Cross-Layer Collaborative In-Network Processing in Multihop Wireless Sensor Networks

Yuan Tian and Eylem Ekici, *Member*, *IEEE*

Abstract—Emerging Wireless Sensor Network (WSN) applications demand considerable computation capacity for in-network processing. To achieve the required processing capacity, cross-layer collaborative in-network processing among sensors emerges as a promising solution: Sensors do not only process information at the application layer, but also synchronize their communication activities to exchange partially processed data for parallel processing. However, scheduling computation and communication events is a challenging problem in WSNs due to limited resource availability and shared communication medium. In this work, an application-independent task mapping and scheduling solution in multihop homogeneous WSNs, *Multihop Task Mapping and Scheduling (MTMS)*, is presented that provides real-time guarantees. Using our proposed application model, the multihop channel model, and the communication scheduling algorithm, computation tasks and associated communication events are scheduled simultaneously. The Dynamic Voltage Scaling (DVS) algorithm is presented to further optimize energy consumption. Simulation results show significant performance improvements compared with existing mechanisms in terms of minimizing energy consumption subject to delay constraints.

Index Terms—Wireless sensor network, multihop, cross-layer, in-network processing, task mapping and scheduling.

1 Introduction

Many emerging applications for WSNs are associated with real-time requirements that necessitate in-network processing. As an example, mission-critical target detection and tracking applications [1], [2] impose time limits on the delivery of results. Processing information locally and sending the end results to a central location is generally more energy-efficient than sending raw data across a multihop WSN. The reduced communication volume also reduces the delivery latency and, hence, improves real-time performance.

Collaborative in-network processing is a viable solution to provide the required processing power not available in stand-alone sensor nodes. Collaborative in-network processing partitions applications into smaller tasks executed in parallel on different sensor nodes. Dependencies between tasks are maintained through the exchange of intermediate results between sensor nodes. Hence, collaborative in-network processing methods are inherently cross-layer solutions that involve coordination of computation as well as communication events. Though local communication overhead is introduced during in-network processing, the resulting volume is significantly smaller than the raw data. Thus, the communication load and energy consumption are reduced for long distance communication over multiple hops.

The benefits of in-network processing are especially pronounced in wireless camera sensor networks (WCSN) [3], [4]. In WCSNs, applications such as image registration [5] and distributed visual surveillance [6] involve computationally intensive operations. The inherent real-time requirements of multimedia applications further exacerbate this problem. For instance, distributed intelligence and information processing for multiple-camera surveillance are the primary methods used in Third Generation Surveillance Systems (3GSS), where a large number of cameras are connected with networks [6]. A real-time multicamera surveillance system is presented in [7], where multiple cameras collaboratively detect and classify objects. As part of the system requirements, vision-based localization is implemented after an object is detected and recognized [7]. Data fusion is applied in these operations to integrate data from different cameras. Vision-based localization is a natural solution in 3GSS to recover objects' 3D information, albeit with intensive processing, where images are periodically captured. Most 3GSS research proposals assume local area network (LAN) connections. However, as envisioned in [3], [4], camera networks equipped with wireless connections are promising due to the ease of deployment and the flexibility of topology adjustment. A simple visual surveillance example is shown in Fig. 1, where four calibrated wireless camera sensors collaboratively detect an object's location. Instead of sending original images, the object's location is calculated and delivered to the base station: Sensors first estimate the object's location by themselves, then fuse the intermediate results to further eliminate estimation errors. As such, the communicated data volume is reduced by several orders of magnitude.

To enable collaborative in-network processing, the following problems must be solved:

- the assignment of tasks to sensors,
- determining the execution sequence of tasks, and
- scheduling communication between sensors.

- *The authors are with the Department of Electrical and Computer Engineering, The Ohio State University, 2015 Neil Ave., Columbus, OH 43202. E-mail: {tiany, ekici}@ece.osu.edu.*

Manuscript received 25 Apr. 2006; revised 10 July 2006; accepted 4 Aug. 2006; published online 16 Jan. 2007.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-0117-0406.

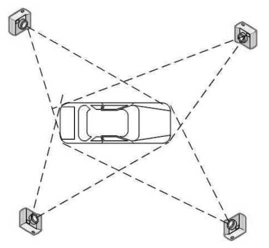


Fig. 1. A simplified distributed video surveillance example.

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 [8], [9], [10]. However, these existing solutions cannot directly be implemented in WSNs as the wireless communication scheduling is not addressed. Furthermore, these solutions do not explicitly consider energy consumption during communication and task execution, which is one of the major constraints in WSNs. Though task mapping and scheduling have recently been discussed in the literature for WSNs [11], [12], [13], [14], [15], [16], they all consider single-hop clustered networks without addressing communication scheduling in multihop networks, which hinders their generalization to arbitrary WSN scenarios. Clustering sensors into single-hop clusters can lead to a large number of clusters and, consequently, comes with the cost of greater communication and routing overhead in large-scale WSNs [17]. Thus, multihop clustering and, consequently, collaborative in-network processing solutions in multihop clusters are preferable in large-scale WSNs.

In this work, we propose an *application-independent* solution, *Multihop Task Mapping and Scheduling (MTMS)*, to provide the in-network computation capacity required by arbitrary real-time applications in multihop WSNs. MTMS aims to guarantee application deadlines with minimum energy consumption. MTMS not only maps and schedules *computation tasks* to sensors in parallel to accelerate execution, but also addresses *communication scheduling* among sensors to exchange intermediate results in a *multihop* cluster of sensor nodes. To the best of our knowledge, this is the first work that addresses the joint scheduling of communication and computation in multihop WSNs. A novel high-level application model, Hyper-DAG, is proposed to abstract arbitrary applications. As a cross-layer solution, MTMS schedules computation tasks at the application layer as well as their associated communication at Medium Access Control (MAC) and network layer. For this purpose, a novel communication model is presented to abstract multihop wireless channels. Based on this channel model, the multihop communication scheduling algorithm is integrated as a part of MTMS with the collision avoidance feature. The resulting start and finish times of communication events constitute the schedule used by the MAC. MTMS has two phases: *task mapping and scheduling phase* and *DVS phase*. In the *task mapping and scheduling phase*, computation tasks are scheduled at the highest processing power to find a feasible solution. A Dynamic Voltage Scaling (DVS) algorithm is presented to further reduce the energy consumption.

2 Related Work

Localized task mapping and scheduling problems in WSNs have been studied in the literature recently. These solutions consider applications executed independently within clusters composed of geographically close nodes, following locally generated schedules. In [11], an online task scheduling mechanism (CoRAl) is proposed to allocate the network resources between the tasks of periodic applications in WSNs in an iterative manner: The upper-bound frequencies of applications are first evaluated according to the bandwidth and communication requirements between sensors. The frequencies of the tasks on each sensor are then optimized subject to the upper-bound execution frequencies. However, CoRAl assumes that the tasks are already assigned to sensors without addressing the task mapping problem. Furthermore, energy consumption is not explicitly discussed in [11]. Different from CoRAl, our proposed MTMS solution addresses task mapping and task scheduling simultaneously. In addition, energy consumption is explicitly considered in the objective function of MTMS.

Distributed Computing Architecture (DCA) is presented in [12], 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 the cluster heads' computation power. Furthermore, the application-specific design of DCA limits its implementation for generic applications. On the other hand, our application-independent MTMS solution can distribute the computation load among multiple sensors, which provides higher computation capacity.

Localized task mapping and task scheduling have been jointly considered for mobile computing [18] and for WSNs [14], [15], [16] recently. Task mapping and scheduling heuristics are presented in [18] for heterogeneous mobile ad hoc grid environments. However, the communication model adopted in [18] is not well-suited for WSNs, which assumes individual channels for each node and the concurrent data transmission and reception ability of every node. MTMS, on the other hand, is specifically developed for WSNs adopting a more realistic communication model. In [15], the EcoMapS algorithm is proposed for energy-constrained applications in single-hop clustered WSNs to map and schedule communication and computation simultaneously. EcoMapS aims to schedule tasks with the minimum schedule length subject to energy consumption constraints. However, EcoMapS does not provide execution deadline guarantees for applications, which is addressed in MTMS. In [14], Energy-balanced Task Allocation (EbTA) is introduced to minimize balanced energy consumption subject to application deadline constraints. In [14], communications over multiple wireless channels are first modeled as additional linear constraints of an Integer Linear Programming (ILP) problem. Then, a heuristic algorithm is presented to provide a practical solution. However, the communication scheduling model in [14] does not exploit the broadcast nature of wireless communication, which can conserve energy and reduce schedule length. RT-MapS is presented in [16] to provide a deadline guarantee with minimum application energy consumption. The broadcast nature of wireless communication is utilized in RT-MapS to leverage energy consumption. In MTMS, broadcast and multicast are realized through the presented communication scheduling algorithm and the task mapping and scheduling algorithms. Furthermore, all localized mechanisms, namely, CoRAl, DCA, EcoMapS, EbTA, and RT-MapS, assume a single-hop cluster environment, which prevents their application to general implementations. Different from these existing works, MTMS provides a more general solution for multihop clustered WSNs.

In summary, MTMS is a generic task mapping and scheduling solution for multihop wireless sensor networks with the following salient properties different from the existing work:

- The multihop wireless channel is modeled to reflect the broadcast nature of wireless communication.
- Based on the channel model, multihop communication events and computation tasks are jointly scheduled.
- Task mapping and task scheduling are considered simultaneously.
- Based on realistic energy models for computation and communication in WSNs, MTMS aims to guarantee application deadline constraints with minimum energy consumption to prolong the network lifetime.

3 Preliminaries

3.1 Network Assumptions

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

- Homogeneous sensors are grouped into k-hop clusters. In this paper, we define a k-hop network as a connected network G with diameter $diam(G) \leq k$, where k is the hop count of the longest path connecting any two nodes.
- Each cluster executes an application which is either assigned during the network setup time or remotely distributed by base stations during the network operation. Once assigned, applications are independently executed within each cluster unless new applications arrive. 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.
- Computation and communication can occur simultaneously on sensor nodes as supported by various platforms including MICA2DOT running TinyOS.
- Communication within a cluster is isolated from other clusters through time division or channel-hopping mechanisms with appropriate hardware support such as Chipcon CC2420.
- Sensors are equipped with DVS processors such as StrongARM SA-1100 [19]. The CPU speed can be changed by dynamically adjusting the supply voltage with a finite number of levels. The delay of speed and voltage adjustment for a DVS processor can be on the order of 10-100 μs [20] [21]. Such DVS adjustment overhead can be accounted for in the adjusted task execution time. For the sake of simplicity, we assume the DVS adjustment overhead to be negligible.

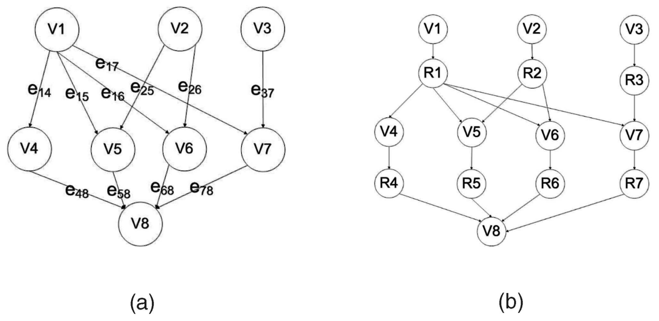


Fig. 2. DAG examples. (a) An example DAG. (b) The Hyper-DAG representation.

It should be noted that, while the intracluster communications are isolated from each other, 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 intracluster communication isolation.

3.2 Application Model

Applications can be represented by Directed Acyclic Graph (DAG) to have an application-independent solution [14]. 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*. 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 equal to zero. Fig. 2a shows an example of a DAG, where $V1$, $V2$, and $V3$ are entry-tasks, $V8$ is an exit-task, and $V5$ is the immediate successor and immediate predecessor of $V1$ and $V8$, respectively.

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 the *Entry-task Assignment Constraint*.

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 the same node, the result delivery latency is considered to be zero and v_j can start to execute after v_i is finished. This

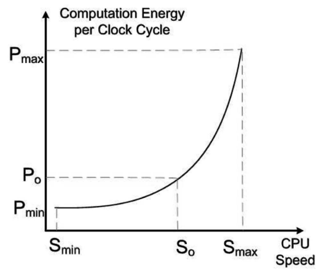


Fig. 3. CPU power consumption versus CPU speed.

execution dependency between tasks is referred to as the *Communication Dependency Constraint*.

3.3 Energy Consumption Model and Dynamic Voltage Scaling

The energy consumption 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:

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

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

where E_{elec} and ε_{amp} are hardware-related parameters [12]. In our communication scheduling algorithm, the energy consumption of the sender and receivers of a packet will be updated according to (1) and (2).

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

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

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

where V_T is the thermal voltage and C, I_o, n, K, and c are processor-dependent parameters [19], [12].

Due to the discrete nature of task mapping and scheduling, a schedule that meets a deadline may do so with slack time until the deadline. The unbalanced load of sensors and communication scheduling also results in CPU idle time. DVS is a technique to exploit the CPU idle time by jointly decreasing CPU speed and supply voltage while still meeting deadlines. According to (3) and (4), a decrease in CPU supply voltage leads to an approximately linear increase in execution time and an approximately quadratic decrease in computation energy consumption. The relationship of the CPU speed and the unit energy consumption is approximately shown in Fig. 3. Assume that the execution load of a task v_i is N clock cycles and its deadline is t. Given the CPU speed of S_{max}, v_i can be finished at time t' with the CPU slack time of $t - t'$ before the deadline. The corresponding computational energy consumption is $N \cdot P_{max}$. By reducing the CPU speed to S_o, v_i's execution time is increased to t. On the other hand, the computational energy consumption decreases to $N \cdot P_o$, which leads to an energy savings of $N \cdot (P_{max} - P_o)$. Here, P_{max} and P_o stand for the original and adjusted computation energy consumption per clock cycle of v_i with CPU speeds of S_{max} and S_o, respectively.

It should be noted that the energy consumption model presented above only considers the energy expenditure directly related with application execution. Thus, energy consumption during idle time is not taken into account. However, our communication and computation schedules may also be used to determine the sleep schedules of sensors, where sensors go to sleep when no communication and computation activities are scheduled for them.

3.4 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 $(v_i, m_k, s_{v_i,m_k}, f_{v_i,m_k}, t_{v_i,m_k}, c_{v_i,m_k})$, where m_k represents the node to which a computation task v_i is assigned, s_{v_i,m_k}, f_{v_i,m_k}, and t_{v_i,m_k} represent the start time, finish time, and execution length of v_i, and c_{v_i,m_k} represents the energy consumption of v_i on node m_k, respectively. It should be noted that task start times, finish times, and application deadlines are relative to the application execution start time, which is the reference time $t_o = 0$.

Let $CommEng(m_k)$ represent the communication energy consumption of a node m_k including data transmission, reception, and forwarding. The design objective of MTMS is to find a schedule $H^o \in \{H^x\}$ that has the minimum energy consumption under the delay constraint:

$$\text{Find } H^o = arg\min energy(H), \tag{5}$$

$$\text{where } energy(H) = \sum_{i,k} c_{v_i,m_k} + \sum_k CommEng(m_k), \tag{6}$$

$$\text{subject to } length(H) = \max_{i,k} f_{v_i,m_k} \leq DL, \tag{7}$$

where $energy(H)$ and $length(H)$ are the overall energy consumption and the schedule length of H, respectively, and DL is the deadline of the application. The DAG scheduling problem is shown to be an NP-complete problem in general [22]. 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, and
- $T_{st}^{ft}(m_k)$ denotes the tasks assigned on node m_k during the time interval $[st, ft]$.

4 The Proposed MTMS Algorithm

The proposed MTMS 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,

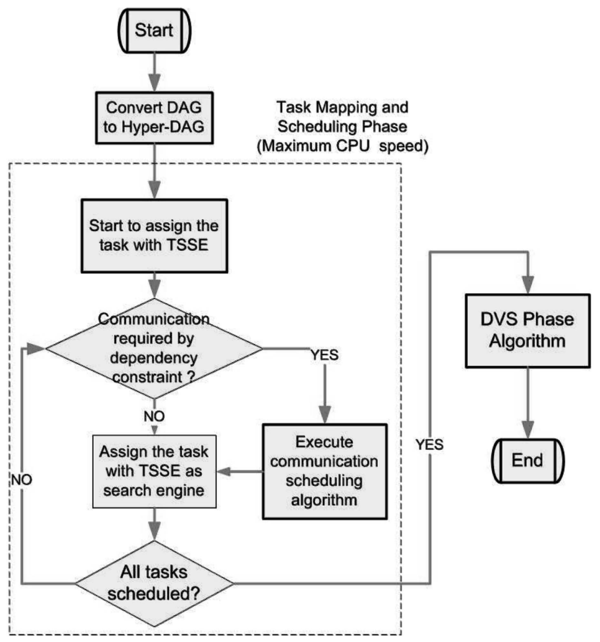

Start
Convert DAG to Hyper-DAG
Task Mapping and Scheduling Phase (Maximum CPU speed)
Start to assign the task with TSSE
Communication required by dependency constraint ?
YES
NO
Assign the task with TSSE as search engine
Execute communication scheduling algorithm
NO
All tasks scheduled?
YES
DVS Phase Algorithm
End

Fig. 4. The flowchart of MTMS.

and communications between sensors are scheduled based on the *Communication Dependency Constraints*. We extend the Min-Min algorithm [18] as the multihop *Task Schedule Search Engine (TSSE)* in this phase. The Min-Min algorithm delivers satisfactory performance with relatively low complexity [10], [18]. The objective of the TSSE algorithm is to minimize energy consumption subject to deadline constraints. To guarantee deadlines, sensors are scheduled with the maximum CPU speed f_{cpu}^{max} by the TSSE algorithm. Then, schedules created in the first phase are further optimized in the *DVS phase* by reducing CPU speeds to exploit CPU slack times.

The MTMS algorithm is executed on cluster heads when applications are assigned to clusters. In case of a loss of a cluster head, a new cluster head is selected via the clustering algorithm in use and schedules will be regenerated by the new cluster head.

The original Min-Min algorithm is designed for traditional parallel processing without considering wireless communication scheduling. To solve this problem, we developed a new *multihop communication scheduling algorithm* based on our proposed Hyper-DAG representation of tasks and multihop channel model. The communication scheduling algorithm is utilized by the TSSE algorithm during task scheduling to satisfy the *Communication Dependency Constraints*. Schedules generated by the modified Min-Min algorithm are further optimized with the DVS algorithm. The flowchart of the MTMS solution is shown in Fig. 4. In the following sections, the main components of our proposed MTMS algorithm, namely, Hyper-DAG extension and multihop channel modeling, the communication scheduling algorithm, the TSSE algorithm, and the DVS algorithm, are presented.

4.1 Hyper-DAG Extension and Multihop 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 reduces both the execution time and 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 the resulting 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 *Hyper-DAG*. The Hyper-DAG representation of the DAG in Fig. 2a is shown in Fig. 2b. 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. Based on the Hyper-DAG representation, (6) is rephrased as follows:

$$energy(H) = \sum_{v_i \in V,k} c_{v_i,m_k} + \sum_{v_i \in R,k} c_{v_i,m_k} = \sum_{v_i \in V',k} c_{v_i,m_k}, \quad (8)$$

where c_{v_i,m_k} of $v_i \in R$ is the energy consumption of node m_k for sending or receiving communication task v_i through the wireless channel.

To properly schedule communication events, we model the multihop channel as a virtual node $\mathcal{C}$ on which only communication tasks can be executed. Different from the virtual node model in [11], [15], where only single-hop channels are considered, our channel model takes potential interference between simultaneous communications in multihop networks into consideration.

Unlike in single-hop networks, there can be multiple simultaneous communications in multihop networks. Thus, the virtual node $\mathcal{C}$ in the multihop 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 the 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:

$$P_{st}^{ft}(v) = \begin{cases} \infty, if\ \exists\gamma \in T_{st}^{ft}(\mathcal{C}) : S(\gamma) \in N(R(v))\ or\ R(\gamma) \in N(S(v)) \\ 0,\ \ otherwise, \end{cases} \quad (9)$$

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 multihop channel model is presented as follows:

- The wireless channel of a cluster is modeled as a virtual node $\mathcal{C}$. All cluster members are considered to be directly connected with $\mathcal{C}$.
- The channel node $\mathcal{C}$ executes communication tasks only. All communication tasks exchanged between sensor nodes must be routed through $\mathcal{C}$. The available time, start time, execution time, and finish time of a communication task v_i scheduled on $\mathcal{C}$ are represented by $at_{v_i,\mathcal{C}}$, $s_{v_i,\mathcal{C}}$, $t_{v_i,\mathcal{C}}$, and $f_{v_i,\mathcal{C}}$, respectively.
- A communication task assigned on $\mathcal{C}$ stands for an ongoing data communication. Its execution time on $\mathcal{C}$ equals its communication length via the wireless channel. The corresponding data transmission and reception energy consumption are accounted for by the sender and receivers following (1) and (2), respectively.
- There can be multiple communication tasks scheduled on $\mathcal{C}$ in time interval $[st, ft]$, which are denoted as $T_{st}^{ft}(\mathcal{C})$.
- The cost of executing communication task v_i on $\mathcal{C}$ in time interval $[st, ft]$ is

  $$cost(v_i, st, ft) = P_{st}^{ft}(v_i) + g(st - at_{v_i,\mathcal{C}}),$$

  where $g(x)$, $x \geq 0$, is a monotonically increasing function. The penalty function $P_{st}^{ft}(v_i)$ represents the scheduling feasibility in $[st, ft]$. If a schedule causes interference, the cost function becomes infinite since $P_{st}^{ft}(v_i) = \infty$. For such scenarios, the communication scheduling algorithms search for another time interval to avoid packet collisions. Otherwise, the cost function is determined by $g(st - at_{v_i,\mathcal{C}})$ as $P_{st}^{ft}(v_i) = 0$. Since $st - at_{v_i,\mathcal{C}}$ denotes the delay between v_i's available time and scheduled start time, minimizing $g(st - at_{v_i,\mathcal{C}})$ leads to the selection of the earliest feasible execution of v_i on $\mathcal{C}$.

With the Hyper-DAG representation and channel model, the *Communication Dependency Constraint* in Section 3.2 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 available on the same node.

It should be noted that the penalty function presented here just takes communication interference into account. However, the penalty function can be further extended with factors such as link quality. We will defer the discussion of alternative penalty functions to our future work.

4.2 Communication Scheduling Algorithms

To meet the *Communication Dependency Constraint* in Hyper-DAG scheduling, communication scheduling between nodes is required if a computation task depends on a communication task assigned on another node. The communication scheduling algorithm presented in this section is used in conjunction with the task mapping and scheduling algorithm described in Section 4.3.

In multihop clusters, the sender and the receivers of a communication task can be one or more hops away from each other. We schedule multihop 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 multihop channel models presented in Section 4.1, unicasting communication task v_i from sensor m_s to its single-hop neighbor m_r through the wireless channel can be modeled as follows: v_i is first duplicated from m_s to $\mathcal{C}$, which stands for originating the data transmission. The duplicated copy v_i^c is then executed on $\mathcal{C}$ for the duration of the communication length, which represents the data transmission. After v_i^c is finished on $\mathcal{C}$, v_i^c is duplicated to m_r from $\mathcal{C}$, which represents the end of the data transmission. After v_i^c is duplicated to m_r, the transmitted data is available to computation tasks assigned to m_r. Any given transmission can potentially reach multiple receivers if they do not interfere with neighboring communications. From the perspective of task scheduling, broadcasting is similar to unicast communication, except that v_i^c will be duplicated to multiple receivers after it is finished on $\mathcal{C}$. Broadcasting may lead to significant energy savings compared with multiple unicasts between the sender and receivers. Thus, in communication scheduling, we always consider the possibility of receiving broadcast data first. The detailed description of the single-hop communication scheduling algorithm is presented in Fig. 5.

In Fig. 5, Steps 6-23 stand for originating a new communication from m_s to m_r. If a communication task has multiple immediate successors to be scheduled on different sensors, multiple receptions of the broadcast data without interference can be scheduled in Steps 28-36. Compared with originating a new communication for each recipient, the broadcast reception method leads to an energy savings of one data transmission for each additional data reception.

In our multihop communication scheduling algorithm, a routing 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 $m_n = m_r$. In this paper, we employ the low complexity stateless geographic routing algorithm, GPSR [23]. After obtaining the path, the communication task will be iteratively duplicated from the source to the destination following the $OneHopSchedule()$ algorithm.

Similar to that of the one-hop communication scheduling, a communication task may be requested by several destinations that are multiple hops away. Thus, multicasting is desirable to shorten communication latencies as well as to decrease energy consumption. The first time a communication task v_i is requested from m_s to m_r, unicast path is formed from the source to the destination, which is a distribution tree with no branches. In the subsequent scheduling steps, each time v_i is requested by another sensor m_k, the distribution tree branches and expands to m_k by connecting m_k with the nearest node on the existing tree. The detailed description of the multihop communication scheduling is presented in Fig. 6.

In our communication scheduling algorithm, collision avoidance is achieved by implementing the penalty function. In WSNs, packet losses may also occur due to channel conditions. Such packet losses can be handled by retransmitting the erroneous packets. As packet retransmission

```
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 $m_r$
OneHopSchedule($v_i$,$m_s$,$m_r$):
________________________________________
1. /*No need to communicate if $v_i$ already on $m_r$*/
2. IF $v_i \in T(m_r)$
3.     Return;
4. /*$m_s$ sent $v_i$ before?*/
5. Find a copy of $v_i$: $v_i^c \in T(\mathcal{C})$, $S(v_i^c) = m_s$
6. /*No, unicast scheduling from scratch*/
7. IF $v_i^c$ does not exist
8.     Find $v_i \in T(m_s)$
9.     Find time interval [st,ft]:
10.        /*Find interval with minimum cost*/
11.        $cost(v_i, st, ft) = \min$
12.        /*Make sure $v_i$ can be executed on $\mathcal{C}$
13.        $st \geq f_{v_i,m_s}$, $ft - st = t_{v_i,\mathcal{C}}$
14.    Schedule a copy of $v_i$ to $\mathcal{C}$:
15.        $s_{v_i^c,\mathcal{C}} \leftarrow st$, $f_{v_i^c,\mathcal{C}} \leftarrow ft$
16.        $T(\mathcal{C}) \leftarrow T(\mathcal{C}) \cup \{v_i^c\}$
18.        Update the energy consumption of $m_s$
19.    Schedule a copy of $v_i^c$ to $m_r$:
20.        $s_{v_i^r,m_r} \leftarrow f_{v_i^c,\mathcal{C}}$, $f_{v_i^r,m_r} \leftarrow f_{v_i^c,\mathcal{C}}$
21.        $T(m_r) \leftarrow T(m_r) \cup \{v_i^r\}$
22.        Update the energy consumption of $m_r$
23.    Return
24. /*Yes, try broadcast reception first*/
25. ELSE
26.    /*Consider $v_i^c$'s transmission duration*/
27.    $st \leftarrow s_{v_i^c,\mathcal{C}}$, $ft \leftarrow f_{v_i^c,\mathcal{C}}$
28.    /*No interference when receiving $v_i^c$*/
29.    IF $\nexists \gamma \in T_{st}^{ft}(\mathcal{C}) : S(\gamma) \in N(m_r)$
30.        /*Receive the broadcasted packet*/
31.        Schedule a copy of $v_i^c$ to $m_r$:
32.            $s_{v_i^k,m_r} \leftarrow f_{v_i^c,\mathcal{C}}$, $f_{v_i^k,m_r} \leftarrow f_{v_i^c,\mathcal{C}}$
33.            $T(m_r) \leftarrow T(m_r) \cup \{v_i^r\}$
34.            $R(v_i^c) \leftarrow R(v_i^c) \cup \{m_r\}$
35.            Update the energy consumption of $m_r$
36.        Return
37.    ELSE
38.        /*Another transmission of $v_i$ from $m_s$*/
39.        Goto Step 8
```

Fig. 5. The single-hop communication task scheduling algorithm.

```
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. /*Has $v_i$ been distributed before?*/
2. Find a copy of $v_i$: $v_i^c \in T(\mathcal{C})$
3. /*No, initialize a communication of $v_i$*/
4. IF $v_i^c$ does not exist:
5.     /*Find the sender of $v_i$*/
6.     Find the sensor node $m_s$: $v_i \in T(m_s)$
7.     Find the path from $m_s$ to $m_r$:$path = (m_1, ..., m_n)$
8.     /*Iteratively forward $v_i$ to $m_r$*/
9.     For $m_k = m_2$ to $m_n$
10.        OneHopSchedule($v_i, m_s, m_k$)
11.        $m_s \leftarrow m_k$
12.    Return
13. /*Yes, branching from the nearest node to $m_r$*/
14. ELSE
15.    Find a copy of $v_i$:
16.      $v_i^o \in T(\mathcal{C})$ s.t.:
17.        Distance between $S(v_i^o)$ and $m_r$ is minimum
18.    $m_s \leftarrow S(v_i^o)$
19.    Goto Step 4
```

Fig. 6. The communication task scheduling algorithm.

may delay application finish time, sensors should compensate the retransmission time by speeding up the subsequent task executions. Such speedup is a reverse procedure of our DVS algorithm, which is to be introduced in Section 4.4 to reduce CPU speed. Further discussion of the adaptive communication failure handling algorithm is deferred to our future work.

4.3 Merging Communication Scheduling with the TSSE Algorithm

In the *Task Mapping and Scheduling Phase*, tasks of a Hyper-DAG are assigned to sensor 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 to sensor nodes.
- A communication task can be assigned to sensors or $\mathcal{C}$.
- A communication task assigned to a sensor denotes data stored in sensor memory and is ready to be processed on the same node. Thus, its execution time and energy consumption are zero.
- A nonentry computation task assigned to a sensor must have all its immediate predecessors available before its execution, i.e., 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, *multihop channel model*, *Communication Scheduling Algorithm*, and the task mapping constraints presented above, task mapping and scheduling in multihop wireless networks can be tackled as a generic task mapping and scheduling problem with additional constraints. This problem is NP-complete in general [22] and heuristic algorithms are needed to obtain practical solutions. Due to its satisfactory performance at a relatively low complexity, the Min-Min algorithm [18] is modified and implemented in MTMS. The modified Min-Min algorithm is based on Hyper-DAG scheduling and is referred to as the TSSE algorithm.

The core of the TSSE algorithm is the fitness function. For each task-node combination (v_i, m_k), the fitness function $fit(v_i, m_k, \alpha)$ indicates the combined cost in time and energy domain of assigning task v_i to node m_k, where α is the weight parameter trading off the time cost for the energy consumption cost. At each step of the TSSE algorithm, the task-node combination that gives the minimum fitness value among all combinations is always assigned first. To extend and describe the fitness function of the Min-Min Algorithm in [18], the following notations are introduced first:

- DL is the application deadline relative to the application start time.
- f_{v_i,m_k} is the scheduled finish time of v_i on m_k relative to the application start time. f_{v_i,m_k} denotes the partial schedule length of the application after assigning v_i.
- $EPA(v_i)$ is the amount of energy consumption on all nodes for the application so far before the assignment of v_i.

```
Input: Hyper-DAG; sensor set: $SS$
Output: Optimal schedule $H^o$ of tasks in Hyper-DAG
TSSE Algorithm:
----------------------------------------------------------------
1. /*Scan $\alpha$ values in 0.1 steps*/
2. FOR $\alpha = 0$; $\alpha \leq 1.0$; $\alpha += 0.1$
3.    Assign entry-tasks
4.    Initialize the mappable task list $L$
5.    /*Repeat until all tasks assigned*/
6.    WHILE $L$ is not empty
7.       /*Calculate with all (task,sensor) combinations*/
8.       FOR task $v_i \in L$
9.          FOR all computing sensor $m_k$
10.            /*Ensure dependency constraint*/
11.            IF $pred(v_i) \nsubseteq T(m_k)$
12.               FOR $v_n \in pred(v_i) - T(m_k)$
13.                  CommTaskSchedule($v_n$,$m(v_n)$,$m_k$)
14.            Assign $v_i$ to $m_k$, calculate $fit(v_i, m_k, \alpha)$
15.         Find $m_i^o$ $s.t.$ $fit(v_i, m_i^o, \alpha)$ is minimum
16.      Find the task-sensor pair ($v$,$m$):
17.         $fit(v, m, \alpha)$ is minimum
18.      Assign $v$ to $m$, remove $v$ from $L$
19.      /*Locally assign communication task on sensor*/
20.      assign $succ(v)$ on $m$
21.      Update $L$ with new unassigned mappable tasks
22. Among all schedules with different values of $\alpha$
23.   IF $\exists H : length(H) \leq DL$ with $\min energy(H)$
24.      return $H$
25.   ELSE
26.      return $H : length(H)$ is minimum
```

Fig. 7. The Task Schedule Search Engine (TSSE) algorithm.

- $PE(v_i, m_k)$ represents the application energy consumption increase for assigning v_i to m_k. A computation task v_i cannot be executed on m_k unless all of its immediate predecessors (which are communication tasks) are available on m_k. Thus, a copy of all v_i's immediate predecessors that are not stored in m_k must be scheduled to m_k. $PE(v_i, m_k)$ is the sum of communication energy consumption of sending all missing data to m_k and the computation energy consumption associated with v_i's execution on m_k.
- The trade-off between schedule length and energy consumption is achieved by taking a weighted sum of two unitless entities. The first one is the normalized partial schedule length $NPT(v_i, m_k) = \frac{f_{v_i,m_k}}{DL}$. We also normalize $PE(v_i, m_k)$ by $EPA(v_i)$ and use it as the second contributor to the fitness function: $NPE(v_i, m_k) = \frac{PE(v_i,m_k)}{EPA(v_i)}$.

Thus, the fitness function of assigning v_i to m_k is defined as:

$$fit(v_i, m_k, \alpha) = \alpha \cdot NPT(v_i, m_k) + (1 - \alpha) \cdot NPE(v_i, m_k). \tag{10}$$

The TSSE Algorithm is presented in Fig. 7. In the description of TSSE, a "mappable" task is either an entry-task or a task that has all immediate predecessors already scheduled and the "mappable task list" is the list that contains currently mappable tasks of the Hyper-DAG. During the initial scheduling, sensors are scheduled with full speed f_{cpu}^{max}. For each application, we compare schedules with different α values ranging from 0 to 1 in 0.1 increments. The schedule with the minimum energy consumption under the deadline constraint is chosen as the optimal solution among these candidate schedules. If none of the candidate schedules meets the deadline, the one with the shortest schedule is chosen. Since different values of α represent different trade-offs between scheduling cost in time and energy domains, the α value is kept unchanged in Steps 3-21. However, different applications may find optimal schedules with different α values.

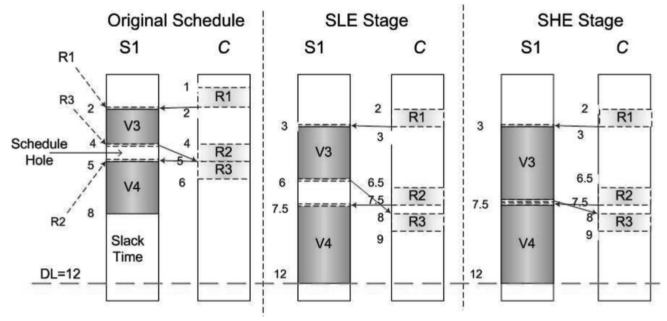


Fig. 8. The partial DVS adjustment example.

In WSNs, sensors are prone to failures. In case of sensor failures, the former schedule will not be a feasible solution. For such a situation, rescheduling with MTMS is needed to recover the functionality. Adjusting the previous schedule is also a viable solution to quickly recover sensor failures, which will be part of our future work.

4.4 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 CPU idle time until the deadline, which is referred to as "slack time." The unbalanced load of sensors and the communication scheduling also result in CPU idle time between computation and communication tasks, which is referred to as a "schedule hole." In the *DVS Phase*, the CPU idle time is exploited by decreasing the CPU speed to reduce computation energy consumption.

Our DVS algorithm is composed of two stages: the Schedule Length Extension (SLE) Stage and the Schedule Hole Elimination (SHE) Stage. In the SLE stage, the slack time between schedule length $length(H)$ and application deadline DL is eliminated by proportionally slowing down all sensors' CPU speed. Let β be defined as $\beta = \frac{length(H)}{DL} < 1$ and the rescale factor γ as $\gamma = \lceil \beta \cdot f_{cpu}^{max} \rceil / f_{cpu}^{max}$. Here, the function $\lceil f \rceil$ is the ceiling function that returns the minimum available CPU speed greater than or equal to f. All processors are slowed down to $\gamma \cdot f_{cpu}^{max}$, which increases computation tasks' execution lengths. To accomplish this, a computation task's start time, execution time, and finish time are multiplied by γ^{-1}. To match the start time of its immediate successors, a communication task v_i's finish time f_{v_i,m_k} is also multiplied by γ^{-1}. Since a communication task v_i's execution length t_{v_i,m_k} is assumed to be independent of the CPU operation, its start time s_{v_i,m_k} is adjusted to $\gamma^{-1} f_{v_i,m_k} - t_{v_i,m_k}$.

An example of DVS adjustment is shown in Fig. 8. For the sake of simplicity, we only consider a partial schedule of a sensor S_1 with two data receptions R_1 and R_2 from $\mathcal{C}$ and one data transmission R_3 to $\mathcal{C}$. It should be noted that R_1, R_2, and R_3 are assigned to $S1$ with zero execution times, while their execution times on $\mathcal{C}$ are all 1 time units (tu).

```
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.   /*Initialize DVS adjust interval parameters*/
3.   $ds \leftarrow 0$, $df \leftarrow \infty$
4.   Scan $v_i \in T_{ds}^{\infty}(m_k)$ in increasing order of $s_{v_i,m_k}$
5.     /*Update $ds, df$ based on the comm. tasks on $\mathcal{C}$*/
6.     IF $\exists$ a copy of $v_i \in R$: $v_i^c \in T(\mathcal{C})$
7.       Find the computation task $v_j$ following $v_i$
8.       IF $m_k$ is the sender of $v_i^c$
9.         /*Comp. task ends before sending results*/
10.        $df \leftarrow \min(s_{v_i^c,\mathcal{C}}, s_{v_j,m_k})$
11.        SpeedAdjust($m_k$,$ds$,$df$,$\gamma \cdot f_{cpu}^{max}$)
12.        $ds \leftarrow df$
13.      ELSE    /*$m_k$ is the receiver of $v_i^c$*/
14.        /*Comp. task starts after receiving data*/
15.        $ds \leftarrow \max(ds, f_{v_i^c,m_k}, s_{v_j,m_k})$
16.    /*Adjustment bounded by deadline*/
17.    ELSE IF $v_i$ is exit-task and $f_{v_i} < DL$
18.      SpeedAdjust($m_k$,$ds$,$DL$,$\gamma \cdot f_{cpu}^{max}$)
```

Fig. 9. The SHE algorithm.

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

1. /*Initialization*/
2. $e_{ds}^{df} \leftarrow 0$, $tt \leftarrow ds$
3. /*Calculate CPU execution time $e_{ds}^{df}$ in $[ds, df]$*/
4. FOR $v_i \in T_{ds}^{ft}(m_k)$ and $v_i \in V$
5.   $e_{ds}^{df} \leftarrow e_{ds}^{df} + t_{v_i,m_k}$
6. /*CPU utility in [ds,df]*/
7. $\eta \leftarrow e_{ds}^{df}/(df - ds)$
8. /*Adjusted CPU speed in [ds,df]*/
9. $f_{cpu}^o \leftarrow \lceil f_{cpu} \cdot \eta \rceil$
10. /*Adjust computation tasks*/
11. FOR $v_i \in T_{ds}^{df}(m_k)$ and $v_i \in V$
12.   $s_{v_i,m_k} \leftarrow tt$
13.   $t_{v_i,m_k} \leftarrow t_{v_i,m_k} \cdot \frac{f_{cpu}}{f_{cpu}^o}$
14.   $f_{v_i,m_k} \leftarrow s_{v_i,m_k} + t_{v_i,m_k}$
15.   $tt \leftarrow f_{v_i,m_k}$
16. /*Immediate successors of the adjusted comp. tasks*/
17. FOR $v_i \in T_{ds}^{ft}(m_k)$ and $v_i \in R$
18.   IF $pred(v_i) \in T(m_k)$
19.     $s_{v_i,m_k} \leftarrow f_{pred(v_i),m_k}$, $f_{v_i,m_k} \leftarrow f_{pred(v_i),m_k}$
20. Update the energy consumption of $m_k$
```

Fig. 10. The CPU speed adjust algorithm in a given time interval $[ds, df]$.

Assuming that the original schedule length is 8 tu with CPU speed f_{cpu}^{max}, the deadline DL = 12 tu and the calculated rescale factor $\gamma^{-1} = 1.5$. In the SLE Stage of Fig. 8, the CPU speed is reduced to $f_{cpu}^{max}/1.5$. Consequently, v_4's start time and finish time are adjusted from 5 tu and 8 tu to 7.5 tu and 12 tu, respectively. Therefore, the slack time before the deadline is eliminated. On the other hand, v_3's start time and finish time are adjusted from 2 tu and 4 tu to 3 tu and 6 tu, respectively. Thus, the schedule hole between v_3 and v_4 still exists, which is eliminated in the SHE stage. The time interval $[ds, df]$ that contains the schedule hole is first decided as follows: v_3 cannot start execution before R_1 reception finishes at 3 tu, which makes $ds = 3$ tu. v_3 must be finished before v_4 starts at 7.5 ut and $R3$ is transmitted at 8 tu. Thus, $df = min(7.5, 8) = 7.5$ tu. The CPU speed in $[ds, df]$ is further reduced. The schedule of v_3 is consequently adjusted to finish at 7.5 tu, and the schedule hole is eliminated. It should be noted that, due to the discrete nature of DVS, smaller slack time and schedule holes may still exist after adjustment in general.

The SHE algorithm is presented in Fig. 9. The SHE algorithm iteratively scans each sensor's schedule to detect time intervals $[ds, df]$ that contain schedule holes. As demonstrated in Fig. 8, a communication tasks' reception finish time is taken as the lower bound of calculating ds, as a computation task cannot be executed before all of its immediate predecessors (which are communication tasks) are available. df equals the minimum of the start times of the following computation task and transmission event launched by m_k. Once a time interval $[ds, df]$ that contains a schedule hole is found, $SpeedAdjust()$ is executed to eliminate the schedule hole by reducing the CPU speed in $[ds, df]$. The $SpeedAdjust()$ algorithm is presented in Fig. 10. In $SpeedAdjust()$, the *CPU utility* η during a time interval $[ds, df]$ is defined as:

$$\eta = e_{ds}^{df}/(df - ds), \tag{11}$$

where e_{ds}^{df} is the overall CPU execution time during $[ds, df]$. We first reduce the CPU speed in $[ds, df]$ to $\lceil f_{cpu} \cdot \eta \rceil$. In Steps 10-15, execution times of computation tasks in $[ds, df]$ are increased according to the updated CPU speed. Note that SpeedAdjust() does not change the communication schedule on $\mathcal{C}$.

4.5 Computational Complexity Analysis

We first assume that there are s sensors in a k-hop network, and the DAG has n tasks with e edges. Thus, the extended Hyper-DAG has n computation tasks, n communication tasks, and the average in-degree of computation tasks is e/n.

In the TSSE Algorithm, the loop from Step 12 to Step 13 is executed in $O(\frac{e}{n})$ time, the loop from Step 9 to Step 14 is executed in $O(s)$ time, and the loops starting from Step 8 and Step 6 are both executed in $O(n)$ time. The communication scheduling algorithm has a complexity determined by a routing algorithm. The geographic algorithm GPSR has a complexity of $O(k)$. Thus, the complexity of the *task mapping and scheduling phase* is $O(\frac{e}{n} \cdot s \cdot n^2 \cdot k) = O(ensk)$.

Regarding the DVS algorithm, the SLE stage needs to adjust all tasks once. Thus, it has a complexity of $O(v)$. The *SpeedAjust Algorithm* of the SHE stage scans and adjusts tasks assigned on each sensors with a complexity of $O(\frac{n}{s})$. The SHE Algorithm scans and adjusts all unadjusted tasks at most once with a complexity of $O(n \cdot \frac{n}{s}) = O(\frac{n^2}{s})$. Thus, the complexity of the DVS algorithm is $O(n + n^2/s)$.

Taking both the *task mapping and scheduling phase* and the *DVS phase* into account, the overall complexity is $O(ensk + n + \frac{n^2}{s})$. If we assume that sensors are uniformly distributed in a network, we have $s = O(k^2)$ and the overall complexity is $O(enk^3 + \frac{n^2}{k^2})$.

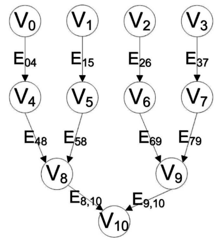


Fig. 11. The DAG of the visual surveillance example.

TABLE 1
Simulation with the Visual Surveillance Example

DL	Metrics	MTMS	EbTA	DCA
3 ms	Schedule Length (ms)	3.00	3.00	5.64
	Overall Energy Consumption (uJ)	2194	2743	2238
	MECpN (uJ)	592	298	1139
8 ms	Schedule Length (ms)	7.88	8.00	7.88
	Overall Energy Consumption (uJ)	1689	2362	1802
	MECpN (uJ)	976	298	703

5 Simulation Results

The performance of the MTMS algorithm with DVS adjustment is evaluated through simulations. The performance of an extended version of DCA with DVS is also evaluated as a benchmark. DCA is extended with our proposed communication scheduling algorithm to deliver the intermediate results of entry-tasks to the cluster head for further processing. We first simulate the application in Fig. 1 and compare with DCA and EbTA algorithms. To further evaluate MTMS performance, simulations are run on arbitrary applications with randomly generated DAGs and WSN topologies. Our simulations study the following scenarios:

- effects of the application deadline constraints and DVS adjustment,
- effect of the number of tasks in applications,
- effect of the cluster size,
- effect of the communication load, and
- comparison with EbTA [14] in single-hop clustered networks.

In these simulations, we observe schedule length, energy consumption, and deadline missing ratio. The schedule length is defined as the finish time of the exit-task of an application. The energy consumption includes the computation and communication energy expenditure of all sensors. The deadline missing ratio is defined as the ratio of the number of the schedules with lengths larger than the deadline to all schedules generated. To evaluate energy balance, we also observe the maximum energy consumption per node (MECpN) when compared with EbTA. As indicated in [14], MECpN can serve as a measurement of energy consumption balance and network lifetime.

5.1 Simulation Parameters

In our simulation study, the bandwidth of the channel is set to 1 Mb/s and the transmission range $r = 10$ meters. Sensors are equipped with the StrongARM SA-1100 microprocessor, whose speed ranges from 59 MHz to 206 MHz with 30 discrete levels. The parameters of (1) through (4) are in coherence with [19], [12] as follows: $E_{elec} = 50\ \mathrm{nJ/b}$, $\varepsilon_{amp} = 10\ \mathrm{pJ/b}/m^2$, $V_T = 26\ \mathrm{mV}$, $C = 0.67\ \mathrm{nF}$, $I_o = 1.196\ \mathrm{mA}$, $n = 21.26$, $K = 239.28\ \mathrm{MHz/V}$, and $c = 0.5\ \mathrm{V}$.

5.2 The Real-Life Example of Distributed Visual Surveillance

To demonstrate in-network processing in WCSNs, we consider the simplified intrusion detection system in Fig. 1. Rather than sending large volumes of data to a base station for processing, we locally process the data and send the location information of the object. The in-network processing application is abstracted as the DAG in Fig. 11, where tasks $V_0 - V_3$ represent background subtracting and bounding box abstracting [6]. After these steps, detected moving objects are approximated with rectangles (bounding boxes) in images from each camera and represented by the vertices of their bounding boxes. The vertice coordinates and camera calibration data are passed to the next localization stage. Object locations can be estimated by "passing a viewing ray through the bottom of the object in the image and intersecting it with a model representing the terrain" [24] with data from each camera. To evaluate the estimation error range, the locations of the points on the bounding boxes' bottom lines are calculated, which are represented by tasks $V_4 - V_7$. Then, the location estimation results from different cameras are fused to eliminate estimation errors in tasks $V_8 - V_{10}$. The edges $E_{04} - E_{37}$ stand for the communication of bounding box coordinates and camera calibration data, and $E_{46} - E_{9,10}$ denote the communication of the estimated object locations with estimation error ranges. After V_{10}, the object's location is recovered from 2D images and sent to a base station. As we can expect, the data volume to be delivered is much smaller than that of original images.

In the simulation, we evaluate the performance of MTMS, DCA, and EbTA algorithms. Since EbTA is designed for single-hop clusters, we do not obtain results for multihop clusters. We assume a single object, 256×256 gray-scale images, the task computation load of 200 KCC for $V_0 - V_7$, computation load of 10 KCC for $V_8 - V_{10}$, communication volume of 20 bytes for $E_{04} - E_{37}$, and the communication volume of 40 bytes for $E_{48} - E_{9,10}$. We present two sets of results in Table 1 for deadlines DL = 3 ms and DL = 8 ms. For the shorter deadline, DCA fails to meet the deadline since it does not leverage the parallel processing potential of the cluster. For the larger deadline, all algorithms satisfy deadline requirements, and DVS decreases energy consumption. In both cases, MTMS can exploit more slack time before the deadlines than DCA, which leads to smaller energy consumption. Regarding the energy consumption of MTMS and EbTA, MTMS has smaller application energy consumption than EbTA because EbTA distributes computation tasks to more sensors to lower down MECpN, which leads to more communication events scheduled on the channel and higher overall energy consumption. In this specific application, where all communications are unicast, the MECpN of EbTA is lower than MTMS.

It should be noted that, in the example above, sending these four 64 KByte images will consume about 0.2 J per hop. According to Table 1, with the smaller deadline DL = 3 ms, the energy consumption of MTMS is 2194 uJ, which is much smaller than transmitting all images over one hop. After the in-network processing, the resulting data volume is reduced to 40 bytes, which consumes only 32.32 uJ to be

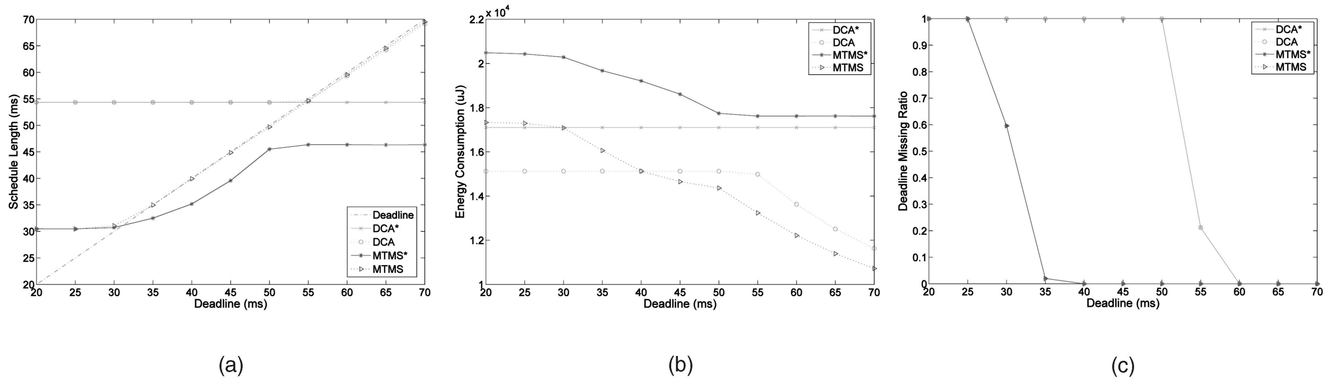


Fig. 12. Effect of application deadlines. (a) Schedule length. (b) Energy consumption. (c) Deadline missing ratio.

delivered over one hop. Thus, the overall energy consumption of processing information and transmitting the result is drastically reduced when compared with directly delivering original images over one hop. In large-scale WSNs where base stations are located multiple hops away, energy savings through in-network processing become more pronounced.

5.3 Simulation with Random DAGs

To evaluate MTMS performance for arbitrary applications, simulations are run on randomly generated DAGs which are scheduled on randomly created multihop 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 predecessors *maxPred*. Unless specifically stated, the number of each nonentry task's 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 $\pm$ 10 percent], and [800 bits $\pm$ 10 percent], respectively. The sensors are uniformly distributed on disc of radius $k \cdot r$ to form a k-hop connected cluster. We assume that there are $n = 5$ sensors in a single-hop cluster. Thus, there are $5k^2$ sensors in a k-hop cluster. During simulations, the entry-tasks are randomly assigned to sensors. The simulation results presented in this section correspond to the average of 250 random (DAG, cluster) combinations. For each pair of DAG and cluster, different deadlines are imposed to evaluate performances.

5.3.1 Effect of the Application Deadlines and DVS Adjustment

We investigate the effect of application deadlines and DVS adjustment with 250 pairs of randomly created DAGs and 3-hop clusters. The parameters of DAGs considered for this set of simulations are *numTask* = 40, *numEntry* = 10, and *maxPred* = 10. To evaluate the effect of DVS, the schedule length, energy consumption, and deadline missing ratio of DCA and MTMS before the voltage adjustment (denoted as DCA* and MTMS*, respectively) are also investigated.

As shown in Fig. 12a and Fig. 12c, MTMS has a better capability to meet deadlines compared with DCA. When deadlines are very small, even though the deadline missing ratio of MTMS and DCA are both high, the average schedule length of MTMS 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 MTMS can have more sensors involved in parallel, which speeds up execution.

Regarding energy consumption, DCA* has better energy consumption performance than MTMS* for most scenarios according to Fig. 12b. However, by implementing the DVS algorithm, this energy consumption difference is significantly reduced. The DVS adjustment is a promising technique to optimize energy consumption, as shown in Fig. 12. Even when deadlines are relatively small and there is very little slack time before application deadlines, the DVS adjustment of MTMS can still save about 15 percent energy compared with the scenarios without the DVS adjustment (MTMS*). 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. 12a), the deadline missing ratio is not affected (Fig. 12c) for any of the simulated deadline values.

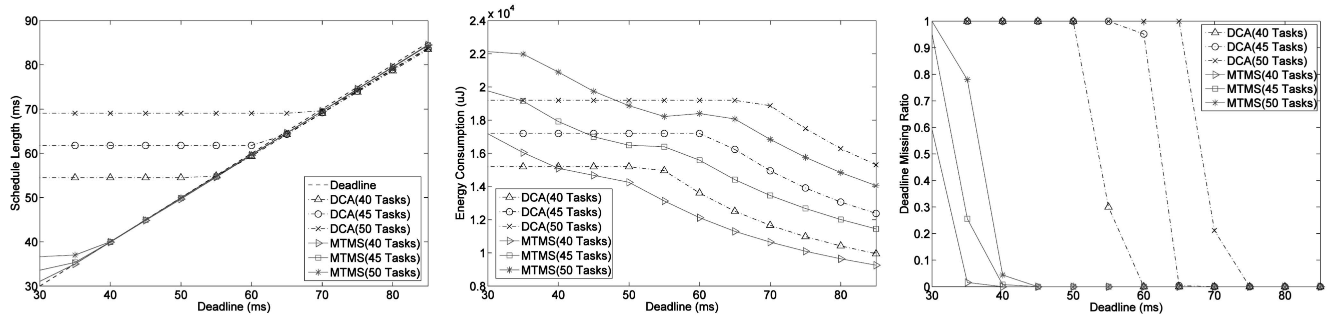
5.3.2 Effect of the Number of Tasks

To investigate the effect of number of tasks in applications, simulations are run on randomly generated DAGs with 40, 45, and 50 tasks (*numEntry* = 10, *maxPred* = 10). For fair comparison, each set of 40, 45, and 50 task DAGs are scheduled on the same randomly created 3-hop cluster. The presented results are the average of 250 simulation runs, and each simulation corresponds to one set of randomly generated 3-hop cluster and DAG.

According to the simulation results in Fig. 13b, energy consumption is dominated by the number of tasks. When the number of tasks increases, the energy consumption of DCA and MTMS both increase proportionally, and MTMS has higher energy consumption. However, when deadline is increasing, the energy consumption of MTMS decrease faster than DCA by exploiting the available CPU slack time due to its better capacity to meet deadlines. Regarding the deadline missing ratio, DCA is dramatically affected with task volume increment while MTMS is less affected (Fig. 13c). This property is also reflected with schedule length presented in Fig. 13a. Thus, MTMS has a better scalability compared with DCA regarding schedule length and deadline missing ratio.

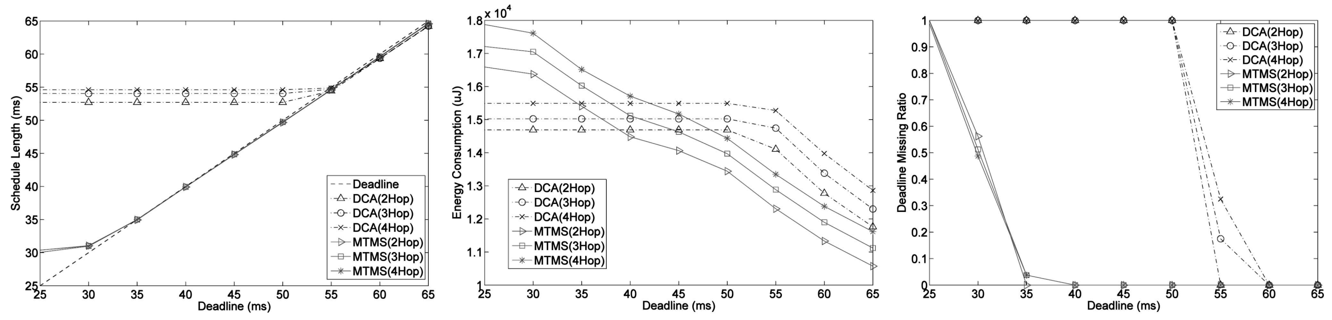
5.3.3 Effect of the Cluster Size

In this section, the effect of the cluster size is evaluated with random DAGs scheduled on 2-hop, 3-hop, and 4-hop random clusters. Each result represents the average of



(a) (b) (c)

Fig. 13. Effect of number of tasks (40 tasks versus 45 tasks versus 50 tasks). (a) Schedule length. (b) Energy consumption. (c) Deadline missing ratio.



(a) (b) (c)

Fig. 14. Effect of cluster size. (a) Schedule length. (b) Energy consumption. (c) Deadline missing ratio.

250 simulation runs. 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.

The simulation results are shown in Fig. 14. As the cluster size increases, the performance of DCA degrades correspondingly. With regard to MTMS, the energy consumption proportionally increases with increasing cluster size. However, an interesting observation is that, when the cluster size increases from 3-hop to 4-hop, the deadline missing ratio slightly decreases around the deadline of 30 ms. This deadline missing ratio improvement stems from the better parallelism achieved with larger network sizes. When the cluster size increases, there can be more communication scheduled simultaneously, which may lead to more computation tasks executed in parallel. Thus, MTMS has a better capacity to adapt to larger cluster sizes.

5.3.4 Effect of the Communication Load

In task mapping and scheduling, the relationship between communication and computation load may affect the performance. This factor is evaluated by changing the average result data volume with fixed average computation load. Simulations are run with randomly generated DAGs with *numTask* = 40, *numEntry* = 10, and *maxPred* = 10 on 3-hop random clusters. The three different settings of DAGs have communication data volume uniformly distributed in [600 bit, ± 10 percent], [800 bit, ± 10 percent], and [1,000 bit, ± 10 percent] with the task computation load uniformly distributed in [300 KCC, ± 10 percent]. For each random network, there are 10 sets of DAGs scheduled. The simulation results are the average with 25 random networks.

As shown in Fig. 15, the performance of DCA and MTMS are both affected by the communication load: As communication load increases, the energy consumption, schedule lengths, and deadline missing ratio increase as well. Compared to MTMS, the performance of DCA degrades less with increased communication load. DCA's robustness against communication load increase stems from the fact that DCA has most of its tasks executed on cluster heads. Since the execution time and energy consumption of a communication task on the same sensor node is zero, increasing the communication load will not affect these tasks' execution. On the other hand, the MTMS algorithm assigns tasks on different sensors to speed up execution, which leads to more communication tasks scheduled on $\mathcal{C}$. Thus, the MTMS algorithm is affected more by the communication load changes. However, it can be observed that, even when the communication load increases, MTMS still significantly outperforms DCA in terms of meeting deadline constraints, and its energy consumption decreases faster when deadline increases.

5.3.5 Comparison with EbTA [14]

To further evaluate our proposed solution, we compare the performance of MTMS with EbTA [14]. Since EbTA is not designed for multihop networks, we run simulations for single-hop, single-channel clusters. Due to the small scale of a single-hop cluster (five sensors as assumed in Section 5.1), performances are evaluated with applications of less computation load. The presented results are the average of 250 simulation runs of random DAGs with *numTask* = 20,

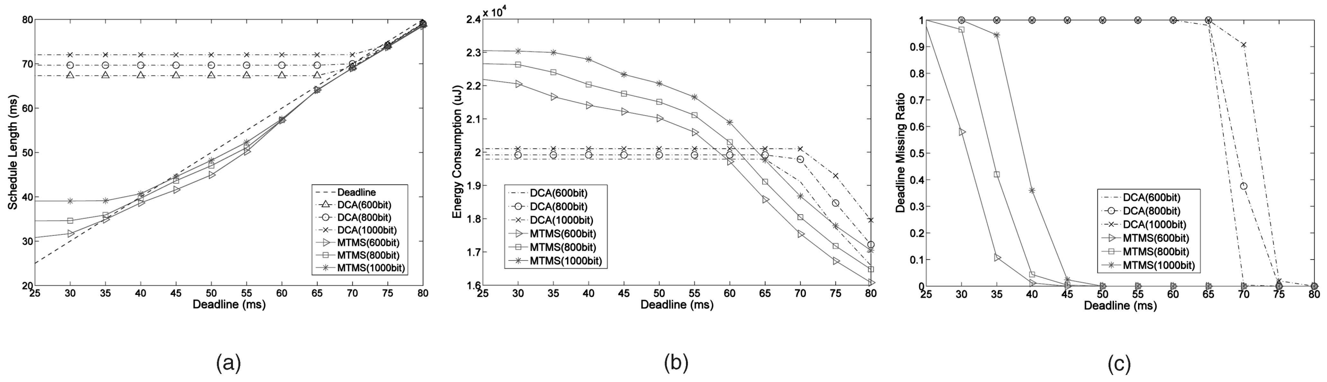


Fig. 15. Effect of communication load. (a) Schedule length. (b) Energy consumption. (c) Deadline missing ratio.

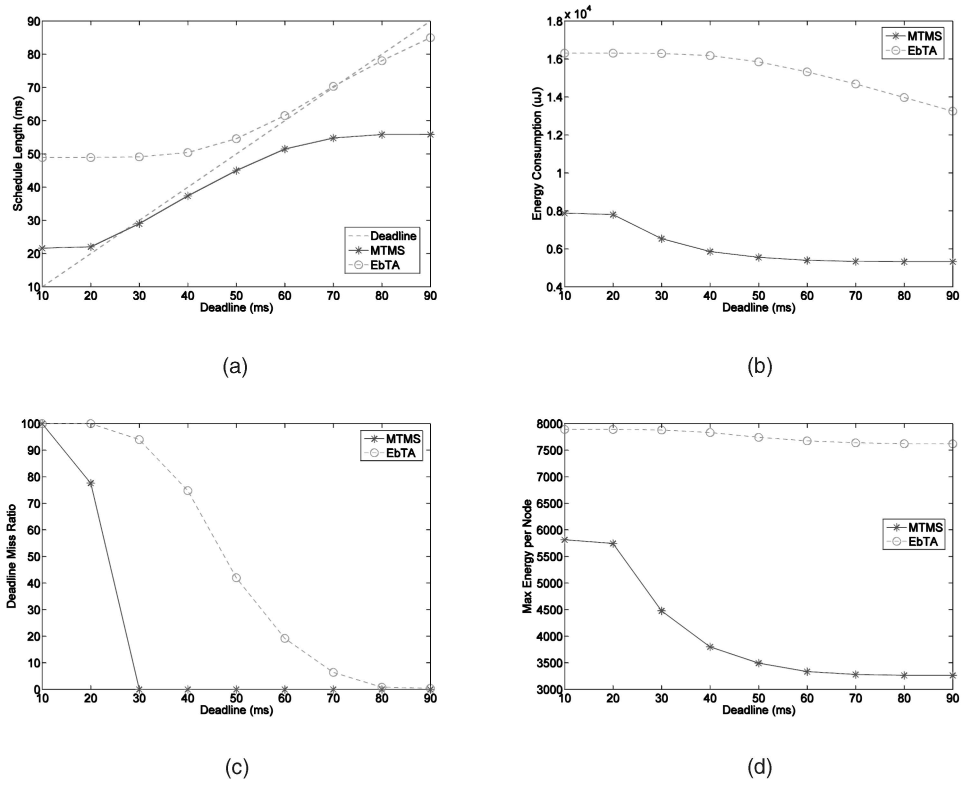


Fig. 16. Comparison with EbTA. (a) Schedule length. (b) Application energy consumption. (c) Deadline missing ratio. (d) Max energy consumption per node.

numEntry = 5, and *maxPred* = 5. The metrics we observe are schedule length, application energy consumption, deadline missing ratio, and MECpN.

As shown in Fig. 16, MTMS outperforms the energy-balanced solution, EbTA, with smaller application energy consumption, MECpN, schedule length, and deadline missing ratio for all simulated scenarios. The superior performance of MTMS mainly stems from the fact that MTMS exploits the broadcast feature of wireless channel when scheduling communication events, while a task in EbTA must send information individually to its immediate successors. Another factor is that EbTA aims to balance sensor energy consumption by evenly distributing computation tasks, which leads to more communication tasks scheduled on the channel and higher energy consumption. Thus, MTMS is less affected by the communication load between computation tasks of an application compared with EbTA.

6 Conclusions and Future Work

In this paper, we propose an application-independent task mapping and scheduling solution for multihop WSNs—Multihop Task Mapping and Scheduling (MTMS). We consider applications executed in multihop clusters of a WSN with delay constraints. The design objective of MTMS is to map and schedule the tasks of an application with the minimum energy consumption subject to delay constraints. The multihop 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, the task scheduling algorithm schedules tasks with minimum energy consumption subject to deadline constraints. Simulations with randomly generated DAGs and networks show significant performance improvements compared with

DCA and EbTA in terms of minimizing energy consumption subject to delay constrains.

In our future work, we plan to develop a low complexity algorithm to quickly recover functionality when sensor failures occur. Furthermore, our future work will address the handling of communication failures by retransmitting erroneous packets and adaptively adjusting sensor schedules to meet deadlines. We will also extend the penalty function to reflect factors such as link quality and develop an adaptive routing algorithm. With this approach, the communication scheduling will choose reliable links and balance communication load among cluster nodes, which will increase the communication reliability and the network lifetime.

Acknowledgments

The authors would like to thank Yang Yu for providing the simulator of EbTA [14].

References

[1] J. Liu, J. Reich, and F. Zhao, "Collaborative In-Network Processing for Target Tracking," *EURASIP J. Applied Signal Processing*, no. 4, pp. 378-391, Mar. 2003.

[2] T. Vercauteren, D. Guo, and X. Wang, "Joint Multiple Target Tracking and Classification in Collaborative Sensor Networks," *IEEE J. Selected Areas in Comm.*, vol. 23, no. 4, pp. 714-723, Apr. 2005.

[3] P. Kulkarni, D. Ganesan, and P. Shenoy, "Multimedia Sensing: The Case For Multitier Camera Sensor Networks," *Proc. 15th Int'l Workshop Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '05)*, pp. 141-146, June 2005.

[4] W.-C. Feng, E. Kaiser, W.C. Feng, and M.L. Baillif, "Panoptes: Scalable Low-Power Video Sensor Networking Technologies," *ACM Trans. Multimedia Computing, Comm., and Applications*, vol. 1, no. 2, pp. 151-167, May 2005.

[5] B. Zitova and J. Flusser, "Image Registration Methods: A Survey," *Elsevier Image and Vision Computing*, vol. 21, no. 11, pp. 977-1000, Oct. 2003.

[6] M. Valera and S.A. Velastin, "Intelligent Distributed Surveillance Systems: A Review," *IEE Proc. Vision, Image and Signal Processing*, vol. 152, no. 2, pp. 192-204, Apr. 2005.

[7] D. Thirde, M. Borg, J. Aguilera, H. Wildenauer, J. Ferryman, and M. Kampe, "Robust Real-Time Tracking for Visual Surveillance," *J. Applied Signal Processing*, 2006.

[8] L.D. de Cerio, M. Valero-Garcia, and A. Gonzalez, "Hypercube Algorithms on Mesh Connected Multicomputers," *IEEE Trans. Parallel and Distributed Systems*, vol. 13, no. 12, pp. 1247-1260, Dec. 2002.

[9] A. Dogan and F. Özgüner, "Matching and Scheduling Algorithms for Minimizing Execution Time and Failure Probability of Applications in Heterogenous Computing," *IEEE Trans. Parallel and Distributed Systems*, vol. 13, no. 3, pp. 308-323, Mar. 2002.

[10] T.D. Braun, H.J. Siegel, N. Beck, L.L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen, and R.F. Freund, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *J. Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810-837, June 2001.

[11] S. Giannecchini, M. Caccamo, and C.-S. Shih, "Collaborative Resource Allocation in Wireless Sensor Networks," *Proc. Euromicro Conf. Real-Time Systems (ECRTS '04)*, pp. 35-44, June/July 2004.

[12] A. Wang and A. Chandrakasan, "Energy-Efficient DSPs for Wireless Sensor Networks," *IEEE Signal Processing Magazine*, pp. 68-78, July 2002.

[13] R. Kumar, V. Tsiatsis, and M.B. Srivastava, "Computation Hierarchy for In-Network Processing," *Proc. Second ACM Int'l Conf. Wireless Sensor Networks and Applications (WSNA '03)*, pp. 68-77, Sept. 2003.

[14] Y. Yu and V.K. Prasanna, "Energy-Balanced Task Allocation for Collaborative Processing in Wireless Sensor Networks," *ACM/Kluwer J. Mobile Networks and Applications*, vol. 10, nos. 1-2, pp. 115-131, Feb. 2005.

[15] Y. Tian, E. Ekici, and F. Özgüner, "Energy-Constrained Task Mapping and Scheduling in Wireless Sensor Networks," *Proc. IEEE Int'l Workshop Resource Provisioning and Management in Sensor Networks (RPMSN '05)*, pp. 211-218, Nov. 2005.

[16] Y. Tian, B. Jarupan, E. Ekici, and F. Özgüner, "Real-Time Task Mapping and Scheduling for Collaborative In-Network Processing in DVS-Enabled Wireless Sensor Networks," *Proc. IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS '06)*, pp. 1-10, Apr. 2006.

[17] O. Younis, M. Krunz, and S. Ramasubramanian, "Node Clustering in Wireless Sensor Networks: Recent Developments and Deployment Challenges," *IEEE Network*, vol. 20, no. 3, pp. 20-25, May/June 2006.

[18] 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," *Proc. Parallel and Distributed Processing Symp.*, Apr. 2004.

[19] 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," *Proc. ACM MobiCom '01*, pp. 272-286, July 2001.

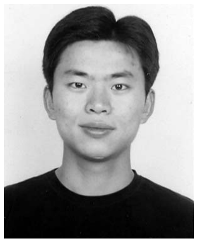
[20] T. Pering, T. Burd, and R. Brodersen, "Dynamic Voltage Scaling and the Design of a Low-Power Microprocessor System," *Proc. Driven Microarchitecture Workshop*, pp. 107-112, 1998.

[21] J. Pouwelse, K. Langendoen, and H. Sips, "Dynamic Voltage Scaling on a Low-Power Microprocessor," *Proc. ACM MobiCom '01*, pp. 251-259, July 2001.

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

[23] B. Karp and H.T. Kung, "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks," *Proc. ACM MobiCom '00*, pp. 243-254, Aug. 2000.

[24] R.T. Collins, A.J. Lipton, H. Fujivoshi, and T. Kanade, "Algorithms for Cooperative Multisensor Surveillance," *IEEE Proc.*, vol. 89, no. 10, pp. 1456-1477, Oct. 2001.



Yuan Tian received the BS and MS degrees in information engineering and communication and information systems from Zhejiang University, Hangzhou, China, in 1998 and 2001, respectively. He recently received the PhD degree in electrical and computer engineering from the Ohio State University, Columbus, Ohio, in 2006. He is currently with Bosch Research and Technology Center North America, Palo Alto, California. His current research interests include medium access control protocols, wireless sensor networks, and network resource allocation and optimization.

Eylem Ekici received the BS and MS degrees in computer engineering from Bogazici University, Istanbul, Turkey, in 1997 and 1998, respectively. He received the PhD degree in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, Georgia, in 2002. Currently, he is an assistant professor in the Department of Electrical and Computer Engineering of the Ohio State University, Columbus, Ohio. His current research interests include wireless sensor networks, vehicular communication systems, next generation wireless systems, and space-based networks, with a focus on routing and medium access control protocols, resource management, and analysis of network architectures and protocols. He is a member of the IEEE.