Distributed Optimal On-line Task Allocation Algorithm for Wireless Sensor Networks

Wanli Yu, *Student Member, IEEE,* Yanqiu Huang, *Member, IEEE* and Alberto Garcia-Ortiz, *Member, IEEE*

***Abstract*—Complex wireless sensor network (WSN) applications as those in Internet-of-Things or in-network processing are pushing the requirements for energy efficiency and data processing drastically. Energy-aware task allocation becomes crucial to efficiently distribute the tasks of the applications for the nodes to extend the network lifetime. In this paper, we propose a distributed optimal on-line task allocation (DOOTA) algorithm, by considering the energy cost of communicating, computing, sensing and sleeping activities, to optimally balance the workload distribution among the sensor nodes. Through an in-depth analysis, this work proves that the optimal partition solution for each node consists of at most 2 partition cuts with the corresponding weights. This observation enables the proposed on-line algorithm to maximize the network lifetime with low execution complexity. The simulation results show that our proposed algorithm extends the network lifetime by 12.26 times compared with the strategy of no scheduling, which is 2.22 times more than previous off-line task allocation methods. Moreover, the energy spent on executing the on-line algorithm is so small that it can be neglected.**

***Index Terms*—Wireless sensor networks (WSNs), energy efficiency, workload scheduling, task allocation, network lifetime, on-line optimization.**

I. Introduction

WIRELESS sensor networks (WSNs) have been applied to a wide variety of applications with vastly varying requirements and characteristics. Energy efficiency is a primary concern in almost any WSN application: the nodes in WSNs are usually supplied by the limited battery power, and it is hard to recharge or replace the dying nodes due to large quantities or the harsh physical environments, which would lead to fragmentations of the network and loss of potentially critical information. Therefore, many research and industrial communities are devoted to studying how to achieve the energy efficiency and extend the WSN lifetimes.

In traditional WSN applications, the workloads are simple and wireless communication is usually the most energy intensive process. Specifically, a single bit transmission requires 1000 times the energy cost of a 32-bit computation in a classical architecture [1]. Thus, most of the previous researches mainly focus on reducing the communication cost. For example, energy efficient clustering and routing approaches have been proposed to conserve communication energy by reducing the transmission distance and balancing the transmission loads within the clusters [2]–[4]; alternatively, there are numerous compression-based techniques that either focus on reducing the volumes of the transmitting packets [5]–[8] or aim to decrease the transmission rate to achieve the energy efficiency [9]–[11]; also a large number of sleep/wakeup schemes have been studied to reduce the energy spent on idle states of the radio component [12], [13].

However, the energy consumption of sensing and computation are not insignificant any more, as more complex WSN applications, e.g., in Internet-of-Things or in-network processing scenarios, appear in the past decade. The sensing cost of several off-the-shelf sensors, like the TDA0161 of STM or the CP18, VL18 produced by VISOLUX, are significantly larger than the communication cost [14]. Processing a vast amount of data or executing complex algorithms makes the computation cost comparable with or even larger than the communication cost. For instance, the computation of ECDH-ECDSA requires 5 times more energy than the communication on MICAz node [15]. Thus, to efficiently maximize the network lifetime, it is necessary to balance the energy cost of nodes by simultaneously taking the workload of sensing, computation and communication tasks of the given applications into account.

Energy-aware task allocation can efficiently solve the workload balance problem. Although numerous task allocation approaches have been studied in previous wired networks [16], [17], they cannot be directly applied to WSNs due to the limited battery energy and the special wireless communication mechanism. The design of task allocation approaches for WSNs is an ongoing research. It is attracting significant attentions [18]–[26]. One the one hand, studies aim to select the appropriate groups of nodes to cooperatively complete a global application, typically using bio-inspired meta-heuristic algorithms, e.g., particle swarm optimization (PSO) [18], [19], genetic algorithm (GA) [20], [21], etc. In [18], a modified version of binary PSO (MBPSO) is designed for the allocation of multiple computationally intensive tasks for WSNs; [19] proposes a fault-tolerant task allocation algorithm (FTAOA) based on discrete PSO (DPSO) to support the fault tolerance of workload allocation. The common drawback of the PSO based algorithms is that they would easily get stuck in local optimum. The authors in [20], [21] focus on the research of GA based approaches to address the workload distribution problems. Although the modified versions of GA have high probability to obtain the optimal solution, numerous generations are needed which takes long time sometimes even several days. An additional issue is the lack of real applications in the evaluation of the above mentioned works. On the other hand, task allocation approaches for the scenarios that each node has to execute an individual application are studied, which are designed targeting at real applications [22]–[26]. The authors

W. Yu, Y. Huang and A. Garcia-Ortiz are with the Institute of Electrodynamics and Microelectronics, University of Bremen, 28359, Bremen, Germany (e-mail: {wyu, huang, agarcia}@item.uni-bremen.de).

in [22], use an exhaustive search method to optimally distribute the workload of the tasks for real applications with a static partition cut for the nodes. Due to the complexity of the exhaustive algorithms, heuristic algorithms are designed to reduce the complexity [23], [24], but they cannot reach the optimal solutions and are restricted to symmetrical networks. An optimal task allocation scheme based on integer linear programming algorithm is proposed by [25] to maximize the network lifetime. This approach is later on improved by [26], which provides the dynamic partition scheduling to further prolong the lifetime. Nevertheless, most of the existing approaches for real applications are centralized algorithms and are preferred to be executed off-line due to the computational complexity. They require to know all the network parameters in advance. It is very hard or even impossible to satisfy this requirement in realistic WSN scenarios. Even if the parameters can be obtained beforehand, the changes of the networks, like network sizes or positions of the nodes, make the off-line schemes inefficient. Thus, efficient on-line task allocation algorithms are strongly required.

This work proposes a distributed optimal on-line task allocation (DOOTA) algorithm to maximize the network lifetime, targeting at the real applications. It enables each slave to parallelly calculate its own workload distribution solution with very lightweight complexity. The main contributions of this work are:

- It models the workload distribution problem of a given WSN application as a SDF graph partition problem, and systematically formulates the energy consumption and the time constraints of the nodes in WSNs.
- It proves that using at most 2 partition cuts with the corresponding weights for each node is enough to achieve the maximum network lifetime, and finds the possible partition cuts based on the binary decision diagram (BDD) theory.
- To the best of our knowledge, it is the first work that proposes an optimal on-line task allocation algorithm for multi-partition schedules to flexibly respond the dynamic changes of WSNs, i.e., network size or positions of the nodes.

The rest of this paper is organized as follows. Sec. II models the workload distribution problem, the energy consumption and the time latencies of the nodes in WSNs. In Sec. III, we firstly introduce the problem of the network lifetime maximization and prove that the optimal solution is always made up of at most 2 partition cuts; then the DOOTA algorithm is proposed. The simulation results are reported in Sec. IV. We summarize our work in Sec. V.

II. Modeling The Workload Distribution and Energy Consumption for Cluster-based WSNs

This work focuses on cluster-based WSNs as shown in Fig. 1, since current studies have already proved that such hierarchical architectures increase the network scalability and lifetime efficiently [2], [27]. For a given WSN, it can be firstly grouped into numbers of clusters according to clustering approaches as in [2], [27]. We consider the leaf nodes (slaves) transmit to the cluster head (master) by one hop, due to the fact that direct transmission is more energy efficient than multi-hop routing in small-to-medium size WSNs [8], [27]. All the slaves are sources and complete the applications with the cooperation of the master node, respectively. Each slave either transmits the raw data to the master or pre-processes it before the transmission. The master node is in charge of receiving data and executing further processing; after that, it forwards the processed data to the sink node.

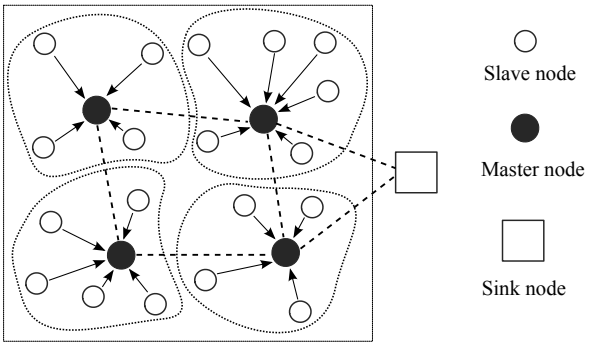


Fig. 1. A cluster-based wireless sensor network.

Our goal is to optimize the network lifetime which has a diversity of definitions by considering the number of alive nodes, sensing coverage, connectivity, etc. [28]. In one hop networks, those definitions, in a certain extent, are equivalent to the time until κ nodes die. The time when the first node dies ($\kappa = 1$) is the most frequently used definition in literatures such as [7], [8], [23]–[26], [28], [29]. Here, we use this definition for the comparison with previous approaches. Note that our algorithm also can be extended to satisfy the lifetime definition of κ nodes die. Since each cluster can work independently [8], [23], the minimum lifetime among the clusters is actually the network lifetime. Thus, we only need to focus on the optimization within each cluster.

This section firstly models the workload distribution problem, then formulates the energy consumption and time constraints of the slave and master nodes in the cluster.

A. Modeling Workload Distribution

Synchronous data-flow (SDF) graphs are widely used for modeling WSN workloads [23], [25], [30]. The structure of the data-flow is not limited to a static SDF graph, it can also have conditional branches. This paper only presents the static ones due to the space constraint. Fig. 2 shows one example of a static SDF graph, $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$. Each actor $v \in \boldsymbol{V}$ is a computational module for executing related task; each edge $e \in \boldsymbol{E}$ represents a buffer for actors to store and fetch data (tokens). When an actor fetches the fixed amount of tokens from its input edges, $in(v)$, it is fired to execute the task and generate tokens on its output edges, $out(v)$. The tokens consumed and generated by an actor, $k_c(v)$ and $k_g(v)$, are constant and can be formulated as:

$$k_c(v) = \sum_{e \in in(v)} k_o(e) \quad \text{and} \quad k_g(v) = \sum_{e \in out(v)} k_i(e)$$

where $k_o(e)$ and $k_i(e)$ are the output and input tokens on edge e, respectively. For each edge, the consistency property during one scheduling period has to be satisfied, i.e., $q\big(src(e)\big)k_i(e) = q\big(snk(e)\big)k_o(e)$, where $q(\cdot)$ is a unique minimal positive integer which represents the execution times of each actor; $src(e)$ and $snk(e)$ are the source and sink actors of edge e.

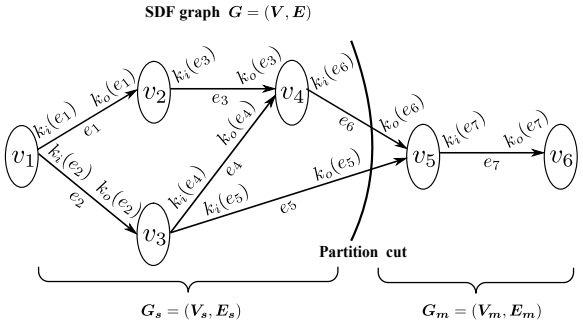


Fig. 2. An example of the synchronous dataflow (SDF) graph.

Distributing the workload for the slave and master nodes is equivalent to divide the modeled SDF graph, $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, into two subgraphs, $\boldsymbol{G_s} = (\boldsymbol{V_s}, \boldsymbol{E_s})$ and $\boldsymbol{G_m} = (\boldsymbol{V_m}, \boldsymbol{E_m})$, with a partition cut. As the workload is completed by the cooperation of the slave and master nodes, $\boldsymbol{G_s}$ and $\boldsymbol{G_m}$ has to satisfy *constraint* (1):

$$\begin{aligned} \boldsymbol{G_s} \cup \boldsymbol{G_m} = \boldsymbol{G} \quad \text{and} \quad \boldsymbol{G_s} \cap \boldsymbol{G_m} = \varnothing; \\ \boldsymbol{G_s} \neq \varnothing \quad \text{and} \quad \boldsymbol{G_m} \neq \varnothing \end{aligned} \tag{1}$$

To guarantee that the data is transmitted from the slave to the master, each edge that across the partition cut has to meet the *constraint* (2):

$$src(e) \in \boldsymbol{G_s} \quad \text{and} \quad snk(e) \in \boldsymbol{G_m} \tag{2}$$

The amount of the transmitted data, k_d, in one scheduling period can be expressed as:

$$\begin{aligned} k_d &= \sum_{v \in \boldsymbol{G_m}} q(v)\big(k_c(v) - k_g(v)\big) \\ &= \sum_{v \in \boldsymbol{G_s}} q(v)\big(k_g(v) - k_c(v)\big) \end{aligned} \tag{3}$$

To formulate the workload distribution as a linear programming problem, we introduce an $L \times 1$ binary vector, $\boldsymbol{X} = [x(v_1), \cdots, x(v_l), \cdots, x(v_L)]^T$, to represent the partition cut. L is the number of actors in the SDF graph, and $x(v_l)$ is a boolean parameter, it indicates that actor v_l belongs to the slave node or the master node as defined by:

$$x(v_l) = \begin{cases} 0, & \text{if } v_l \in \boldsymbol{G_s} \\ 1, & \text{if } v_l \in \boldsymbol{G_m} \end{cases} \tag{4}$$

Then, the *constraints*, (1) and (2), can be rewritten as the following matrix expressions:

$$\begin{aligned} 1 \leq \mathbb{1}_{1\times L}\boldsymbol{X} \leq L - 1 \\ \mathbb{B}^T\boldsymbol{X} \leqq \mathbb{0} \end{aligned} \tag{5}$$

where $\mathbb{1}_{1\times L}$ is an $1 \times L$ all one vector; $\mathbb{0}$ is an all zero vector; $\mathbb{B}$ is the incidence matrix of the SDF graph, it has the row for each actor and column for each edge: $\mathbb{B}(v, e)$ equals 1 if edge e leaves actor v, -1 if edge e enters v and 0 otherwise.

Let $\boldsymbol{K_v} = [k_n(v_1), \cdots, k_n(v_l), \cdots k_n(v_L)]$ denote the vector of the net consumed tokens of each actor, where $k_n(v_l) = q(v_l)\big(k_c(v_l) - k_g(v_l)\big)$. Then, (3) can be rewritten as:

$$k_d = \boldsymbol{K_v}\boldsymbol{X} = \boldsymbol{K_v}(\boldsymbol{X} - \mathbb{1}^T_{1\times L}) \tag{6}$$

$\boldsymbol{K_v}\mathbb{1}^T_{1\times L} = 0$, because the net consumed tokens of the complete SDF graph is 0.

B. Modeling the Energy Consumption

In cluster-based WSNs, the slave node mainly spends energy on sensing, computing, transmitting and sleeping processes; the energy consumption of the master node includes receiving, computing and sleeping cost.

The sensing energy consumption, E_{sen}, depends on the types of sensors. Most WSN applications rely on a synchronous philosophy where the sensors are always working with a given sampling ratio [31]. Thus, E_{sen} is a constant value when a specific WSN application is executed by the slave and master nodes.

The computational activities of executing one actor involves three aspects: fetching the tokens from the input edges of the actor, processing them, and storing processed tokens onto its output edges [23], [25]. The energy cost of executing actor v of slave node i, $e_{cp\_si}(v)$, and the master node, $e_{cp\_m}(v)$, in one scheduling period can be formulated as:

$$\begin{aligned} e_{cp\_si}(v) =& q(v)\Big(k_c(v)P_{e\_si}t_{e\_si} + P_{si}(v)t_{si}(v) \\ &+ k_g(v)P_{e\_si}t_{e\_si}\Big) \\ e_{cp\_m}(v) =& q(v)\Big(k_c(v)P_{e\_m}t_{e\_m} + P_m(v)t_m(v) \\ &+ k_g(v)P_{e\_m}t_{e\_m}\Big) \end{aligned} \tag{7}$$

where P_{e_si}, P_{e_m} and t_{e_si}, t_{e_m} represent the mean power and time dissipation of fetching (or storing) one token from (or onto) the edges in slave node i and the master node, respectively; $P_{si}(v)$, $P_m(v)$ and $t_{si}(v)$, $t_m(v)$ are the mean processing power and time consumption of the actor when it is executed in slave node i and the master node, respectively.

Let $\boldsymbol{E_{cp\_si}} = \big[e_{cp\_si}(v_1), \cdots, e_{cp\_si}(v_L)\big]$ and $\boldsymbol{E_{cp\_m}} = \big[e_{cp\_m}(v_1), \cdots, e_{cp\_m}(v_L)\big]$ denote the computation cost of each actor when they are executed by slave node i and the master node, respectively. The computation energy consumption of slave node i and the master node for a given partition cut $\boldsymbol{X_i}$ in one scheduling period are:

$$\begin{aligned} E_{cp\_si}(\boldsymbol{X_i}) =& \boldsymbol{E_{cp\_si}}\Big(\mathbb{1}^T_{1\times L} - \boldsymbol{X_i}\Big) \\ E_{cp\_m}(\boldsymbol{X_i}) =& \boldsymbol{E_{cp\_m}}\boldsymbol{X_i} \end{aligned} \tag{8}$$

As reported in [25], the communication cost of a WSN node includes data packets communication and overhead activities. Thus, the communication energy cost of slave node i and the master node for transmitting and receiving k_d bits data are:

$$
\begin{aligned}
E_{cm\_si}(\boldsymbol{X_i}) &= e_{o\_si} + e_{tx\_si}(d_i)k_d \\
&= e_{o\_si} + e_{tx\_si}(d_i)\boldsymbol{K_v}(\boldsymbol{X_i} - \mathbb{1}_{1\times L}^T) \\
E_{cm\_m}(\boldsymbol{X_i}) &= e_{o\_m} + e_{rx}k_d \\
&= e_{o\_m} + e_{rx}\boldsymbol{K_v}\boldsymbol{X_i}
\end{aligned} \tag{9}
$$

where e_{o_si} and e_{o_m} are the energy that the slave node i and the master node spend on the overhead activities; $e_{tx\_si}(d_i)$ and e_{rx} are the corresponding energy cost for transmitting and receiving one bit data, respectively. According to [25], [36], $e_{rx} = P_{T0}t_{rx}$, where P_{T0} is the energy consumed in the electronic circuits of the transceiver for receiving or transmitting 1 bit of data and t_{rx} is the time cost of receiving 1 bit of data; $e_{tx\_si}(d_i)$ is a function of distance: $e_{tx\_si}(d_i) = t_{tx\_si}P_{tx}(d_i) = t_{tx\_si}(P_{T0} + \frac{10^{(L(d_i)+P_{rin})/10}}{\eta})$, where t_{tx_si} is the time cost of transmitting one bit data, P_{rin} is the receiver sensitivity, η is the drain efficiency, and $L(d_i) = 20\log_{10}(f) + 10\alpha\log_{10}(d_i) - 27.55$ in which f is the RF frequency and α is the path loss exponent.

The sleeping energy cost of slave node i and the master node can be formulated by:

$$
\begin{aligned}
E_{slp\_si} &= P_{slp\_si}T_{slp\_si} \\
E_{slp\_m} &= P_{slp\_m}T_{slp\_m}
\end{aligned} \tag{10}
$$

where P_{slp_si} and P_{slp_m} are the power consumption when slave node i and the master node are in sleep mode; T_{slp_si} and T_{slp_m} are their sleeping time which can be easily calculated as shown in the following Sec. II-C.

As the master node is in charge of its n slave nodes, it has to iterate n times to complete the workloads for the slave nodes in one scheduling period. Combining (8), (9) and (10), the energy consumption of slave node i and the master node are:

$$
\begin{aligned}
E_{si}(\boldsymbol{X_i}) &= P_{slp\_si}T_{slp\_si} + E_{sen} + e_{o\_si} + \\
&\quad \big(\boldsymbol{E_{cp\_si}} - e_{tx\_si}(d_i)\boldsymbol{K_v}\big)\big(\mathbb{1}_{1\times L}^T - \boldsymbol{X_i}\big) \\
E_m(\boldsymbol{X_1},\cdots,\boldsymbol{X_n}) &= P_{slp\_m}T_{slp\_m} + \sum_{i=1}^{n} e_{o\_m} + \\
&\quad \big(\boldsymbol{E_{cp\_m}} + e_{rx}\boldsymbol{K_v}\big)\boldsymbol{X_i}
\end{aligned} \tag{11}
$$

C. Modeling the Time Constraints

The sensing time, T_{sen}, is considered to be a constant value. According to (8), the execution time of running one actor in slave node i, $t_{cp\_si}(v)$, and the master node, $t_{cp\_m}(v)$, in one scheduling period can be expressed as:

$$
\begin{aligned}
t_{cp\_si}(v) &= q(v)\Big(k_c(v)t_{e\_si} + t_{si}(v) + k_g(v)t_{e\_si}\Big) \\
t_{cp\_m}(v) &= q(v)\Big(k_c(v)t_{e\_m} + t_m(v) + k_g(v)t_{e\_m}\Big)
\end{aligned} \tag{12}
$$

The time of executing the schedule defined by a given partition $\boldsymbol{X_i}$ of the slave i, $T_{cp\_si}(\boldsymbol{X_i})$, and the master node, $T_{cp\_m}(\boldsymbol{X_i})$, are:

$$
\begin{aligned}
T_{cp\_si}(\boldsymbol{X_i}) &= \boldsymbol{T_{cp\_si}}\big(\mathbb{1}_{1\times L}^T - \boldsymbol{X_i}\big) \\
T_{cp\_m}(\boldsymbol{X_i}) &= \boldsymbol{T_{cp\_m}}\boldsymbol{X_i}
\end{aligned} \tag{13}
$$

where $\boldsymbol{T_{cp\_si}} = \big[t_{cp\_si}(v_1),\cdots,t_{cp\_si}(v_L)\big]$ and $\boldsymbol{T_{cp\_m}} = \big[t_{cp\_m}(v_1),\cdots,t_{cp\_m}(v_L)\big]$ represent the time cost of the actors executed by slave node i and the master node, respectively.

Correspondingly, according to (9), the communication time cost of slave node i, and the master node, when exploiting partition $\boldsymbol{X_i}$ are:

$$
\begin{aligned}
T_{cm\_si}(\boldsymbol{X_i}) &= t_{o\_si} + t_{tx\_si}\boldsymbol{K_v}(\boldsymbol{X_i} - \mathbb{1}_{1\times L}^T) \\
T_{cm\_m}(\boldsymbol{X_i}) &= t_{o\_m} + t_{rx}\boldsymbol{K_v}\boldsymbol{X_i}
\end{aligned} \tag{14}
$$

where t_{o_si} and t_{o_m} are the time that slave node i and the master node spend on the overhead activities; t_{rx} is the time cost of receiving one bit data.

Let T_{sch} denote the duration of one scheduling period, which is set up by the users according to different WSN applications. The sleeping time of slave node i, T_{slp_si}, and the master node, T_{slp_m}, can be calculated by:

$$
\begin{aligned}
T_{slp\_si}(\boldsymbol{X_i}) &= T_{sch} - T_{sen} - T_{cp\_si}(\boldsymbol{X_i}) - T_{cm\_si}(\boldsymbol{X_i}) \\
T_{slp\_m}(\boldsymbol{X_i}) &= T_{sch} - \sum_{i=1}^{n} T_{cp\_m}(\boldsymbol{X_i}) + T_{cm\_m}(\boldsymbol{X_i})
\end{aligned}
$$

Since time constraint is also very important and needs to be considered in many WSN applications, we formulate the timing constraint as follows. For concreteness, we assume that time division multiple access (TDMA) protocol is used in the network as for example in [8], [23]. The fixed slot is assigned to each slave node for the transmission; the duration of each slot, T_{slot}, is a uniform time period. Considering that the Micro-processor and the RF of the WSN node work sequentially, the master node has to complete the current computation before it receives the data from the next slave node. The diagrams of the activities of the slave and master nodes are shown in Fig. 3.

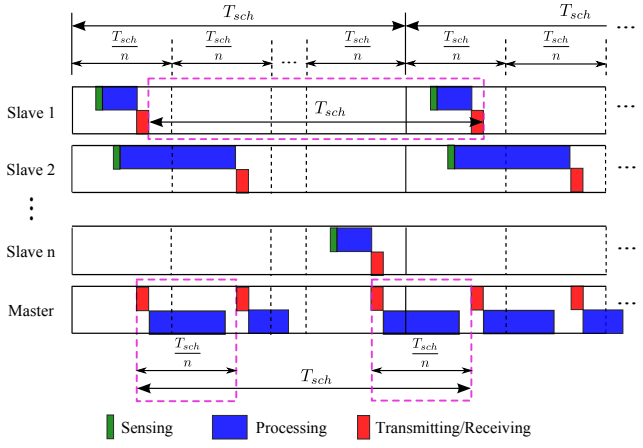


Fig. 3. The diagrams of the activities of the slave and master nodes.

Thus, the time constraints of slave node i and the master node in one scheduling period should subject to:

$$
\begin{aligned}
&T_{sen} + T_{cp\_si}(\boldsymbol{X_i}) + T_{cm\_si}(\boldsymbol{X_i}) \le T_{sch}, \\
&T_{cm\_si}(\boldsymbol{X_i}) \le T_{slot} \\
&T_{cp\_m}(\boldsymbol{X_i}) + T_{cm\_m}(\boldsymbol{X_i}) \le \frac{T_{sch}}{n}, \\
&T_{cm\_m}(\boldsymbol{X_i}) \le T_{slot}
\end{aligned} \tag{15}
$$

III. Distributed Optimal On-line Task Allocation Algorithm

In this section, we firstly present the network lifetime maximization problem that the task allocation approach aims to solve. Then, the composition of the optimal solution is analyzed. Based on the analysis, we present the distributed optimal on-line task allocation (DOOTA) algorithm including the off-line preparation and the on-line algorithm. Note that our algorithm is carried out within each cluster independently.

A. Problem Statement

To maximize the network lifetime, it is necessary to balance the energy cost of each node. This can be achieved by appropriately distributing the workload by using the optimal partition solutions. For generality, asymmetrical networks[1] are considered, where each slave node may have different energy parameters and distances to the master. The problem is to find the best partition solutions for each individual node to achieve the maximum network lifetime. We formulate the problem using the models created in Sec. II in the following.

Let $\boldsymbol{X_i^j}$ denote the partition cut that slave node i exploits in the jth scheduling period. Assuming the network dies after T periods, the problem is then to find $\boldsymbol{X_i^j}(i = 1, \cdots, n; j = 1, \cdots, T)$ to maximize T, which can be formulated as:

$$\arg\max_{\boldsymbol{X_i^j}} \ T$$

$$\text{subject to :}$$

$$\sum_{j=1}^{T} E_{si}(\boldsymbol{X_i^j}) \leq B_{si}, \ i = 1, \cdots, n$$

$$\sum_{j=1}^{T} E_m(\boldsymbol{X_1^j}, \cdots, \boldsymbol{X_n^j}) \leq B_m$$

$$(5) \text{ and } (15)$$

where B_{si} and B_m are the battery energy of the slave node i and the master node, respectively.

Although [23], [25] and [26] has solved this problem, even [25] and [26] present the off-line optimal solutions, they are centralized algorithms and require to obtain the detailed network parameters in advance, which is hard or even impossible in realistic scenarios. Moreover, the networks are not always static: the number of the nodes and their positions may change over time. Off-line methods are not efficient enough to maximize the network lifetime; even worse, they may decrease the network lifetime. Nevertheless, the results obtained by these methods can be treated as the metrics to quantify the quality of our proposed DOOTA algorithm.

B. Analysis of the Optimal Partition Solution

We analyze the composition of the optimal partition solution in this section. This lays the foundation for us to propose the DOOTA algorithm. It is inspired by [26], in which the authors present a surprising report that the optimal partition solution always consists of two partition cuts. However, they conclude this result only from the simulation observations without any theoretical support. We provide an in-depth analysis to fill this gap.

When considering a specific SDF graph in the cluster, the number of the valid partition cuts is limited. Each partition cut can be associated with a point, (E_{si}, E_{mi}), on the energy plane, which represents the energy cost of slave node i, E_{si}, and the master node, E_{mi}. Using (13) and (14), the time latencies of slave node i and the master node when exploiting each partition cut can be calculated. According to the time requirement, we remove those points which cannot satisfy the time constraint, (15). The corresponding energy points of the left partition cuts and their combinations construct a convex set as shown in Fig. 4.

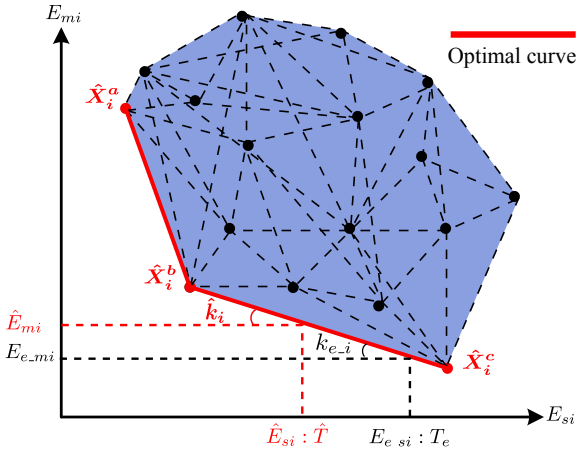


Fig. 4. All combinations of the valid partition cuts and the corresponding energy consumption of slave node i and the master node.

The smaller the energy consumption of the node, the longer it can survive. Thus, we target at the points that minimize the energy consumption of slave node i under the same energy consumption of the master node and vice versa. These points correspond to a subset of the boundary of the convex set, which is called *optimal curve* (see Fig. 4). It is actually a convex curve. Its start and end points, $\hat{X}_i^a$ and $\hat{X}_i^c$, stand for the minimum energy cost of the slave node i and the master, respectively. Given an arbitrary point in the convex set, i.e., one arbitrary possible partition solution, there exists at least one point on the *optimal curve* where both the energy cost of slave i and the master are smaller than or equal to the given point. In other words, there is always at least one partition solution on this *optimal curve* that makes the lifetimes of both the slave and master nodes longer or equal to other solutions in the convex set. Therefore, the optimal solution is definitely on this *optimal curve*, which is formed by the *important partition cuts* and their linear combinations. As shown in Fig. 4, for example, the optimal solution is either one of the *important partition cuts*, $\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{a}}$, $\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{b}}$ and $\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{c}}$, or the linear combinations of $\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{a}}$ and $\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{b}}$ or $\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{b}}$ and $\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{c}}$.

Therefore, we only need to keep the *important partition cuts* and construct the *optimal curve* to obtain the optimal solution.

[1]Note that, the workload distribution schedules can be easily applied in the symmetrical networks, where all slave nodes just need to have one same partition solution.

C. DOOTA Algorithm

The DOOTA algorithm consists of two phases: an off-line preparation and an on-line algorithm. Each slave node firstly stores the *optimal curve* off-line, and then runs the on-line algorithm to calculate its own partition solution.

1) Off-line Preparation: Based on the analysis in Sec. III-B, this section aims to obtain the *important partition cuts* and formulate the *optimal curve*. The cuts can be obtained off-line because the SDF graphs of the applications can be easily modeled before deploying the networks in realistic scenarios.

Since the *important partition cuts* are among the valid partition cuts, we firstly find the valid ones using a binary decision diagram (BDD). For a SDF graph with L actors, the complexity of finding the valid partition cuts which satisfy the *constraints* (1) and (2) by enumeration is $\mathcal{O}(2^L)$: it exponentially increases with L. Using a recursive boolean description of the problem and implementing the computations with BDDs, we can improve notably the performance of the algorithm. A BDD is a data structure used to represent a boolean function [32]; it can be also considered as a compressed representation of sets or relations. The partition cuts are made up of L boolean variables, since each actor of the SDF graph belongs to either the slave or master. Thus, we can compress a set of valid partitions using a BDD with L variables.

The key idea is to use the implicit partial order defined by the SDF graph. There may exist different valid paths from actor v_l to the end actor v_L in the SDF graph. According to *constraint* (2), actor v_l cannot be assigned to the master if its destination actors belong to the slave node. Taking Fig. 2 as an example, there are two paths from v_3 to v_6: $v_3 \rightarrow v_4 \rightarrow v_5 \rightarrow v_6$ and $v_3 \rightarrow v_5 \rightarrow v_6$. When v_3 is assigned to the master node, its destinations v_4 and v_5 must belong to the master. We use v_l and $\bar{v_l}$ to represent that actor v_l is distributed to the master and slave nodes, respectively. Let Ψ_l denote a boolean function of actors on the paths from v_l to v_L which returns true when v_l belongs to the master and *constraint* (2) is satisfied. It can be expressed recursively as:

$$\Psi_l = v_l \bigwedge_{v_j \in dest(v_l)} \Psi_j$$

where $dest(v_l)$ is the set of the destination actors of v_l. Further on, when actor v_l is assigned to the slave node, its destination actors do not affect its assignment. Thus, boolean function, ψ_l, describing the paths satisfying *constraint* (2) and starting at point l is:

$$\psi_l = \Psi_l \vee \left(\bar{v_l} \bigwedge_{v_j \in dest(v_l)} \psi_j \right)$$

According to *constraint* (1) that the slave and master nodes have to cooperate to complete the whole workload, it satisfies $\psi_L = \Psi_L = v_L$ and $\Psi_1 = \emptyset$. Through the reversed iterative computations from ψ_L to ψ_1, the representation of the valid partition cuts ψ_1 can be obtained. However, the complexity may exponentially increase with L. Therefore, efficient ways to obtain the valid partition cuts are needed. This work uses the CUDD package [33], which can efficiently manipulate BDDs of boolean functions, to create the final BDD of ψ_1. When obtaining the BDD, the valid partition cuts are actually the paths from the top actor to terminal 1. For instance, the BDD with the reversed order of the SDF graph in Fig. 2 is created as shown in Fig. 5. The dotted and solid edges represent that their source actors belong to the slave and master nodes, respectively. We can easily obtain that there are 6 paths from the actor v_6 to terminal 1. Each path corresponds one valid partition cut that represents the distribution result of each actor.

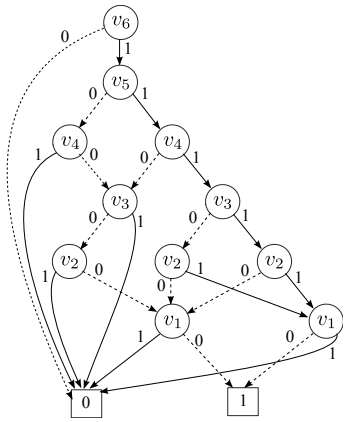


Fig. 5. The BDD graph of the SDF graph in Sec. II-A.

Next, we calculate the *important partition cuts*. Once the valid partition cuts are obtained, we compute their energy consumption using (11) and obtain the energy plane as Fig. 4 shows. The point that makes the slave node consume the least energy is selected as the first important point. We calculate the slopes of the segments which start from the first important point to the others. The point which corresponds to the minimum slope is the second important point. Then, we start from the current important point to find the next one until the last one is found. The computation complexity depends on the specific energy points. In the worst case, it is $\mathcal{O}(\frac{N_{vp}(N_{vp}-1)}{2})$, where N_{vp} is the number of the valid partition cuts.

We further formulate the *optimal curve* using the obtained *important partition cuts*. Taking Fig. 4 for example, the obtained *important partition cuts* are $\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{a}}$, $\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{b}}$ and $\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{c}}$. The relation between E_{mi} and E_{si} on this *optimal curve* can be formulated as (16), which will be stored in slave node i before deploying it into the network.

$$E_{mi} = \begin{cases} A_0 E_{si} + B_0, & \text{if} E_{si} \in \left[E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{a}}), E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{b}})\right] \\ A_1 E_{si} + B_1, & \text{if} E_{si} \in \left[E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{b}}), E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{c}})\right] \end{cases} \quad (16)$$

where

$$A_0 = \frac{E_m(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{a}}) - E_m(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{b}})}{E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{a}}) - E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{b}})}, A_1 = \frac{E_m(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{b}}) - E_m(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{c}})}{E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{b}}) - E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{c}})},$$

$$B_0 = \frac{E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{a}}) E_m(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{b}}) - E_m(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{a}}) E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{b}})}{E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{a}}) - E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{b}})} \quad \text{and}$$

$$B_1 = \frac{E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{b}}) E_m(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{c}}) - E_m(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{b}}) E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{c}})}{E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{b}}) - E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{c}})}.$$

Note that $E_{si} = E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{a}})$ when $E_{si} < E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{a}})$ and $E_{si} = E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{c}})$ when $E_{si} > E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{c}})$, due to the energy limitations of the slave and master nodes.

2) DOOTA: Based on the analysis and the results in Secs. III-B and III-C1, we propose the DOOTA algorithm in this section. It is actually an on-line negotiation among the slave and master nodes considering the parameters of the network, after each slave node stores the function of the *optimal curve* off-line. We firstly present a naive method, which may require a large quantity of intra-cluster communications. In order to reduce the communication overhead, a lightweight method is further proposed.

Naively, the master node firstly broadcasts an expected network lifetime T_e, when the network starts to work. Each slave node computes its own expected energy consumption $E_{e\_si} = B_{si}/T_e$ and the corresponding E_{e_mi} according to (16), and then transmits them to the master node. After receiving the messages from all slave nodes, the master node examines whether its battery energy is enough to last T_e time. If it still has residual battery energy, i.e., $B_m > T_e \sum_{i=1}^{n} E_{e\_mi}$, it broadcasts a larger T_e, otherwise a smaller one. The slave nodes calculate the corresponding E_{e_si} and E_{e_mi} again until $B_m = T_e \sum_{i=1}^{n} E_{e\_mi}$. At last, the master node broadcasts one confirm message, and the slave nodes individually calculate their own partition solutions based on the final E_{e_si}. Although this naive method is simple, it may need a large quantity of message exchanges, which results in too much overhead.

To address the overhead problem, we propose a lightweight on-line algorithm. It can dramatically reduce the number of iterations based on Theorem 1.

Theorem 1. *When the lifetimes of the slave and master nodes are equivalent i.e., $T_{s1} = \cdots = T_{sn} = T_m$, the time until the first node dies is maximized.*

Proof. Assuming $T_{s1} = \cdots = T_{sn} = T_m = T^*$, the average energy consumption of slave node i, E_{si}^*, and the master node, E_m^*, over T^* periods are:

$$E_{si}^* = \frac{B_{si}}{T^*} \quad \text{and} \quad E_m^* = \sum_{i=1}^{n} E_{mi}^* = \frac{B_m}{T^*}$$

When the lifetimes of the nodes do not equal each other, e.g., $T_{si} = T^* + \varepsilon > T^*$, where ε is an arbitrary positive real number, there exists:

$$E_{si} = \frac{B_{si}}{T^* + \varepsilon} < \frac{B_{si}}{T^*} = E_{si}^*$$

According to the monotone decreasing property of (16), E_{mi} is bigger than E_{mi}^*, which causes $E_m = \sum_{i=1}^{n} E_{mi}$ is bigger than E_m^* too. Thus, we can obtain that:

$$T_m = \frac{B_m}{E_m} < \frac{B_m}{E_m^*} = T^*$$

The network lifetime, $min\{T_{s1}, \cdots, T_{sn}, T_m\} = T_m$, is smaller than T^*. The proof is similar when $T_{si} = T^* - \varepsilon < T^*$, in which the lifetime, $min\{T_{s1}, \cdots, T_{sn}, T_m\} = T_{si}$, is smaller than T^* too. □

Unlike the naive method where the master node randomly adjusts T_e according to the received messages, the improved algorithm enables the master node to calculate the temporary optimal lifetime $\hat{T}$. After each slave receives T_e, it transmits not only its expected energy cost $E_{e\_si} = B_{si}/T_e$ and the corresponding E_{e_mi} calculated by (16), but also the slope, k_{e_i}, of the segment which includes the point (E_{e_si}, E_{e_mi}) as shown in Fig. 4. Let $(\hat{E}_{si}, \hat{E}_{mi})$ represent the energy point corresponding to the optimal lifetime $\hat{T}$. Assuming it is still on the current segment, $\hat{E}_{mi}$ can be calculated by:

$$\hat{E}_{mi} = E_{e\_mi} + k_{e\_i}(\hat{E}_{si} - E_{e\_si}) \tag{17}$$

According to Theorem 1, there exists:

$$\hat{T} = \frac{B_{si}}{\hat{E}_{si}} = \frac{B_m}{\sum_{i=1}^{n} \hat{E}_{mi}} \tag{18}$$

Since $E_{e\_si} = B_{si}/T_e$, combining (17) and (18), $\hat{T}$ can be calculated in the master node by:

$$\hat{T} = \frac{B_m - T_e \sum_{i=1}^{n} k_{e\_i} E_{e\_si}}{\sum_{i=1}^{n} E_{e\_mi} - k_{e\_i} E_{e\_si}} \tag{19}$$

Then, the master node compares $\hat{T}$ with T_e. If they are different, it broadcasts the current expected lifetime $T_e = \hat{T}$. Slave node i repeats the calculation of E_{e_si}, E_{e_mi} and k_{e_i}, and sends them to the master node. Once $\hat{T}$ equals T_e, the master node broadcasts a $confirm$ message. The last received T_e is actually the final maximum lifetime; slave node i can easily calculate its own optimal partition cuts and the weights based on it. For instance, assuming $\hat{T}$ in Fig. 4 is the final lifetime, the optimal partition solution for slave node i consists of two partition cuts, $\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{b}}$ and $\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{c}}$, with the related weights w_b and w_c.

$$\begin{aligned} w_b &= \frac{E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{c}}) - \hat{E}_{si}}{E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{c}}) - E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{b}})} \\ w_c &= \frac{E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{b}}) - \hat{E}_{si}}{E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{b}}) - E_{si}(\hat{\boldsymbol{X}}_{\boldsymbol{i}}^{\boldsymbol{c}})} \end{aligned} \tag{20}$$

The pseudo codes that executed in the master node and slave node i are shown in Algorithm 1 and Algorithm 2.

Algorithm 1 Master node algorithm

```
Initialize T_e and broadcast it
for each calculation round do
    Receive E_e_mi, k_e_i and E_e_si
    Calculate T̂ using (19)
    if T̂ == T_e then
        Broadcast confirm
        Break
    else
        T_e = T̂, and broadcast T_e
    end if
end for
```

IV. Simulations and Analysis

This section evaluates the performance of the DOOTA algorithm and demonstrates its superiority by comparing it with the approaches [23], [25], [26] that use the same application

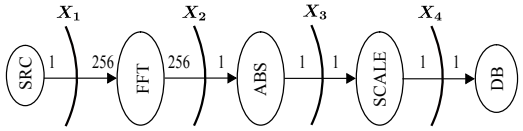


Fig. 6. The SDF graph of spectrum computation application.

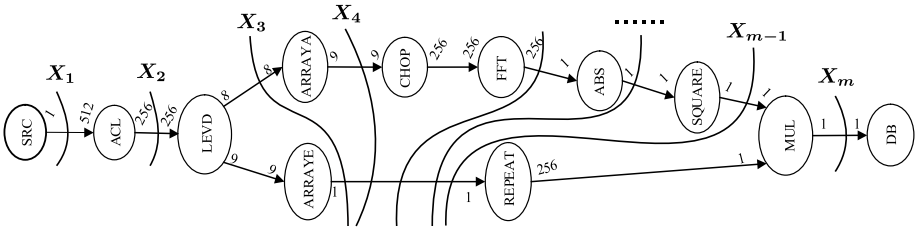


Fig. 7. The SDF graph of MEPS computation.

Algorithm 2 Slave node algorithm

```
for each received message do
    if not confirm then
        Calculate E_e_si, E_e_mi and k_e_i, using (16),
        and transmit them
    else
        Ê_si = B_si/T_e
        Calculate partition weights using (20)
        Break
    end if
end for
```

and network modeling. We firstly generate an arbitrary cluster based WSN, with which the lifetime increase of the network using DOOTA is presented and compared. Secondly, we study the performance and overhead cost of DOOTA within the cluster, since each cluster can work independently. By adjusting the number of slave nodes in the cluster, numerous simulation results regarding the lifetime increase, the running time and the number of exchange messages are presented. Afterwards, the flexibility of DOOTA is illustrated by dynamically changing the density of the cluster.

A. Parameters and Off-line Preparation for DOOTA

1) simulation parameters: The values of the energy related parameters of the slave and master nodes in the networks are obtained from [23], [34]: $P_m(v) = P_{si}(v) = P_{e\_m} = P_{e\_si} = 36.9mW$, $t_{o\_m} = t_{o\_si} = 100\mu s$, $e_{o\_m} = e_{o\_si} = 3.69\mu J$, $t_{rx} = t_{tx\_si} = t_{e\_m} = t_{e\_si} = 4\mu s$,$e_{rx} = 0.239\mu J$, the related parameter values for calculating $e_{tx\_si}(d_i)$ are $P_{T0} = 59.8mW$, $P_{rin} = -85dBm$, $\eta = 0.05$, $f = 2.4G\ Hz$ and $\alpha = 2$. The sleep energy cost is so small that is not considered in the simulations.

The battery energy of each node is randomly generated from 1 to 10 kJ. Two typical computation applications, *spectrum* computation that converts signals from time domain to frequency domain and *maximum entropy power spectrum* (MEPS) computation adapted from Ptolemy II design environment [22], [23], [35] are used. The corresponding SDF graphs of the two applications and the execution time of each actor in each SDF graph related to the CC2430 node are shown in Figs. 6 and 7, Tabs. I and II [23], [25], respectively. The energy consumption of sensing unit is included in actor SRC.

TABLE I
EXECUTION TIME OF ACTORS IN SPECTRUM APPLICATION SDF GRAPH.

Actors	Average execution cycle on CC2430	Average execution time (sec.) on CC2430 with 32MHz CLK	Numbers of iterations, $q(\cdot)$
SRC	2532.99	7.92E-5	256
FFT	8163758	0.26	1
ABS	1409	4.4E-5	256
SCALE	1606.3	5.02E-5	256
DB	707	2.21E-5	256

TABLE II
EXECUTION TIME OF ACTORS IN MEPS APPLICATION SDF GRAPH.

Actors	Average execution cycle on CC2430	Average execution time (sec.) on CC2430 with 32MHz CLK	Numbers of iterations, $q(\cdot)$
SRC	2534.21	7.92E-5	512
ACL	52830479	1.65	1
LEVD	17977524	0.56	1
ARRAYA	2186	6.83E-5	1
ARRAYE	325	1.01E-5	1
REPEAT	28093	8.77E-4	1
CHOP	39672	1.24E-3	1
FFT	7479003	0.23	1
ABS	1428.24	4.46E-5	256
SQUARE	478.85	1.5E-5	256
MUL	618.29	1.93E-5	256
DB	722.25	2.26E-5	256

2) off-line preparation for DOOTA: In our simulations, each slave node executes the same application, MEPS or spectrum. The corresponding important partition cuts of each application can be obtained using the method mentioned in Sec. III-C1. There are total $N = 2^L$ partition cuts of a given SDF graph with L actors, since each actor can belong to either the slave node or the master node. For MEPS application, $N = 2^{12}$. After filtering the partition cuts by using the binary decision diagram, 21 valid partition cuts remain. Among these cuts, there are only 3 *important partition cuts*, $\boldsymbol{X_1}$, $\boldsymbol{X_3}$ and $\boldsymbol{X_m}$ as shown in Fig. 7. Similarly, there are also 3 *important partition cuts* for the spectrum graph. Those cuts are stored in each slave node before the deployment of the network.

B. Evaluation of DOOTA: general WSNs

This section evaluates the DOOTA algorithm on a common WSN scenarios. As shown in Fig. 8, we generate a WSN with 100 nodes randomly located in a two dimensional network area of 100×100 square meters. The sink node is located at the point $(50,\ 150)$ which is not shown in the figure. Using LEACH [27] clustering protocol, the 100 nodes are grouped into 6 clusters, $C_1, \cdots, C_6$. In the cluster, each slave node connects with one master node by one wireless hop.

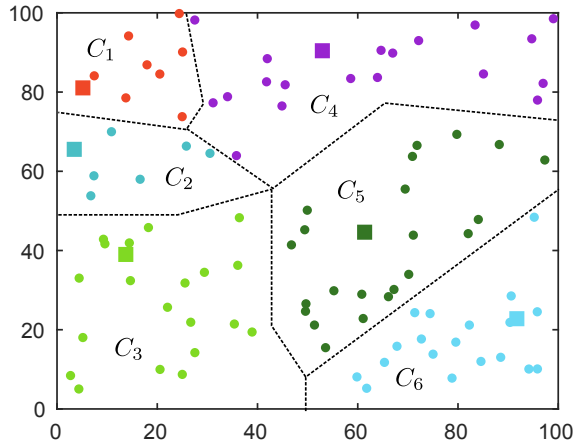
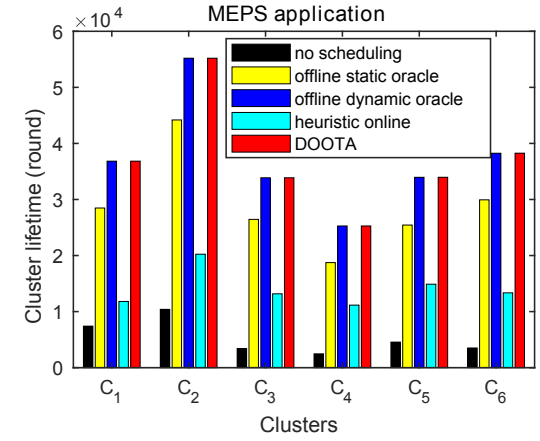


Fig. 8. The 100-node random test WSN of size $100 \times 100\ m^2$ is grouped into 6 clusters based on the idea of LEACH [27]. The master and slave nodes are marked by ● and ■ with different color in different clusters, respectively. The sink node is located at the point $(50,\ 150)$ which is not shown.

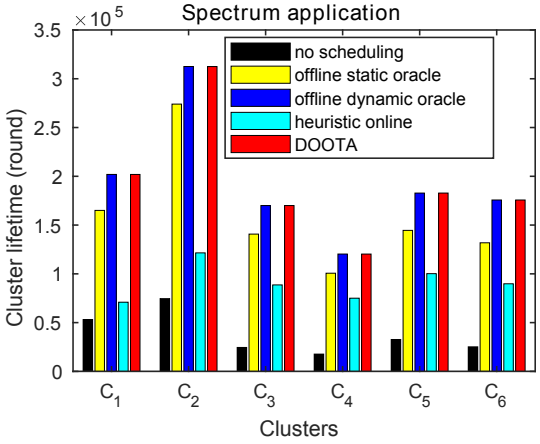
We evaluate the DOOTA algorithm on the above described scenario in terms of network lifetime increase with respect to the strategy of *no scheduling*, in which each slave node just executes the task of the first actor of the SDF graph. The performance of DOOTA is compared with three similar approaches:

- *Heuristic on-line* [23]: It calculates different partition solutions as the number of slave nodes varies. The result is stored in the nodes before deploying the network. At run time, the partition solution is selected based on the number of slave nodes.
- *Off-line static* [25]: It formulates the task allocation problem as an integer linear programming (ILP) problem and provides an optimal static partition cut for each slave and master node.
- *Off-line dynamic* [26]: It formulates the task allocation problem as a LP problem and provides multiple partition cuts with the corresponding weights. It provides optimal solutions when all the network parameters are known.

Each cluster independently executes the above algorithms. For the off-line approaches [25] and [26], they are termed as *off-line static oracle* and *off-line dynamic oracle*, respectively, if the center (oracle) knows all of the network parameters in advance. Fig. 9 depicts the corresponding lifetimes of the 6 clusters by using different schemes. It is obvious that DOOTA performs better than off-line static oracle and the heuristic on-line schemes in each cluster for both MEPS and spectrum applications. It extends the lifetime of each cluster as much as



(a)



(b)

Fig. 9. The lifetimes of the 6 clusters by applying the no scheduling strategy, off-line static and dynamic oracles, heuristic on-line and DOOTA algorithms for (a) MEPS and (b) spectrum applications.

the off-line dynamic oracle. The network lifetime is defined as the minimum lifetime among the clusters. Taking Fig. 9a, the MEPS application, for example, the lifetime of cluster C_4 is considered as the network lifetime. Among the four approaches, the heuristic on-line provides the lowest extension of the network lifetime. It considers that all slave nodes have the same battery and transmission distance to the master node. This assumption makes the heuristic on-line scheme very simple and easy to implement, while it also brings a limitation in the asymmetrical networks. Rather than using the static partition, DOOTA and the off-line dynamic oracle achieve the maximum network lifetime by providing different partition cuts with the corresponding weights. Specifically, both DOOTA and the off-line dynamic oracle increase the network lifetime by 10.24 times with respect to the *no scheduling* strategy, while the off-line static oracle and the heuristic on-line methods extend the network lifetime by 7.59 and 4.52 times, respectively. The similar phenomenon holds for spectrum application as shown in Fig. 9b.

Although the off-line dynamic oracle can maximumly extend the network lifetime, it is too complex and needs to know all the network parameters in advance. While the overhead cost of running the DOOTA algorithm is so small that can be neglected as detailedly presented in the next section.

C. Evaluation of DOOTA: independent cluster

Since each cluster executes DOOTA independently, we present the detailed estimations of DOOTA within the cluster in this section. The cluster is randomly generated in a two dimension area of 100×100 square meters with one master located at the center and n randomly distributed slave nodes. The reported results are obtained by generating 500 instances of the simulations.

Firstly, a set of simulations are conducted to estimate the effect of the number of the slave nodes, n, on the increase of network lifetime with respect to the *no scheduling* strategy. The results reported in Fig. 10 show that all of the four task allocation approaches dramatically extend the network lifetime for both MEPS and spectrum applications. As the number of slave nodes in the cluster, n, increases, their improvements become more significant. It is due to the fact that the rapid increasing workload in each cluster makes the master node overburdened and die soon. While through the efficient workload scheduling, the energy consumption of each node is well or even optimally balanced. For example, DOOTA extends the network lifetime in average from 5.90 to 20.94 times when the cluster size increases from 5 to 40 as shown in Fig 10a. It performs as well as the off-line dynamic oracle and outperforms the off-line static oracle and heuristic on-line schemes, which is consistent with the simulation results in Sec. IV-B. Moreover, DOOTA has more advantages for handling complex task allocation problems. The superiority of DOOTA for the MEPS application (see Fig. 10a) is more significant than the spectrum application (see Fig. 10b). Since there is only one master node in the cluster, executing the task of the last actor of the application graph is always the best partition solution as n becomes a very large number. This application caused limitation makes the complex applications have more task allocation possibilities than the simple ones.

In the next sets of simulations, we estimate the overhead cost of executing DOOTA algorithm. The overhead cost mainly consists of two parts: the computation and communication costs. The computation cost is measured by the execution time of running the algorithm in Matlab. Fig. 11 illustrates the algorithm runtimes of the above mentioned approaches for MEPS and spectrum applications. It is very fast to execute the on-line approaches due to the lightweight complexity, while the time consumption of the two off-line approaches are hundreds of times higher than them. The heuristic on-line approach requires the least runtime, since it only look-ups the table, where the relation between the partition solutions and the corresponding number of slave nodes are stored off-line.

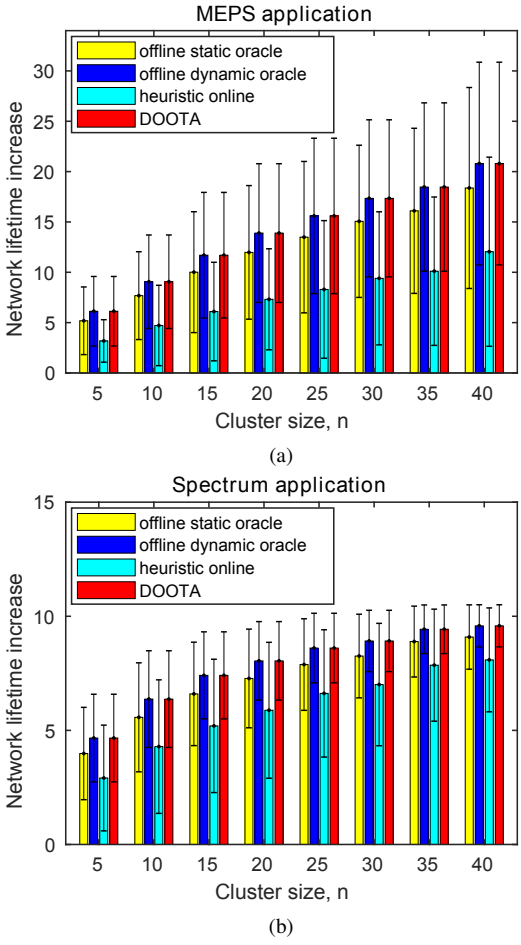


Fig. 10. The network lifetime increase by applying the heuristic on-line, off-line static and dynamic oracles, and DOOTA algorithms with respect to *no scheduling* for different cluster size. (a) MEPS application. (b) spectrum application.

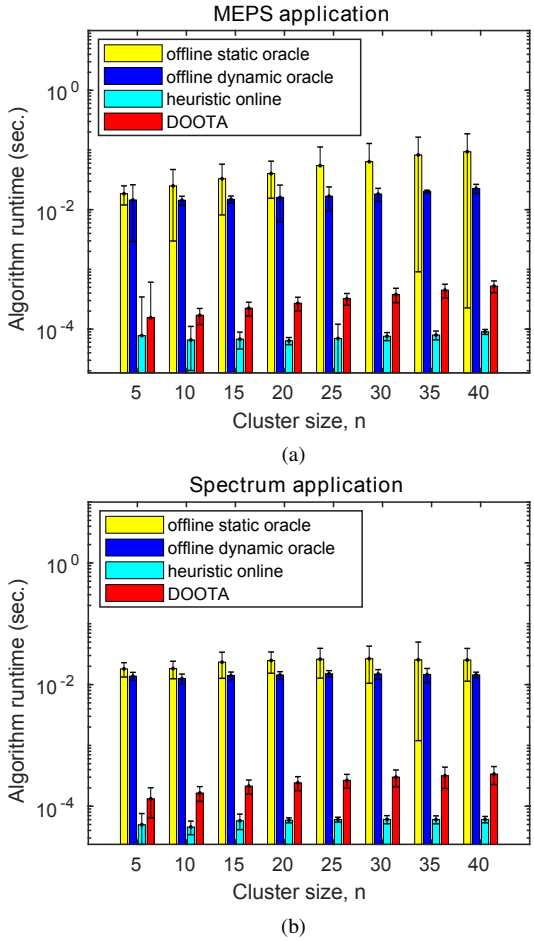


Fig. 11. The algorithm runtime of the heuristic on-line, off-line static and dynamic oracles, and DOOTA algorithms for different cluster size. (a) MEPS application. (b) spectrum application.

Although DOOTA needs slightly more time, it provides the optimal partition solutions and achieves the maximum network lifetime. Thus, the energy cost of the simple computation of DOOTA can be neglected. Note that the complexities of the applications do not affect the runtime of DOOTA, which is only influenced by the number of the *important partition cuts* of the application graph as presented in Sec. III-C2. The runtimes of DOOTA for both MEPS and spectrum applications are very close as shown in Fig. 11, since both the MEPS and spectrum application graphs have the same number of *important partition cuts* as illustrated in Sec. IV-A2.

The communication overhead is measured by the number of the message exchanges between each slave and the master node. As shown in Fig. 12, the number of the message exchanges is very small. Specifically, each node only needs in average up to 5 message exchanges as n changes from 5 to 40 for MEPS application. Since DOOTA is able to calculate the temporary optimal network lifetime according to (19), which significantly reduces the number of the message exchanges. Therefore, compared with the network lifetime which lasts hundreds of thousands scheduling rounds, the overhead of executing DOOTA algorithm can be neglected. Besides, there is a slight difference between the number of message exchanges per node for MEPS and spectrum applications, since DOOTA is operated based on the *important partition cuts* of the application graph.

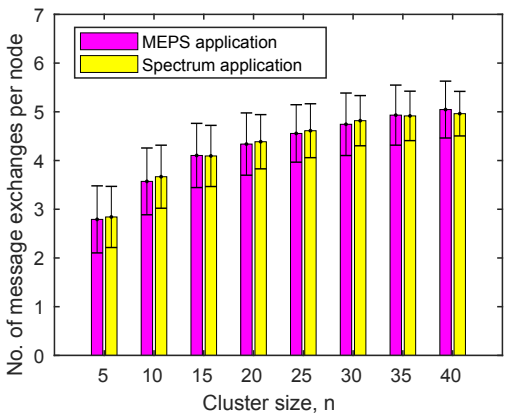


Fig. 12. Number of message exchanges per node of DOOTA algorithm for MEPS and spectrum applications.

Further on, DOOTA algorithm can flexibly respond the dynamic changes of the network, which is the major limitation of the off-line algorithms or complex centralized algorithms. When the network changes at run time, e.g., the cluster size suddenly changes from 10 to 20, DOOTA and the heuristic on-line scheme can immediately respond the dynamic changes by few number of message exchanges and table look-up operation, respectively. While the off-line static and dynamic approaches cannot provide any partition solution for the new added nodes[2]. Fig. 13 depicts the improvements of network lifetime of the four approaches for the suddenly changed

[2]We consider that the *oracles* are not available when dynamic changes happen.

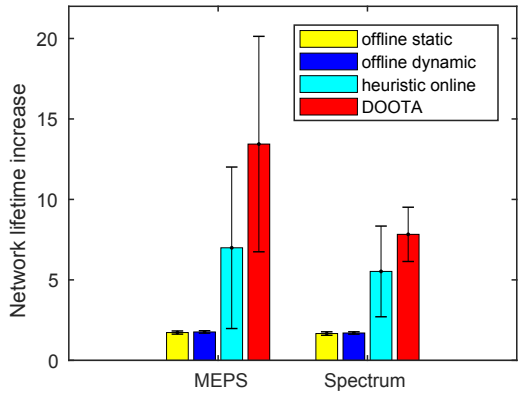


Fig. 13. The network lifetime increase by by applying the heuristic on-line, off-line static and dynamic, and DOOTA algorithms, with respect to *no scheduling* when the cluster size suddenly changes from 10 to 20 at run time.

cluster size. As expected, both the DOOTA and heuristic on-line scheme increase the network lifetime much longer than the off-line static and dynamic approaches. DOOTA outperforms the heuristic on-line scheme, since it provides the optimal partition solution. Meanwhile, the superiority of DOOTA is more significant for complex MEPS application than spectrum application, which is consistent with the results in Fig. 10.

V. Conclusion

This paper proposes a distributed optimal on-line task allocation (DOOTA) algorithm for cluster-based wireless sensor networks. It takes the energy consumption of sensing, computing, communicating and sleeping into account, and provides optimal partition solution to maximize the network lifetime. Unlike the previous centralized and off-line approaches which have to know all of the network parameters in advance, e.g., battery energy and location of each node, this work enables each node to calculate its own partition solution on-line with very lightweight complexity. Through an in-depth analysis, we prove that the optimal solution for each node is made up of at most two of the *important partition cuts* with the proper weights. This can be obtained with the help of the binary decision diagram theory. The simulation results demonstrate the efficiency of DOOTA algorithm. With very fast execution time and typically 3 to 5 message exchanges between each slave and the master, DOOTA maximizes the network lifetime as much as the off-line dynamic algorithm. It always outperforms the heuristic on-line and off-line static task allocation algorithms. The superiority of DOOTA becomes more significant for complex applications. In addition, DOOTA can flexibly address the dynamic network changes at runtime.

References

[1] K. C. Barr and K. Asanovi, "Energy-aware lossless data compression," *ACM Trans. Comput. Syst.*, vol. 24, no. 3, pp. 250–291, 2006.

[2] S. Tyagi and N. Kumar, "A systematic review on clustering and routing techniques based upon LEACH protocol for wireless sensor networks," *Journal of Network and Computer Applications*, vol. 36, no. 2, pp. 623 – 645, 2013.

[3] F. Ren, J. Zhang, T. He, C. Lin, and S. K. D. Ren, "Ebrp: Energy-balanced routing protocol for data gathering in wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 12, pp. 2108–2125, 2011.

[4] M. Zhao, Y. Yang, and C. Wang, "Mobile data gathering with load balanced clustering and dual data uploading in wireless sensor networks," *IEEE Transactions on Mobile Computing*, vol.14, no.4, pp.770–785, 2015.

[5] Y. Liang and W. Peng, "Minimizing energy consumptions in wireless sensor networks via two-modal transmission," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 12–18, 2010.

[6] M. Vecchio, R. Giaffreda, and F. Marcelloni, "Adaptive lossless entropy compressors for tiny iot devices," *IEEE Transactions on Wireless Communications*, vol. 13, no. 2, pp. 1088–1100, 2014.

[7] W. Yu, Y. Huang, and A. Garcia-Ortiz, "An altruistic compression-scheduling scheme for cluster-based wireless sensor networks," in *Sensing, Communication, and Networking (SECON), 12th Annual IEEE International Conference on*, pp. 73–81, 2015.

[8] A.T. Hoang and M. Motani, "Collaborative broadcasting and compression in cluster-based wireless sensor networks," *ACM Transactions on Sensor Networks*, vol. 3, no. 3, pp. 17, 2007.

[9] C. Liu, K. Wu, and J. Pei, "An energy-efficient data collection framework for wireless sensor networks by exploiting spatiotemporal correlation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 7, pp. 1010–1023, 2007.

[10] Y. Huang, W. Yu, and A. Garcia-Ortiz, "PKF: A communication cost reduction schema based on kalman filter and data prediction for wireless sensor networks," in *Proceedings of 26th IEEE Inernational system-on-chip conference*. CAS, pp. 73–78, 2013.

[11] Y. Huang, W. Yu, C. Osewold, and A. Garcia-Ortiz, "Analysis of PKF: A communication cost reduction scheme for wireless sensor networks," *Wireless Commu., IEEE Transactions on*, vol.15, no.2, pp.843–856, 2016.

[12] L. Tang, Y. Sun, O. Gurewitz, and D. B. Johnson, "Pw-mac: An energy-efficient predictive-wakeup mac protocol for wireless sensor networks," in *INFOCOM, Proceedings IEEE*, pp. 1305–1313, 2011.

[13] Y. Wu, X. Y. Li, Y. Li, and W. Lou, "Energy-efficient wake-up scheduling for data collection and aggregation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 2, pp. 275–287, 2010.

[14] C. Alippi, G. Anastasi, M. D. Francesco, and M. Roveri, "Energy management in wireless sensor networks with energy-hungry sensors," *IEEE Instr. Meas. Magazine*, vol. 12, no. 2, pp. 16–23, 2009.

[15] G. de Meulenaer, F. Gosset, F. X. Standaert, and O. Pereira, "On the energy cost of communication and cryptography in wireless sensor networks," in *2008 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, pp. 580–585, 2008.

[16] M. Kafil and I. Ahmad, "Optimal Task Assignment in Heterogeneous Distributed Computing Systems," *IEEE Concurrency*, vol. 6, no. 3, pp. 42–51, 1998.

[17] D. Bozdag, F. Ozguner, E. Ekici, and U. Catalyured, "A task duplication based scheduling algorithm using partial schedules ," in *International Conference on Parallel Processing*, pp. 630–637, 2005.

[18] J. Yang, H. Zhang, Y. Ling, C. Pan, and W. Sun, "Task allocation for wireless sensor network using modified binary particle swarm optimization," *IEEE Sensors Journal*, vol. 14, no. 3, pp. 882–892, 2014.

[19] W. Guo, J. Li, G. Chen, Y. Niu, and C. Chen, "A pso-optimized real-time fault-tolerant task allocation algorithm in wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 12, pp. 3236–3249, 2015.

[20] Y. Jin, J. Jin, A. Gluhak, K. Moessner, and M. Palaniswami, "An intelligent task allocation scheme for multihop wireless networks," *IEEE Trans. on Para. and Dist. Systems*, vol.23, no.3, pp.444–451, 2012.

[21] Y. Jin, S. Vural, A. Gluhak and K. Moessner, "Dynamic Task Allocation in Multi-Hop Multimedia Wireless Sensor Networks with Low Mobility," *Sensors*, vol. 13, no. 10, pp. 13998–14028, 2013.

[22] D.-I. Ko, C.-C. Shen, S. Bhattacharyya, and N. Goldsman, "Energy-driven partitioning of signal processing algorithms in sensor networks," in *Proceedings of International Workshop on Embedded Computer Systems* vol. 4017, pp. 142–154, 2006.

[23] C.-C. Shen, W. L. Plishker, D.-I. Ko, S. S. Bhattacharyya, and N. Goldsman, "Energy-driven distribution of signal processing applications across wireless sensor networks," *ACM Trans. Sen. Netw.*, vol. 6, no. 3, pp. 24:1–24:32, 2010.

[24] Y. Yu and V. K. Prasanna, "Energy-Balanced Task Allocation for Collaborative Processing in Wireless Sensor Networks," *Mobile Networks and Applications*, vol. 10, no. 1, pp. 115–131, 2005.

[25] Y. Huang, W. Yu, and A. Garcia-Ortiz, "Accurate energy-aware workload distribution for wireless sensor networks using a detailed communication energy cost model," *Journal of Low Power Electronics*, vol. 10, no. 2, pp. 183–193, 2014.

[26] W. Yu, Y. Huang, and A. Garcia-Ortiz, "Modeling optimal dynamic scheduling for energy-aware workload distribution in wireless sensor networks," in *Proceedings of 12th International conference on distributed computing in sensor systems (DCOSS)*. IEEE/ACM, pp. 116–118, 2016.

[27] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, 10 pp. vol.2, IEEE, 2000.

[28] I. Dietrich and F. Dressler, "On the Lifetime of Wireless Sensor Networks," *ACM Trans. Sen. Netw.*, vol. 5, no. 1, 2009.

[29] M. A. Abd and S. F. M. Al-Rubeaai and B. K. Singh and K. E. Tepe and R. Benlamri, "Extending Wireless Sensor Network Lifetime With Global Energy Balance," *IEEE Sen. Journal*, vol.15, no.9, pp.5053–5063, 2015.

[30] S. S. Bhattacharyya, W. Plishker, etc., "Modeling and optimization of dynamic signal processing in resource-aware sensor networks," in *Advanced Video and Signal-Based Surveillance (AVSS), 2011 8th IEEE International Conference on*, pp. 449–454, 2011.

[31] K. Romer and F. Mattern, "The design space of wireless sensor networks," *IEEE Wireless Communications*, vol. 11, no. 6, pp. 54–61, 2004.

[32] R. Drechsler and B. Becker, "Binary decision diagrams: theory and implementation," *Springer Science & Business Media*, 2013.

[33] F. Somenzi, "CUDD: CU Decision Diagram Package Release 3.0.0," *Department of Electrical, Computer, and Energy Engineering, University of Colorado at Boulder*, 2015.

[34] Texas-instruments, *CC2430 Datasheet*, Chipcon Products from Texas Instruments, USA: Texas Instruments.

[35] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, etc., "Taming heterogeneity - the ptolemy approach," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127–144, 2003.

[36] Q. Wang and M. Hempstead and W. Yang, "A Realistic Power Consumption Model for Wireless Sensor Network Devices," in *3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks*, pp. 286–295, 2006.