Available online at www.sciencedirect.com
ScienceDirect
Signal Processing: *Image Communication* 22 (2007) 237–251
SIGNAL PROCESSING: IMAGE COMMUNICATION
www.elsevier.com/locate/image

Real-time multimedia processing in video sensor networks

Yaoyao Gu[a,*], Yuan Tian[b], Eylem Ekici[c]

[a]*Texas Instrument, USA*
[b]*Bosch Research and Technology Center North America, USA*
[c]*Department of Electrical and Computer Engineering, The Ohio State University, USA*

Received 27 December 2006; accepted 28 December 2006

Abstract

Video sensor networks (VSNs) has become the recent research focus due to the rich information it provides to address various data-hungry applications. However, VSN implementations face stringent constraints of limited communication bandwidth, processing capability, and power supply. In-network processing has been proposed as efficient means to address these problems. The key component of in-network processing, task mapping and scheduling problem, is investigated in this paper. Although task mapping and scheduling in wired networks of processors has been extensively studied, their application to VSNs remains largely unexplored. Existing algorithms cannot be directly implemented in VSNs due to limited resource availability and shared wireless communication medium. In this work, an application-independent task mapping and scheduling solution in multi-hop VSNs is presented that provides real-time guarantees to process video feeds. The processed data is smaller in volume which further releases the burden on the end-to-end communication. 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.

Keywords: Video sensor networks; Real-time multimedia processing; Energy efficiency

1. Introduction

Recently, video sensor networks (VSNs) have attracted more and more research interest due to the rich information it offers which benefits numerous data-hungry applications [8,18,22]. A number of camera sensor nodes collaboratively capture visual information and send data through multi-hop communication to base stations. However, this transmission of high volume multimedia data stream challenges the limited bandwidth and energy supplies of VSNs. 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 VSN. Hence, it is plausible to process the data in-network, extract features of interests, and send the data with critical importance back to base stations.

*Corresponding author. Tel.: +1 714 398 9394; fax: +1 619 542 1222.
E-mail addresses: ygu@ti.com (Y. Gu), tiany@ece.osu.edu (Y. Tian), ekici@ece.osu.edu (E. Ekici).

0923-5965/$ - see front matter © 2007 Published by Elsevier B.V.
doi:10.1016/j.image.2006.12.013

The real-time feature of video streaming also necessitates in-network processing. The reduced communication volume reduces the delivery latency, and, hence, improves real-time performance. Many algorithms used for in-network processing require significant processing power. Applications such as image registration [22] and distributed visual surveillance [18] 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 task allocation problem must be solved which includes two prospects as follows:

- Assigning tasks on sensor nodes.
- Determining execution sequence of tasks on each sensor.

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 [1,3,5,11,12]. However, these existing solutions cannot directly be implemented in VSNs: 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 VSNs.

Task allocation problem has been investigated in large-scale WSNs, as well. 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 [6], 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 [19], 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.

Localized task mapping and task scheduling have been jointly considered for mobile computing [14] and for WSNs [16,17,21] recently. Task mapping and scheduling heuristics are presented in [14] for heterogeneous mobile ad hoc grid environments. However, the communication model adopted in [14] is not well suited for WSNs, which assumes individual channels for each node and concurrent data transmission and reception capacity of every node. In [17], the EcoMapS algorithm is proposed for energy-constrained applications in single-hop clustered WSNs to map and schedule communication and computation simultaneously. While EcoMapS does not provide execution deadline guarantees for applications, RT-Maps algorithm proposed in [16] addresses this problem from another perspective. It presents a real-time solution which minimizes the energy consumption subject to certain schedule deadline. In [21], energy-balanced task allocation (EbTA) is introduced to minimize balanced energy consumption subject to application deadline constraints. In [21], communications over multiple wireless channels are first modeled as additional linear constraints. However, the communication scheduling model in [21] does not exploit the broadcast nature of wireless communication, which can conserve energy and reduce schedule length. The algorithms proposed in [16] aim to meet application deadline constraints with minimum energy consumption. The broadcast nature of wireless communication is exploited in [16]. However, 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 VSN 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) [21], 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 dynamic voltage scaling (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 DVS is implemented in the *DVS phase* to reduce the energy consumption.

2. Preliminaries

2.1. Network assumptions

Our proposed task mapping and scheduling mechanism is designed for applications executed within a multi-hop cluster of VSNs. We assume the following VSN 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 $\mathrm{diam}(G) \leqslant k \cdot r$, 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 [2].
- Sensors are equipped with DVS processors [19] 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.

2.2. Application and energy consumption model

To have an application-independent solution, we represent applications executed in clusters with DAGs [21]. 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 its 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 that 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 pseudoexit-task with computation cost equals zero. Fig. 1(a) shows an example of a DAG.

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

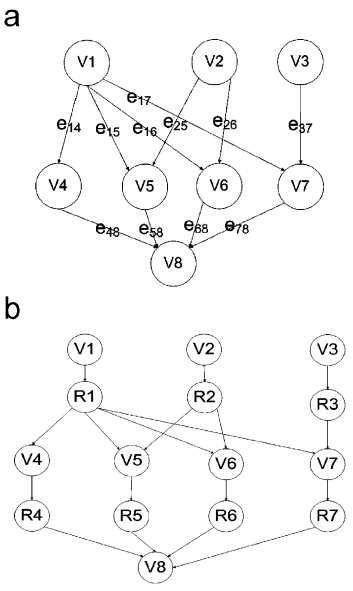

Fig. 1. DAG examples. (a) An example DAG, and (b) Hyper-DAG representation.

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_0 are defined as $E_{tx}(l,d)$ and $E_{rx}(l)$, respectively,

$$E_{tx}(l,d) = E_{elec} \cdot l + \varepsilon_{amp} \cdot l \cdot d^2, \tag{2.1}$$

$$E_{rx}(l) = E_{elec} \cdot l, \tag{2.2}$$

where E_{elec} and ε_{amp} are hardware related parameters [7,19].

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

$$E_{comp}(V_{dd},f) = NCV_{dd}^2 + V_{dd}(I_o e^{V_{dd}/nV_T})\left(\frac{N}{f}\right), \tag{2.3}$$

$$f \simeq K(V_{dd} - c), \tag{2.4}$$

where V_T is the thermal voltage and C, I_o, n, K and c are processor-dependent parameters [13,19].

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.

2.3. 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 $H^x = \{h_1^x, h_2^x, \ldots, h_n^x\}$ 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 $H^0 \in \{H^x\}$ that has the minimum energy consumption under the delay constraint:

$$\min \quad \text{energy}(H^0) = \sum_{i,k} c_{i,m_k} \tag{2.5}$$

$$\text{s.t.} \quad \text{length}(H^0) = \max_{i,k} f_{i,m_k} \leqslant \text{DL}, \tag{2.6}$$

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 [4]. 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 $[st, ft]$.

3. 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.

3.1. Hyper-DAG extension and multi-hop channel modeling

In VSNs, 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 reduces both 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 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, $V = \{v_i\} = \{\text{Computation Tasks}\}$, and $R = \{R_i\} = \{\text{Communication Task}\}$.

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 $\mathscr{C}$ on which only communication tasks can be executed. Different from the virtual node model in [6,17], 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 $\mathscr{C}$ 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 $\mathscr{C}$ during time interval $[st, ft]$ is defined as

$$P_{st}^{ft}(v) = \begin{cases} \infty & \text{if } \exists \gamma \in T_{st}^{ft}(\mathscr{C}) : S(\gamma) \in N(R(v)) \text{ or} \\ & \quad R(\gamma) \in N(S(v)), \\ 0 & \text{otherwise,} \end{cases} \tag{3.1}$$

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. Fig. 2 illustrates an example for the above penalty function. Given start time s_1 and finish time f_1 for a scheduled communication task r_1 on the virtual channel node, the penalty function for scheduling task r_2 starting anytime between s_1 and f_1 is $P_{s_1}^{f_1}(r_2) = \infty$, because node n_3 is n_2's one-hop neighbor. The data transmission from n_3 to n_4 will destroy the data received at node n_2. However, as shown in Fig. 2(b), task r_1 and r_3 can be scheduled simultaneously, since r_1 and r_3 do not have interference between each other. With the penalty function defined above, the multi-hop channel model is presented as follows:

- Wireless channel is modeled as a virtual node $\mathscr{C}$.
- $\mathscr{C}$ executes communication tasks only.
- There can be multiple tasks on $\mathscr{C}$ in time interval $[st, ft]$, denoted as $T_{st}^{ft}(\mathscr{C})$.
- The cost of executing communication task v_i on $\mathscr{C}$ in time interval $[st, ft]$ is $\text{cost}(v_i, st, ft) = st + P_{st}^{ft}(v_i)$.

With the Hyper-DAG representation and the channel model, the *communication dependency constraint* in Section 2.2 is rephrased as follows: in the

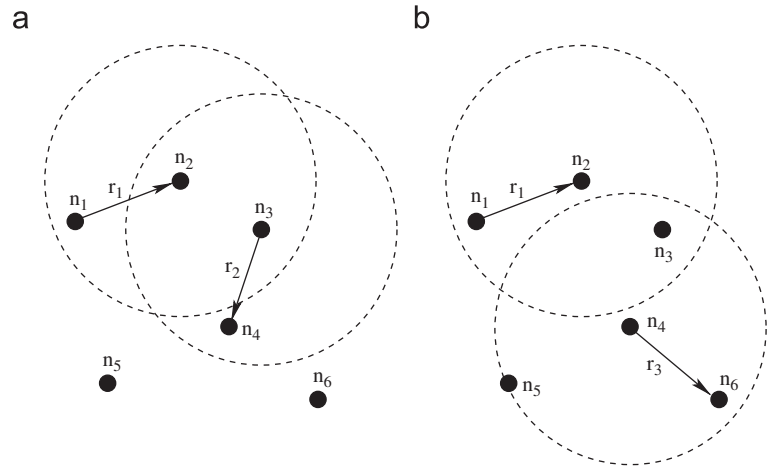

Fig. 2. Illustration of multi-hop communication modeling penalty function. (a) $P_{s_1}^{f_1}(r_2)=\infty$, where s_1 and f_1 are the start time and finish time for task r_1, and (b) $P_{s_1}^{f_1}(r_2)=0$. r_1 and r_3 can be scheduled simultaneously on the same multi-hop channel.

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 $\mathscr{C}$, 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.

3.2. 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 3.1, scheduling communication between single-hop neighbors is equivalent to first duplicating a communication task from the sender to $\mathscr{C}$, and then from $\mathscr{C}$ 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 $\mathscr{C}$ 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: $v_i^c \in T(\mathscr{C})$, $S(v_i^c) = m_s$
2. **IF** v_i^c does not exist
3. Find $v_i \in T(m_s)$
4. Find time interval $[st,ft]$:
5. $\text{cost}(v_i,st,ft) = \min$
6. $st \geqslant f_{v_i,m_s}$, $ft - st \geqslant t_{v_i,\mathscr{C}}$
7. Schedule a copy of v_i to $\mathscr{C}$:
8. $s_{v_i^c,\mathscr{C}} \leftarrow st, T(m_k) \leftarrow T(m_k) \cup \{v_i^c\}$
9. Schedule a copy of v_i^c to m_r:
10. $s_{v_i^k,m_k} \leftarrow f_{v_i^c,\mathscr{C}}$, $T(m_k) \leftarrow T(m_k) \cup \{v_i^k\}$
11. **ELSE**
12. $st \leftarrow s_{v_i^c,\mathscr{C}}$, $ft \leftarrow f_{v_i^c,\mathscr{C}}$
13. **IF** $\nexists \gamma \in T_{st}^{ft}(\mathscr{C}) : S(\gamma) \in N(m_r)$
14. Schedule a copy of v_i^c to m_r:
15. $s_{v_i^k,m_k} \leftarrow f_{v_i^c,\mathscr{C}}$, $T(m_k) \leftarrow T(m_k) \cup \{v_i^k\}$
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 Steps 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 [10] algorithm is used to obtain the $path = (m_1, \ldots, m_n)$ 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: $v_i^c \in T(\mathscr{C})$
2. Find the sensor node m_s: $v_i \in T(m_s)$
3. Calculate the path from m_s to m_r:path $= (m_1, \ldots, m_n)$
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: $v_i^0 \in T(\mathscr{C})$, dist($S(v_i^0)$, m_r) = min
10. Find $m_s \in N(S(v_i^0))$: dist(m_s, m_r) = min
11. IF v_i does not have a copy on m_s
12. OneHopCommTaskSchedule($v_i, S(v_i^0), m_s$)
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 3.3.

3.3. Scheduling with DCTMS algorithm

In the *task mapping and scheduling phase*, tasks of a Hyper-DAG are assigned to sensors nodes and $\mathscr{C}$. 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 $\mathscr{C}$. 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 $\text{pred}(v_i) \neq \emptyset$, then $\text{pred}(v_i) \subset T(m(v_i))$ and $s_{v_i,m(v_i)} \geqslant \max f_{\text{pred}(v_i),m(v_i)}$.

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 [4] 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 DCTMS algorithm composed by the following procedures:

- Dynamic critical-path evaluation and optimal task selection (DCOS).
- Optimal sensor searching and task assignment (OSTA).

The *DCOS* 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 *OSTA*. 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 the 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.

3.3.1. DCOS procedure

The core of the DCTMS scheduling algorithm is the *DCOS 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 VSNs. 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 $\mathrm{AVG}_{\mathrm{hop}}$ between active sensors, where $\mathrm{AVG}_{\mathrm{hop}}$ is dynamically updated in the *OSTA procedure*.

The *DCOS procedure* dynamically calculates critical-paths as follows: similar to the E-CNPT algorithm in [17], DCOS first iteratively calculates the earliest start time $\mathrm{EST}(v_i)$ 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

$$\mathrm{EST}(v_i) = \max_{v \in \mathrm{pred}(v_i)} \{\mathrm{EST}(v) + t_v\}, \tag{3.2}$$

$$t_v = \begin{cases} C_v / f_{\mathrm{CPU}}^{\max}, & v \in R, \\ \mathrm{AVG}_{\mathrm{hop}} \cdot R_v / \mathrm{BW}, & v \in R, \end{cases} \tag{3.3}$$

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

$$\mathrm{LST}(v_i) = \min_{v \in \mathrm{succ}(v_i)} \{\mathrm{LST}(v)\} - t_{v_i}, \tag{3.4}$$

where t_{v_i} has the same definition as t_v in Eq. (3.2).

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 OSTA 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 OSTA for further process.

3.3.2. OSTA procedure

In the OSTA procedure, the to-be-assigned task from DCOS will be scheduled on all active sensors, then the schedule with the minimum schedule length will be chosen. During the scheduling, sensors are scheduled with full speed $f_{\mathrm{cpu}}^{\max}$.

Input: Hyper-DAG; sensor set: SS
Output: Schedule of v^0
OSTA 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^0 with the DCOS procedure
5.    IF v^0 ∈ R
6.      Assign v^0 to m(pred(v^0))
7.    ELSE
8.      FOR all active sensors m_k
9.        IF pred(v^0) ⊈ T(m_k)
10.         FOR v_n ∈ pred(v^0) − T(m_k)
11.           CommTaskSchedule(v_n, m(v^0), m_k)
12.       Assign v^0 to m_k
13.     Keep the schedule with m^0: f_{v^0,m^0} = min
14.     Update AVG_hop if new active sensors involved
```

3.4. 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

$$\eta = e_{st}^{ft}/(ft - st), \tag{3.5}$$

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 further energy optimization (FEO) stage. In the SLE stage, the slack time between schedule length length(H) and application deadline DL is eliminated if any: assume $\beta = \text{length}(H)/\text{DL} < 1$, we define the re-scale factor as $\gamma = \lceil \beta \cdot f_{\text{cpu}}^{\max} \rceil / f_{\text{cpu}}^{\max}$. 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 schedule is extended in proportion to γ: all processors are slowed down to $\gamma \cdot f_{\text{cpu}}^{\max}$ with less energy consumption; computation tasks' start time, execution time, and finish time are multiplied by factor γ^{-1}. To satisfy dependency constraints, a communication task v_i finish time f_{v_i} is multiplied by factor γ^{-1}, while its execution time is unchanged and the start time $s_{v_i} = \gamma \cdot f_{v_i} - t_{v_i}$.

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, and is actually even larger. Thus, the adjusted schedule from the SLE stage needs to be further optimized in the FEO 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}^0 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. $\quad e_{st}^{ft} \leftarrow e_{st}^{ft} + t_{v_i,m_k}$
4. $\eta \leftarrow e_{st}^{ft}/(ft - st)$
5. $f_{\text{cpu}}^0 \leftarrow \lceil f_{\text{cpu}} \cdot \eta \rceil$
6. **FOR** $v_i \in T_{st}^{ft}(m_k)$ and $v_i \in V$
7. $\quad s_{v_i,m_k} \leftarrow tt$
8. $\quad t_{v_i,m_k} \leftarrow t_{v_i,m_k} \cdot \frac{f_{\text{cpu}}}{f_{\text{cpu}}^0}$, $f_{v_i,m_k} \leftarrow s_{v_i,m_k} + t_{v_i,m_k}$
9. $\quad tt \leftarrow f_{v_i,m_k}$
10. **FOR** $v_i \in T_{st}^{ft}(m_k)$ and $v_i \in R$
11. $\quad$ **IF** $\text{pred}(v_i) \in T(m_k)$
12. $\quad\quad s_{v_i,m_k} \leftarrow f_{\text{pred}(v_i),m_k}$, $f_{v_i,m_k} \leftarrow f_{\text{pred}(v_i),m_k}$
13. Update the energy consumption of m_k

In the FEO algorithm, the communication tasks on $\mathscr{C}$ 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 FEO 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^0
FEO Algorithm:

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

4. Simulation results

The performance of the DCTMS algorithm with DVS adjustment is evaluated and compared with the DCA algorithm [13,19]. To provide persuasive results, DCA is extended using our multi-hop communication model. It is also implemented with DVS adjustment using the same algorithm as DCTMS. We first run simulations on a real-life application model as a proof of concept. Then simulations are run on randomly generated DAGs 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 [21] in single-hop clusters.

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.

4.1. Simulation parameters

In our simulation study, the bandwidth of the channel is set to 1 MB/s and the transmission range $r = 10$ m. Energy consumption model adopts sensors equipped with the StrongARM SA-1100 microprocessor, whose speed ranges from 59 to 206 MHz with 30 discrete levels. The parameters of Eqs. (2.1)–(2.4) are the same as in [7,13,19] as follows: $E_{elec} = 50\,nJ/b$, $\varepsilon_{amp} = 10\,pJ/b/m^2$, $V_T = 26\,mV$, $C = 0.67\,nF$, $I_0 = 1.196\,mA$, $n = 21.26$, $K = 239.28\,MHz/V$ and $c = 0.5\,V$.

Simulations are first run on a real-life application model as a proof of concept. Then they 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 byte) of a task are uniformly distributed over [1, maxPred], [300 KCC ±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.

4.2. Simulation with a real-life application example

A practical visual data processing application in VSNs is considered in this section. Camera sensor nodes work collaboratively to monitor various objects in a given area. The conventional methods transmit all data through multi-hop communication to base stations and process information centrally. Information from neighboring sensor nodes is highly correlated, which implies that if data can be processed in the network exploiting correlations, smaller data volume will be sent through VSNs. Due to the limited communication bandwidth, it is plausible to extract features of interests and send the data with critical importance back to base stations in VSNs.

In a distributed visual object recognition scenario, neighboring sensor nodes pair up to exchange information for object recognition. The algorithm proposed in [15,20] for real-time object recognition is exploited. With this approach, features are extracted locally, followed by the voting operation [20]. The local feature selection procedure applies an edge detector to each input image, and extracts interest points. Then for each interest point, all captured features from different images are fused and votes are made according to interest points relative positions. Finally, features and their votes are aggregated at a single node in the cluster. After this collection, messages with object information are sent back to base stations. Fig. 3 shows the DAG representing this application. Tasks v_1–v_4 are entry-tasks which convert original images to binary images using edge detection and interest point detection [20]. Image size, hence, communicated information volume is significantly reduced here. Tasks v_5–v_8 extract features and fuse the image data from neighboring sensor pairs to improve the feature recognition ratio [22]. The object recognition in each image is done by "comparing the extraction of feature points and the interest points over edge detectors" [20]. The Hausdorff distances [9] are used as the criteria for image matching and voting. Object information from different cameras are fused to eliminate redundancy in v_9–v_{11}.

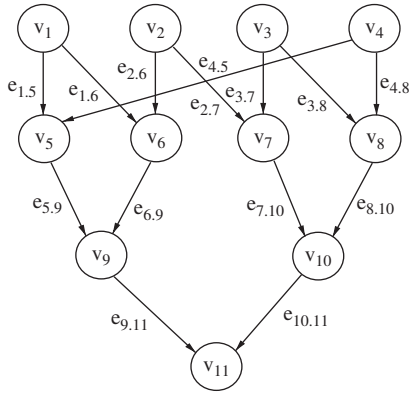

Fig. 3. DAG for distributed feature extraction application.

We assume that data size generated at each camera sensor is 128×128 bytes. We further assume the task computation load of v_1–v_4 to be 1000 KCC, the computation load of v_5–v_8 is 40 000 KCC, the computation load for matching in v_9–v_{11} is 1 KCC. We also assume the communication task for $E_{1,5}$–$E_{4,8}$ are 500 bytes, communication volumes of $E_{5,9}$–$E_{8,10}$ are 40 bytes. We compare the performance of our proposed DCTMP algorithm with the DCA and EbTA algorithm in Table 1. Since, EbTA is a scheduling algorithm for single-hop cluster only, these algorithms are evaluated in single-hop environment for fair comparison. The investigated metrics are the energy consumption and schedule length.

We compare the energy consumption and schedule length for all three algorithm under two different deadline conditions: deadline = 0.4 and 0.8 s. In both scenario, according to the simulation results, DCTMP schedule provides less energy consumption compared with DCA and EbTA to guarantee schedule deadlines. It should be noted that the simplified example is just for the proof of concept purposes. Real applications may have more complicated algorithms with larger image size and higher computation load. In the example above, sending these four 16 KB-images will consume about 51 mJ per hop. According to Table 1, the energy consumption of processing these images with deadline = 0.8 s is about 73 mJ. After the in-network processing, the resulting data volume is reduced to 40 bytes, which consumes only 0.032 mJ to delivery over one hop. Thus, the overall energy consumption of processing information and transmitting the results is saved compared with directly delivering original images over more two hops. This is satisfied in most large-scale VSNs, where energy savings through in-network processing is significant.

4.3. Simulation with randomly generated DAG

4.3.1. 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. 4(a) and (c), DCTMS has a better capability to meet deadlines compared with DCA. When 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. 4(b). However, by implementing DVS algorithm, this energy consumption difference is significantly reduced. As shown in Fig. 4(a) and (b), 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. 4(a)), the DMR is not affected (Fig. 4(c)) for any of the simulated deadline values.

Table 1
Simulation results for object recognition example

Deadline(s)	Metrics	DCA	EbTA	DCTMP
0.4	Energy consumption (mJ)	218.491	131.715	93.620
	Schedule length (s)	0.794	0.334	0.400
	Meet deadline	No	Yes	Yes
0.8	Energy consumption (mJ)	218.491	92.644	72.738
	Schedule length (s)	0.794	0.702	0.751
	Meet deadline	Yes	Yes	Yes

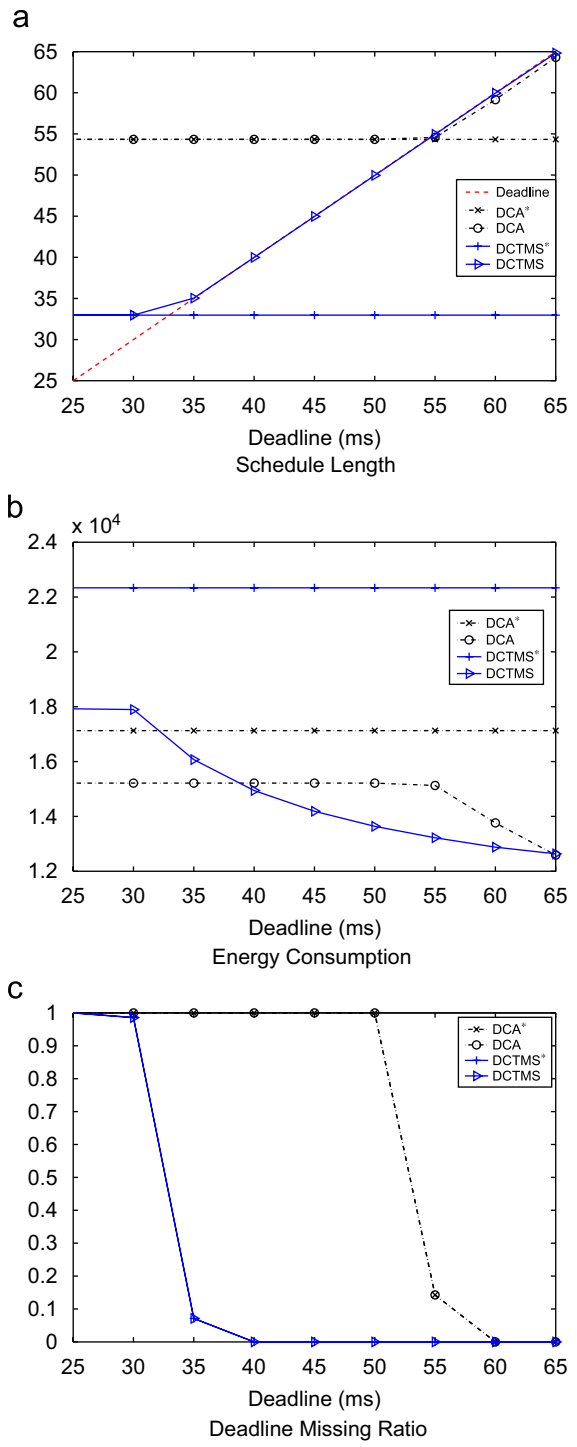

Fig. 4. Effect of application deadlines. (a) Schedule length, (b) energy consumption, (c) deadline missing ratio.

4.3.2. *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.

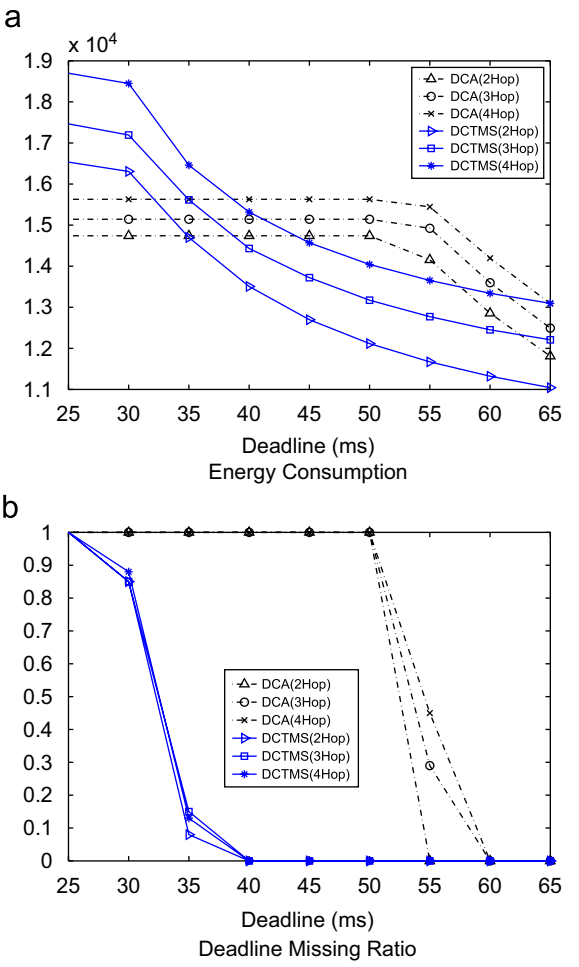

Fig. 5. Effect of cluster size. (a) Energy consumption and, (b) deadline missing ratio.

As shown in Fig. 5, 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.

4.3.3. 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. 6(a), energy consumptions are 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. 6(b)). Thus, DCTMS has a better scalability compared with DCA regarding schedule length and DMR.

4.3.4. 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 (five 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. 7, DCTMS outperforms EbTA in terms of deadline guarantee and energy consumption. The superior performance of DCTMS mainly stems from the fact that DCTMS exploits

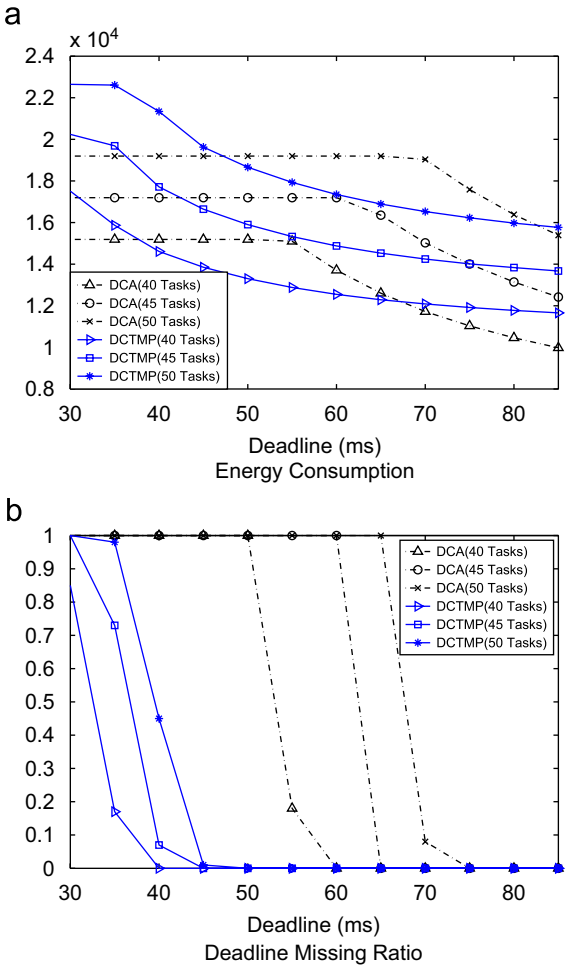

Fig. 6. Effect of number of tasks (40 tasks vs. 45 tasks vs. 50 tasks). (a) Energy consumption and, (b) deadline missing ratio.

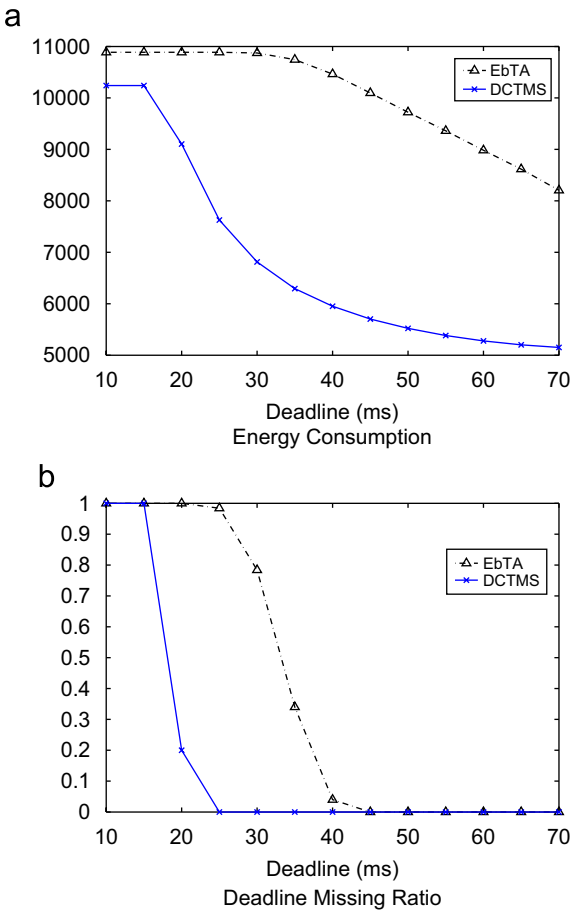

Fig. 7. DCTMS vs. EbTA. (a) Energy consumption and, (b) deadline missing ratio.

the broadcast feature of the wireless channel when scheduling communication, while a task in EbTA must send information individually to its immediate successors.

5. Conclusion

In this paper, we propose an energy-efficient real-time multimedia processing solution to enhance the in-network processing capability for multi-hop VSNs, called dynamic critical-path task mapping and scheduling (DCTMP) algorithm. 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.

Acknowledgement

The authors would like to thank Yang Yu for providing the simulator of EbTA [21].

References

[1] T.D. Braun, H.J. Siegel, N. Beck, L. Boloni, R.F. Freund, D. Hensgen, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, J. Parallel Distributed Comput. 61 (6) (June 2001) 810–837.

[2] CC2420 Data Sheet, 2006. Available: ⟨www.chipcon.com⟩ (Online).

[3] E. Fernadez, B. Bussell, Bounds on the number of processors and time for multiprocessor optimal schedules, IEEE Trans. Comput. 22 (8) (1973) 745–751.

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

[5] A. Gerasoulis, T. Yang, A comparison of clustering heuristics for scheduling dalz's on multiprocessors, J. Parallel Distributed Comput. 16 (4) (1992) 276–291.

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

[7] W.B. Heinzelman, A. Chandrakasan, H. Balakrishnan, An application-specific protocol architecture for wireless microsensor networks, IEEE Trans. Wireless Commun. 1 (4) (October 2002) 660–670.

[8] E. Horster, R. Lienhart, W. Kellermann, J. Bouguet, Calibration of visual sensors and actuators in distributed computing platforms, in: Proceedings of the Third ACM International Workshop on Video Surveillance and Sensor Networks (VSSN'05), 2005.

[9] D. Huttenlocher, G. Klanderman, W. Rucklidge, Comparing images using the hausdorff distance, IEEE Trans. Pattern Anal. Machine Intell. PAMI-15 (1993) 850–863.

[10] B. Karp, H.T. Kung, GPSR: greedy perimeter stateless routing for wireless networks, in: Proceedings of ACM MobiCom'00, August 2000, pp. 243–254.

[11] H. Kasahara, S. Narita, Practical multiprocessor scheduling algorithms for efficient parallel processing, IEEE Trans. Comput. 33 (11) (1984) 1023–1029.

[12] Y.-K. Kwok, I. Ahmad, Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors, IEEE Trans. Parallel Distributed Syst. 7 (5) (May 1996) 506–521.

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

[14] 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, J. Velazco, Static mapping of subtasks in a heterogeneous ad hoc grid environment, in: Proceedings of Parallel and Distributed Processing Symposium, April 2004.

[15] Y. Tatsuya, K. Masataka, I. Katsushi, Local-feature based vehicle recognition system using parallel vision board, in: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, 1999.

[16] Y. Tian, J. Boangoat, E. Ekici, F. Özgüner, Real-time task mapping and scheduling for collaborative in-network processing in DVS-enabled wireless sensor networks, in: Proceedings of Parallel and Distributed Processing Symposium (IPDPS'06), 2006.

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

[18] M. Valera, S.A. Velastin, Intelligent distributed surveillance systems: a review, IEEE Proc. Vision Image Signal Process. 152 (2) (April 2005) 192–204.

[19] A. Wang, A. Chandrakasan, Energy-efficient DSPs for wireless sensor networks, IEEE Signal Process. Mag. (July 2002) 68–78.

[20] J. You, P. Bhattaharya, S. Hungenahally, Real-time object recognition: hierarchical image matching in a parallel virtual machine environment, in: Proceedings of 14th International Conference on Pattern Recognition, vol. 1, 1998, pp. 275–277.

[21] Y. Yu, V.K. Prasanna, Energy-balanced task allocation for collaborative processing in wireless sensor networks, ACM/Kluwer J. Mobile Networks Appl. 10 (1–2) (2005) 115–131.

[22] B. Zitova, J. Flusser, Image registration methods: a survey, Elsevier Image Vision Comput. 21 (11) (Oct. 2003) 977–1000.