm 3 Online Algorithm 3(OL-3)

Parameters: real number $\gamma < 1/2$, number of additional active instances k, rounds R.

```
for each super round ℝ ≤ R/k do
    Clear the previous schedule but keep the updated probability distribution p_i(r+1);
    for each round r ∈ {r_1, ··· , r_k} do
        Apply algorithm OL-1(for 1 round) to assign one active instance;
        Update corresponding probability p_i(r+1) for each interval;
    end for
end for
```

(a) (b)

Fig. 2. (a) Solar motes used in our experiment; (b) The outdoor testbed.

VI. IMPLEMENTATION AND EVALUATION

In order to validate the performance and feasibility of all the proposed algorithms in practice, we implement them on a real testbed consisting of 40 sensors. Our outdoor experiment lasts for 30 days, and we compare the performance of each algorithm in terms of the average CTD under various weather conditions, *e.g.*, sunny, rainy, and cloudy.

A. Experimental Setup

The solar powered mote used in our experiment contains three main parts: solar panel, Li-ion Battery, and TelosB sensor node. The solar motes and outdoor testbed are shown in Figure 2. The reasons why we chose the Li-ion battery are as follows: (1) high open circuit voltage, (2) extremely low self-discharge rate, (3) no memory effect, and (4) high power density. The internal protection circuit is equipped to protect the battery from over-charging and discharging. We adopt the TelosB Mote [15] with a MSP430 processor and CC2420 transceiver. It integrates humidity, temperature, and light sensors, and provides flexible interconnection with peripherals. We randomly deploy 40 sensors in the outdoor field to test our protocol, and we run our outdoor experiment for a period of 30 days from June 1st to June 30th, during which the average harvested energy was 20Ah. The range of the duty-cycle varies from around 2% to 25%. Each sensor will dynamically adjust its duty-cycle and schedule according to the current energy supply and our algorithms. For fairness,

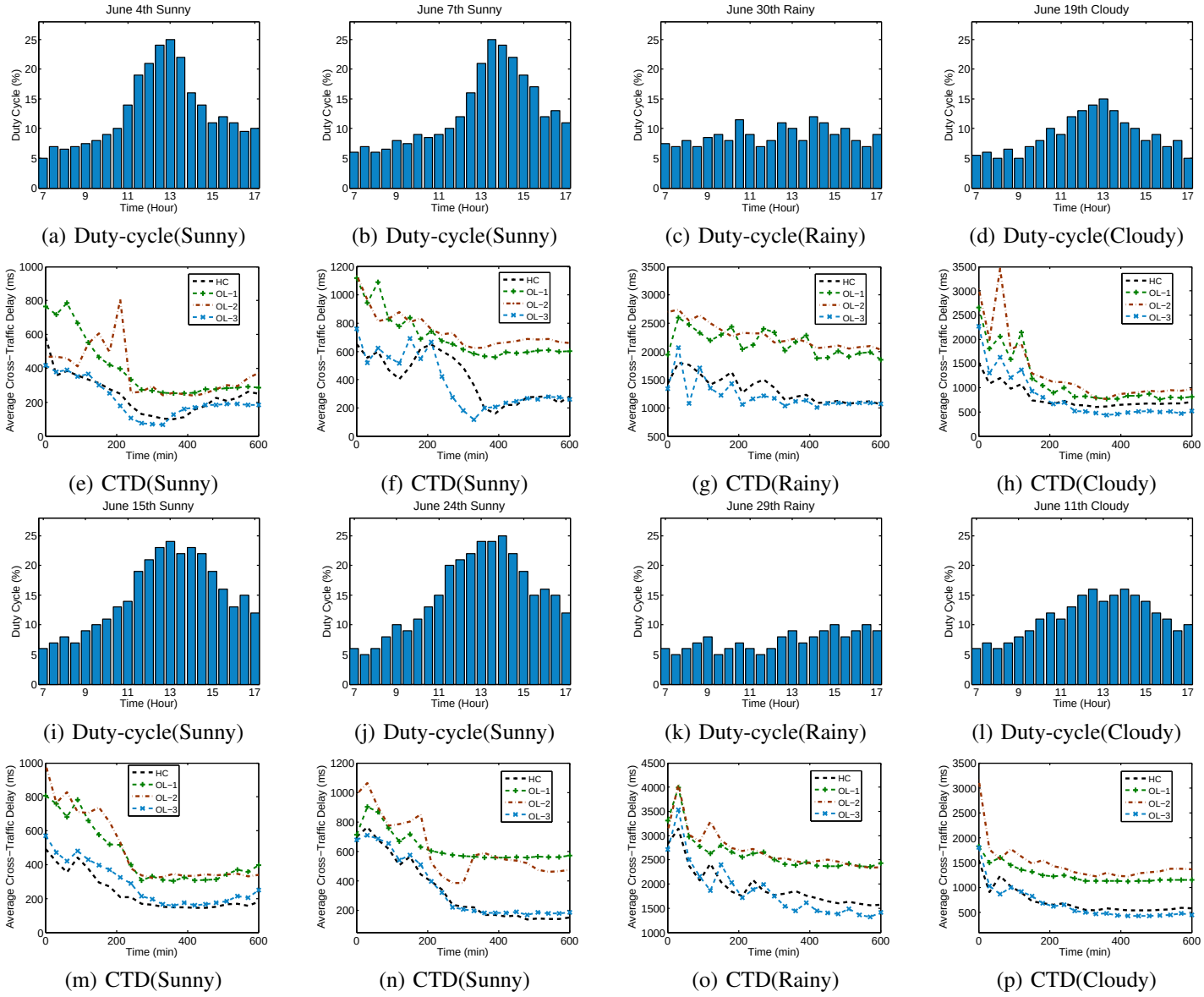

Fig. 3. (a)-(d), (i)-(l): duty-cycle change during one day under different weather conditions; (e)-(h), (m)-(p): cross traffic delay of four algorithms on each day.

we chose the same routing protocol, link-quality-based ETX [5], for all algorithms in this work.

B. Experimental Results

1) Impact of Weather Conditions: First of all, we are interested in how the duty-cycle changes during one day for each node under different weather conditions. We select eight days worth of data from one sensor node to report here. As illustrated in Figures. 3(a)-(d), (i)-(l), these eight days almost cover all of the typical weather conditions, *e.g.*, sunny, cloudy, and rainy. From the duty-cycle change, it is not hard to observe that: (1) The average duty-cycle heavily depends on current weather conditions. Under a sunny day, the average duty-cycle is much higher than rainy and cloudy. Specifically, the average duty-cycle under sunny day is 15%, rainy day is 7%, and cloudy day is 10%; (2) the variance of the duty-cycle change during one day under different weather conditions also varies significantly. For a sunny day, the duty-cycle increases/decreases dramatically when approaching/leaving the peak, and it stays stable at noon for around $2-3$ hours. For cloudy weather, the duty-cycle changes around every 1 hour point, but the variance is much smaller. For rainy weather, the variance is even smaller, and the duty-cycle keeps stable around the average level.

2) Cross traffic delay: In this section, we compare the CTD for the offline greedy algorithm and the three online algorithms. To ensure fairness, we implement four algorithms at the same time to minimize the impact of environmental energy supply variation. We partition 28 sensors into 4 groups, each group has the exact same topology: three successors, one relay node, and three predecessors. Each successor generates

a packet every 30 seconds and sends it to a randomly selected destination node (with the same probability). In a later stage, we can easily calculate the traffic distribution based on the above setup. Then, we implement four algorithms on different groups starting at the same time.

To reveal the relation between the duty-cycle change and the CTD, we still choose the eight days worth of data to report here. In Figures 3(e)-(h), (m)-(p), we plot four lines, each of which stands for the CTD resulting from one algorithm. For all three online algorithms, we set $R = 10$ and $\gamma = 0.2$. By comparing Figures 3(a)-(d), (i)-(l) and Figures 3(e)-(h), (m)-(p), we have the following insights:

► The CTD matches the available duty-cycle well, *e.g.*, the minimum delay occurs when the node duty-cycle increases to the peak, no matter which algorithm we are implementing.

► Compared with the other three online algorithms, the offline Hill-Climbing (HC) algorithm has the most *stable* performance. Although the average performance of OL-3 is very close to or even better than HC, it suffers from a larger variance. The larger variance mainly comes from the penalty of failed explorations, as HC is able to pre-compute a "good" schedule (with a performance guarantee as shown in Theorem 3) based on pre-known traffic distribution and link reliability. For online algorithms, they must try enough possible choices before finding a good schedule.

► When the duty-cycle changes significantly and frequently, the performance of the online algorithm becomes worse compared with HC. This is because the more frequently the duty-cycle changes, the less time online algorithm can afford on exploration. In an extreme case when the duty-cycle keeps changing every one period, the online algorithm starts to always randomly choose a schedule at each period, which surely leads to a poor performance.

► Among the three online algorithms, OL-3 has the best performance. This is because the search space of OL-3 is reduced from $O(n^k)$ to $O(nk)$ compared with OL-1, which dramatically reduces its exploration time. Compared with OL-2, OL-3 will utilize all available active instances every k rounds; this makes a significant difference when the duty-cycle changes very frequently and dramatically.

By summarizing above discussions, we find that offline HC has the most stable performance due to its pre-known knowledge of both the link reliability and the traffic distribution. Surprisingly, the average CTD resulting from OL-3 is very close to the offline one. Especially in cloudy or rainy days when the duty-cycle changes less frequently, OL-3 performs even better. The average performance of the other two online algorithms OL-1, OL-2 is less sufficient. OL-1 mainly suffers from its huge searching space, and OL-2 needs to take the risk of wasting available resources.

3) End to End Delay: In this section, we study the average end to end delay of the four algorithms over 30 days. The results are illustrated in Figure. 4, similar to the results shown in the cross traffic delay evaluation, Offline HC performs better than the online algorithms, 90% of HC packets reach their destinations within 800 ms; the same percentile for OL-1 is 1800 ms, for OL-2 is 2200 ms, and for OL-3 is 1000 ms. Due to the random characteristic of online algorithms, they have a longer tail than the offline algorithm.

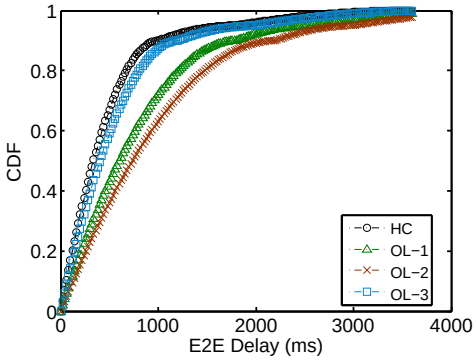

Fig. 4. Cumulative distribution function (CDF) of average end to end delay.

VII. Conclusion

In this paper, we first present an offline duty-cycle adjustment by assuming that the link reliability and traffic distribution is known a priori. We then propose a class of online algorithms without requiring any knowledge regarding the link reliability and traffic distribution. For each of them, we give a theoretical bound on the performance. Our experimental results over a real world testbed confirm the efficiency of the proposed methods.

VIII. Acknowledgement

The research of Xiang-Yang Li is partially supported by NSF CNS-0832120, NSF CNS-1035894, National Natural Science Foundation of China under Grant No. 61170216, No. 61228202, China 973 Program under Grant No.2011CB302705. The work of Jie Wu is partially supported by NSF grants ECCS 123182, CNS 1138963, ECCS 1128209, CNS 1065444, and CCF 1028167. The work of Guihai Chen is partly supported by China NSF grants (60825205, 61073152, 61133006) and China 973 project (2012CB316200).

References

[1] Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing 32*, 1 (2003), 48–77.

[2] Berry, D., and Fristedt, B. *Bandit problems: sequential allocation of experiments*. Routledge, 1985.

[3] Cao, Q., Abdelzaher, T., He, T., and Stankovic, J. Towards optimal sleep scheduling in sensor networks for rare-event detection. In *IPSN 2005*.

[4] Cornuejols, G., Fisher, M., and Nemhauser, G. Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management Science 23*, 8 (1977), 789–810.

[5] Couto, D., Aguayo, D., Bicket, J., and Morris, R. A high-throughput path metric for multi-hop wireless routing. *Wireless Networks 11*, 4 (2005), 419–434.

[6] Foster, D., and Vohra, R. Regret in the on-line decision problem. *Games and Economic Behavior 29*, 1-2 (1999), 7–35.

[7] Gu, Y., and He, T. Data forwarding in extremely low duty-cycle sensor networks with unreliable communication links. In *SenSys 2007*.

[8] Gu, Y., and He, T. Bounding communication delay in energy harvesting sensor networks. In *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on* (2010), IEEE, pp. 837–847.

[9] GU, Y., HE, T., LIN, M., AND XU, J. Spatiotemporal delay control for low-duty-cycle sensor networks. In *2009 30th IEEE Real-Time Systems Symposium* (2009), IEEE, pp. 127–137.
[10] GU, Y., ZHU, T., AND HE, T. ESC: Energy Synchronized Communication in Sustainable Sensor Networks. ICNP.
[11] HE, S., CHEN, J., YAU, D., SHAO, H., AND SUN, Y. Energy-efficient capture of stochastic events by global-and local-periodic network coverage. In *MobiCom 2009*.
[12] KANSAL, A., HSU, J., ZAHEDI, S., AND SRIVASTAVA, M. Power management in energy harvesting sensor networks. *ACM Transactions on Embedded Computing Systems (TECS) 6*, 4 (2007), 32.
[13] KANSAL, A., POTTER, D., AND SRIVASTAVA, M. Performance aware tasking for environmentally powered sensor networks. In *Proceedings of the joint international conference on Measurement and modeling of computer systems* (2004), ACM, pp. 223–234.
[14] NEMHAUSER, G., WOLSEY, L., AND FISHER, M. An analysis of approximations for maximizing submodular set functionsII. *Mathematical Programming 14*, 1 (1978), 265–294.
[15] POLASTRE, J., SZEWCZYK, R., AND CULLER, D. Telos: Enabling ultra-low power wireless research. In *IPSN* (2005), IEEE Press, p. 48.
[16] ROBBINS, H. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society 58*, 5 (1952), 527–535.
[17] SU, L., LIU, C., SONG, H., AND CAO, G. Routing in intermittently connected sensor networks. In *ICNP 2008*.
[18] WU, Y., FAHMY, S., AND SHROFF, N. Energy efficient sleep/wake scheduling for multi-hop sensor networks: Non-convexity and approximation algorithm. In *IEEE INFOCOM 2007. 26th IEEE International Conference on Computer Communications* (2007), pp. 1568–1576.
[19] ZHU, T., ZHONG, Z., GU, Y., HE, T., AND ZHANG, Z. Leakage-aware energy synchronization for wireless sensor networks. In *Proceedings of the 7th international conference on Mobile systems, applications, and services* (2009), ACM, pp. 319–332.

IX. APPENDIX

A. Proof for Lemma 2:

Proof: Firstly, it is easy to prove that $g(\Gamma_i) = D_{max} - D_{\Gamma_i}$ is non-negative and monotone. This is simply because $D_{max} \geq D_{\Gamma_i}$ and $D_{\Gamma_i \cup \{t\}} \leq D_{\Gamma_i}$, where $\Gamma_i \cup \{t\}$ denotes the work schedule by augmenting an active instance into Γ_i at interval t.

In order to prove that $g(\Gamma_i)$ is submodular, it is equivalent to prove the following property:

$$D_{\Gamma_S} - D_{\Gamma_S \cup \{t\}} \geq D_{\Gamma_T} - D_{\Gamma_T \cup \{t\}} \tag{2}$$

where $\Gamma_S \subseteq \Gamma_T$. To establish this result, we need to look, implicitly or explicitly, at the expression $D_{\Gamma_S} - D_{\Gamma_S \cup \{t\}}$ for arbitrary work schedule Γ_S and interval t. In other words, what decrease do we get in the expected expected CTD when we add an active instance into the current work schedule at interval t. This decrease is very difficult to analyze directly, because it is hard to work with quantities of the form Equation 1. Our proof deals with this difficulty by formulating an equivalent view of the process, which provides an alternate way to reason with the submodularity property.

For a given node b, consider only one packet, and assume that its predecessor is p and successor is s; the schedule for nodes p and s is fixed. Consider a point in the transmission process when the packet has just arrived at p. Then p attempts to send it to node b, succeeding with probability p_{pb}. We can view the outcome of this random event as being determined by flipping a coin of bias p_{pb}. From the point of view of the process, we can in fact create a graph with nodes p, b, and s, by adding an "edge" from p to b for each possible attempted transmission (we thus have infinity edges), and each edge is associated with a weight indicating the sleep latency of this attempt. Similarly, we create set of weighted "edges" from b to s.

For ease of analysis, we decompose node b into T virtual nodes, each of which stands for an interval in period T. We further distribute the edges from p to b to corresponding virtual nodes at which the attempted transmission happens. Similarly, we assign the edges from b to p to corresponding virtual nodes at which the packet is ready at b. Then, for any schedule Γ_i, we have an induced subgraph by excluding those nodes (intervals) that are not contained in Γ_i. A coin of bias p_{pb} is flipped at the very beginning of the process on each edge from p to any of b's virtual nodes. With all of the coins flipped in advance, the process can be viewed as follows. The edges for which the coin flip indicated a transmission will be successful are declared to be live; the remaining edges are declared to be blocked. If we fix the outcomes of the coin flips and then initially assign b a schedule Γ_i, it is clear how we can determine the CTD the end of the process:

Claim 1: The CTD is the weight of the (weighted) shortest path from p to s consisting entirely of live edges in the reduced graph of Γ_i.

Consider the probability space in which each sample point specifies one possible set of outcomes for all of the coin flips on the edges. Let Y denote one sample point in this space, and define $D_Y(\Gamma_i)$ to be the CTD of the process when Γ_i is the schedule for node b, and Y is the set of outcomes of all coin flips on edges.

First, we claim that for each fixed outcome Y, the function $D_Y(\Gamma_i)$ satisfies Equation 2. To see this, let Γ_S and Γ_T be two schedules such that $\Gamma_S \subseteq \Gamma_T$, and consider the quantity: $D_Y(\Gamma_T) - D_Y(\Gamma_T \cup \{t\})$. This is the difference between the weight of shortest path in $D_Y(\Gamma_T)$ and in $D_Y(\Gamma_T \cup \{t\})$; then, there are two possible cases:

► $D_{\Gamma_T} - D_{\Gamma_T \cup \{t\}} > 0$: it indicates that a new shortest path can be found from p to s through t. Together with $S \subseteq T$, we have $D_Y(\Gamma_T \cup \{t\} = D_Y(\Gamma_S \cup \{t\})$, *e.g.*, the new shortest path also has the smallest weight in $D_Y(\Gamma_S \cup \{t\})$. Due to the fact that $D_Y(\Gamma_T) \leq D_Y(\Gamma_S)$, we get $D_Y(\Gamma_S) - D_Y(\Gamma_S \cup \{t\}) \geq D_Y(\Gamma_T) - D_Y(\Gamma_T \cup \{t\})$.

► $D_{\Gamma_T} - D_{\Gamma_T \cup \{t\}} = 0$: it implies that the original shortest path in D_{Γ_T} still has the smallest weight after adding t. Since $D_Y(\Gamma_S) - D_Y(\Gamma_S \cup \{t\}) \geq 0$ always holds, we have $D_Y(\Gamma_S) - D_Y(\Gamma_S \cup \{t\}) \geq D_Y(\Gamma_T) - D_Y(\Gamma_T \cup \{t\})$.

This finishes the proof for $D_Y(\Gamma_S) - D_Y(\Gamma_S \cup \{t\}) \geq D_Y(\Gamma_T) - D_Y(\Gamma_T \cup \{t\})$ for a fixed outcome Y. We finally have:

$$D_{\Gamma_S} = \sum_{\text{outcomes } Y} \text{Prob}[Y] \cdot D_Y(\Gamma_S)$$

Since the expected CTD is just the weighted average over all outcomes, because a non-negative linear combination of $D_Y(\Gamma_S)$ still satisfies Equation 2, hence Equation 2 also holds. Continuing this reasoning, we can extend the result to multiple packets and predecessor-successor pairs. ■