Multiple Task Scheduling for Low-Duty-Cycled Wireless Sensor Networks

Shuguang Xiong[1], Jianzhong Li[1], Mo Li[2], Jiliang Wang[3], Yunhao Liu[3,4]
[1]Schoole of Computer Science and Technology, Harbin Institute of Technology, China
[2]School of Computer Engineering, Nanyang Technological University, Singapore
[3]CSE Department, Hong Kong University of Science and Technology, Hong Kong, China
[4]Tshinghua National Laboratory for Information Science and Technology, Tsinghua University, China
Email: n2xiong@gmail.com, lijzh@hit.edu.cn, limo@ntu.edu.sg, {aliang, liu}@cse.ust.hk

***Abstract*—For energy conservation, a wireless sensor network is usually designed to work in a low-duty-cycle mode, in which a sensor node keeps active for a small percentage of time during its working period. In applications where there are multiple data delivery tasks with high data rates and time constraints, low-duty-cycle working mode may cause severe transmission congestion and data loss. In order to alleviate congestion and reduce data loss, the tasks need to be carefully scheduled to balance the workloads among the sensor nodes in both spatial and temporal dimensions. This paper studies the load balancing problem, and proves it is NP-Complete in general network graphs. Two efficient scheduling algorithms to achieve load balance are proposed and analyzed. Furthermore, a task scheduling protocol is designed relying on the proposed algorithms. To the best of our knowledge, this paper is the first one to tackle multiple task scheduling for low-duty-cycled sensor networks. The simulation results show that the proposed algorithms greatly improve the network performance in most scenarios.**

I. Introduction

Wireless sensor networks (WSNs) have great potential to be used in many long-term applications such as environmental surveillance [17] [21], structure monitoring [13] [9], habitat research [15], and etc. To bridge the gap between limited energy supplies of the sensor nodes [6] [8] and the system lifetime, many research studies suggest the WSNs operated in low-duty-cycle mode [8] [7] [6]. In low-duty-cycled sensor networks, a sensor node keeps its radio on for a small percentage (e.g., less than 5%) of time during each working period. As reported in recent literatures [8] and [10], idle listening is a major source of energy consumption that accounts for most of the energy cost at sensor nodes. The low-duty-cycle working mode notably reduces the energy consumption in idle listening, thus prolonging the lifetime of a WSN. In low-duty-cycled sensor networks, the working period of a sensor node is divided into a number of time slots with equal length. A sensor node chooses one time slot as its active time, and keeps radio on to receive data only at that time slot.

The low-duty-cycled WSN extensively prolongs the network lifetime at the expense of extremely shortened available time period for sensors to receive data. Two inevitable problems, however, arise with the low-duty-cycle working mode. First, severe transmission congestion will be introduced when multiple nodes send data to the same node during its extremely shortened active time, which causes packet loss and decreases the network performance [19] [2]. Second, due to the transmission congestion and the increased per-hop transmission delay, a node may not acquire adequate bandwidth to forward the data packets it has received in time. Data packets are prone to get dropped due to buffer overflow in high data rate applications [9] [24]. The two problems may become even more severe when multiple tasks of data forwarding exist in the network. The transmission schedule without careful design may lead to highly unbalanced use of the forwarding capability of the network in both spatial and temporal dimensions. The unbalanced traffic burden may further intensify transmission congestion and increase transmission delay.

In order to coordinate multiple data forwarding tasks with time constraints, the tasks need to be carefully scheduled so that the workloads can be balanced across the sensors in compliance with their own working schedules. However, very few works have focused on improving the efficiency of multiple task scheduling in low-duty-cycled WSNs. To the best of our knowledge, the problem, which is to find out an optimal schedule for given data delivery requests with specified time constraints such that the workloads are evenly distributed over the sensor nodes, remains unsolved.

In this paper, we thoroughly investigate the multiple task scheduling problem for low-duty-cycled WSNs, and propose efficient algorithms to schedule the tasks for balanced use of sensors. In summary, we (1) formulate the Load Balancing (LB) problem, and propose a polynomial-time algorithm for scheduling the tasks in a tree, (2) prove that the LB problem in general network graphs is NP-Complete, and propose an approximation algorithm with performance analysis, (3) design a distributed task scheduling protocol for practical networks, and (4) conduct extensive simulations to evaluate the performances of the algorithms. The results show that the network performance is notably improved in most scenarios by the algorithms. As far as we know, this paper is the first one to tackle multiple task scheduling for low-duty-cycled WSNs.

The rest of this paper is organized as follows. Section II briefly summarizes related works. Section III introduces the network model and gives the problem description. Section IV proposes an algorithm that achieves the optimal schedule for tasks in a tree. Section V further investigates the hardness of this problem in general network graphs, and proposes

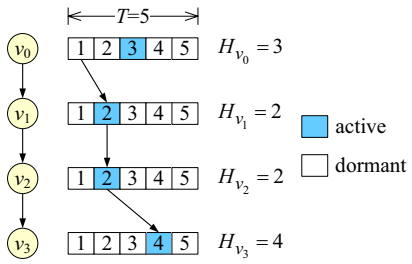

Fig. 1. A sample low-duty-cycled network, in which $T = 5$, $P = 2$ and $D = 4$. The data generated by node v_0 are delivered through path $v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3$ resulting in time delay 4.

a heuristic algorithm. Section VI describes the design of a distributed protocol for practical networks. In Section VII, we present the simulation results. Section VIII concludes this paper and suggests possible future works.

II. Related Work

There are a number of works that focus on scheduling algorithms in WSNs, with the goals to minimize communication latency [14] [18], avoid collision [3] [2] [26], or achieve energy efficiency [23] or fairness [19].

In [14], Lu et al. study how to minimize the communication latency given that each sensor has a duty cycling requirement of being awake for only $1/k$ time slots on an average. [18] proposes a heuristic scheduling algorithm to reduce the time delay in data aggregation applications. Gandham et al. propose a distributed edge coloring algorithm to derive a collision-free schedule [3]. For high data rate sensor network applications, a novel scheduling technique called Dynamic Conflict-free Query Scheduling (DCQS) is proposed in [2]. [26] presents a distributed algorithm to generate a collision-free schedule for data aggregation in WSNs. Rao et al. propose a practical distributed algorithm to compute a time-slot based schedule that provides end-to-end Max-Min fairness to multi-hop wireless networks [19]. Recently, Tan et al. explore distributed opportunistic scheduling (DOS) with delay constraints for throughput maximization with respect to two different types of average delay constraints [20].

Our work differs from above existing works in that we consider the load balancing problem and time schedules for multiple data delivery tasks rather than the schedules for individual links that mainly try to avoid collisions. The algorithms are designed specifically for low-duty-cycled WSNs, especially for data collection applications.

Data delivery and dissemination mechanisms in low-duty-cycled WSNs are studied in recent works [8] [7] [6]. In [8], Guo et al. study opportunistic flooding for low-duty-cycled sensor networks with unreliable links. Gu et al. propose a method for increasing duty-cycle at individual node, and a scheme on placement of sink nodes to provide real-time guarantee of communication delay [7]. A dynamic data forwarding (DSF) scheme is presented in [6] with experiments conducted on low-duty-cycled WSNs. Compared with the works above, this paper aims to achieve load balance among the sensor nodes without increased duty cycles and additional sink nodes.

A number of studies have investigated routing tree construction for data gathering in WSNs [25] [11] [27]. The problem of constructing a data gathering tree to maximize the network lifetime is shown to be NP-Complete in [25]. Khan et al. present a distributed algorithm that constructs an $O(\log n)$-approximate minimum spanning tree (MST) in a network [11]. Using real-time reinforcement learning strategies, Zhang et al. propose an adaptive spanning tree routing mechanism [27].

III. Network Model and Problem Description

A. Network Model

In this paper, a sensor network is regarded as an undirected graph $G(V, E)$, where V refers to the set of sensor nodes, and E stands for the set of radio links between the nodes in V. For energy conservation, the nodes work in low-duty-cycle mode [8] [7] [6], in which the working period T of a node v is divided into a number of equal-length time slots. In each working period, v turns on its radio to receive data in only one time slot, which is called the active time of v, denoted as H_v. In the rest time slots, v remains dormant unless it sends data. For simplicity, the length of a time slot is set to 1, which is considered as the minimum time unit.

A *task* specifies a data delivery request from a source node to a destination node through a given path. Consider n tasks in the network, and each task TASK_i $(1 \leq i \leq n)$ is represented by $\langle v_{s_i}, v_{d_i}, \text{PATH}_i, \text{NODE}_i, D_i \rangle$, in which v_{s_i} and v_{d_i} are the source node and the destination node, resp., PATH_i and NODE_i refer to the edges and the nodes on the data delivery path from v_{s_i} to v_{d_i}, resp., and D_i is the *time constraint* of the task.

The data of a task can be delivered in one hop from node u to node v at time slot j if the data has been generated by u or u received the data at time slot i, where $i \leq j \leq i + P$ and P is called the per-hop time constraint.

A *time schedule* S for the tasks records the times for the sensor nodes to receive data. Specifically, $S(i, j)$ in the schedule refers to the time for node $v_j \in \text{NODE}_i \setminus \{v_{s_i}\}$ to receive the data of TASK_i, and $S(i, d_i)$ is regarded as the *time delay* of TASK_i. S is feasible if $S(i, p) \leq S(i, q) \leq S(i, p) + P$ $\forall v_p \rightarrow v_q \in \text{PATH}_i$, and $S(i, d_i) \leq D_i$. Given a time schedule S, the *workload* of node v_i at time j, denoted as $w(i, j)$, is the total number of data received by v_i at time j. For convenience, the time schedule of TASK_i can also be expressed by $v_{s_i} \rightarrow v_{k_1}(t_{k_1}) \rightarrow \ldots \rightarrow v_{d_i}(t_{d_i})$, where t_j in the brackets is the time when v_j receives data from the precedent node along the data delivery path. Clearly, $t_{d_i} = S(i, d_i)$.

We assume that the sensor nodes are synchronized [16] and have the same working period T, and each node knows the active times of its neighbors in advance.

Figure 1 illustrates a low-duty-cycled sensor network with a line topology, and a task for delivering v_0's data to v_3. The data are generated at time 1, and sent to v_1, v_2 and v_3 at time 2, 2, 4, resp. Note that when any other data received by v_0 at time 3, they cannot be delivered to v_3 in time no more than T since there is no valid non-descending order of time within $[3, D]$ for the nodes in path $v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3$.

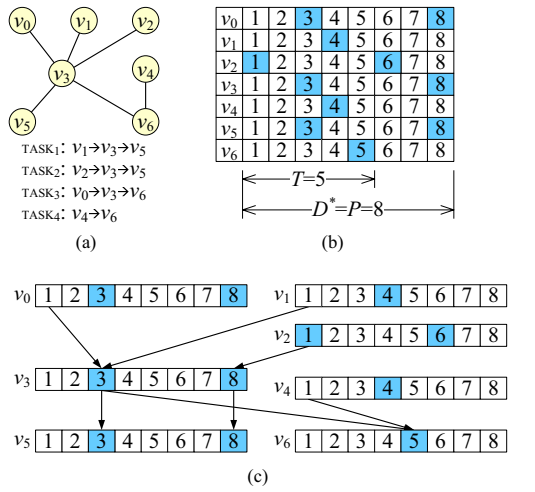

Fig. 2. An instance of the LB problem and an optimal schedule. (a) The network topology and the tasks. (b) The active and dormant states of the nodes with $T = 5$ and $D = P = 8$. (c) An optimal time schedule with maximum workload $W(S) = 2$, which appears at v_3 at time 3, and v_6 at time 5.

B. Problem Description

The Load Balancing (LB) Problem can be described as: given a sensor network with working period T, per-hop time constraint P, active time H_v $\forall v \in V$, and n tasks $\langle v_{s_i}, v_{d_i}, \text{PATH}_i, \text{NODE}_i, D_i \rangle$ for $1 \leq i \leq n$, to derive a schedule S so as to minimize $W(S)$, the max workload of the nodes in each time slot. Formally, let $x(i, j, k)$ be a 1-0 integer variable indicating whether the data of TASK_i are received by node v_j at time k, $S(i, j) = k$ means $x(i, j, k) = 1$, and we have

$$\sum_{k=1}^{D_i} x(i, j, k) = 0, \forall v_j \notin \text{NODE}_i \setminus \{v_{s_i}\} \tag{1}$$

$$x(i, j, k) = 0, \forall k \leq D^*, (k-1)|T + 1 \neq H_{v_j} \tag{2}$$

$$\sum_{k=1}^{D_i} x(i, j, k) = 1, \forall v_j \in \text{NODE}_i \setminus \{v_{s_i}\} \tag{3}$$

$$x(i, p, k) \leq \sum_{l=k}^{k+P} x(i, q, l), \forall v_p \rightarrow v_q \in \text{PATH}_i \tag{4}$$

Let $D^* = \max\{D_i\}$ and $w(j, k) = \sum_{i=1}^{n} x(i, j, k)$ for $1 \leq i \leq n$, the goal is to minimize

$$W(S) = \max \left\{ \sum_{k=1}^{D^*} w(j, k) \right\}, \forall v_j \in V \tag{5}$$

Equation (1) ensures that the data for each task are delivered only by the nodes involved in the task. Equation (2) restricts the ability of each node to receive data when it is in dormant state. Equation (3) guarantees that the data for each task can be sent to their destination along the path in D_i time, while Inequality (4) limits that the data for each task are forwarded hop-by-hop with a delay no more than P. $w(j, k)$ in Equation (5) refers to the workload of node v_j at time k.

Figure 2 depicts an instance of the LB problem, in which part of the sensor nodes have only one time to receive data, e.g., v_1 and v_4, while the other sensor nodes have two time slots available for receiving data, e.g., v_3 and v_5. An optimal schedule with maximum workload 2 is shown in Figure 2(c), which can be represented as $v_1 \rightarrow v_3(3) \rightarrow v_5(3)$, $v_2 \rightarrow v_3(8) \rightarrow v_5(8)$, $v_0 \rightarrow v_3(3) \rightarrow v_6(5)$, and $v_4 \rightarrow v_6(5)$.

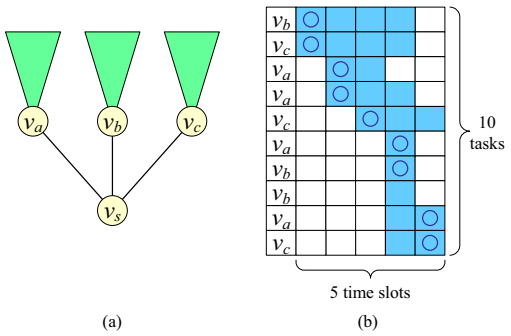

Fig. 3. Compute $W(S)$ for the tasks in a tree. (a) The tasks induce a tree topology rooted at v_s. (b) A task table of v_s, in which the time slots available for scheduling are shadowed. The cycles refer to the selected time slots in an infeasible schedule (since the 8-th task cannot be selected) with threshold $k = 2$ when the greedy algorithm ends. This implies that no feasible schedule exists for $W(S) \leq 2$. In fact, we can see that $W(S) = 3$ for this instance.

Different time schedules result in variant maximum workloads. For example, if the time schedule of TASK_2 changes to $v_2 \rightarrow v_3(3) \rightarrow v_5(3)$ from the above schedule, then the maximum workload increases to 3. Another related problem is "whether a feasible time schedule exists such that all the tasks can be done within their time delay constraints?". This problem is easy to solve by constructing a schedule in which a node always forwards data to its next-hop neighbor in a task as soon as possible. The answer to this question is "no" *iff* such a construction fails due to no time slot available during the construction. In the rest of the paper, we only consider the scenarios where a feasible time schedule exists.

IV. Scheduling Algorithm for Tasks in A Tree

We begin with a special case of the LB problem, in which all the tasks have a common destination v_s, and the paths induce a directed tree rooted at v_s (see Figure 3(a)), i.e., once two paths intersect at some node v, the rest parts of the two paths starting from v are identical. This special case is often in accordance with real applications, e.g., data collection of all the sensor nodes via a routing tree. We assume that $P = D^*$ for simplicity in this section. However, the algorithms apply to the problem without this restriction as well.

Lemma 1. *In any optimal time schedule S, the workload of v_s at some time is equal to $W(S)$.*

Proof: For contradiction, suppose that the workload of v_s at any time t ($1 \leq t \leq D^*$) is less than $W(S)$, then there must be a node v_j and time k such that $w(j, k) = W(S)$. Since v_s is the only destination of the tasks, the data received by v_j at time k will finally arrive at v_s in p ($p \geq 1$) time slots t_1, t_2, ..., t_p, $t_i < t_j \forall 1 \leq i < j \leq p$. Because part of the data received by v_j at time k arrive at v_s at time t_1, there is a non-descending order of active time of the nodes along the path, varying from k to t_1. Accordingly, there is a non-

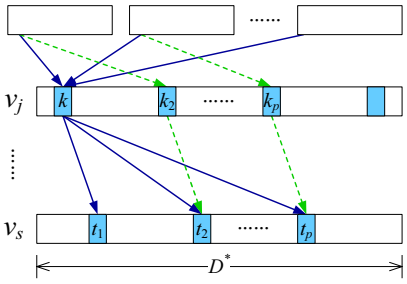

Fig. 4. Postpone the data transmissions from other nodes to node v_j at time k in a tree topology, so that the altered workload $w'(j,k)$ is no greater than $w(s,t_1)$, while the workloads on v_s remains unchanged.

descending order of active time varying from k_q to t_q where $k_q = k + (t_q - t_1)$ for $2 \le q \le p$. This implies that any data received by v_j at time k_q can be delivered to v_s at time t_q. Thus, for the $w(s,t_q)$ data that will be received by v_s at time t_q ($2 \le q \le p$), we can postpone the time for v_j to receive them from k to k_q. As shown in Figure 4, this operation results in a feasible schedule without alternating $w(s,t)$, $1 \le t \le D^*$.

Let the workload of v_j be $w'(j,k)$ after the operation, we have $w'(j,k) \le w(s,t_1)$ since v_s may receive data from the nodes other than v_j. Furthermore, $w'(j,k) < W(S)$ due to $w(s,t_1) < W(S)$. For $k = 1$ to D^*, we conduct the operation if $w(j,k) = W(S)$, and when all the operations are done, the maximum workload of v_j is less than $W(S)$.

Next, the operations are carried out for all the nodes in a topological order, i.e., the workload of a node u can be re-balanced only when the workloads of all its children have already been re-balanced. Since the topology is a tree, this order guarantees that u's workload cannot be alternated by the operations for its ancestors. Hence, when this process finishes, all the nodes except v_s have a maximum workload less than $W(S)$, which is contradict to the assumption. ■

Lemma 1 suggests that if we can find a feasible schedule that minimizes the maximum workload of v_s, then this feasible schedule is overall optimal. We design a polynomial-time algorithm *SAT* (Scheduling Algorithm for Tree topology), which (1) computes a *task table* of v_s recording the time range of each task available for scheduling in the *data preparation step*, (2) computes an optimal redistribution of the workloads of v_s under which the maximum workload of v_s is minimized in the *workload minimization step*, and (3) derive a feasible schedule for all the nodes in the *schedule generation step*.

The data preparation step. $\forall v_p \rightarrow v_q \in \text{PATH}_i$ in TASK_i ($1 \le i \le n$), the earliest time for v_q to receive the data is $t_e(i,q) = \min\{t | t \ge t_e(i,p)$, v_q is active at time $t\}$. For consistency, let $t_e(i,s_i) = 0$, and $t_e(i,s)$ can be computed. On the other hand, the latest time for v_s to receive the data is defined as $t_l(i,s) = \max\{t | t \le D_i$, v_s is active at time $t\}$.

Suppose there are m time slots when v_s is active in D^*, indexed by $1, 2, \ldots, m$, denote the indices of time $t_e(i,s)$ and $t_l(i,s)$ of TASK_i as $\text{TASK}_i.e$ and $\text{TASK}_i.l$, resp. The schedule for TASK_i at v_s can be expressed as an $1 \times m$ 1-0 vector, in which the entry in column k indicates whether the data of TASK_i are received by v_s at the k-th active time of v_s. TASK_i has a higher priority than TASK_j if (1) $\text{TASK}_i.e < \text{TASK}_j.e$, or (2) $\text{TASK}_i.l < \text{TASK}_j.l$ otherwise, or (3) $\text{ID}(v_i) < \text{ID}(v_j)$ otherwise, where $\text{ID}(v_k)$ means the ID of the node forwarding data to v_s in TASK_k, or (4) $i < j$ otherwise. The n vectors are sorted by the priority in descending order, and combined into an $n \times m$ task table. A feasible schedule must set one entry between column $\text{TASK}_i.e$ and $\text{TASK}_i.l$ to 1, and remain the other entries as 0 for each row i. An example is shown in Figure 3(b).

Algorithm 1 The Greedy Algorithm

INPUT: The task table organized as a priority queue Q, and a threshold k ($1 \le k \le n$).
OUTPUT: Whether a feasible schedule A exists with a maximum workload no more than k.

```
1: num=0; /* the number of unscheduled tasks */
2: while Q is not empty do
3:    count=0;
4:    i=top(Q).e;
5:    while Q is not empty, and top(Q).e==i do
6:       if count< k then
7:          A(top(Q).r, i)=1; /* schedule the task at time i */
8:          extract top(Q) from Q;
9:          count++;
10:      else if i < top(Q).l then
11:         top(Q).e=i + 1; /* update Q */
12:      else
13:         extract top(Q) from Q; /* an unscheduled task*/
14:         num++;
15: if num==0 then
16:    return true;
17: else
18:    return false;
```

The workload minimization step. Given the task table of v_s, for row i ($1 \le i \le n$), the workload minimization step needs to set one entry between column $\text{TASK}_i.e$ and $\text{TASK}_i.l$ to 1, so that the maximum sum of each column is minimized.

The key idea is a greedy algorithm that derives a feasible schedule with a given threshold k on the sum of each column. Given the task table organized as a priority queue Q, the algorithm schedules no more than k tasks at time i for $i = 1$ to m. Specifically, if there are no more than k tasks with earliest time i in Q, it schedules all the tasks at i, and extract them from Q. Otherwise, only the first k tasks are scheduled at i and extracted, while the earliest times of the rest tasks are altered to $i+1$. If some tasks are not scheduled at the end of the procedure, the algorithm returns false, otherwise it returns true and a feasible schedule A at v_s, in which $A(i,j) = 1$ records that the i-th task is scheduled at the j-th active time of v_s. The pesudo code is shown in Algorithm 1.

Algorithm 1 schedules at most n tasks, and each scheduling requires $O(n)$ extract and update operations of Q, each of which consumes $O(\log n)$ time. Hence, the time complexity of the greedy algorithm is $O(n^2 \log n)$. Furthermore, this upper bound is tight. In a worst case, $\text{TASK}_i.e = 1$ and $\text{TASK}_i.e = m$ ($m \ge n$) for $1 \le i \le n$, and $k | n = 0$. The algorithm conducts $n - (i-1)k$ extract and update operations to schedule k tasks at the i-th time, hence the running time is $\sum_{i=1}^{n/k} (n - (i-1)k) \log n = \frac{1}{2k}(n^2 + kn) \log n$.

Lemma 2. *The greedy algorithm returns true if and only if there is a feasible schedule with threshold k.*

Proof: The reason is two-fold. First, if it returns true, all the tasks are scheduled in the derived schedule, and the sum of each column is no more than k. Second, if it returns false, suppose there is a feasible schedule, denoted as B, we perform the following transformations on B.

Let us consider the first column of the task table, and $\sum_{i=1}^{n} B(i,1) \leq \sum_{i=1}^{n} A(i,1)$ since the latter is the maximum possible sum of this column in any feasible schedule. If $\sum_{i=1}^{n} B(i,1) < \sum_{i=1}^{n} A(i,1)$, we can find $\sum_{i=1}^{n} A(i,1) - \sum_{i=1}^{n} B(i,1)$ tasks not scheduled at time 1 in B, then alter all their schedules to 1, so as to make the two sums equal.

If $\exists p$ such that $B(p,1) = 1$ and $A(p,1) = 0$, then $\exists q$ such that $B(q,1) = 0$ and $A(q,1) = 1$, and $\exists c > 1$ such that $B(q,c) = 1$. According to condition (2), schedule A always chooses the first $x \leq k$ tasks sorted by the latest time in non-descending order, and the latest time of task p is no less than that of task q, hence we can set $B(q,c) = 0$ and $B(p,c) = 1$. For load balance, we also set $B(p,1) = 0$ and $B(q,1) = 1$. Repeat this procedure until no such p and q can be found, and it is clear that A and B are identical in the first column.

Suppose $\sum_{i=1}^{n} B(i,j) \leq \sum_{i=1}^{n} A(i,j)$ after the transformations performed on column 1 to j, the transformation is also conducted on column $j+1$. The only difference is that $B(q,j+1) = 0$ if $A(q,j+1) = 0$ and $A(s,j+1) = 1$ for some $s < q$. Hence, $\sum_{i=1}^{n} B(i,j+1) \leq \sum_{i=1}^{n} A(i,j+1)$.

We conduct this transformation for column 1 to m to obtain a schedule C. On one hand, C is feasible since no more than k entries are selected in each column in C, and all the tasks are scheduled in C. On the other hand, the sum of each column in C is no more than that in A, thus C is infeasible since A is infeasible. Therefore, no feasible schedule exists unless Algorithm 1 returns true. ∎

According to Lemma 2, $W(S)$ is the minimum threshold k that makes the greedy algorithm return true. Hence $W(S)$ can be determined by executing a binary search on k in the range from 1 to n. The workload minimization step requires $O(n \log n)$ time to build the priority queue, and calls Algorithm 1 in each step of the binary search, so the time complexity of this step is $O(n^2 \log^2 n)$.

The schedule generation step. According to the computed schedule of v_s, this step schedules the tasks on the rest nodes. Recall that $\forall v_p \rightarrow v_q \in \text{PATH}_i$ in TASK_i $(1 \leq i \leq n)$, the earliest time for v_q to receive the data is $t_e(i,q) = \min\{t | t \geq t_e(i,p)$ and v_q is active at time $t\}$. Let v_q $(v_q \neq v_s)$ be scheduled at time $t_e(i,q)$ to receive the data of TASK_i, while v_s receives the data of TASK_i at its j-th active time, which is no less than $t_e(i,s)$. By this approach, we can obtain an optimal schedule.

A better approach can be employed to balance the workloads of v $(v \neq v_s)$. Specifically, if the times for v_q to receive the data from v_p for the tasks passing v_p are all determined, we can employ an algorithm similar to the workload minimization step to compute the schedule on v_p. The two differences only lie in the input of the greedy algorithm. First, the priority queue Q_p consists of $n_p < n$ tasks, and $v_p \rightarrow v_q \in \text{PATH}_i$, $\forall$ $\text{TASK}_i \in Q_p$. Second, the latest time for v_p to receive the data of TASK_i, $t_l(i,p)$, is defined as $t_l(i,p) = \max\{t | t \leq t_l(i,q)$ and v_p is active at time $t\}$, and the definition of $\text{TASK}_i.l$ is changed accordingly.

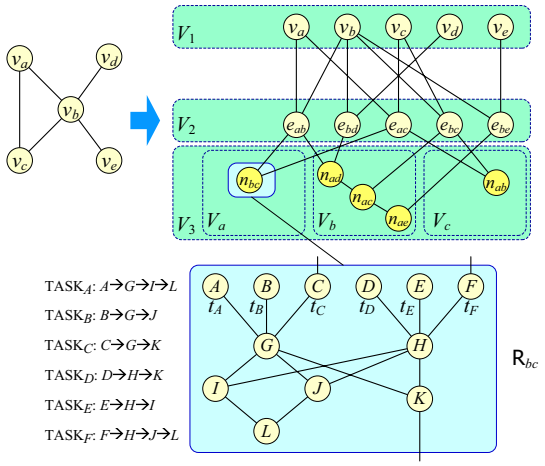

Fig. 5. The construction from an instance of the G3C problem to an instance of the LBS problem. We can see the restriction components enforce the tasks starting from a node in V_1 must be scheduled at the same time slot whenever the instance of G3C is 3-colorable.

It is easy to see that $\forall v_p \rightarrow v_q \in \text{PATH}_i$, $\forall$ $\text{TASK}_i \in Q_p$, the scheduled time of v_p to receive the data of TASK_i is no more than that of v_q, hence the schedule generation step outputs a feasible optimal schedule that also balances the workloads of the intermediate sensor nodes.

Theorem 1. *Given n tasks and the induced tree with m nodes, the SAT algorithm computes an optimal schedule in $O(mn^2 \log^2 n)$ time.*

Proof: According to Lemma 1 and Lemma 2, the SAT algorithm computes an optimal schedule S with $W(S) = w(v_s,t)$ for some $1 \leq t \leq D^*$. As discussed above, the data preparation step and the workload minimization step require $O(mn + n \log n)$, and $O(n^2 \log^2 n)$ time, resp. Since the schedule generation step calls Algorithm 1 once for each node in the tree, its time complexity $O(mn^2 \log^2 n)$ dominates the running time of the SAT algorithm. ∎

V. Scheduling Algorithm for General Case

In general case of the LB problem, the graph induced by the paths of the tasks is not necessarily a tree. This section proves that the LB problem is NP-Complete, and then provides a heuristic algorithm with the analysis for its performance.

A. Hardness of the LB Problem

Consider a special case of the LB problem (denoted as the LBS problem) with additional restrictions: (1) $T_1 = T_2 = \ldots T_{|V|} = T = 1$, (2) $P = 0$, which implies that the nodes in NODE_i must be scheduled to receive the data of TASK_i at the same time, and (3) $D_1 = \ldots = D_n = 3$, which means that each task needs to be scheduled within delay no more than 3.

A *restriction component* involves a 12-node graph and 6 tasks, as shown in Figure 5. The nodes are labeled from A to K, and the paths of the 6 tasks are $\text{PATH}_A = A \rightarrow G \rightarrow I \rightarrow L$, $\text{PATH}_B = B \rightarrow G \rightarrow J$, $\text{PATH}_C = C \rightarrow G \rightarrow K$, $\text{PATH}_D = D \rightarrow H \rightarrow K$, $\text{PATH}_E = E \rightarrow H \rightarrow I$, and $\text{PATH}_F = F \rightarrow$

$H \to J \to L$. The scheduled time of the tasks are denoted as t_A, t_B, t_C, t_D, t_E, t_F, resp. The restriction component is used to enforce that the assigned time of TASK_C and TASK_F are equal in a schedule with $W(S) = 1$, as Lemma 3 indicates.

Lemma 3. *A schedule S for the six tasks exists with $W(S) = 1$ if and only if $t_A = t_D$, $t_B = t_E$, and $t_C = t_F$.*

Proof: A schedule S exists with $W(S) = 1$ indicates that (1) $t_A \neq t_B \neq t_C$ since $\text{NODE}_A \cap \text{NODE}_B \cap \text{NODE}_C = \{G\}$, (2) $t_B \neq t_F$ since $\text{NODE}_B \cap \text{NODE}_F = \{J\}$, and (3) $t_A \neq t_F$ since $\text{NODE}_A \cap \text{NODE}_F = \{L\}$. From the above three facts and $t_A, \ldots, t_F \in \{1, 2, 3\}$, $t_C = t_F$. Furthermore, $t_A \neq t_E$ because $\text{NODE}_A \cap \text{NODE}_E = \{I\}$, hence $t_A = t_D$ and $t_B = t_E$.

Conversely, if $t_A = t_D$, $t_B = t_E$, and $t_C = t_F$, let $t_A = t_D = 1$, $t_B = t_E = 1$, and $t_C = t_F = 3$, and it is easy to see that in each time slot, a node receives data from at most one other node, thus the derived schedule S has $W(S) = 1$. ■

Theorem 2. *The LBS problem is NP-Complete.*

Proof: Given a schedule S for a LBS instance composed of n tasks, the claim that $W(S) > 1$ can be verified in polynomial time in n. Hence LBS $\in$ NP. Next, we construct a polynomial-time reduction from the Graph 3-Colorability (G3C) problem [4] to LBS, as shown in Figure 5. G3C can be described as: given an undirected graph $G(V, E)$, whether $G(V, E)$ is 3-colorable, that is, does there exist a function $f : V \to \{1, 2, 3\}$ such that $f(u) \neq f(v)$ whenever $uv \in E$.

Let $G(V, E)$ be an arbitrary instance of G3C, the construction of the instance of LBS consists of two steps: the construction of the graph $G'(V', E')$, and that of the tasks.

First, $\forall v_i \in V$, a node also labeled as v_i is added in V'. The set of the $|V|$ nodes in V' is denoted as V_1. After that, $\forall v_i v_j \in E$, a node labeled as e_{ij} is added in V'. The set of the $|E|$ nodes in V' is denoted as V_2. $\forall e_{ij} \in V_2$, two edges $v_i e_{ij}$ and $v_j e_{ij}$ $v_i, v_j \in V_1$ are added in E'. Note that e_{ij} and e_{ji} refers to the same node. Next, a set of nodes V_i are added in V' $\forall v_i \in V$ and $|N_{v_i}| > 1$, where N_{v_i} refers to the neighbors of v_i in G. Let $p = N_{v_i}$, for v_i and v_{i_1}, v_{i_2}, …, $v_{i_p} \in N_{v_i}$, there are $p-1$ nodes in V_i, denoted as $n_{i_1 i_2}$, $n_{i_1 i_3}$, …, $n_{i_1 i_p}$, resp. Then in G', $n_{i_1 i_2}$ is connected to e_{ii_1} and e_{ii_2}, and $n_{i_1 i_j}$ is connected to $n_{i_1 i_{j-1}}$ and e_{ii_j} in G' for $j = 3$ to p. Define the union of such V_i as V_3. Finally, each node $n_{i_1 i_j} \in V_i$ is replaced by a restriction component $\mathsf{R}_{i_1 i_j}$. The edge that connects $n_{i_1 i_{j-1}}/e_{ii_1}$ and $n_{i_1 i_j}$ now connects K in $\mathsf{R}_{i_1 i_{j-1}}/\mathsf{R}_{ii_1}$ and C in $\mathsf{R}_{i_1 i_j}$. Similarly, the edge that connects e_{ii_j} and $n_{i_1 i_j}$ now connects e_{ii_j} and F in $\mathsf{R}_{i_1 i_j}$.

Second, let $|N_{v_i}|$ tasks start from $\forall v_i \in V_1$. The first task has a path $v_i \to e_{ii_1} \to (C \to G \to K)\mathsf{R}_{i_1 i_2} \to \ldots \to (C \to G \to K)\mathsf{R}_{i_1 i_p}$, where $(C \to G \to K)\mathsf{R}_{i_1 i_j}$ refers to a path inside $\mathsf{R}_{i_1 i_j}$. The j-th ($2 \leq j \leq |N_{v_i}|$) task has a path $v_i \to e_{ii_j} \to (F \to H \to J \to L)\mathsf{R}_{i_1 i_j}$. TASK_A, TASK_B, TASK_D and TASK_E for each restriction component are reserved.

Suppose G is 3-colorable, we set the time of the tasks starting from $v_i \in V_1$ as the color of $v_i \in V$. As a result, $e_{ij} \in V_2$ receives the data from v_i and v_j in different time slots since the colors of v_i and v_j are different in G. Besides, node C and F in each restriction component receive data at the same time. By Lemma 3, the four tasks starting from A, B, D, E can be scheduled at a time so that each node in the component receives data from at most one other node per time slot. Hence a schedule S with $W(S) = 1$ can be obtained.

Conversely, if $\exists S$ with $W(S) = 1$, by Lemma 3, the assigned time of the tasks that starting from $v_i \in V_1$ must be the same. Thus we can set the color of $v_i \in V$ as the assigned time of the tasks. Since $W(S) = 1$, $e_{ij} \in V_2$ receives the data from v_i and v_j ($v_i, v_j \in V_1$) in different time slots, hence the colors of v_i and v_j in G must be different if $v_i v_j \in E$, which implies that G is 3-colorable.

It is easy to see that the construction takes polynomial time in the input size of G. Because G3C is known as NP-Complete [4], the LBS problem is NP-Complete. ■

B. A Heuristic Algorithm

Since the problem is NP-Hard, we present a heuristic algorithm, named *SAG* (Scheduling Algorithm for General case). The basic idea is to compute an initial schedule in which a node always forwards data to its next-hop neighbor in each task as soon as possible, and then postpone the time of a task at some nodes in order to reduce current maximum workload.

As shown in Algorithm 2, the output of SAG is a time schedule S, represented by $I(i, j)$ for each TASK_i and node $v_j \in \text{NODE}_i$, which indicates that the data of TASK_i is received by v_j in its $I(i, j)$-th active time. Recall that $S(i, j)$ refers to the time when v_j receive the data, the conversion from $I(i, j)$ to $S(i, j)$ is denoted as $S(i, j) = \textit{time}(I(i, j))$.

At first, SAG computes the minimum index $I(i, j)$ of node v_j's active time to receive the data of TASK_i. Then the initial workload of v_j in its t-th active time is set to the number of tasks scheduled in its t-th active time. Next, SAG continues to find out a node v_j whose workload at time k is equal to current maximum workload, and then tries to find out TASK_i and delay δ so that the time for v_j to receive the data can be postponed to its $(I(i, j) + \delta)$-th active time, by which $w(i, j)$ and $W(S)$ may be reduced. Here two types of operations are employed: (1) all the times of TASK_i on v_j and v_j's successors in PATH_i are postponed by δ. (2) all the times of TASK_i on v_j's predecessors in PATH_i are postponed by δ.

The algorithm checks whether (1) $\textit{time}(I(i, d_i) + \delta) \leq D_i$, i.e., the postponed time on v_{d_i} is no more than D_i, and (2) $\textit{time}(I(i, j) + \delta) - \textit{time}(I(i, p)) \leq P$, where $v_p \to v_j \in \text{PATH}_i$. If so, and operation (1) is beneficial, i.e., the altered schedule by operation (1) has a less maximum workload, then SAG performs operation (1). Besides, if operation (1) is performed and operation (2) is beneficial, SAG performs operation (2).

SAG terminates if no operation can be performed to reduce $W(S)$. Because it requires $O(|V|^2 D^* n)$ time to check whether part of a task can be postponed by δ, and an operation makes at least one $I(i, j)$ increased by $\delta \geq 1$ ($1 \leq I(i, j) \leq D^*$ for each TASK_i and $v_j \in V$), it terminates in $O(|V|^3 D^{*2} n^2)$ time.

VI. Protocol Design and Analysis

Based on the proposed algorithms, we design a task scheduling protocol TSP for low-duty-cycled WSNs.

Algorithm 2 The Heuristic Algorithm SAG

INPUT: n tasks TASK$_i$ ($1 \leq n$) and the induced graph $G(V, E)$.
OUTPUT: $I(i, j)$ for each TASK$_i$, $v_j \in V$.

for all $v_j \in V$ **do**
  for all TASK$_i$ **do**
    $I(i, j)$=the earliest time for v_j to receive the data;
  compute $w(j, t)$ at each time t;
while *end*==false **do**
  compute $W(S) = \max\{w(j, k)\}$ $\forall v_j \in V$ and $\forall k \leq D^*$;
  select a node v_j such that $w(j, t) = W(S)$ for some t;
  let k =argmin$\{w(j, k) = W(S)\}$, *end*=true;
  if $\exists$ TASK$_i$ and $\delta \geq 0$ such that *time*$(I(i, d_i) + \delta) \leq D_i$ **then**
    if *time*$(I(i, j) + \delta)$− *time*$(I(i, p)) \leq P$, where $v_p \rightarrow v_j \in$ PATH$_i$ **then**
      if $w(x, I(i, x) + \delta) < W(S) - 1$ for all v_p's successor v_x in PATH$_i$ **then**
        end=false;
        for all v_p's successor v_x in PATH$_i$ **do**
          Update(i, x, δ);
        if $w(x, I(i, x) + \delta) < W(S) - 1$ for all v_j's predecessor v_x in PATH$_i$ **then**
          for all v_j's predecessor v_x in PATH$_i$ **do**
            Update(i, x, δ);

/* update the schedule for TASK$_i$ at node v_x */
Procedure Update(i, x, δ):
$w(x, I(i, x) + \delta) + +$;
$w(x, I(i, x)) - -$;
$I(i, x) += \delta$;

A. Protocol Description

The TSP protocol consists of the following two phases: the setup phase and the working phase.

The setup phase derives a *task schedule list* L for each node u, in which the value of the i-th entry $L(i) = S(i, v)$ records the time when the node should forward the data of TASK$_i$ to node v. A coordinator node (e.g., the sink) is required to execute the proposed algorithms to compute the schedule lists. If the coordinator has all the information of the tasks initially, it just disseminates the derived schedule lists to the nodes, otherwise the information of the tasks needs to be sent to the coordinator before the computations. However, we note that in a common case in which the sink collects the data of the sensor nodes via a routing tree as discussed in Section IV, the computation for the task schedule lists can be executed in a distributed manner, and a node only needs to send its own schedule list rather than the schedule lists of the predecessors, so the communication cost can be reduced.

When all the nodes involved in the tasks obtain their time schedule lists, they begin to forward data according to the schedule lists. The behavior of a node u in the working phase is regulated as follows. At the beginning of each time slot, by looking up its task schedule list, u checks whether there are data of some TASK$_i$ in its buffer which should be forwarded to the next-hop node in PATH$_i$ at or before this time slot. If yes, u turns its radio on and sends the data to the next-hop node. If this transmission is successful, u removes the data from its buffer. After that, u determines whether to keep its radio on during this time period according to its own time schedule. If yes, it can receive data in this time slot. When u receives the data of TASK$_i$, it checks whether the data should be forwarded in the same time slot by looking up the list. If yes, u forwards the data immediately, otherwise it stores the data in its buffer. u drops the data if the buffer is full.

B. Practical Issues

To make the TSP protocol available for various applications, two related practical issues must be addressed including local time synchronization, and computation and storage overhead of the sensors on which the protocol runs.

Local time synchronization. In Section III, we assume that the sensor nodes work in a synchronized mode. In fact, it is sufficient for a node to know the active time slot of its predecessors and successors, hence global synchronization is not a necessary requirement. To know those active time slots, simple and low-cost local synchronization techniques [16] can achieve an accuracy of $2.24\mu s$ with the cost of exchanging a few bytes of packets among neighboring nodes every 15 minutes. Since a time slot is typically longer than $2000\mu s$ [7] and the data can be transmitted at any time in the time period, the accuracy of $2.24\mu s$ is far more than sufficient.

Computation and storage overhead. By Theorem 1, when the task schedules are computed in a distributed manner, a node requires $O(n^2 \log^2 n)$ time to compute the schedule list on it. Besides, a node requires $O(n)$ space to store the task table, and the size of the schedule list is linear to the number of tasks that pass the node. Since the memory and the internal flash of a node is limited (e.g., $10k + 48k$ bytes on a TelosB mote [1]), and the time can be represented by a 4-byte integer, the TSP protocol can support thousands of tasks. If a node cannot afford the space for a packet-level scheduling, k tasks can be combined as one task, as long as the schedule list can be stored on the node.

VII. Simulation Evaluation

In order to evaluate the performance of the proposed algorithms, we conduct extensive simulations on the TOSSIM [12] simulator under variant network settings. We use *network yield* [22] as the primary measure, which is calculated by

$$\textit{Yield} = \frac{\#\text{ of data pkts received by the sink during } D^*}{\#\text{ of data pkts sent by all nodes during } D^*} \quad (6)$$

Besides, the simulations record the time delays of the tasks, the storage overflows on the sensor nodes, and the transmission losses between any two sensors. For comparison, the simulations also use the best-effort strategy (denoted BST) in forwarding the data of the tasks. The simulation results reveal that there is indeed an urgent need to deploy efficient schedules for multiple tasks with high data rates, and our proposed algorithms improve network performance notably.

A. Simulation Setup

In the simulations, 30∼100 sensor nodes are randomly deployed in a 100m ×100m square field with default transmission power. A time slot is set to 2 seconds, and the working period T of each node is 20 time slots, resulting in a 5% duty-cycled network. Initially, each node randomly selects a time slot in $[1, T]$ as its active time in each working period.

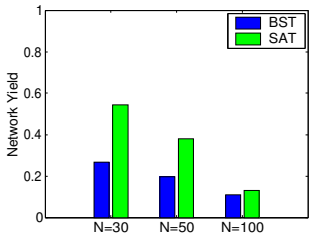

Fig. 6. Network Scale v.s. Network Yield.

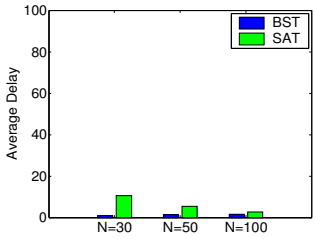

Fig. 7. Network Scale v.s. Average Delay of the Tasks ($D^* = 100$).

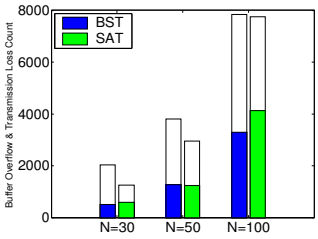

Fig. 8. Network Scale v.s. Buffer Overflow and Transmission Loss.

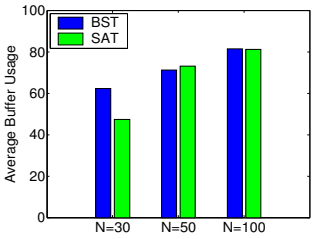

Fig. 9. Network Scale v.s. Average Buffer Usage ($B = 100$).

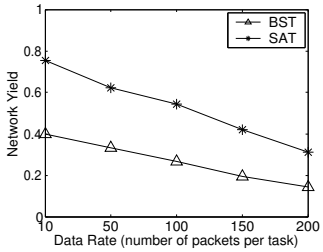

Fig. 10. Data Rate v.s. Network Yield.

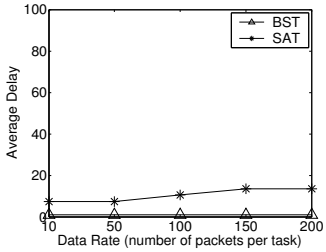

Fig. 11. Data Rate v.s. Average Delay of the Tasks ($D^* = 100$).

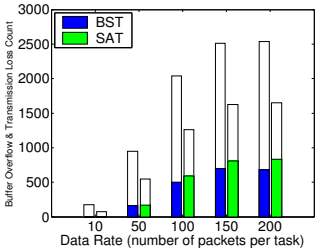

Fig. 12. Data Rate v.s. Buffer Overflow and Transmission Loss.

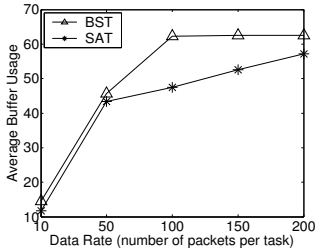

Fig. 13. Data Rate v.s. Average Buffer Usage ($B = 100$).

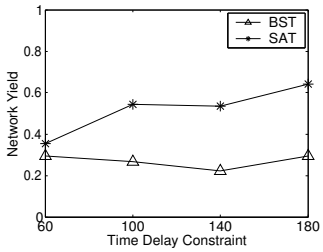

Fig. 14. Time Delay Constraint v.s. Network Yield ($N = 30$).

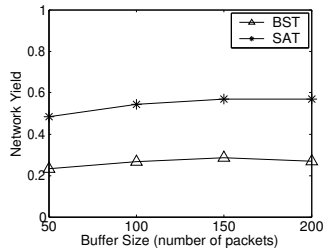

Fig. 15. Buffer Size v.s. Network Yield ($N = 30$).

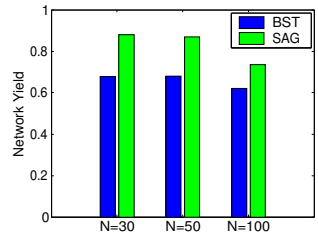

Fig. 16. Network Scale v.s. Network Yield in General Case.

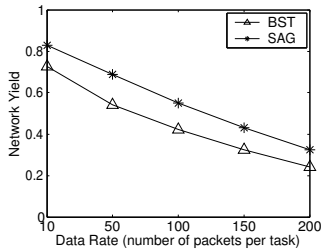

Fig. 17. Data Rate v.s. Network Yield in General Case.

The simulations mainly focus on the case in which the tasks induce a tree, as discussed in Section IV. The paths of the tasks derive from a routing tree constructed by routing protocols such as CTP [5]. For the tasks in general case, the paths are constructed by a random walk of a probe message on the graph with a given length. The considered parameters include (1) the network scale N, (2) the time constraint D^* of the tasks, (3) the data rate R, measured by the number of packets per task, and (4) the buffer size B on a node.

B. Impact of Network Scale

In this simulation, the number of the sensor nodes is set to 30, 50, and 100, and each sensor except the sink has a task of 100 packets. The time constraint of each task is set to 100, and the buffer size on each sensor is set to 100 packets. As the network size increases, the results are shown in Figure 6∼Figure 9. In Figure 6, we can see that the network yield under SAT is much higher than that under BST.

The average time delay of the tasks under SAT are larger than that under BST (Figure 7), however, both the two strategies result in low delays compared with the time constraint $D^* = 100$. It turns out that SAT improves network yield at the expense of time delay. To further investigate the cause of network yield degrading, the simulation counts the times of buffer overflows and transmission losses. As Figure 8 depicts, both the buffer overflow and transmission loss increase as the network scale increases. The white bars in Figure 8 refers to transmission losses, and we can see that the number of packet overflows under BST is less than that under SAT when $N = 30$ and $N = 100$, while the number of packet loss under BST is larger than that under SAT, resulting in a worse performance.

As the network becomes larger, the average buffer usage increases under both the two strategies (Figure 9), and there are only marginal differences between the two strategies.

C. Impact of Data Rate

The data rate can be adjusted by altering the number of packets per task. As the data rate increases, the network yield inevitably decreases since the both the congestion and storage burden increase. As illustrated in Figure 10, SAT achieves a network yield almost as twice as BST. Figure 11 shows that the average delay of the tasks under SAT is always larger than that under BST. As the data rate increases from 10 to 200, both the packet loss and buffer overflow increase. BST

has a much higher packet loss than SAT, while the buffer overflow under SAT is slightly higher than that under BST (Figure 12). These results suggests SAT significantly alleviates transmission congestion in a time slot with a lower buffer occupancy (Figure 13) with $N = 30$.

D. Impact of User-Assigned Parameters

In real applications, user may want to vary the size of the buffer provided for the tasks, as well as the time delay constraint. To investigate the impact of the two user-assigned parameters, the simulation examines the network yield with variant setups. The result shown in Figure 14 reveals that SAT is more sensitive to the time delay constraint, while the time delay has little impact on the network yield under BST. As available buffer size increases, the network yield under SAT improves slightly, while that under BST remains stable, and even decreased when $B = 200$ (Figure 15).

E. The Performance of SAG

Finally, we test the performance of the proposed SAG algorithm. It can be seen that the network yield is improved by nearly 20% under SAG (Figure 16) when the size of the network varies. Furthermore, SAG always has a better performance than BST as the data rate varies (Figure 17). Compared with the results shown in Figure 6, the network yields decrease more slowly. The reason lies in the inherence of the tasks, i.e., the data flows of the tasks are more evenly distributed in the general case than in the tree topology, resulting in less transmission congestions and buffer overflows.

VIII. Conclusion and Future Work

In this paper, we thoroughly investigate the multiple task scheduling problem for low-duty-cycled sensor networks. We accordingly formulate the Load Balancing (LB) problem, prove its NP-Completeness, and propose corresponding algorithms in achieving maximized efficiency. Based on the proposed algorithms we design a distributed task scheduling protocol TSP for practical networks. Extensive simulations on TOSSIM simulator validate our protocol design. Compared with a best-effort strategy, the TSP protocol achieves notably improved performance in most scenarios.

The algorithms proposed in this paper mainly apply to application scenarios with static routing and foreseeable data rates of the tasks. We plan to extend our work to consider more adaptive strategies applicable to dynamic routing and data rates, as well as topological changes, such as node/link failures. Furthermore, we will seek for better analytical results for the general case of the problem.

Acknowledgments

This work is supported in part by the NSFC under Grant No. 60803152, 60933001, 61033015, the NSFC - RGC program under Grant No. 60831160525, and SUG COE_SUGRSS_20Aug2010_1314 in Nanyang Technological University of Singapore. The authors would like to thank the anonymous reviewers for their valuable comments.

References

[1] Crossbow TelosB Datasheet. http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf.

[2] O. Chipara, C. Lu, and J. Stankovic. Dynamic Conflict-free Query Scheduling for Wireless Sensor Networks. In *Proc. of ICNP*, 2006.

[3] S. Gandham, M. Dawande, and R. Prakash. Link Scheduling in Sensor Networks: Distributed Edge Coloring Revisited. In *Proc. of IEEE INFOCOM*, 2005.

[4] M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[5] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection Tree Protocol. In *Proc. of SENSYS*, 2009.

[6] Y. Gu and T. He. Data Forwarding in Extremely Low Duty-Cycle Sensor Networks with Unreliable Communication Links. In *Proc. of ACM SENSYS*, 2007.

[7] Y. Gu, T. He, M. Lin, and J. Xu. Spatiotemporal Delay Control for Low-Duty-Cycle Sensor Networks. In *Proc. of IEEE RTSS*, 2009.

[8] S. Guo, Y. Gu, B. Jiang, and T. He. Opportunistic Flooding in Low-Duty-Cycle Wireless Sensor Networks with Unreliable Links. In *Proc. of ACM MOBICOM*, 2009.

[9] G. Hackmann, F. Sun, N. Castaneday, C. Lu, and S. Dyke. A Holistic Approach to Decentralized Structural Damage Localization Using Wireless Sensor Networks. In *Proc. of IEEE RTSS*, 2008.

[10] R. Jurdak, P. Baldi, and C. V. Lopes. Adaptive Low Power Listening for Wireless Sensor Networks. *IEEE Transactions on Mobile Computing*, 6(8):988–1004, 2007.

[11] M. Khan and G. Pandurangan. A Fast Distributed Approximation Algorithm for Minimum Spanning Trees. *Distributed Computing*, 20(6):391–402, 2008.

[12] P. Levis and N. Lee. TOSSIM: A Simulator for TinyOS Networks. User's Manual in TinyOS, 2003.

[13] M. Li and Y. Liu. Underground Structure Monitoring with Wireless Sensor Networks. In *Proc. of ACM/IEEE IPSN*, 2007.

[14] G. Lu, N. Sadagopan, B. Krishnamachari, and A. Goel. Delay Efficient Sleep Scheduling in Wireless Sensor Networks. In *Proc. of IEEE INFOCOM*, 2005.

[15] A. Mainwarin, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *ACM WSNA*, 2002.

[16] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi. The Flooding Time Synchronization Protocol. In *Proc. of ACM SENSYS*, 2004.

[17] L. Mo, Y. He, Y. Liu, J. Zhao, S.-J. Tang, X.-Y. Li, and G. Dai. Canopy Closure Estimates with GreenOrbs: Sustainable Sensing in the Forest. In *Proc. of ACM SENSYS*, 2009.

[18] Y. Pan and X. Lu. Energy-efficient Lifetime Maximization and Sleeping Scheduling Supporting Data Fusion and QoS in Multi-SensorNet. *Signal Processing*, 87(12):2949–2964, 2007.

[19] A. Rao and I. Stoica. Adaptive Distributed Time-slot based Scheduling for Fairness in Multi-hop Wireless Networks. In *Proc. of ICDCS*, 2008.

[20] S.-S. Tan, D. Zheng, J. Zhang, and J. Zeidler. Distributed Opportunistic Scheduling for Ad-Hoc Communications Under Delay Constraints. In *Proc. of IEEE INFOCOM*, 2010.

[21] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A Macroscope in the Redwoods. In *Proc. of ACM SENSYS*, 2005.

[22] G. W.-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and Yield in a Volcano Monitoring Sensor Network. In *Proc. of OSDI*, 2006.

[23] L. Wang and Y. Xiao. A Survey of Energy-efficient Scheduling Mechanisms in Sensor Networks. *Mobile Networks and Applications*, 11(5):723–740, 2006.

[24] G. Werner-Allen, S. Dawson-Haggerty, and M. Welsh. Lance: Optimizing High-Resolution Signal Collection in Wireless Sensor Networks. In *Proc. of ACM SENSYS*, 2008.

[25] Y. Wu, S. Fahmy, and N. B. Shroff. On the Construction of a Maximum-Lifetime Data Gathering Tree in Sensor Networks: NP-Completeness and Approximation Algorithm. In *Proc. of IEEE INFOCOM*, 2008.

[26] B. Yu, J. Li, and Y. Li. Distributed Data Aggregation Scheduling in Wireless Sensor Networks. In *Proc. of IEEE INFOCOM*, 2009.

[27] Y. Zhang and Q. Huang. A Learning-based Adaptive Routing Tree for Wireless Sensor Networks. *Journal of Comm.*, 1(2):12–21, 2006.