Dynamic Critical-Path Task Mapping and Scheduling for Collaborative In-Network Processing in Multi-Hop Wireless Sensor Networks

Yuan Tian, Yaoyao Gu, Eylem Ekici, and Füsun Özgüner
Department of Electrical and Computer Engineering, The Ohio State University
Email: {tiany,guy,ekici,ozguner}@ece.osu.edu

Abstract— **Although task mapping and scheduling in wired networks of processors has been extensively studied, their application to WSNs remains largely unexplored. Existing algorithms cannot be directly implemented in WSNs due to limited resource availability and shared communication medium. In this work, an application-independent task mapping and scheduling solution in multi-hop WSNs is presented that provides real-time guarantees. Using a novel multi-hop channel model and a communication scheduling algorithm, computation tasks and associated communication events are scheduled simultaneously with a dynamic critical-path scheduling algorithm. Dynamic Voltage Scaling (DVS) mechanism is implemented to further optimize energy consumption. According to the simulation results, the proposed solution outperforms existing mechanisms in terms of guaranteeing application deadlines with minimum energy consumption.**

I. Introduction

Many emerging applications for WSNs are associated with real-time requirements that necessitate in-network processing. Processing information locally and sending the end results to a central location is generally more energy-efficient than sending raw data across a multi-hop WSN. The reduced communication volume also reduces the delivery latency, and, hence, improves real-time performance. Many algorithms used for in-network processing require significant processing power. In case of video sensor networks, applications such as image registration [1] and distributed visual surveillance [2] involve computationally intensive operations. *Collaborative in-network processing* is a viable solution to provide the required processing power not available in stand-alone sensor nodes. To enable collaborative in-network processing, the following problems must be solved:

- Assignment of tasks to sensors,
- Determining execution sequence of tasks,
- Scheduling communication between sensors.

In high-performance computing, the first problem is referred to as *task mapping* and the second one as *task scheduling*. Both problems have been extensively studied in the past for interconnected processors in wired networks. However, these existing solutions cannot directly be implemented in WSNs: Wireless communication scheduling such as collision avoidance is not addressed. Furthermore, most of these solutions do not explicitly consider energy consumption during communication and task execution, which is one of the major constraints in WSNs.

In large-scale WSNs, events of interest generally occur in remote regions that only local sensors can detect. Consequently, local information processing, and localized task mapping and scheduling is more suitable for large-scale WSNs. In [3], an online task scheduling mechanism (CoRAl) is proposed to allocate network resources between the tasks of periodic applications in WSN clusters iteratively: The frequencies of the tasks on each sensor are optimized subject to the previously evaluated upper-bound execution frequencies. However, CoRAl does not address mapping tasks to sensor nodes. Distributed Computing Architecture (DCA) is proposed in [4], which executes low level tasks on sensing sensors and offloads all other high level processing tasks to cluster heads. However, processing high level tasks can still exceed the capacity of cluster heads' computation power. Furthermore, application-specific design of these solutions limit their implementation for generic applications.

Task mapping and scheduling heuristics are presented in [5] for heterogeneous mobile ad hoc grid environments. However, the communication model adopted in [5] is not well-suited for WSNs, which assumes individual channels for each node, and concurrent data transmission and reception capacity of every node. The EcoMapS algorithm in [6] aims to minimize the schedule length subject to the energy consumption constraint in single-hop clustered WSNs. However, EcoMapS does not provide execution deadline guarantees for applications. In [7], Energy-balanced Task Allocation (EbTA) is introduced to minimize balanced energy consumption subject to application deadline constraints. In [7], communications over multiple wireless channels are modeled as additional linear constraints of an Integer Linear Programming (ILP) problem, and a heuristic algorithm with Dynamic Voltage Scaling (DVS) mechanism is presented. However, the communication scheduling model in [7] does not exploit the broadcast nature of wireless communication, which can conserve energy and time expenditure. All localized mechanisms above assume a single-hop cluster environment, which hinders their application to general implementations.

The aim of this work is to develop an *application independent* solution to provide the in-network computation capacity required by arbitrary real-time applications while minimizing energy consumption. We consider delay-constrained applications executed in multi-hop clusters of homogeneous wireless sensors. We propose the Dynamic Critical-path Task Mapping and Scheduling (DCTMP) solution that jointly schedules communication and computation tasks of an application with minimum energy consumption subject to delay constraints. The proposed DCTMP algorithm is based on the high-level application model that describes applications through a Directed Acyclic Graph (DAG) [7], which can be used to model arbitrary applications. A novel communication model is proposed to model multi-hop wireless channels. Based on this channel model, a multi-hop communication scheduling algorithm is integrated as part of DCTMP. In DCTMP, communication and computation are jointly scheduled in two phases: *task mapping and scheduling phase* and *DVS phase*. In the *task mapping and scheduling phase*, communication and computation events are scheduled at highest processing power to find a feasible solution. The Dynamic Voltage Scaling (DVS) is implemented in the *DVS phase* to reduce the energy consumption.

II. Preliminaries

A. Network Assumptions

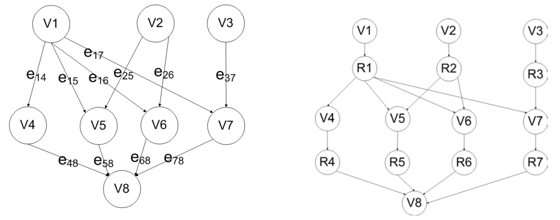
Our proposed task mapping and scheduling mechanism is designed for applications executed within a multi-hop cluster of WSNs. We assume the following WSN properties:

- Sensors are grouped into k-hop clusters with a clustering algorithm. In this paper, we define a k-hop network as a network G with diameter $diam(G) \leq k \cdot r$, where r is the sensor transmission range.
- Each cluster executes an application which is either assigned during the network setup time or remotely distributed by base stations during the network operation. With application arrivals, cluster heads create schedules for execution within clusters.
- Calculated schedules are used to run the associated applications as many times as required by applications.
- Location information is locally available within clusters.
- Communication within a cluster is isolated from other clusters through time division or channel hopping mechanisms with appropriate hardware support such as the Chipcon CC2420 transceiver [8].
- Sensors are equipped with DVS processors [4] whose speed and supply voltage can be dynamically adjusted with finite number of levels. The overhead of speed and voltage adjustment is assumed to be negligible.

It should be noted that while the intra-cluster communication is isolated from each other, the communication across clusters is assumed to be handled over common time slots or channels orthogonal to those used inside a cluster. As such, information flow across the network is not hindered by intra-cluster communication isolation.

B. Application and Energy Consumption Model

To have an application-independent solution, we represent applications executed in clusters with Directed Acyclic Graphs (DAG) [7]. A DAG *T = (V,E)* consists of a set of vertices V representing the tasks to be executed and a set of directed edges E representing communication dependencies among tasks. The edge set E contains directed edges e_{ij} for each task $v_i \in V$ that task $v_j \in V$ depends on. The computation weight of a task is represented by the number of CPU clock cycles to execute the task. Given an edge e_{ij}, v_i is called the immediate predecessor of v_j, and v_j is called the immediate successor of v_i. An immediate successor v_j depends on its immediate predecessors such that v_j cannot start execution before it receives results from all of it immediate predecessors. A task without immediate predecessors is called an *entry task* and a task without immediate successors is called an *exit task*. We assume a DAG may have multiple entry tasks and one exit task. If there are more than one exit-tasks, they will be connected to a pseudo exit-task with computation cost equals zero. Fig. 1(a) shows an example of a DAG.



(a) An Example DAG (b) Hyper-DAG Representation

Fig. 1. DAG Examples

In this paper, we assume that an entry task is a sensing-task to detect certain physical events, and its sensor assignment is determined according to application requirements. This entry-task assignment requirement is referred to as *Entry-task Assignment Constraint* throughout the paper.

In the DAG scheduling problem, if a task v_j scheduled on one node depends on a task v_i scheduled on another node, a communication between these nodes is required. In such a case, v_j cannot start its execution until the communication is completed and the result of v_i is received. However, if both of the tasks are assigned on same node, the result delivery latency is considered to be zero and v_j can start to execute after v_i is finished. This execution dependency between tasks is referred to as *Communication Dependency Constraint* throughout the paper.

The energy consumptions of transmitting and receiving l-bit data over a distance d that is less than a threshold d_o are defined as $E_{tx}(l, d)$ and $E_{rx}(l)$, respectively:

$$E_{tx}(l, d) = E_{elec} \cdot l + \varepsilon_{amp} \cdot l \cdot d^2, \quad (1)$$

$$E_{rx}(l) = E_{elec} \cdot l, \quad (2)$$

where E_{elec} and ε_{amp} are hardware related parameters [4] [9].

The energy consumption of executing N clock cycles with CPU clock frequency f is given as:

$$E_{comp}(V_{dd}, f) = NCV_{dd}^2 + V_{dd}(I_o e^{\frac{V_{dd}}{nV_T}})(\frac{N}{f}), \quad (3)$$

$$f \simeq K(V_{dd} - c), \quad (4)$$

where V_T is the thermal voltage and C, I_o, n, K and c are processor dependent parameters [10] [4].

It should be noted that the energy consumption model presented above only considers the energy expenditure directly related with application executions, thus energy consumption during idle time is not taken into account.

C. Problem Statement

The task mapping and scheduling problem is to find a set of task assignments and their execution sequences on a network that minimizes an objective function such as energy consumption or schedule length. Let $H^x = \{h_1^x, h_2^x, ..., h_n^x\}$ denote a task mapping and scheduling solution of the application DAG T on a network G, where x is the index of the task mapping and scheduling solution space. Each element $h_i^x \in H^x$ is a tuple of the form $(v_i, m_k, s_{i,m_k}, t_{i,m_k}, f_{i,m_k}, c_{i,m_k})$, where m_k represents the node to which task v_i is assigned, s_{i,m_k}, t_{i,m_k}, f_{i,m_k}, and c_{i,m_k} represent the start time, execution time, finish time, and energy consumption of v_i on node m_k, respectively. The design objective of DCTMP is to find an $H^o \in \{H^x\}$ that has the minimum energy consumption under the delay constraint:

$$\min energy(H^o) = \sum_{i,k} c_{i,m_k}; \tag{5}$$

$$\text{subject to } length(H^o) = \max_{i,k} f_{i,m_k} \leq DL, \tag{6}$$

where $length(H)$ and $energy(H)$ are the schedule length and energy consumption of H, respectively, and DL is the deadline of the application. DAG scheduling problem is shown to be an NP-complete problem in general [11]. Therefore, heuristic algorithms are needed to solve this problem in polynomial time.

Some notations are listed here for convenience:

- $pred(v_i)$ and $succ(v_i)$ denote the immediate predecessors and successors of task v_i respectively,
- $m(v_i)$ denotes the node on which v_i is assigned,
- $T(m_k)$ denotes the tasks assigned on node m_k,
- $T_{st}^{ft}(m_k)$ denotes the tasks assigned on node m_k during the time interval $[st, ft]$.

III. The Proposed Scheduling Solution

The proposed scheduling solution consists of two phases: *task mapping and scheduling phase* and *DVS phase*. In the *task mapping and scheduling phase*, applications are scheduled across application, MAC and network layers: Computation tasks are assigned to sensors, their execution sequence are decided, and communications between sensors are scheduled based on the *Communication Dependency Constraints*. The task mapping and scheduling phase aims to guarantee application deadline constraints. We design a Dynamic Critical-path Task Mapping and Scheduling (DCTMS) algorithm as the multi-hop task schedule search engine. DCTMS dynamically evaluates critical-paths of task graphs, and assigns the most critical task to shorten schedule lengths. To guarantee application deadline constraints, sensors are scheduled with the maximum CPU speed f_{cpu}^{max} by the DCTMS algorithm. Then, energy consumption of the schedules created in the first phase are optimized in the *DVS phase* by reducing CPU speeds to exploit CPU slack times.

Unlike traditional dynamic critical-path scheduling algorithms such as [12] without considering wireless communication, DCTMS is designed for multi-hop WSN applications. We developed a new *multi-hop communication scheduling algorithm* based on our proposed Hyper-DAG representation of tasks and multi-hop channel model. The communication scheduling algorithm is utilized by the DCTMS algorithm during task scheduling to satisfy the *Communication Dependency Constraints*. In the following sections, the main components of our proposed task mapping and scheduling algorithm, namely, Hyper- DAG extension and multi-hop channel modeling, communication scheduling algorithm, DCTMS algorithm, and DVS algorithm, are presented.

A. Hyper-DAG Extension and Multi-Hop Channel Modeling

In WSNs, communication is broadcast in nature. When a node transmits information, it is potentially received by multiple nodes in the cluster. This property can be leveraged to relay information generated by a task to all its successors in a single transmission rather than multiple, sequential transmissions. This approach both reduces the execution time as well as the energy consumption. To represent the broadcast feature of wireless communication, the DAG representation of applications is extended as follows: For a task v_i in a DAG, we replace the edges between v_i and its immediate successors with a net R_i. The weight of R_i equals to the result data volume of v_i. R_i represents the communication task to send the result of v_i to all its immediate successors in the DAG. This extended DAG is a hypergraph and is referred to as *Hpyer-DAG*. The Hyper-DAG representation of the DAG in Fig. 1(a) is shown in Fig. 1(b). A Hyper-DAG is represented as $T' = (V', E')$, where $V' = \{\gamma_i\} = V \cup R$ denotes the new set of tasks to be scheduled and E' represents the dependencies between tasks. Here, $V = \{v_i\}$ = {*Computation Tasks*}, and $R = \{R_i\}$ = {*Communication Task*}.

With Hyper-DAGs, communication events between computation tasks are explicitly represented in task graphs. To properly schedule communication events, we model multi-hop channel as a virtual node $\mathcal{C}$ on which only communication tasks can be executed. Different from the virtual node model in [3] [6], where only single-hop channels are considered, our channel model takes potential interference between simultaneous communications in multi-hop networks into consideration.

Unlike in single-hop networks, there can be multiple simultaneous communications in multi-hop networks. Thus, the virtual node $\mathcal{C}$ in multi-hop channel model should be able to execute multiple communication tasks simultaneously. To avoid interference between scheduled communication tasks, a "*penalty function*" is introduced into the cost function of communication scheduling. Under unit disc graph model, the "penalty" of scheduling a communication task is zero if it does not cause interference; otherwise, it is infinite. The communication scheduling algorithms will only schedule a communication task with the minimum finite cost. The penalty function $P_{st}^{ft}(v)$ of assigning a communication task v onto $\mathcal{C}$ during time interval $[st, ft]$ is defined as:

$$P_{st}^{ft}(v) = \begin{cases} \infty, if\ \exists \gamma \in T_{st}^{ft}(\mathcal{C}) : S(\gamma) \in N(R(v))\ or\ R(\gamma) \in N(S(v)) \\ 0,\ otherwise, \end{cases} \tag{7}$$

where $S(\gamma)$ and $R(\gamma)$ are the sender and receivers of communication task γ, respectively, and $N(m_k)$ is the set of sensor $m_k's$ one-hop neighbors. With the penalty function defined above, the multi-hop channel model is presented as follows:

- Wireless channel is modeled as a virtual node $\mathcal{C}$.
- $\mathcal{C}$ executes communication tasks only.

- There can be multiple tasks on $\mathcal{C}$ in time interval $[st, ft]$, denoted as $T_{st}^{ft}(\mathcal{C})$.
- The cost of executing communication task v_i on $\mathcal{C}$ in time interval $[st, ft]$ is $cost(v_i, st, ft) = st + P_{st}^{ft}(v_i)$.

With the Hyper-DAG representation and the channel model, the *Communication Dependency Constraint* in Section II-B is rephrased as follows: In the Hyper-DAG scheduling problem, if a computation task v_j scheduled on node m_k depends on a communication task v_i scheduled on another sensor node or $\mathcal{C}$, a copy of the communication task v_i needs to be scheduled to m_k, and v_j cannot start to execute until all of its immediate predecessors are received on the same node.

B. Communication Scheduling Algorithms

To meet the *Communication Dependency Constraint* in Hyper-DAG scheduling, communication between nodes is required if a computation task depends on a communication task assigned on another node. In multi-hop clusters, the sender and the receiver of a communication task can be one or more hops away from each other. We schedule multi-hop communication following the paths generated by a routing algorithm. In every hop, we use the one-hop communication scheduling algorithm.

We first introduce the one-hop communication scheduling algorithm. With the Hyper-DAG and the multi-hop channel models presented in Section III-A, scheduling communication between single-hop neighbors is equivalent to first duplicating a communication task from the sender to $\mathcal{C}$, and then from $\mathcal{C}$ to the receiver. If the requested communication task has been scheduled from the sender to another node before, the receiver will directly duplicate the communication task from $\mathcal{C}$ if it is sent within its communication range and not interfered by other scheduled neighboring communication. This process is equivalent to receiving broadcast data, which can lead to significant energy saving compared with multiple unicasts between the sender and the receivers. The detailed description of the single-hop communication scheduling algorithm is presented below.

Input: Communication task: v_i; sender of v_i: m_s; receiver of v_i: m_r
Output: Schedule of duplicating v_i from m_s to m_r
OneHopCommTaskSchedule(v_i,m_s,m_r)**:**

```
1. Find a copy of $v_i$: $v_i^c \in T(\mathcal{C})$, $S(v_i^c) = m_s$
2. IF $v_i^c$ does not exist
3.     Find $v_i \in T(m_s)$
4.     Find time interval [st,ft]:
5.         $cost(v_i, st, ft) = \min$
6.         $st \geq f_{v_i,m_s}$, $ft - st \geq t_{v_i,\mathcal{C}}$
7.     Schedule a copy of $v_i$ to $\mathcal{C}$:
8.         $s_{v_i^c,\mathcal{C}} \leftarrow st$, $T(m_k) \leftarrow T(m_k) \cup \{v_i^c\}$
9.     Schedule a copy of $v_i^c$ to $m_r$:
10.        $s_{v_i^k,m_k} \leftarrow f_{v_i^c,\mathcal{C}}$, $T(m_k) \leftarrow T(m_k) \cup \{v_i^k\}$
11.ELSE
12.    $st \leftarrow s_{v_i^c,\mathcal{C}}$, $ft \leftarrow f_{v_i^c,\mathcal{C}}$
13.    IF $\nexists \gamma \in T_{st}^{ft}(\mathcal{C}) : S(\gamma) \in N(m_r)$
14.        Schedule a copy of $v_i^c$ to $m_r$:
15.            $s_{v_i^k,m_k} \leftarrow f_{v_i^c,\mathcal{C}}$, $T(m_k) \leftarrow T(m_k) \cup \{v_i^k\}$
16.            $R(v_i^c) \leftarrow R(v_i^c) \cup \{m_r\}$
17.    ELSE
18.        Goto Step 3
```

Steps 2-10 stand for originating a new communication from m_s to m_r, and Step 13-16 represent reception of a broadcast data without interference. Compared with originating a new communication, the broadcast reception method leads to energy saving of one data transmission for each additional data reception.

In our multi-hop communication scheduling algorithm, the low complexity stateless geographic routing algorithm, GPSR [13] algorithm is used to obtain the $path = (m_1, ..., m_n)$ from sender m_s to receiver m_r, where $m_1 = m_s$ and $m_n = m_r$. After obtaining the path, the communication task will be iteratively duplicated from the source to the destination. Similar to that of the one-hop communication scheduling, the requested data might have been scheduled from the source to another node before. Thus, the requested communication task may have duplicate copies distributed in the network. To shorten communication latencies and to decrease communication energy consumption, the communication task should be forwarded starting from the location closest to the destination. The detailed description of the multi-hop communication scheduling is as follows:

Input: Communication task: v_i; receiver of v_i: m_r; sensor set SS
Output: Schedule of duplicating v_i to m_r
CommTaskSchedule(v_i,m_r)**:**

```
1. IF $\nexists$ a copy of $v_i$: $v_i^c \in T(\mathcal{C})$
2.     Find the sensor node $m_s$: $v_i \in T(m_s)$
3.     Calculate the path from $m_s$ to $m_r$:$path = (m_1, ..., m_n)$
4.     For $m_k = m_2$ to $m_n$
5.         OneHopCommTaskSchedule($v_i, m_s, m_k$)
6.         $m_s \leftarrow m_k$
7.     Return
8. ELSE
9.     Find a copy of $v_i$: $v_i^o \in T(\mathcal{C})$, $dist(S(v_i^o), m_r) = \min$
10.    Find $m_s \in N(S(v_i^o))$: $dist(m_s, m_r) = \min$
11.    IF $v_i$ does not have a copy on $m_s$
12.        OneHopCommTaskSchedule($v_i, S(v_i^o), m_s$)
13.    Goto Step 3
```

A data transmission may reach multiple destinations under our communication scheduling. Effectively, we achieve *multicast* distribution of data, which has been proved to be energy efficient. The communication scheduling algorithm is used in conjunction with the task scheduling algorithm as described in Section III-C.

C. Scheduling with DCTMS Algorithm

In the *Task Mapping and Scheduling Phase*, tasks of a Hyper-DAG are assigned to sensors nodes and $\mathcal{C}$. During task mapping, several constraints must be satisfied. These constraints together with the *Communication Dependency Constraint* are represented as follows:

- A computation task can be assigned only on sensor nodes
- A communication task can be assigned both on sensors and $\mathcal{C}$. If a communication task has its immediate predecessor and immediate successors assigned on the same node, it has zero execution length and energy cost.

- If $v_i \in V$ and $pred(v_i) \neq \emptyset$, then $pred(v_i) \subset T(m(v_i))$ and $s_{v_i,m(v_i)} \geq \max f_{pred(v_i),m(v_i)}$

With the *Hyper-DAG* representation, *multi-hop channel model*, *Communication Scheduling Algorithm*, and the task mapping constraints presented above, task mapping and scheduling in multi-hop wireless networks can be tackled as a generic task mapping and scheduling problem with additional constraints. This problem is NP-complete in general [11] and heuristic algorithms are needed to obtain practical solutions. Dynamic critical-path scheduling is known for its relatively low complexity with satisfying schedule length performance. We propose our Dynamic Critical-path Task Mapping and Scheduling (DCTMS) algorithm composed by the following procedures:

- Dynamic critical-path evaluation and optimal task selection (DCEOTS)
- Optimal sensor searching and task assignment (OSSTA)

The *DCEOTS* procedure calculates the critical-path of Hyper-DAGs, and finds the next task to be assigned accordingly. The selected task will then be independently assigned on "active sensors" one by one to find the optimal sensor giving the shortest schedule length in *OSSTA*. Here, an "active sensor" is a sensor that either runs computation tasks or participates communication activities by sending, receiving or routing communication tasks. These procedures are iteratively executed until all tasks are assigned. In both procedures, network topology and communication scheduling are embedded into the decision making procedure to reflect the multi-hop wireless network features. The details of DCTMS are described below.

1) DCEOTS Procedure: The core of the DCTMS scheduling algorithm is the *DCEOTS Procedure* that dynamically evaluates critical-paths, along which tasks potentially have the largest execution time and may determine schedule lengths. Unlike traditional dynamic critical-path scheduling algorithms that have full connections between processors with fixed communication latency, DCTMS is executed on Hyper-DAG for wireless communication in multi-hop WSNs. Thus, the execution time of a communication task is not only determined by the communication data volume but the assignment of the communication task: Depending on locations of senders and receivers, communication tasks may travel various number of hops. Since the selected task will be experimentally assigned on each active sensor, we estimate the communication latency with the average hop-distance AVG_{hop} between active sensors, where AVG_{hop} is dynamically updated in the *OSSTA procedure*.

The *DCEOTS Procedure* dynamically calculates critical-paths as follows: Similar to the E-CNPT algorithm in [6], DCEOTS first iteratively calculates the earliest start time $EST(v_i)$ of task v_i by traveling downward Hyper-DAGs. For tasks that have already been assigned, their EST equals their scheduled start time; Otherwise, their EST is given by:

$$EST(v_i) = \max_{v \in pred(v_i)} \{EST(v) + t_v\}, \tag{8}$$

$$t_v = \{ {}^{C_v/f_{CPU}^{max}, v \in R}_{AVG_{hop} \cdot R_v/BW, v \in R}, \tag{9}$$

where C_v, R_v, and BW are the computation load, communication data volume, and channel bandwidth, respectively.

Similar to EST, the latest start time (LST) is calculated by traveling upward Hyper-DAGs from the exit task. For exit-tasks and assigned tasks, their LST equals to their EST. Otherwise, their LST is given by:

$$LST(v_i) = \min_{v \in succ(v_i)} \{LST(v)\} - t_{v_i}, \tag{10}$$

where t_{v_i} has the same definition as t_v in Equation 9.

Starting from the exit-task, the path along which tasks have the same value of EST and LST is the critical-path. A dynamic critical-path ends at assigned tasks, and the unassigned "top" task closest to assigned tasks is called a primary critical-node (PC). A "mappable" PC with all immediate predecessors already assigned will be passed to OSSTA for further process; Otherwise, a *secondary critical-path* will be recursively found: Starting from the PC, a task's immediate predecessor with the minimum LST is added to the path until an assigned task is reached. The unassigned "top" task closest to assigned tasks is called a secondary critical-node (SC), and is passed to OSSTA for further process.

2) OSSTA Procedure: In the OSSTA Procedure, the to-be-assigned task from DCEOTS will be experimentally scheduled on active sensors, and the schedule with the minimum schedule length will be kept. During the scheduling, sensors are scheduled with full speed f_{cpu}^{max}.

```
Input: Hyper-DAG; sensor set: $SS$
Output: Schedule of $v^o$
OSSTA Procedure:
1. Assign entry-tasks according to Entry-task Assignment Constraint
2. Initialize $AVG_{hop}$
3. WHILE not all tasks assigned
4.     Find the next PC or SC $v^o$ with the DCEOTS procedure
5.     IF $v^o \in R$
6.         Assign $v^o$ to $m(pred(v^o))$
7.     ELSE
8.         FOR all active sensors $m_k$
9.             IF $pred(v^o) \not\subseteq T(m_k)$
10.                FOR $v_n \in pred(v^o) - T(m_k)$
11.                    CommTaskSchedule($v_n$,$m(v^o)$,$m_k$)
12.            Assign $v^o$ to $m_k$
13.        Keep the schedule with $m^o$: $f_{v^o,m^o} = \min$
14.        Update $AVG_{hop}$ if new active sensors involved
```

D. The DVS Algorithm

Due to the discrete nature of task mapping and scheduling, a schedule that meets a deadline may do so with some more slack time until the deadline. The unbalanced load of sensors and the communication scheduling also result in CPU idle time. In the *DVS Phase*, the CPU idle time is exploited by decreasing the CPU speed to reduce computation energy consumption.

Before introducing the DVS Algorithm, a concept of *CPU utility* η during a time interval $[st, ft]$ is first defined as:

$$\eta = e_{st}^{ft}/(ft - st), \tag{11}$$

where e_{st}^{ft} is the CPU execution time during $[st, ft]$. To exploit the CPU slack time, the strategy of the CPU adjustment algorithm is to slow down the CPU in proportion to the *CPU utility*. After adjustment, the *CPU utility* will approach 1 (but smaller than or equal to 1).

Our DVS algorithm has two stages: Schedule Length Extension (SLE) Stage and Schedule Hole Elimination (SHE) Stage. In the SLE stage, the slack time between schedule length $length(H)$ and application deadline DL is eliminated if any: Assume $\beta = \frac{length(H)}{DL} < 1$, we define the re-scale factor as $\gamma = \lceil \beta \cdot f_{cpu}^{max} \rceil / f_{cpu}^{max}$. Here, the function $\lceil f \rceil$ is a ceiling function that returns the minimum available CPU speed larger than or equal to f. Then, the CPU speed is reduced and the schedule length is increased in proportion to γ: All processors are slowed down to $\gamma \cdot f_{cpu}^{max}$ with less energy consumption; Computation tasks' start time, execution time, and finish time are extended with factor γ^{-1}; A communication task v_i's finish time f_{v_i,m_k} is multiplied by γ^{-1}, its execution time t_{v_i,m_k} remains unchanged, and its start time s_{v_i,m_k} is adjusted to $\gamma^{-1} f_{v_i,m_k} - t_{v_i,m_k}$.

After the SLE stage, slack times before application deadlines are decreased. However, the CPU idle time caused by the unbalanced load of sensors and the communication scheduling still exists. Thus, the adjusted schedule from the SLE stage needs to be further optimized in the SHE stage. We first present the procedure to adjust the CPU speed of a single sensor in a given time interval:

Input: sensor m_k; time interval $[st, ft]$; original CPU speed f_{cpu}
Output: Adjusted CPU speed f_{cpu}^o and task scheduling during $[st, ft]$
SpeedAdjust Algorithm(m_k,st,ft,f_{cpu}):

```
1. $e_{st}^{ft} \leftarrow 0$, $tt \leftarrow st$
2. FOR $v_i \in T_{st}^{ft}(m_k)$ and $v_i \in V$
3.     $e_{st}^{ft} \leftarrow e_{st}^{ft} + t_{v_i,m_k}$
4. $\eta \leftarrow e_{st}^{ft}/(ft - st)$
5. $f_{cpu}^o \leftarrow \lceil f_{cpu} \cdot \eta \rceil$
6. FOR $v_i \in T_{st}^{ft}(m_k)$ and $v_i \in V$
7.     $s_{v_i,m_k} \leftarrow tt$
8.     $t_{v_i,m_k} \leftarrow t_{v_i,m_k} \cdot \frac{f_{cpu}}{f_{cpu}^o}$, $f_{v_i,m_k} \leftarrow s_{v_i,m_k} + t_{v_i,m_k}$
9.     $tt \leftarrow f_{v_i,m_k}$
10. FOR $v_i \in T_{st}^{ft}(m_k)$ and $v_i \in R$
11.     IF $pred(v_i) \in T(m_k)$
12.         $s_{v_i,m_k} \leftarrow f_{pred(v_i),m_k}$, $f_{v_i,m_k} \leftarrow f_{pred(v_i),m_k}$
13. Update the energy consumption of $m_k$
```

In the SHE algorithm, the communication tasks on $\mathcal{C}$ are kept unchanged, and their start time and finish time are taken as the upper and lower bound to adjust the corresponding sensors' speed with the **SpeedAdjust** procedure. The SHE algorithm is described in details as follows:

Input: schedule H from the *Mapping and Scheduling Phase*, sensor set SS, application deadline DL
Output: Adjusted schedule H^o
SHE Algorithm:

```
1. FOR sensor $m_k \in SS$
2.     $st \leftarrow 0$, $ft \leftarrow \infty$
3.     FOR tasks $v_i \in T_{st}^{\infty}(m_k)$
4.         IF There is a copy of $v_i$: $v_i^c \in T(\mathcal{C})$
5.             Find the computation task $v_j$ following $v_i$
6.             IF $m_k$ is the sender of $v_i^c$
7.                 $ft \leftarrow \min(s_{v_i^c,\mathcal{C}}, s_{v_j,m_k})$
8.                 SpeedAdjust($m_k$,$st$,$ft$,$\gamma \cdot f_{cpu}^{max}$)
9.                 $st \leftarrow ft$
10.            ELSE /*$m_k$ is the receiver of $v_i^c$*/
11.                $st \leftarrow \max(f_{v_i^c,m_k}, s_{v_j,m_k})$
12.        ELSE IF $v_i$ is exit-task and $f_{v_i} < DL$
13.            SpeedAdjust($m_k$,$st$,$DL$,$\gamma \cdot f_{cpu}^{max}$)
```

IV. Simulation Results

The performance of the DCTMS algorithm with DVS adjustment is evaluated and compared with the DCA algorithm [10] [4]. To provide persuasive results, DCA is extended using our multihop communication model. It is also implemented with DVS adjustment using the same algorithm as DCTMS. We have run simulations to investigate the following aspects:

- Effect of the application deadline constraints
- Effect of the cluster size
- Effect of the number of tasks in applications
- Comparison with EbTA [7] in single-hop clusteres

In these simulations, the metrics are schedule length, energy consumption, and deadline missing ratio (DMR). DMR is defined as the ratio of the schedules that miss the deadlines.

A. Simulation Parameters

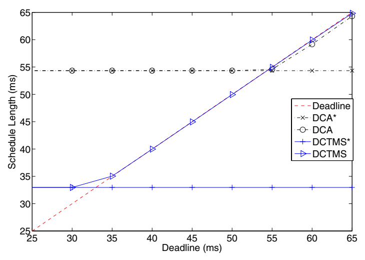
In our simulation study, the bandwidth of the channel is set to 1Mb/s and the transmission range r = 10 meters. Energy consumption model adopts Sensors equipped with the StrongARM SA-1100 microprocessor, whose speed ranges from 59 MHz to 206 MHz with 30 discrete levels. The parameters of Equation 1 - 4 are the same as in [10], [4], [9] as follows: E_{elec} = 50 nJ/b, ε_{amp} = 10 pJ/b/m^2, V_T = 26 mV, C = 0.67 nF, I_o = 1.196 mA, n = 21.26, K = 239.28 MHz/V and c = 0.5 V.

Simulations are run on randomly generated DAGs which are scheduled on randomly created multi-hop clusters. Random DAGs are created based on three parameters: The number of tasks *numTask*, the number of entry-tasks *numEntry*, and the maximum number of immediate predecessors *maxPred*. The number of immediate predecessors, the computation load (in units of kilo-clock-cycle, KCC), and the communication data volume (in units of bit) of a task are uniformly distributed over [1, *maxPred*], [300K CC ±10%], and [800 bits ±10%], respectively. The sensors are uniformly distributed in a disc area with radius of $k \cdot r$ and form a $k - hop$ connected cluster. We assume that there are $n = 5$ sensors in a single-hop cluster and $5k^2$ sensors in a $k - hop$ cluster. During simulations, the entry-tasks are randomly assigned to sensors. The simulation results are the average of 100 runs with different randomly (DAG, cluster) combinations.

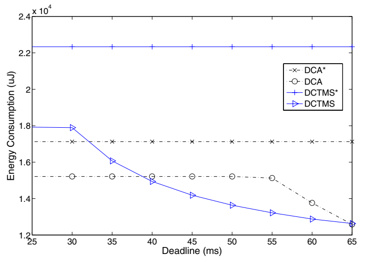
B. Effect of the Application Deadlines

We investigate the effect of application deadlines and DVS adjustment with randomly created 3-hop clusters and DAGs with *numTask* = 40, *numEntry* = 10, and *maxPred* = 10. To evaluate the effect of DVS, the schedule length, energy consumption and DMR of DCA and DCTMS before the voltage adjustment (denoted as DCA* and DCTMS*, respectively) are also investigated.

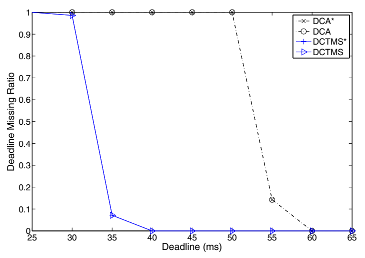
As shown in Fig. 2(a) and Fig. 2(c), DCTMS has a better capability to meet deadlines compared with DCA. When



(a) Schedule Length



(b) Energy Consumption

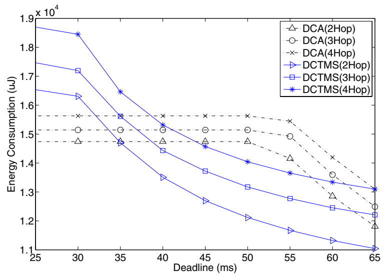


(c) Deadline Missing Ratio

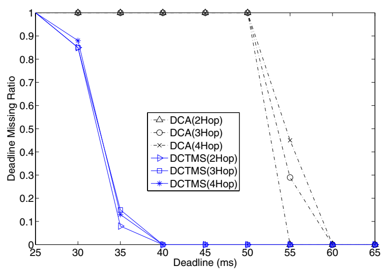
Fig. 2. Effect of Application Deadlines

deadlines are very small, even though DMR of DCTMS and DCA are both high, the average schedule length of DCTMS is much smaller and closer to deadlines compared with DCA. The reason is that DCA has only one sensor for high-level data processing while DCTMS can have more sensors involved in parallel, which speeds up execution.

Regarding energy consumption, DCA* has better energy consumption performance than DCTMS* for most scenarios according to Fig. 2(b). However, by implementing DVS algorithm, this energy consumption difference is significantly reduced. As shown in Fig. 2(b) and Fig. 2(a), DCTMS even outperforms DCA in terms of energy consumption by exploiting the much larger CPU slack time before deadlines for scenarios with large deadlines. Even when deadlines are relatively small and there is very little slack time before application deadlines, the DVS adjustment of DCTMS can still save about 22% energy compared with DCTMS*. This energy saving stems from exploiting the slack time caused by the unbalanced load of sensors and communication scheduling. Though the DVS adjustment may increase schedule lengths (Fig. 2(a)), the DMR is not affected (Fig. 2(c)) for any of the simulated deadline values.



(a) Energy Consumption



(b) Deadline Missing Ratio

Fig. 3. Effect of Cluster Size

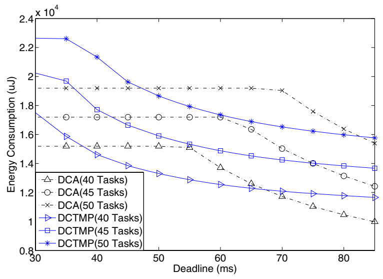
C. *Effect of the Cluster Size*

In this section, the effect of the cluster size is evaluated. In each simulation run, one random DAG with *numTask* = 40, *numEntry* = 10 *maxPred* = 10, and one set of 2-hop, 3-hop, and 4-hop random clusters are generated.

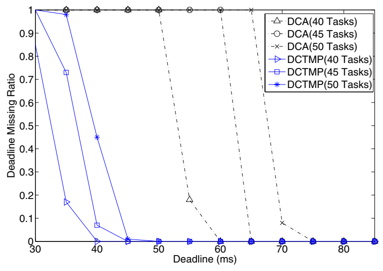
As shown in Figure 3, when the cluster size increases, the performance of DCA degrades correspondingly. Regarding DCTMS, the energy consumption proportionally increases with increasing cluster size. An interesting observation is that when the cluster size increases from 3-hop to 4-hop, the DMR slightly decreases around the deadline of 35 ms. This DMR improvement stems from better parallelism achieved with larger network size: There can be more communication scheduled simultaneously with larger network sizes, which may lead to more computation tasks executed in parallel. Thus, DCTMS has a better capacity to adapt to larger cluster sizes.

D. *Effect of the Number of Tasks*

Simulations are run on randomly generated DAGs with 40, 45, 50 tasks (*numEntry* = 10, *maxPred* = 10) to investigate the effect of number of tasks in applications, and each set of 40, 45, 50 task DAG are scheduled on one randomly created 3-hop cluster. According to the simulation results in Fig. 4(a), energy consumptions is dominated by the number of tasks. When the number of tasks increases, the energy consumption of DCA and DCTMS both increase proportionally, and DCTMS has higher energy consumption. However, when deadline is increasing, the energy consumption of DCTMS decrease faster than DCA by exploiting the available CPU slack time due to its better capacity to meet deadlines. Regarding DMR, DCA is dramatically affected with task volume increment while DCTMS is less affected (Fig. 4(b)). Thus, DCTMS has a better scalability compared with DCA regarding schedule length and DMR.



(a) Energy Consumption



(b) Deadline Missing Ratio

Fig. 4. Effect of Number of Tasks (40 tasks vs 45 tasks vs 50 tasks)

E. Comparison with EbTA

We compare the performance of DCTMS with EbTA for single-hop, single-channel clusters. Due to the small scale of a single-hop cluster (5 sensors as assumed), performances are evaluated with less computation load: The presented results are the average of 100 simulation runs of random DAGs with *numTask* = 25, *numEntry* = 5 and *maxPred* = 5. As shown in Fig. 5, DCTMS outperforms EbTA in terms of deadline guarantee and energy consumption. The superior performance of DCTMS mainly stems from the fact that DCTMS exploits the broadcast feature of the wireless channel when scheduling communication, while a task in EbTA must send information individually to its immediate successors.

V. Conclusion

In this paper, we propose an application-independent task mapping and scheduling solution for multi-hop WSNs - Dynamic Critical-path Task Mapping and Scheduling (DCTMP). DCTMP aims to map and schedule the tasks of an application with the minimum energy consumption subject to delay constraints. The multi-hop wireless channel is modeled as a virtual node to execute communication tasks, and a penalty function is proposed to avoid communication interference. Incorporating our communication scheduling algorithm, DCTMP schedules tasks with minimum energy consumption subject to deadline constraints. Simulations show significant performance improvements compared with DCA and EbTA in terms of minimizing energy consumption subject to delay constrains.

VI. Acknowledgment

The authors would like to thank Yang Yu for providing the simulator of EbTA [7].

References

[1] B. Zitova and J. Flusser, "Image registration methods: A survey," *Elsevier Image and Vision Computing*, vol. 21, no. 11, pp. 977–1000, Oct. 2003.

[2] M. Valera and S. A. Velastin, "Intelligent distributed surveillance systems: A review," *IEEE Proceedings on Vision, Image and Signal Processing*, vol. 152, no. 2, pp. 192–204, Apr. 2005.

[3] S. Giannecchini, M. Caccamo, and C.-S. Shih, "Collaborative resource allocation in wireless sensor networks," in *Proc. of Euromicro Conference on Real-Time Systems (ECRTS'04)*, June/July 2004, pp. 35–44.

[4] A. Wang and A. Chandrakasan, "Energy-efficient DSPs for wireless sensor networks," *IEEE Signal Processing Magazine*, pp. 68–78, July 2002.

[5] S. Shivle, R. Castain, H. J. Siegel, A. A. Maciejewski, T. Banka, K. Chindam, S. Dussinger, P. Pichumani, P. Satyasekaan, W. Saylor, D.Sendek, J. Sousa, J. Sridharan, P. Sugavanam, and J. Velazco, "Static mapping of subtasks in a heterogeneous ad hoc grid environment," in *Proc. of Parallel and Distributed Processing Symposium*, Apr. 2004.

[6] Y. Tian, E. Ekici, and F. Özgüner, "Energy-constrained task mapping and scheduling in wireless sensor networks," in *Proc. of the Workshop on Resource Provisioning and Management in Sensor Networks (RPMSN'05), in conjunction with MASS'05*, Nov. 2005.

[7] Y. Yu and V. K. Prasanna, "Energy-balanced task allocation for collaborative processing in wireless sensor networks," *ACM/Kluwer Jouranl of Mobile Networks and Applications*, vol. 10, no. 1-2, pp. 115–131, Feb. 2005.

[8] *CC2420 Data Sheet*. [Online]. Available: www.chipcon.com

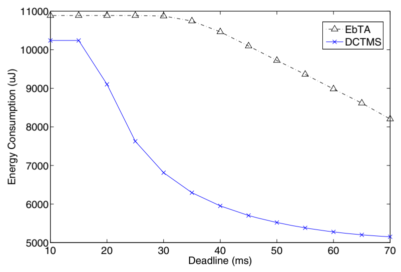
[9] W. B. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," *IEEE Transactions on Wireless Communications*, vol. 1, no. 4, pp. 660–670, Oct. 2002.

[10] E. Shih, S. Cho, N. Ickes, R. Min, A. Sinha, A. Wang, and A. Chandrakasan, "Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks," in *Proc. of ACM MobiCom'01*, July 2001, pp. 272–286.

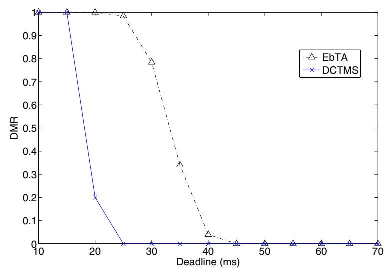
[11] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., 1979.

[12] Y.-K. Kwok and I. Ahmad, "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 5, pp. 506–521, May 1996.

[13] B. Karp and H. T. Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," in *Proc. of ACM MobiCom'00*, Aug. 2000, pp. 243–254.



(a) Energy Consumption



(b) Deadline Missing Ratio

Fig. 5. DCTMS vs EbTA