ENERGY-EFFICIENT COMPUTATION AND COMMUNICATION SCHEDULING FOR CLUSTER-BASED IN-NETWORK PROCESSING IN LARGE-SCALE WIRELESS SENSOR NETWORKS

DISSERTATION

Presented in Partial Fulfillment of the Requirements for

the Degree Doctor of Philosophy in the

Graduate School of The Ohio State University

By

Yuan Tian, B.S., M.S.

* * * * *

The Ohio State University

2006

Dissertation Committee:

Professor Füsun Özgüner, Adviser

Professor Eylem Ekici, Co-Adviser

Professor Charles A. Klein

Approved by

_____

Adviser

_____

Co-Adviser

Graduate Program in
Electrical and Computer
Engineering

ABSTRACT

Emerging Wireless Sensor Networks (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 not only process information at the application layer, but also synchronize their communication activities to exchange partially processed data for parallel processing. Task mapping and scheduling plays an important role in parallel processing. Though this problem has been extensively studied in the high performance computing area, its counterpart in WSNs remains largely unexplored. Scheduling computation and communication events is a challenging problem in WSNs due to limited resource availability and shared communication medium. This research investigates the energy-efficient task mapping and scheduling problem in large-scale WSNs composed of homogeneous wireless sensors. A hierarchical WSN architecture is assumed to be composed of sensor clusters, where applications are iteratively executed. Given this environment, task mapping and scheduling in single-hop clustered WSNs is investigated for energy-constrained applications. Based on the proposed Hyper-DAG model and single-hop channel model, the EcoMapS solution minimizes schedule lengths subject to energy consumption constraints. Secondly, real-time applications are also considered in single-hop clustered WSNs. Incorporating the novel Dynamic Voltage Scaling (DVS) algorithm, the RT-MapS solution provides deadline guarantee with the minimum balanced energy consumption. Next, the task mapping

and scheduling problem is further addressed in its general form for multi-hop clustered WSNs. A novel multi-hop channel model is developed, and a multi-hop communication scheduling algorithm is presented, based on which the MTMS solution minimizes application energy consumption subject to deadline constraints. Finally, low-complexity sensor failure handling algorithms are developed to recover network functionality when sensors failures occur in single-hop and multi-hop clustered WSNs.

Dedicated to my parents and sisters . . .

ACKNOWLEDGMENTS

First and foremost, I would like to thank my family, namely my mother, my father, and my sisters for their unwavering love, patience, support, and encouragement. Without their help, I could not have completed this endeavor.

I would like to thank my advisers Prof. Füsun Özgüner and Prof. Eylem Ekici for their guidance, and intelligent support throughout the course of my graduate studies and research, and throughout the preparation of this dissertation. It is their encouragement, patience, and enthusiasm that makes this dissertation possible. I would also like to thank my other committee member, Prof. Charles. A. Klein, for his valuable time and helpful suggestions.

I would also like to thank my great labmates, who make my life in the High Performance Computing and Networking Laboratory colorful and enjoyable. They are Doruk Bozdag, Atakan Dogan, Mohammed Soleiman Eltayeb, Kavitha Golconda, Yaoyao Gu, Tim Hartley, Boangoat Jarupan, Gokhan Korkmaz, Elif Sariarslan, Justin Teller, and Serdar Vural.

VITA

1998 ..... B.S. in Information Engineering, Zhejiang University, Hangzhou, China

2001 ..... M.S. in Communication and Information System, Zhejiang University, Hangzhou, China

PUBLICATIONS

Research Publications

Yuan Tian, Yaoyao Gu, Eylem Ekici, and Füsun Özgüner, Dynamic Critical-Path Task Mapping and Scheduling for Collaborative In-Network Processing in Multi-Hop Wireless Sensor Networks. In *Proc. of the IEEE Workshop on Ad hoc and Sensor Networks (WASNet'06), in conjunction with ICPP'06*, Columbus, OH, August, 2006.

Yuan Tian, Boangoat Jarupan, Eylem Ekici, and Füsun Özgüner, Real-Time Task Mapping and Scheduling for Collaborative In-Network Processing in DVS-Enabled Wireless Sensor Networks. In *Proc. of the IEEE International Parallel and Distributed Processing Symposium (IPDPS'06)*, pp. 1-10, Rhodes, Greece, April 2006.

Yuan Tian, Eylem Ekici, and Füsun Özgüner, Energy-Constrained Task Mapping and Scheduling in Wireless Sensor Networks. In *Proc. of the IEEE International Workshop on Resource Provisioning and Management in Sensor Networks (RPMSN'05), in conjunction with MASS'05*, pp. 211–218, November 2005.

Yuan Tian, Füsun Özgüner, and Eylem Ekici, Comparison of Hyper-DAG Based Task Mapping and Scheduling Heuristics for Wireless Sensor Networks. In *Proc. of the 20th International Symposium on Computer and Information Sciences (ISCIS 2005)*, pp. 74-83, Istanbul, Turkey, October 2005.

Yuan Tian, Atakan Dogan and Füsun Özgüner, An Urgency-Based Prioritized MAC Layer Protocol for Real-Time Traffic in Ad-Hoc Wireless Networks. In *Proc. of the IEEE International Workshop on Wireless, Mobile and Ad Hoc Networks (WMAN'03), in conjunction with IPDPS'03*, Nice, France, March 2003.

FIELDS OF STUDY

Major Field: Electrical and Computer Engineering

Studies in:

Wireless Sensor Networks
Network Resource Allocation and Optimization
Task Mapping and Scheduling
Distributed MAC Protocol in Ad Hoc Networks

TABLE OF CONTENTS

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

Wireless Sensor Networks (WSNs) are envisioned to observe large environments at close range for extended periods of time. WSNs are generally composed of a large number of sensors with relatively low computation capacity and limited energy supply [3]. One of the fundamental challenges in Wireless Sensor Networks (WSN) is attaining energy efficiency at all levels of design and operation. Many energy efficient communication solutions have recently been proposed for WSNs [23] [29] [41] [43] [57]. In-network processing emerges as an orthogonal approach to significantly decrease network energy consumption [3] [52] by eliminating redundancy and reducing communicated information volume. Example applications include distributed data compression and aggregation [7] [13] [15] [36]. The benefits of in-network processing are especially pronounced in video sensor networks [38] [24] composed of wireless sensors equipped with cameras, where data streams from neighboring nodes can be highly correlated with considerable data volume. A simplified motivating example of video sensor networks is shown in Fig. 1.1, where four calibrated camera sensors collaboratively detect an intruding vehicle's features such as location, vehicle type, and threat level. The sensors first estimate the intruder's features by themselves, then fuse the intermediate results to eliminate estimation errors. Compared to the original images, the resulting data volume can be reduced by several orders of magnitude. Thus, it is

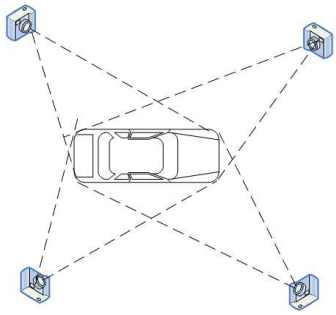


Figure 1.1: A Simplified Distributed Video Surveillance Example

more energy-efficient to send the processed data than delivering the raw data in large-scale WSNs, where base stations can be multiple hops away.

However, such in-network processing applications may require computationally intensive operations to be performed in the network subject to certain constraints. For instance, in target tracking applications [42] [63], sensors collaboratively measure and estimate the location of moving targets or classify targets. To conserve energy and reduce communication load, operations such as Bayesian Estimation and data fusion must be executed in the WSN. In the case of tracking or detecting multiple high-speed moving targets, these operations must be finished in a timely manner with an eye toward limited energy consumption. For video sensor networks, in-network processing such as image registration [72] and distributed visual surveillance [62] may demand considerable computation power that is beyond the capacity of each individual sensor. Thus, it is desirable to develop a general solution to provide the computation capacity required by in-network processing. In WSNs

with densely deployed nodes, a promising solution is to have sensors collaboratively process information with distributed computation load among sensors. To achieve application-independent parallel processing, *task mapping and scheduling* [21] is a problem that must be solved. This dissertation addresses the task mapping and scheduling problem in WSNs to provide the necessary computation power for collaborative in-network processing.

1.1 Task Mapping and Scheduling

Task mapping and scheduling has been intensively studied in the high-performance computing area [11] [10] [18] [21] [22] [27] [28] [53] [46], where applications are generally assumed to be already partitioned into inter-dependent tasks [10]. As such, applications can be represented by Directed Acyclic Graphs (DAG), where the vertices denote the tasks and the edges denote the dependency and communication between the tasks [22]. Two important problems are addressed in many existing task mapping and scheduling solutions for high-performance computing, namely, the assignment of tasks to processing units (task mapping) and the execution sequence of tasks assigned to the same processing units (task scheduling) [19]. As DAG scheduling problems are NP-complete in general [25], these proposed solutions are generally heuristic algorithms that make tradeoffs between schedule performance and computational complexity. Existing task mapping and scheduling solutions for wired networks consider processing units interconnected via different network topologies including tree networks [20] [34] [35], hypercubes [48] [6] [71], and mesh networks [18] [61]. In these networks, one processing unit may have dedicated connections with several neighboring nodes.

However, wireless communication have different constraints than communication in wired networks, which hinders direct implementation of existing solutions for

high-performance computing in WSNs. Different from wired networks, nodes in wireless networks generally share a common wireless channel, and communications of neighboring nodes may interfere each other. In single-hop wireless networks without "hidden nodes" or "exposed nodes" [31] [70], collision avoidance imposes a stringent constraint on task schedules. In such settings, the schedules must ensure that there are no simultaneous communications in networks, which makes most task mapping and scheduling solutions for wired networks impractical in wireless networks. For the case of multiple-hop wireless networks where there can be multiple simultaneous data transmissions, collision avoidance stands out as an even more challenging problem due to hidden and exposed nodes. Thus, task mapping and scheduling solutions in wireless networks should specifically schedule wireless communications between processing units in addition to the aforementioned task mapping and task scheduling problems in wired networks. Furthermore, most existing task mapping and scheduling solutions for wired networks do not explicitly consider energy consumption during communication and task execution as required energy is always available via wired connections. However, energy consumption efficiency is one of the most critical considerations for any WSN solution [3], and should explicitly be considered across all layers of WSNs.

The following problems must be solved to enable collaborative in-network processing in WSNs while considering energy consumption:

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

1.2 Outline

In this dissertation, large-scale WSNs composed of a large number of homogeneous wireless sensors are considered, where sensors can be multiple hops away from each other. In large-scale WSNs, monitored events may occur in remote areas that can only be detected by surrounding sensors. Thus, localized information collecting and processing are preferred in large-scale WSNs. Furthermore, grouping sensors into clusters within which sensors collaboratively process information has gained recognition to enhance network scalability, increase network throughput, as well as to conserve energy consumption and improve network lifetime in large-scale WSNs [3] [54] [68]. Thus, we assume a hierarchical WSN architecture where sensors are grouped into clusters, and information collected by sensors are first processed within clusters and then transmitted by cluster heads to base stations. Consequently, cluster-based task mapping and scheduling solutions are desirable to provide the demanded computation power for energy-efficient in-network processing in large-scale WSNs. In this dissertation, applications are assumed to be executed within clusters either periodically or upon the occurrence of triggering events. These applications are represented by DAGs which leads to *application-independent* solutions. Since cluster heads can easily collect cluster members' information and play coordinating roles among cluster members, our proposed task mapping and scheduling solutions are *centralized* scheduling solutions running on cluster heads. Considering that applications in WSNs are iteratively executed for a relatively long period, and application execution loads are fairly predictable for homogeneous sensor nodes, the developed task mapping and scheduling solutions are *static* algorithms: Schedules are first calculated offline by cluster heads. At the beginning of each execution period or when triggering events occur, sensors collect and process information according to the predetermined schedules. *Schedule adaptivity* is

also addressed in the dissertation where schedules are adjusted based on previously calculated schedules when sensor failures occur. Four main components of this dissertation are listed as follows:

1. A task mapping and scheduling solution for energy-constrained applications in single-hop clusters is presented in Chapter 3. To guarantee network lifetime in energy-scarce WSNs, a practical approach is to restrict the energy expenditure for certain duration of network operation, which consequently imposes energy constraints on application executions. To enhance information processing capacity, the proposed solution aims to minimize schedule lengths subject to energy consumption constraints. The solution assumes a single-hop environment, and intends to form a basis for the following more general solutions in multi-hop network environments.

2. Real-time applications are considered in Chapter 4 for single-hop clustered WSNs. In this chapter, the tradeoff between schedule length and energy consumption is tackled with a different perspective from that in Chapter 3. Realizing that many applications such as multimedia applications have inherent real-time requirements, the solution in Chapter 4 is presented with the objective of providing deadline guarantees with minimum balanced energy consumption.

3. Large-scale WSNs may be grouped into multi-hop clusters, which demands general task mapping and scheduling solutions for multi-hop environments. In Chapter 5, a task mapping and scheduling solution for multi-hop clustered WSNs is developed based on the work presented in Chapter 3 and 4. The solution in Chapter 5 aims to minimize energy consumption subject to deadline constraints.

4. In WSNs, sensors are prone to failures. In case of sensor failures, previously calculated schedules may not be feasible solutions. In such cases, WSN functionality needs to be recovered as soon as possible. Instead of rescheduling from scratch, which can be time consuming, low-complexity recovery algorithms are desirable to quickly recover from sensor and communication failures. In Chapter 6, two sensor failure handling algorithms are proposed for single-hop and multi-hop clusters.

CHAPTER 2

BACKGROUND

In this chapter, prior work relevant to the research problems addressed in this dissertation are presented.

2.1 Related Work

Depending on applications and network scale, task mapping and scheduling can be achieved either network-wide or in a localized manner in WSNs. In small-scale WSNs, it is plausible to take a global approach to optimize the system performance at the network level. In [39], the DFuse framework is proposed to dynamically assign data fusion tasks to sensors in a WSN. The design objective of DFuse is to find mapping from task graph vertices to network nodes with balanced energy consumption. **T**ask **A**llocation among clusters in **C**luster-based **S**ensor **N**etworks (TACSN) is discussed in [66]. The objective of TACSN is to maximize network lifetime via task allocation, which is modeled as a nonlinear optimization problem with constraints such as application deadlines. However, neither DFuse nor TACSN explicitly addresses communication scheduling in WSNs.

Local information processing is more scalable for large-scale WSNs, where events of interest generally occur in remote areas that only local sensors can detect. Localized task mapping and scheduling problems in WSNs have be studied in the literature recently. These

solutions consider applications executed independently within clusters composed of geographically close nodes, following locally generated schedules. Through collective result of local optimizations, these solutions aim to achieve system level optimization.

Collaborative Resource Allocation (CoRAl) is presented in [26] to dynamically allocate resources such as bandwidth and CPU time among multiple periodic applications in fully-connected WSNs. CoRAl considers end-to-end applications composed by a chain of tasks assigned on different sensors. Tasks of an end-to-end application are executed in a pipelined manner.

Subject to resource availability and temporal constraints, CoRAl aims to maximize network utility by adjusting application execution frequencies. In CoRAl, the wireless channel is modeled as a virtual node, and the network bandwidth is allocated in the same manner as sensor CPU time allocation. CoRAl achieves its objective by iteratively executing the following steps until the schedule converges: First, the task execution frequencies on each sensor are locally optimized subject to application execution frequency upper-bounds, whose initial values are set to be infinite. Then the execution frequency upper-bound of each application is reevaluated based on the updated task frequencies and bandwidth allocation. On each node, an extended version of the SLSS algorithm [50] is implemented to compute local optimal frequencies subject to node utility constraints. Different from the original SLSS algorithm, the extended SLSS algorithm in [26] takes each task's application execution frequency upper-bound into consideration. After each iteration of local optimization, the upper-bound frequency of each application is calculated. Assume the **leader** ld_i and **bottleneck** bn_i of application T_i are tasks whose frequency f_i^{ld} and f_i^{bn} are highest and lowest among all tasks of T_i, respectively. The frequency upper-bound of T_i is updated as $f_i^{max} = f_i^{bn} + (f_i^{ld} - f_i^{bn})\sigma$, where σ is the factor controls frequency convergence speed.

The optimization procedure terminates when the weighted difference between leaders' and bottlenecks' frequencies converges.

According to the simulation results, solutions provided by CoRAl are comparable to the optimal solutions obtained by the non-linear optimization tool of Matlab. On the other hand, CoRAl has a much higher execution speed than the Matlab tool. However, in CoRAl, tasks of applications are assumed to be already assigned on sensors, and task mapping remains an open problem. Furthermore, energy consumption is not explicitly considered in [26], which is a fundamental problem in WSNs.

Distributed Computing Architectures (DCA) are proposed in [64] and [40], where low-level tasks are executed on sensing sensors and high-level processing tasks are offloaded to cluster heads. However, processing high level tasks can still exceed the computation capacity of cluster heads. Even with more powerful sensors as cluster heads [40], overloading "hot spot" sensors with extensive computation and communication events can quickly deplete the sensors' battery supply, exposing the WSN to the single-point-failure problem and shortening the network lifetime. Furthermore, application-specific design of these solutions limit their implementation for generic applications.

Real-time task mapping and scheduling heuristics are presented in [53] for heterogeneous mobile ad hoc grid environments. Six static heuristics are presented to minimize energy consumption subject to deadline constraints in an ad hoc grid. The communication model adopted in [53] assumes individual channels for each node and concurrent data transmission and reception capability for every node. However, in large-scale WSNs that are composed of hundreds of sensors, there is not enough network resource to allocate an individual channel for each sensor. Furthermore, concurrent data transmission and reception capacity requires two wireless transceivers working on different channels on a sensor,

which requires even more network resources. Therefore, the communication model in [53] is not well-suited for large-scale WSNs, and the algorithms presented cannot be directly applied in large-scale WSNs.

An Energy-balanced Task Allocation (EbTA) solution is presented in [69]. EbTA assumes single-hop clustered homogeneous WSNs with multiple wireless channels, where sensors are equipped with Dynamic Voltage Scaling (DVS) enabled processors. EbTA considers real-time applications composed by inter-dependent tasks. The design objective of EbTA is to map and schedule applications tasks to sensors with minimal balanced energy consumption subject to deadline constraints. In [69], applications are represented with DAGs, and scheduling is formulated as an Integer Linear Programming (ILP) problem. The exclusive wireless channel access feature is incorporated as an additional constraint in the ILP formulation.

As the formulated ILP problem is computationally costly to solve, a three-phase heuristic is proposed in [69] to provide a practical solution. In Phase 1, tasks are grouped into clusters to minimize overall application execution time assuming infinite number of sensors. Each task first constitutes a cluster by itself. Then all communication tasks are examined in a non-increasing order of their data volume. For each communication event $e(i, j)$ between computation task T_i and T_j, the clusters containing T_i and T_j are merged if it leads to shorter application execution time. When evaluating application execution time, communication events are scheduled to the channel using the First Come First Served (FCFS) policy. In Phase 2, the task clusters from Phase 1 are assigned to sensor nodes with the objective of minimizing the maximum energy expenditure among all sensors. The task clusters from Phase 1 are first sorted in a non-decreasing order of energy dissipation, and stored in a queue Π. The clusters in Π are then assigned to the sensor with the minimum

normalized energy consumption (task execution energy consumption normalized by sensor residue energy, *norm-energy* for short). Every time after a task cluster is assigned to a sensor, the norm-energy of the sensor is updated. This procedure repeats until all task clusters are assigned. Finally, a DVS heuristic is presented for Phase 3 to decrease energy consumption by iteratively adjusting the CPU voltage level of each task. In each DVS adjustment iteration, a *critical node* that has the highest norm-energy ε is selected. Among the tasks assigned on the critical node, a critical task is selected and its CPU speed is adjusted. For a critical task v_c, ε is decreased the most by reducing the CPU speed of executing v_c to the next level. Every time when the CPU speed is adjusted, the application schedule will be iteratively adjusted accordingly to meet inter-task dependency constraints.

EbTA is one of the first work that address task allocation in WSNs that considers both communication and computation tasks. It is shown through simulations that the three-phase heuristic achieves longer lifetime compared with the baseline without DVS. The performance of the three-phase heuristic is also found to be comparable to that of the ILP-based approach via simulations. However, the broadcast feature of wireless communication is not exploited in EbTA, which may lead to significant energy consumption savings.

2.2 Wireless Sensor Network Assumptions

The following assumptions are made regarding the wireless sensor network:

- A wireless sensor network is composed of homogeneous sensors.
- Sensors are grouped into clusters with existing clustering algorithms such as [4] [9] [30] [67].

- 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 the schedules for execution within clusters.

- Calculated schedules are used to run the associated applications according to the application requirements.

- Sensors are synchronized with one of the time synchronization methods discussed in [32] [55] [58]. The time synchronization is necessary for synchronizing scheduled task executions within clusters. Thus, only local time synchronization within clusters is required.

- Computation and communication can occur simultaneously on sensor nodes as supported by various platforms including MICA2DOT running TinyOS [2].

- Communication within a cluster is isolated from other clusters through time division or channel hopping mechanisms such as described in [8] [56] with appropriate hardware support, eg. Chipcon CC2420 transceiver [1].

It should be noted that while the intra-cluster communication is isolated from other clusters, 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.

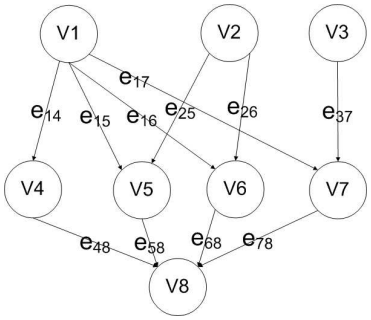


Figure 2.1: An Example DAG

2.3 Application Model

To develop an application-independent solution, we represent applications executed in clusters with Directed Acyclic Graphs (DAG). 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 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 weight of a task is represented by the number of CPU clock cycles to execute the task. Given an edge e_{ij}, v_i is called the immediate predecessor of v_j, and v_j is called the immediate successor of v_i. An immediate successor v_j depends on its immediate predecessors such that v_j cannot start execution before it receives results from all of it immediate predecessors. A task without immediate predecessors is called an *entry-task* and a task without immediate successors is called an *exit-task*. A DAG may have multiple entry-tasks and one exit-task. If there is more than one exit-task, they will be connected to a pseudo exit-task with zero computation cost. Fig. 2.1 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 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 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 *Dependency Constraint* throughout this dissertation.

2.4 Energy Consumption Model

The energy consumption of transmitting and receiving l-bit data over a distance d are defined as $E_{tx}(l, d)$ and $E_{rx}(l)$, respectively:

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

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

where d_o is the distance threshold, E_{elec} and ε_{amp} are hardware related parameters [64] [30].

In this dissertation, we assume that the sensors are equipped with the StrongArm SA1100 processor [64]. The energy consumption of executing N clock cycles with CPU clock frequency f of StrongArm SA1100 is given as:

$$f \simeq K(V_{dd} - c), \tag{2.3}$$

$$E_{comp}(V_{dd}, f) = NCV_{dd}^2 + V_{dd}(I_o e^{\frac{V_{dd}}{nV_T}})(\frac{N}{f}), \tag{2.4}$$

where V_T is the thermal voltage and C, I_o, n, K and c are processor-dependent parameters [52] [64]. The first term of Equation 2.4 denotes the switching energy consumption, which

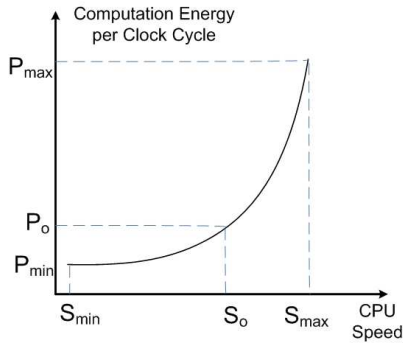


Figure 2.2: CPU Power Consumption vs CPU Speed

dominates the CPU energy consumption. The second term of Equation 2.4 represents the CPU leakage energy consumption. Given Equation 2.3, we can derive from Equation 2.4 that the CPU power consumption (energy consumption per clock cycle) is approximately proportional to V_{dd}^2. The relationship of the CPU power consumption and CPU speed is demonstrated in Fig. 2.2.

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 sleep schedules of sensors, where sensors go to sleep when no communication and computation activities are scheduled for them.

2.5 Notation

The task mapping and scheduling problem is defined as finding a set of task assignments and their execution sequences on a network such that an objective function such

as energy consumption or schedule length is minimized. Let $H^x = \{h_1^x, h_2^x, ..., h_n^x\}$ denote a task mapping and scheduling solution of an application DAG *T* on a network *G*, where x is the solution space index. 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} and f_{i,m_k} represent the start time and finish time of v_i, and t_{i,m_k} and c_{i,m_k} represent the execution length and energy consumption of v_i on node m_k, respectively. The following set of notations are used throughout this dissertation:

- $pred(v_i)$ 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 $[st, ft]$.

CHAPTER 3

ENERGY-CONSTRAINED TASK MAPPING AND SCHEDULING IN SINGLE-HOP CLUSTERED WIRELESS SENSOR NETWORKS

As introduced in Chapter 1, in-network processing is essential for energy-efficiency WSNs applications. On the other hand, the limited energy supply in WSNs [3] imposes stringent energy consumption constraints on in-network processing. WSNs may expect a lifetime ranging from months to years without replacing batteries. To guarantee network lifetime, a practical approach is to restrict the energy expenditure of applications in WSNs. Thus, it is desirable to develop a general solution to provide the computation capacity required by in-network processing subject to energy consumption constraints.

In this chapter, we present a localized task mapping and scheduling solution for energy-constrained applications in WSNs. We consider a single-hop clustered homogeneous WSN. Our proposed solution, **E**nergy-**co**nstrained Task **Map**ping and **S**cheduling (EcoMapS), aims to map and schedule the tasks of an application with the minimum schedule length subject to energy consumption constraints.

Assume that $CommEng(m_k)$ represents the communication energy consumption of a node m_k including data transmitting, receiving and forwarding. The design objective of EcoMapS is to finding a schedule $H^o \in \{H^x\}$ that has the minimum schedule length subject to energy consumption constraints, which can be formulated as follows:

$$Find\ H^o = arg\min length(H), \tag{3.1}$$

$$where\ length(H) = \max_{i,k} f_{v_i,m_k}, \tag{3.2}$$

$$\text{subject to } energy(H) = \sum_{i,k} c_{v_i,m_k} + \sum_{k} CommEng(m_k) \leq EB, \tag{3.3}$$

where $length(H)$ and $energy(H)$ are the schedule length and overall energy consumption of H, respectively, and EB is the energy consumption constraint (also referred to as *Energy Budget*). In EcoMapS, communication and computation are jointly scheduled. A network model and communication scheduling algorithm are presented to exploit the broadcasting nature of wireless networks. Schedules are computed by cluster heads for the entire cluster using our proposed EcoMapS solution. EcoMapS is based on the high-level application model that describes the task dependencies through DAGs. Therefore, EcoMapS can be used with arbitrary applications.

Different from existing work, EcoMapS has the following salient properties:

- Task mapping and task scheduling are considered simultaneously.
- The single-hop wireless channel is modeled as a virtual node, and applications are represented to reflect the broadcast nature of wireless communication.
- Communication and computation events are jointly scheduled.
- Based on realistic energy models, EcoMapS aims to provide energy consumption guarantees with minimum schedule lengths.

3.1 Outline of The Proposed EcoMapS Solution

In our proposed EcoMapS solution, tasks are assigned to sensors, the execution sequence of tasks are decided, and communications between sensors are scheduled with respect to the *Dependency Constraint*. EcoMapS aims to minimize schedule lengths subject to energy consumption constraints. The scheduling algorithms are 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, and schedules will be regenerated by the new cluster head.

In the following sections, the main components of our proposed EcoMapS solution, namely, wireless channel modeling and Hyper-DAG extension, communication scheduling algorithm, and the extended CNPT algorithm [28] and Min-Min algorithm [53] (referred to as E-CNPT and E-MinMin, respectively) are presented. For task mapping and scheduling, either *E-CNPT* or *E-MinMin* is executed as the schedule search engine to find the optimal schedule. The original CNPT and Min-Min algorithms are designed for traditional parallel processing in wired networks. To extend CNPT and Min-Min for WSNs, we developed a *communication scheduling algorithm* based on the *wireless channel model* and the *Hyper-DAG representation* of applications. The communication scheduling algorithm is embedded in the execution of E-CNPT and E-MinMin to satisfy the *Dependency Constraint*.

3.2 Wireless Channel Modeling and Hyper-DAG Extension

In a single-hop cluster, there can be only one transmission on the wireless channel at a given time. Similar to that in [26], the wireless channel can be modeled as a virtual node $\mathcal{C}$ that executes one communication task at any time instance. Hence, a cluster can be

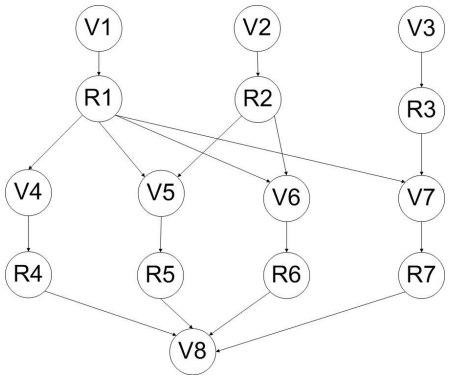


Figure 3.1: The Hyper-DAG Representation of the DAG in Fig. 2.1

modeled as a star-network where all sensors only have connections with the virtual node $\mathcal{C}$. The communication latency between sensor nodes and $\mathcal{C}$ can be considered zero since all wireless communications are accounted for by the tasks executed on $\mathcal{C}$. Assuming that the cluster has p sensors that are denoted as $M = \{m_k\}$ $(0 \leq k < p)$, a cluster can be represented by a connected, undirected graph *G = (*M'*,N)*, where the set $M' = M \cup \{\mathcal{C}\}$, and the set *N* denotes the links between the nodes of M'. With the virtual node representation of $\mathcal{C}$, communication contention can be effectively avoided by serially scheduling communications on $\mathcal{C}$. Another important advantage of the channel model is its suitability to represent the broadcast nature of wireless communication. When a node in a single hop cluster transmits information, it is potentially received by all nodes in the cluster. The broadcast nature of the wireless channel can be leveraged to relay information generated by a task to all its immediate successors in a single transmission rather than multiple, sequential transmissions. This approach reduces schedule lengths as well as communication energy consumption.

To implement this channel model, communication events between computation tasks should be explicitly represented in task graphs. Thus, we extend a DAG as follows: For a task v_i in a DAG, we replace the edges between v_i and its immediate successors with a $net\ R_i$. R_i represents the communication task to send the result of v_i to its immediate successors in the DAG. The weight of R_i equals to the result data volume of v_i. This extended DAG is a hypergraph and is referred to as *Hyper-DAG*. With the Hyper-DAG representation, exclusive channel access constraints and broadcast delivery of results are incorporated into task dependency in a compact way. 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 Tasks*}. The example of converting the DAG in Fig. 2.1 to a Hyper-DAG is shown in Fig. 3.1.

With Hyper-DAGs, communication events between computation tasks are explicitly represented in task graphs. Based on the Hyper-DAG representation, Equation 3.3 is rephrased as follows:

$$\text{subject to } \sum_{v_i \in V,k} c_{v_i,m_k} + \sum_{v_i \in R,k} c_{v_i,m_k} = \sum_{v_i \in V',k} c_{v_i,m_k} \leq EB, \tag{3.4}$$

where c_{v_i,m_k} of $v_i \in R$ is the energy consumption of node m_k for sending, receiving, or forwarding communication task v_i through the wireless channel, and the $\sum_{v_i \in R,k} c_{v_i,m_k}$ equals the communication energy consumption of schedule H^o.

In the Hyper-DAG scheduling problem, the *Dependency Constraint* is rephrased as follows: If a computation task v_j scheduled on node m_k depends on a communication task v_i scheduled on another node, a copy of 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.

It should be noted that the channel model presented above assumes a single-hop clustered environment. However, this model can be generalize to multi-hop networks by relaxing the constraints and taking the inference avoidance into consideration. A general solution in multi-hop clustered WSNs will be presented in Chapter 5.

3.3 Communication Scheduling Algorithm

To meet the *Dependency Constraint* in Hyper-DAG scheduling, communication between nodes is required if a computation task depends on a communication task assigned on another node. Thus, we present our communication scheduling algorithm in this section. As we shall see in Section 3.4, the communication scheduling algorithm is integrated into the execution of our Hyper-DAG task mapping and scheduling algorithms, E-CNPT and E-MinMin.

Based on the Hyper-DAG and the channel model presented in Section 3.2, scheduling communication between single-hop neighbors is equivalent to first duplicating a communication task from the sender to $\mathcal{C}$, and then from $\mathcal{C}$ to the receiver. If the requested communication task has been scheduled from the sender to another node before, the receiver will directly duplicate the communication task from $\mathcal{C}$. This process is equivalent to receiving broadcast data, which can lead to significant energy saving compared with multiple unicasts between the sender and the multiple receivers. The detailed description of the communication scheduling algorithm is shown in Fig. 3.2, where Step 2-15 stands for originating a new communication from m_s to m_r, and Step 18-21 represents reception of a broadcast data. Compared with originating a new communication, the broadcast reception method leads to energy saving of one data transmission for each additional data reception.

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
CommTaskSchedule(v_i**,**m_s**,**m_r**):**

---

1. Find a copy of v_i: $v_i^c \in T(\mathcal{C})$ /*v_i scheduled on $\mathcal{C}$?*/
2. **IF** v_i^c does not exist /*NO, schedule v_i transmission from scratch*/
3. Find $v_i \in T(m_s)$
4. Find time interval [st,ft]:
5. $T_{st}^{ft}(\mathcal{C}) = \emptyset$
6. $ft - st \geq t_{v_i,\mathcal{C}}$
7. $st \geq f_{v_i,m_s}, st = \min$
8. Schedule a copy of v_i to $\mathcal{C}$:
9. $T(\mathcal{C}) \leftarrow T(\mathcal{C}) \cup \{v_i^c\}$
10. $s_{v_i^c,\mathcal{C}} \leftarrow st$
11. Update the energy consumption of m_s
12. Schedule a copy of v_i^c to m_r:
13. $T(m_r) \leftarrow T(m_r) \cup \{v_i^r\}$
14. $s_{v_i^r,m_r} \leftarrow f_{v_i^c,\mathcal{C}}$
15. Update the energy consumption of m_r
16. Return
17. **ELSE** /*YES, receive broadcast data*/
18. Schedule a copy of v_i^c to m_r:
19. $T(m_r) \leftarrow T(m_r) \cup \{v_i^r\}$
20. $s_{v_i^r,m_r} \leftarrow f_{v_i^c,\mathcal{C}}$
21. Update the energy consumption of m_r
22. Return

Figure 3.2: Communication Task Scheduling Algorithm

Under our communication scheduling algorithm, one data transmission may reach multiple receivers.

3.4 Task Mapping and Scheduling Algorithms

In EcoMapS, the tasks of Hyper-DAGs are mapped and scheduled on sensors. During task mapping, several constraints have to be satisfied. These constraints together with the *Dependency Constraint* are represented as follows.

- A computation task can be assigned only to sensor nodes, ie., $\forall v_i \in V : t_{v_i,\mathcal{C}} = \infty, c_{v_i,\mathcal{C}} = \infty$
- A communication task can be assigned to sensors or $\mathcal{C}$
- A communication task assigned to $\mathcal{C}$ stands for an ongoing data communication. Its execution time equals its communication length. The corresponding data transmission and reception energy consumption are accounted for by the sender and receivers following Equation 2.1 and 2.2, respectively.
- A communication task assigned to a sensor denotes data stored in sensor memory, and is ready for processing on the same node. Thus, its execution time and energy length are zero.
- A non-entry computation task assigned on a sensor must have all its immediate predecessors available before it can start execution, i.e., if $v_i \in V$ and $pred(v_i) \neq \emptyset$, then $pred(v_i) \subset T(m(v_i))$ and $s_{v_i,m(v_i)} \geq \max f_{pred(v_i),m(v_i)}$

To meet the *Dependency Constraint* during task mapping and scheduling, if a computation task depends on a communication task assigned on another sensor node, the *communication scheduling algorithm* will be executed to duplicate the absent communication task. With the Communication Scheduling Algorithm and the task mapping constraints presented above, task mapping and scheduling in single-hop wireless networks can be tackled

as a generic task mapping and scheduling problem with additional constraints. This problem is NP-complete in general [25] and heuristic algorithms are needed to obtain practical solutions. In this section, E-CNPT and E-MinMin algorithms are presented as the schedule search engine of EcoMapS with the objective of minimizing schedule lengths subject to energy constraints.

Before presenting the E-CNPT and E-MinMin algorithms, we first introduce a concept of *computing sensor*: A computing sensor is a sensor that can execute non-entry tasks as well as entry-tasks. The concept of computing sensor is an intuitive extension of DCA in [64], where only one sensor in a cluster, i.e., the cluster head, can execute high level tasks. In EcoMapS, there can be more than one computing sensors to speed up execution. However, this approach generally consumes more energy with more computing sensors because of the increased volume of communication between the sensors. Thus, the increment of number of computing sensors must be bounded by energy consumption constraints. In our EcoMapS, E-CNPT and E-MinMin will iteratively search the optimal schedule with different number of computing sensors subject to energy constraints.

3.4.1 E-CNPT Algorithm

The list-scheduling CNPT algorithm [28] is extended and implemented as one of the schedule search engine in EcoMapS, and is denoted as E-CNPT. The objective of E-CNPT is to minimize schedule lengths subject to energy consumption constraints. The strategy of E-CNPT is to assign the tasks along the most critical path first to the nodes with earliest execution start times. By adjusting the number of computing sensors in each scheduling iteration and choosing the schedule with the minimum schedule length under the energy

consumption constraint, the design objective of E-CNPT is achieved. Similar to CNPT, E-CNPT also has two stages: *listing stage* and *sensor assignment stage*. In the *listing stage*, tasks are sequentialized into a queue L such that the most critical path comes the first and a task is always enqueued after its immediate predecessors. In the *sensor assignment stage*, the tasks will be dequeued from L and assigned to the sensors with the minimum execution start time. Several scheduling iterations will be run in the sensor assignment stage with different number of computing sensors, and only one schedule is chosen as the solution according to the design objective. The *listing stage* and *sensor assignment stage* of E-CNPT are introduced individually as follows.

Listing Stage: The Listing Stage of E-CNPT is similar to that of CNPT [28]. In the Listing Stage of E-CNPT, the Earliest Start Time $EST(v_i)$ of task v_i is first calculated for each vertex by traversing the Hyper-DAG downward from the entry-tasks to the exit-task. The Latest Start Time $LST(v_i)$ of task v_i is then calculated in the reverse direction. During the calculation, the entry-tasks have *EST* = 0 and the exit-task has *LST* = *EST*. The formulas to calculate *EST* and *LST* are as follows:

$$EST(v_i) = \max_{v_m \in pred(v_i)} \{EST(v_m) + t_m\}, \tag{3.5}$$

$$LST(v_i) = \min_{v_m \in succ(v_i)} \{LST(v_m)\} - t_i, \tag{3.6}$$

where t_i equals to the execution length on sensor nodes if $v_i \in V$ or to the execution length on $\mathcal{C}$ if $v_i \in E$. Then, the Critical Nodes (CN) are pushed into the stack *S* in the decreasing order of their *LST*. Here, a CN vertex is a vertex with the same value of *EST* and *LST*. Consequently, if *top(S)* has un-stacked immediate predecessors, the immediate predecessor with the minimum *LST* is pushed into the stack; otherwise, *top(S)* is popped and enqueued into a queue *L*. The Listing Phase ends when the stack is empty. After the Listing Phase,

the task graph is sequentialized into *L* and is ready for the Sensor Assignment Phase. It should be noted that the EST and LST are for the purpose of evaluating the critical path of a Hyper-DAG, and EST and LST do not represent the actual execution start time of tasks.

Sensor Assignment Stage: The design objective of our algorithm is to minimize schedule lengths subject to energy consumption constraints. Different from CNPT, E-CNPT iteratively searches the schedule space with different number of computing sensors in the Sensor Assignment Stage. Among these schedules, the one with the minimum schedule length under the energy consumption constraint is chosen as the solution. If no schedule meetings the energy constraint, the best effort is made by choosing the one with the minimum energy consumption. The detailed description of the E-CNPT algorithm is shown in Fig. 3.3.

In the E-CNPT algorithm, SingleCNPT(L,q) is a single round of task scheduling that schedules the tasks in L with q computing sensors. It should be noted that the parameter q is just the upper bound of the number of computing sensors that can be involved in the schedule. The actual number of computing sensors can be smaller than q depending on applications and scheduling algorithms. The core of SingleCNPT(L,q) is the extended CNPT *processor assignment algorithm* embedded with our communication scheduling algorithm. The basic strategy of the algorithm is to assign tasks to the sensor with the minimum Earliest Execution Start Time (EEST). During task scheduling, *Dependency Constraint* has to be satisfied via communication scheduling. SingleCNPT(L,q) is described in Fig. 3.4, where $EAT(m_k)$ is the Earliest Available Time of node m_k, and EEST(v_i, m_k) is the Earliest Execution Start Time of v_i on sensor m_k. Different from EST, EEST represents the actual execution start time of a task if assigned on a sensor node.

Input: Task queue L ; number of available sensors in the cluster p; energy budget EB
Output: Schedule H^o of tasks in L with minimum schedule length under energy budget constraint
E-CNPT Algorithm:

---

1. $L^o \leftarrow \infty$ /*optimal schedule length*/
2. $E_{min} \leftarrow \infty$ /*minimum energy consumption*/
3. **FOR** q = 1 to p /* search computing sensor space*/
4. H = SingleCNPT(L,q)
6. **IF** $energy(H) < E_{min}$
7. $E_{min} \leftarrow energy(H)$
8. $H_{min} \leftarrow H$ /*$eng(H_{min}) = \min$*/
9. **IF** $energy(H) \leq EB$ and $length(H) < L^o$
10. $L^o \leftarrow length(H)$
11. $H^o \leftarrow H$ /*optimal schedule*/
12. **IF** $E_{min} \leq EB$
13. Return H^o
14. **ELSE**
15. Return H_{min}

Figure 3.3: EcoMapS: E-CNPT Algorithm

Input: task queue L; number of computing sensors q
Output: Schedule of tasks in L
SingleCNPT Algorithm:

---

while L is not empty
1. Dequeue v_i from L
2. **IF** $v_i \in R$ /*communication task*/
3. Assign v_i to node $m(pred(v_i))$
4. **ELSE IF** $pred(v_i) = \emptyset$ /*entry-tasks*/
5. Assign v_i to node m_i^o with min $EAT(m_i^o)$
6. **ELSE** /* non-entry computation tasks*/
7. **FOR** all computing sensors $\{m_k\}$
8. Calculate EEST(v_i, m_k):
9. **IF** $pred(v_i) \subseteq T(m_k)$ /*dependency constraint satisfied*/
10. EEST(v_i, m_k) $\leftarrow \max(EAT(m_k), f_{pred(v_i),m_k})$
11. **ELSE** /*communication between sensors is needed*/
12. **FOR** $v_n \in pred(v_i) - T(m_k)$
13. CommTaskSchedule(v_n,$m(v_n)$,m_k)
14. EEST(v_i, m_k) $\leftarrow \max(EAT(m_k), f_{pred(v_i),m_k})$
15. Keep the schedule with minimum (EEST(v_i, m^o))
16. Schedule v_i on m^o: $s_{v_i,m^o} \leftarrow EEST(v_i, m^o)$

Figure 3.4: EcoMapS: SingleCNPT Algorithm

3.4.2 E-MinMin Algorithm

The Min-Min algorithm is reported of satisfying performance with relatively low complexity [11] [53]. Thus, we extend and implement the Min-Min algorithm in [53] as the schedule search engine of EcoMapS. The extended Min-Min algorithm aims to minimize scheduling lengths subject to energy consumption constraints, and is referred to as the E-MinMin algorithm.

Similar to E-CNPT, E-MinMin also searches for a schedule with the optimal number of computing sensors that has the smallest schedule length subject to the energy consumption constraint. The E-MinMin's algorithm of searching for the optimal number of computing sensors is the same as the **E-CNPT Algorithm** in Section 3.4.1 except that the input of the E-MinMin algorithm is the Hyper-DAG instead of the task queue L, and the core of the searching algorithm is the **SingleMinMin** instead of the **SingleCNPT**.

We now introduce the procedure SingleMinMin(Hyper-DAG,q) that schedules the tasks of the Hyper-DAG with q computing sensors. The core of the SingleMinMin algorithm is the fitness function. For each task-node combination (v,m), the fitness function $fit(v, m, \alpha)$ indicates the combined cost in time and energy domain of assigning task v to node m, where α is the weight parameter trading off the energy consumption cost for the time cost. In the SingleMinMin 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 [53], the following notations are introduced first:

- $MFT(v, m)$ is the maximum finish time of the tasks assigned prior to task v.
- $f_{v,m}$ is the scheduled finish time of v on m.

- $PE(v, m)$ is the energy consumption of the application schedule after assigning v to m, which includes the computation energy consumption and communication energy consumption on all nodes so far.

- $NPT(v, m)$ is the normalized partial execution time of assigning v on m: $NPT(v, m) = \frac{f_{v,m}}{MFT(v,m)}$.

- $NPE(v, m)$ is the normalized energy consumption of assigning v on m: $NPE(v, m) = \frac{PE(v,m)}{EB}$.

Thus, the fitness function of assigning v on m is defined as:

$$fit(v, m, \alpha) = \alpha \cdot NPE(v, m) + (1 - \alpha) \cdot NPT(v, m). \tag{3.7}$$

The SingleMinMin Algorithm is presented in Fig. 3.5. In the description of SingleMinMin, a "mappable" task is either an entry-task or a task that has all immediate predecessors already been assigned, and the "mappable task list" is the list that contains currently mappable tasks of the Hyper-DAG. We sweep α values in $\Delta\alpha$ increments to find the best solution, where $\Delta\alpha$ is the α sampling step. For schedules with different α values, the schedule with the minimum schedule length under the energy consumption constraint EB is chosen as the optimal solution among these candidate schedules. If none of the candidate schedules meets EB, the one with the smallest energy consumption is chosen (best-effort solution).

3.5 Computational Complexity Analysis

Assume that the application T is represented as $T = (V, E)$, $|V| = v$, $|E| = e$, the number of entry-tasks is f, and the cluster has p sensor nodes. Thus, the Hyper-DAG is $T' = (V', E')$, where $|V'| = 2v$ and $|E'| = 2e$.

Input: Hyper-DAG; number of computing sensors: q
Output: Schedule H^o of tasks in Hyper-DAG
SingleMinMin Algorithm:

---

1. **FOR** $\alpha = 0$; $\alpha \leq 1.0$; $\alpha += \Delta\alpha$ /*scan α value*/
2. **FOR** entry-tasks v_i /*First assign entry tasks*/
3. Assign v_i on node m_i^o with min $EAT(m_i^o)$
4. Assign $succ(v_i)$ on m_i^o
5. Initialize the mappable task list *L*
6. **WHILE** L is not empty
7. **FOR** task $v_i \in L$ /*Search all task-sensor combinations*/
8. **FOR** all computing sensor m_k
9. **IF** $pred(v_i) \not\subseteq T(m_k)$
10. **FOR** $v_n \in pred(v_i) - T(m_k)$
11. **CommTaskSchedule**(v_n,$m(v_n)$,m_k)
12. Assign v_i to m_k, calculate $fit(v_i, m_k, \alpha)$
13. Find m_i^o: $fit(v_i, m_i^o, \alpha)$ is minimum
14. Find task/sensor pair (v,m): $fit(v, m, \alpha)$ is minimum
15. Assign v to m, remove v from *L*
16. Assign $succ(v)$ on m
17. Update *L* with new unassigned mappable tasks.
18. Among all schedules with different values of α
19. **IF** $\exists H : energy(H) \leq EB$ with $\min length(H)$
20. Return H /*optimal solution*/
21. **ELSE**
22. Return $H : energy(H)$ is minimum /*best-effort solution*/

Figure 3.5: EcoMapS: SingleMin Algorithm

3.5.1 Computational Complexity of EcoMapS with E-CNPT

The time complexity of EcoMapS with E-CNPT is analyzed as follows:

- Listing Stage of E-CNPT: similar to CNPT [28], the complexity is $O(v + e)$.
- SingleCNPT: the communication tasks have complexity of $v \cdot O(1) = O(v)$, the entry-tasks have complexity of $f \cdot O(p) = O(fp)$, other non-entry computation tasks have complexity of $(v - f) \cdot O(p) \cdot O(e/v)$. Hence, the overall complexity of SingleMapSchedule is $O(v) + O(fp) + (v - f) \cdot O(p) \cdot O(e/v)$. For the worst case, $e = O(v^2)$ and $f = O(v)$, thus the complexity of SingleMapSchedule is $O(pv^2)$ for the worst case.
- EcoMapS with E-CNPT: the SingleCNPT algorithm will be called $O(p)$ times. Thus, the complexity of the whole algorithm is $O(v+e)+O(p) \cdot O(v^2p)$ = $O(p^2v^2)$ for the worst case.

3.5.2 Computational Complexity of EcoMapS with E-MinMin

The time complexity of EcoMapS with E-MinMin is analyzed as follows:

- SingleMinMin: the complexity of SingleMinMin is dominated by the loop starting from Step 5, which is executed $O(v)$ times. Similarly to SingleCNPT, the complexity of the loop starting from Step 6 has the complexity of $O(v) \cdot O(p) \cdot O(e/v)$ = $O(pe)$. Thus, SingleMinMin has the complexity of $O(pv^3)$ for the worst case.
- EcoMapS with E-MinMin: Similar to the analysis of E-CNPT, the complexity is $O(p) \cdot O(pv^3)/\Delta\alpha$ = $O(p^2v^3/\Delta\alpha)$ for the worst case.

From the analysis above, the complexity of the EcoMapS with E-MinMin is higher than that of the EcoMapS with E-CNPT with the order of v for a fixed value of $\Delta\alpha$.

3.6 Simulation Results

The performance of our EcoMapS solution with E-CNPT and E-MinMin algorithms is evaluated through simulations. The performance of an extended version of DCA [64] is 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 simplified distributed video surveillance application example. To further evaluate EcoMapS performance, simulations are run on arbitrary applications with randomly generated DAGs. Our simulator is programmed in C++ language and executed in Linux environment. Our simulations with random DAGs study the following scenarios:

- Effect of the $\Delta\alpha$ parameter of the E-MinMin algorithm
- Effect of energy consumption constraints
- Effect of number of tasks in applications
- Effect of inter-task dependency
- Effect of communication load
- Evaluation of energy consumption balance
- Comparison of the heuristic execution time

In these simulations, we observe energy consumption and schedule length metrics unless otherwise stated. The energy consumption includes computation and communication energy expenditure of all sensors. The schedule length is defined as the finish time of the

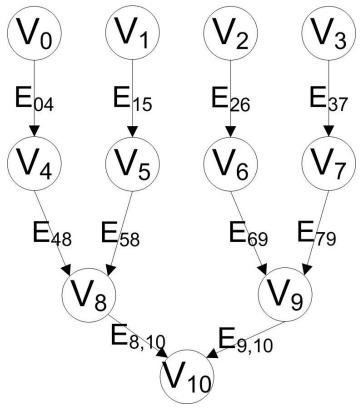


Figure 3.6: The Video Surveillance Example DAG

exit-task of an application. The simulation results presented in this section correspond to the average of five hundred independent runs.

3.6.1 Simulation Parameters

In our simulation study, the bandwidth of the channel is set to 1Mb/s and the transmission range r = 10 meters. We assume that there are 10 sensors in a cluster. The sensors are equipped with the StrongARM SA-1100 microprocessor with the CPU frequency be 100 MHz. The parameters of Equation 2.1, 2.2, 2.4 are in coherence with [52], [64], [30] as follows: E_{elec} = 50 nJ/b, ε_{amp} = 10 pJ/b/m^2, V_T = 26 mV, C = 0.67 nF, I_o = 1.196 mA, n = 21.26, K = 239.28 MHz/V and c = 0.5 V.

3.6.2 Simulation of an Example Application: Distributed Visual Surveillance

To demonstrate in-network processing in WSNs, we consider the intrusion detection system in Fig. 1.1. Here, we consider a simplified application that detects the location of intruders. Rather than sending captured images to a base station, we locally process the images and send the location information of the intruder. The in-network processing application is abstracted as the DAG in Fig. 3.6, where tasks $V_0 - V_3$ represent background subtracting and bounding box abstracting [62]. After these steps, the detected intruder is approximated with rectangles (bounding boxes) in images from each camera. The vertex coordinates of the bounding boxes and camera calibration data are passed to the next localization stage. Object locations are then estimated by "passing a viewing ray through the bottom of the object in the image and intersecting it with a model representing the terrain" [17] with data from each camera sensor. To evaluate the estimation error range, the locations of the points on the bounding boxes's 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 boxes' vertex coordinates and cameras' calibration data, and $E_{48} - E_{9,10}$ 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.

In the simulation, we assume that there is one intruder at any time instance, captured images are 128×128 8-bit gray images, the computation load of $V_0 - V_7$ is 50 KCC, the computation load of $V_8 - V_{10}$ is 1 KCC, the communication volumes of $E_{04} - E_{37}$ are 20 bytes, and the communication volumes of $E_{48} - E_{9,10}$ are 40 bytes. Two scenarios with

EB (uJ)	Metrics	DCA	E-CNPT	E-MinMin
300	Energy Consumption (uJ)	315.4	299.3	299.3
	Schedule Length (ms)	3.17	2.53	2.53
∞	Energy Consumption (uJ)	315.4	331.6	347.7
	Schedule Length (ms)	3.17	1.66	1.85

Table 3.1: EcoMapS: Simulation with the Visual Surveillance Example

energy budget $EB = 300uJ$ and $EB = \infty$ are simulated. As we will discuss in Section 3.6.3, $\Delta\alpha$ is set to be 0.1 during the simulations. According to the results shown in TABLE 3.1, both EcoMapS algorithms have better capacity to meet energy consumption constraints than DCA. With small energy budgets, the performance of E-CNPT and E-MinMin converges. When energy consumption budget is sufficiently large, E-CNPT performs the best with less energy consumption than E-MinMin for this specific application.

In the example above, sending these four 16KByte-images will consume about 0.05 J per hop. According to Table 3.1, even with infinite energy budget, the energy consumption of processing these images with E-MinMin is 347.7 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 deliver 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.

3.6.3 Simulation with Random DAGs

To evaluate EcoMapS performance for arbitrary applications, simulations are run on randomly generated DAGs. 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*. The number of each non-entry task's immediate predecessors, the computation load (in units of kilo-clock-cycle, KCC), and the resulting data volume (in units of bit) of a task are uniformly distributed over [1, *maxPred*], [300K CC $\pm 10\%$], and [800 bits $\pm 10\%$], respectively.

Effect of the $\Delta\alpha$ parameter of the E-MinMin Algorithm

We investigate the effect of different values of $\Delta\alpha$ with $\Delta\alpha = 0.2$, $\Delta\alpha = 0.1$, and $\Delta\alpha = 0.05$. Simulations are run with randomly generated DAGs. The parameters of DAGs considered for this set of simulations are *numTask* = 25, *numEntry* = 6, and *maxPred* = 3. The energy consumption and schedule length are observed for different available energy levels (also referred to as Energy Budget).

As we can see from Fig. 3.7, E-MinMin with different values of $\Delta\alpha$ performs almost the same with respect to meeting energy budget constraints. Regarding schedule lengths, E-MinMin with $\Delta\alpha = 0.05$ performs the best while E-MinMin with $\Delta\alpha = 0.2$ performs the worst. This observation is reasonable because with smaller $\Delta\alpha$, E-MinMin can search the schedule space in a more exhaustive manner and discover better solutions. However, a smaller value of $\Delta\alpha$ also leads to longer heuristic execution time, as discussed in Section 3.5. The important observation from Fig. 3.7 is that the performance difference between different $\Delta\alpha$ values is very small. We set the intermediate value $\Delta\alpha = 0.1$ for E-MinMin in all of the following simulations.

Effect of Energy Consumption Constraints

The effect of the energy consumption constraints is evaluated with randomly generated DAGs with *numTask* = 25, *numEntry* = 6, and *maxPred* = 3.

As shown in Fig. 3.8, both EcoMapS algorithms have better capability to adjust their schedules according to energy budget compared with DCA. When the energy budget is small, EcoMapS algorithms converge to solutions that use one sensor for computation, which is the default behavior for DCA. Instead of sending all sensed data to cluster heads, the EcoMapS algorithms choose one of the sensing sensor for computation, which saves energy and shortens schedule lengths. As the energy budget increases, the EcoMapS algorithms have more sensors involved in computation, which decreases schedule lengths at the cost of larger energy consumption. On the other hand, DCA cannot adjust its schedule to higher availability of energy resources. Compared with DCA, the EcoMapS algorithms can lead up to 67% schedule length improvement for this set of simulations.

Regarding the comparison of the EcoMapS algorithms themselves, both E-CNPT and E-MinMin tend to use one computing sensor with small energy budget, which leads to equal schedule lengths and energy consumption. When the energy budget is sufficiently large, E-CNPT has a slightly shorter schedule length than E-MinMin because of its better perspective of global optimization: The listing stage of E-CNPT enqueues the tasks according to the critical paths of the Hyper-DAG, while E-MinMin just locally calculates the cost of assigning a task. However, this improvement comes at a higher energy consumption cost, as shown in Fig. 3.8(a). For the scenarios with intermediate energy budgets, E-MinMin outperforms E-CNPT up to 39% in terms of schedule lengths, as shown in Fig. 3.8(b). This advantage of E-MinMin stems from its fitness function. Different from E-CNPT, which just takes time cost into account when assigning tasks with the fixed number

of computing sensors, the fitness function of E-MinMin considers time cost as well as energy consumption. Thus, E-MinMin is more likely to find a feasible schedule meeting energy constraints with a larger number of computing sensors than E-CNPT, which leads to shorter schedule lengths.

Effect of Number of Tasks in Applications

To test the effect of number of tasks in applications, three sets of simulations are run on randomly generated DAGs with 20, 25 and 30 tasks (numEntry = 6, maxPred = 3). As shown in Fig. 3.9, energy consumption and schedule lengths are dominated by the number of tasks. When the number of tasks increases, the energy consumption and schedule length of DCA increase proportionally. The EcoMapS algorithms on the other hand adapt themselves to the increasing energy budget. For the extreme scenarios with small and large energy budgets, the schedule lengths and energy consumption of the EcoMapS algorithms increase in proportion to the increment of the number of tasks. For the intermediate scenarios, the EcoMapS algorithms adapt their schedule lengths and energy consumption according to the available energy budget when the number of tasks increases. For all three scenarios, the energy consumption of E-MinMin follows energy budgets closer than E-CNPT, and the schedule length of E-MinMin is shorter than E-CNPT for the scenarios with intermediate energy budgets.

Effect of Inter-task Dependency

The inter-task dependency is determined by the in/out degree of application DAGs. Simulations with sets of DAGs with $maxPred = 3$ and $maxPred = 6$ (*numTask = 25*, *numEntry* = 6) are executed.

According to the simulation results of Fig. 3.10, the inter-task dependency has almost no effect on the performance of DCA. The robustness of DCA against inter-task dependency changes stems from the fact that inter-task dependency affects communication scheduling, and DCA has most of the tasks executed on the cluster head with less needs for communication.

Regarding the EcoMapS algorithms, increasing the in/out degree of DAGs does not introduce new communication task in the Hyper-DAG, but increases the dependency between a communication task and its immediate successors. Greater dependency degree between tasks may lead to a higher number of communication tasks scheduled on $\mathcal{C}$ and less parallelism between sensors, which leads to more energy consumption and longer schedules. Thus, when the energy budget is sufficiently large, the energy consumption of the EcoMapS algorithms increases and the schedule lengths decrease. When the energy budget is relatively tight, both of the EcoMapS algorithms use less computing sensors to meet energy constrains when the inter-task dependency increases, which decreases energy consumption and increases schedule lengths. As we can see from Fig. 3.10(b), although the performance of the EcoMapS algorithms degrade with higher inter-task dependency, the EcoMapS algorithms still outperform DCA with respect to schedule lengths subject to energy consumption constraints.

Effect of Communication Load

In task mapping and scheduling, the relationship between communication and computation load may affect the overall performance. This factor is evaluated by changing the average communication data volume with fixed average computation load. Simulations are run with randomly generated DAGs with *numTask* = 25, *numEntry* = 6, *maxPred* = 3. The three different settings of DAGs have result data volume uniformly distributed in [600bit,

$\pm 10\%$], [800bit, $\pm 10\%$], and [1000bit, $\pm 10\%$] with task computation load uniformly distributed in [300KCC, $\pm 10\%$].

As shown in Fig. 3.11, the performance of DCA and the EcoMapS algorithms are both affected by the communication load. When communication load increases, the schedule lengths increase. Compared to the EcoMapS algorithms, the performance of DCA degrades less with increasing communication load. DCA's robustness against communication load variation stems from the fact that DCA has most of its tasks executed on the cluster head. Since the execution time and energy consumption of a communication task on a sensor are zero, communication load changes will not affect these tasks' execution. On the other hand, the EcoMapS algorithms assign tasks on different sensors to speed up execution, which leads to more communication tasks scheduled on $\mathcal{C}$. Thus, the EcoMapS algorithms are affected more by the communication load changes. However, even when communication load increases, the EcoMapS algorithms still significantly outperform DCA with shorter schedule lengths subject to energy consumption constraints as shown in Fig. 3.11. Compared with E-CNPT, E-MinMin more effectively utilizes the available energy budgets (Fig. 3.11(a)), and is less affected by changes in communication load (Fig. 3.11(b)).

Evaluation of Energy Consumption Balance

The energy consumption balance is another important factor in the WSN design. In this section, the energy consumption balance of the proposed EcoMapS algorithms are evaluated and compared to the DCA algorithm. The random DAGs considered in the simulations have the parameters of *numTask* = 25, *numEntry* = 6, and *maxPred* = 3. The observed metrics are the *Fairness Index (FI)* and the *Maximum of Sensor' Energy Consumption* in addition to the energy consumption of all sensors. Here, the Fairness Index is a variation

of Jain's Fairness Index [49], and is defined as

$$FI = \frac{(\sum_{k=1}^{n} E_k)^2}{n \sum_{k=1}^{n} E_k^2}, \tag{3.8}$$

where E_k is the energy consumption of sensor m_k, and n is the number of active sensors. The "active sensors" are the sensors that execute either entry-tasks or non-entry-tasks. FI varies in [0,1], and the closer of FI to 1 , the better the energy consumption balance of the schedule.

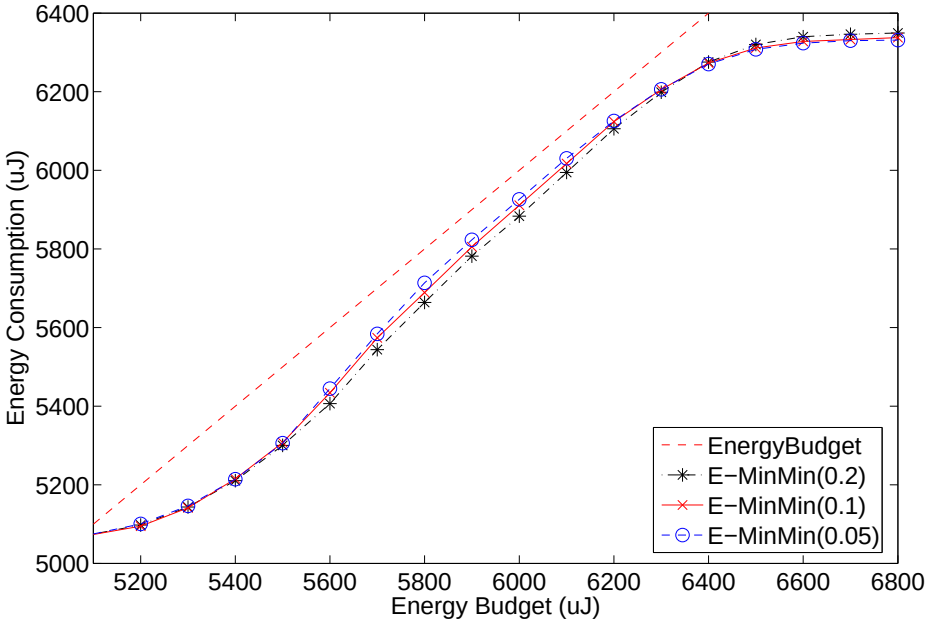
As shown in Fig. 3.12, when the energy budget is small, the EcoMapS algorithms tend to utilize a small number of computing sensors to reserve energy. Thus, computation activities as well as energy consumption are burdened on these sensors (Fig. 3.12(c)), which leads to relatively inferior energy consumption balance (Fig. 3.12(b)). However, when the energy budget increases, more sensors can be involved in the application execution. Though the overall energy consumption increases due to the increased communication volume, the maximum of each sensor's energy consumption decreases (Fig. 3.12(c)) and the energy consumption balance improves (Fig. 3.12(b)) because of the distributed computation load among sensors. Compared to E-CNPT, the energy consumption of E-MinMin is more balanced for the scenarios with intermediate energy budgets. On the other hand, DCA always overloads the cluster head with most computation tasks regardless of energy availability, which causes poor energy consumption balance for all scenarios.

Comparison of Heuristic Execution Time

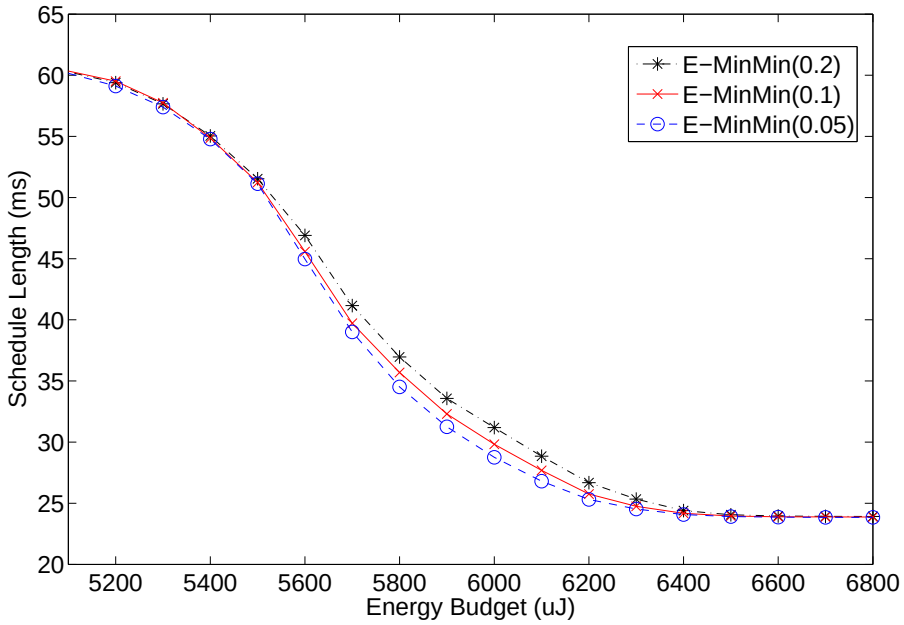
Execution time is also an important factor to evaluate heuristic algorithms. As we have analyzed in Section 3.5, E-MinMin has a higher complexity than E-CNPT. In this section, the relative execution time of E-MinMin over E-CNPT is tested with randomly generated DAGs of different number of tasks with *numEntry* = 6 and *maxPred* = 3. As shown in

Fig. 3.13, E-CNPT is more than 50 times faster than E-MinMin for tested scenarios. When the number of tasks increases, the speed difference between E-CNPT and E-MinMin also slightly increases.

As we have already seen in previous sections, the performance of E-CNPT and E-MinMin are similar when the energy budget is small or sufficiently large. For these scenarios, E-CNPT is more preferable due to its shorter execution time. For the scenarios with medium energy budgets, E-MinMin generally provides shorter schedule lengths with better energy consumption balance than E-CNPT. However, taking the heuristic execution time into account, the tradeoff between the schedule length and the heuristic execution time should be considered. Both E-CNPT and E-MinMin are executed in the Initialization Phase of EcoMapS, and schedules must be regenerated when new applications arrive. For WSNs where applications are not updated frequently, a schedule is executed for a long term. For these WSN applications, the overhead of schedule generations is negligible and E-MinMin is preferred because of its shorter schedule lengths. For a WSN that updates its applications more frequently, E-CNPT can be favored over E-MinMin due to E-CNPT's shorter schedule computation time.

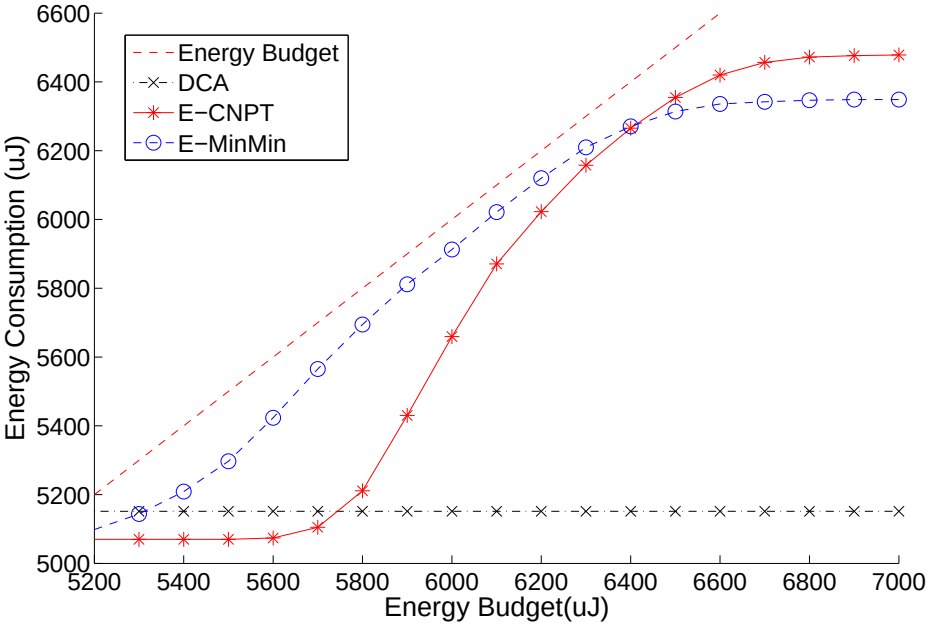


(a) Energy Consumption

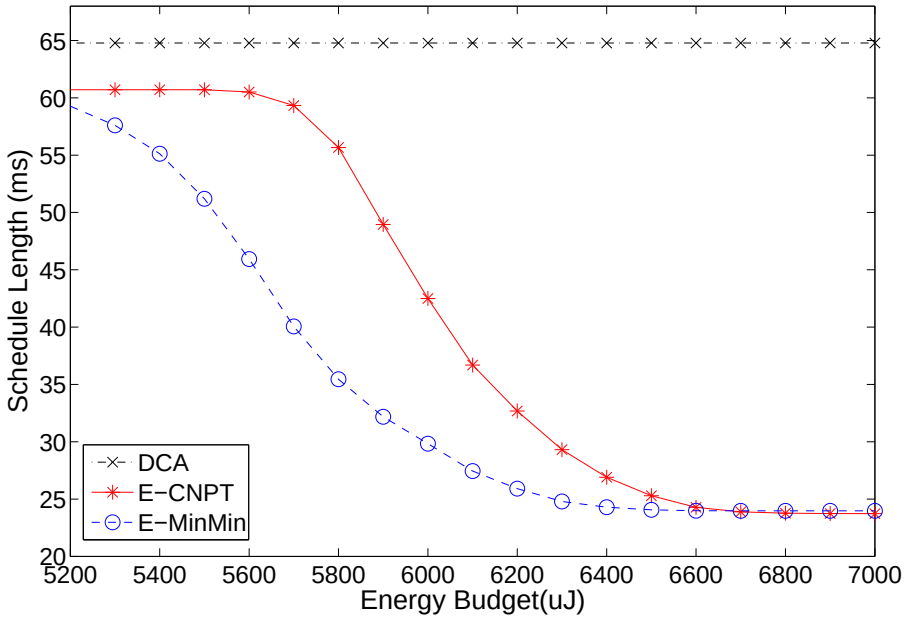


(b) Schedule Length

Figure 3.7: EcoMapS: Effect of Different value of $\Delta\alpha$

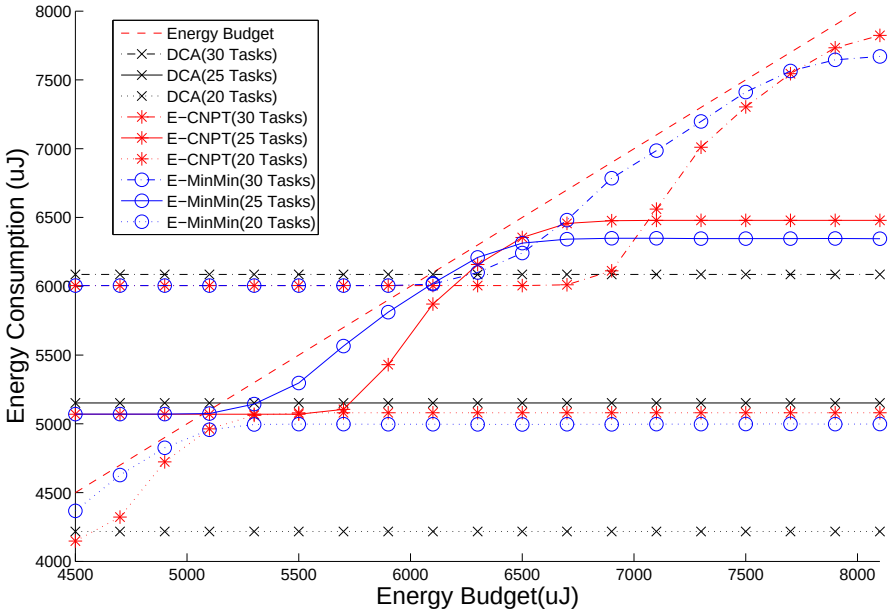


(a) Energy Consumption

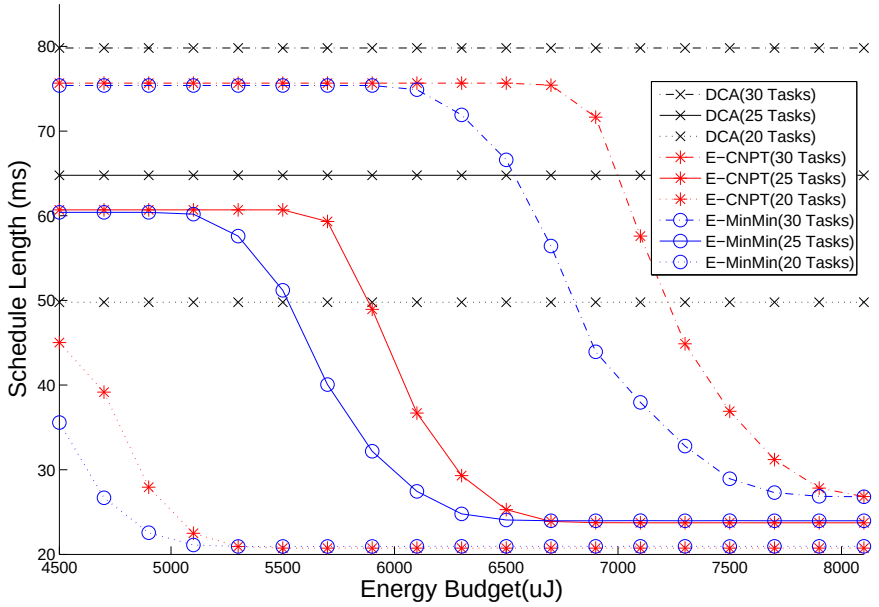


(b) Schedule Length

Figure 3.8: EcoMapS: Effect of the Energy Consumption Constraints

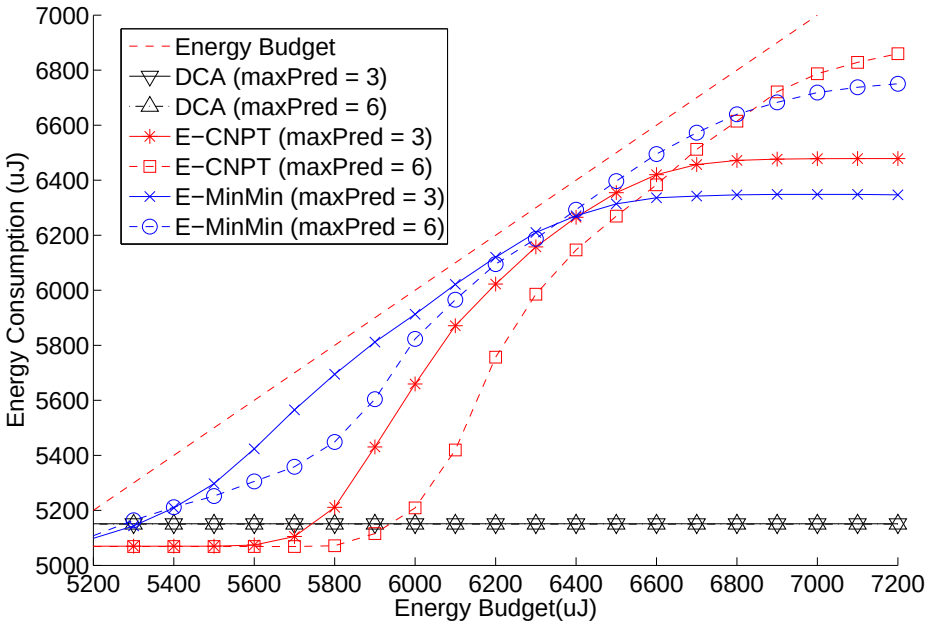


(a) Energy Consumption

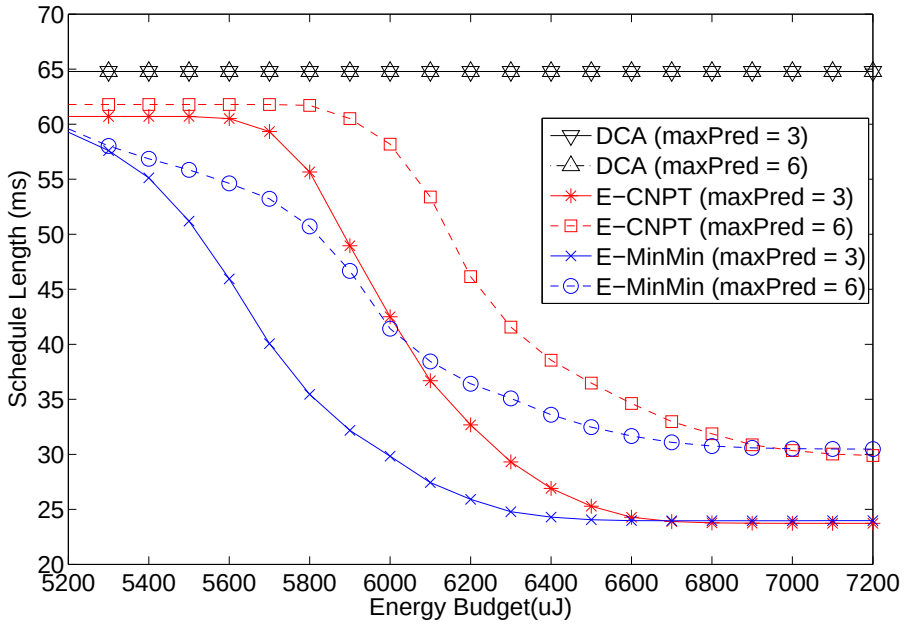


(b) Schedule Length

Figure 3.9: EcoMapS: Effect of Number of Tasks

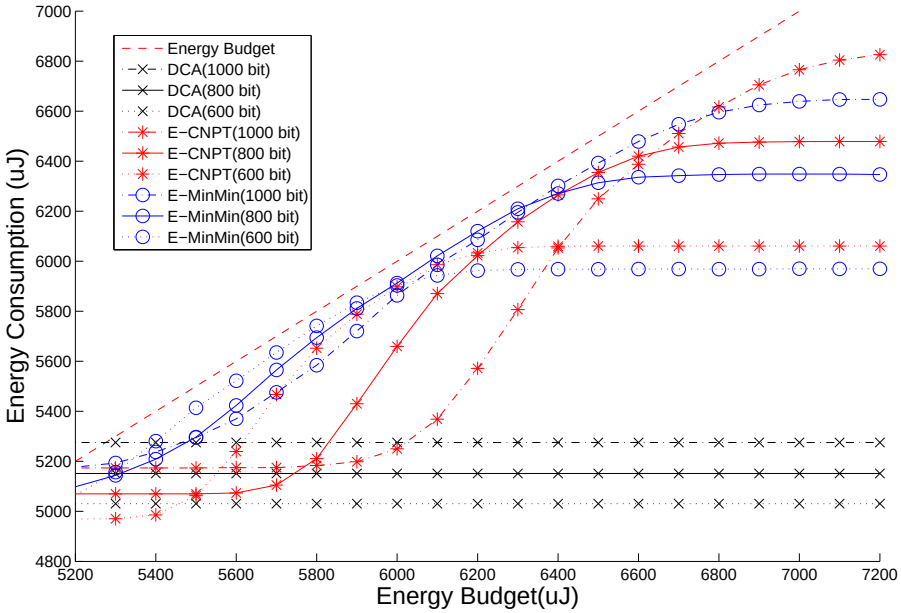


(a) Energy Consumption

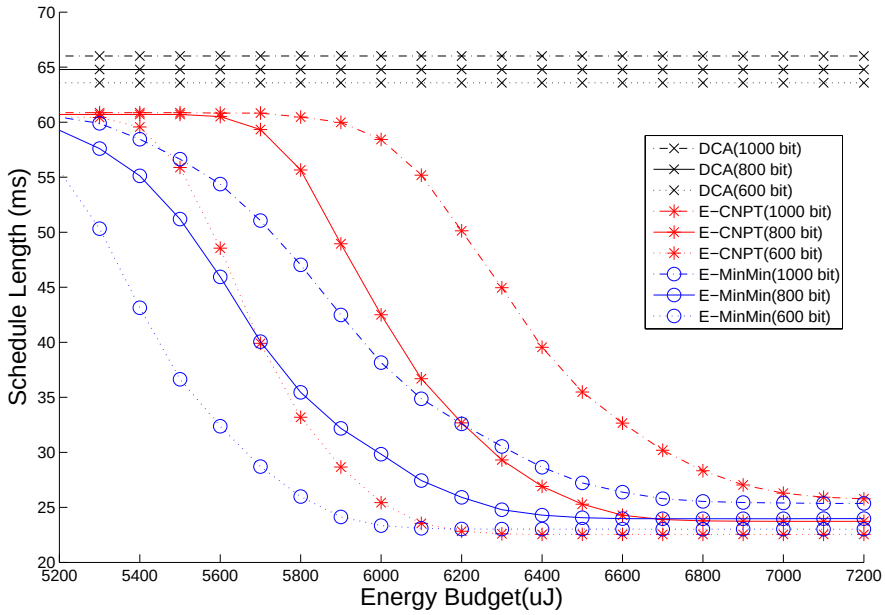


(b) Schedule Length

Figure 3.10: EcoMapS: Effect of Inter-task Dependency

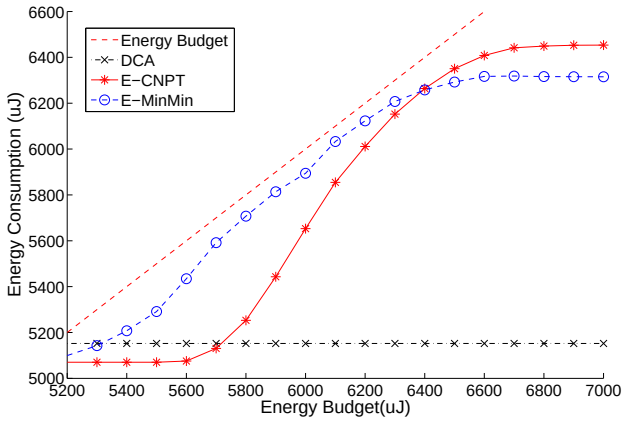


(a) Energy Consumption

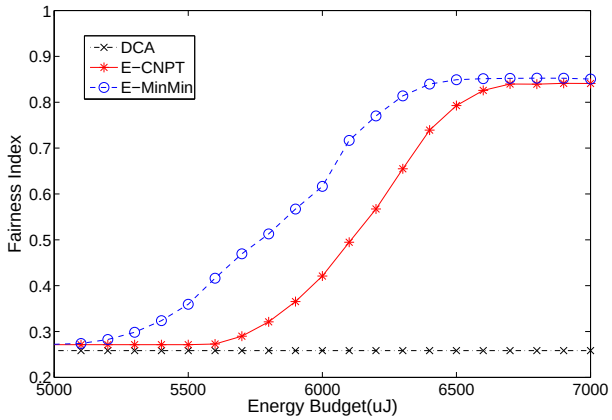


(b) Schedule Length

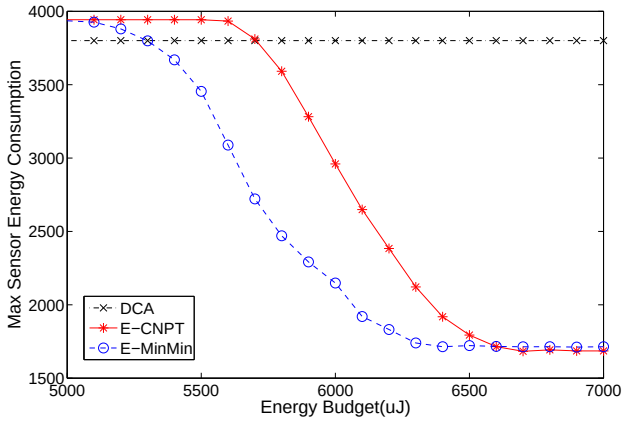
Figure 3.11: EcoMapS: Effect of Communication Load



(a) Overall Energy Consumption



(b) Fairness Index



(c) Maximum of Sensor's Energy Consumption per Sensor

Figure 3.12: EcoMapS: Energy Consumption Balance

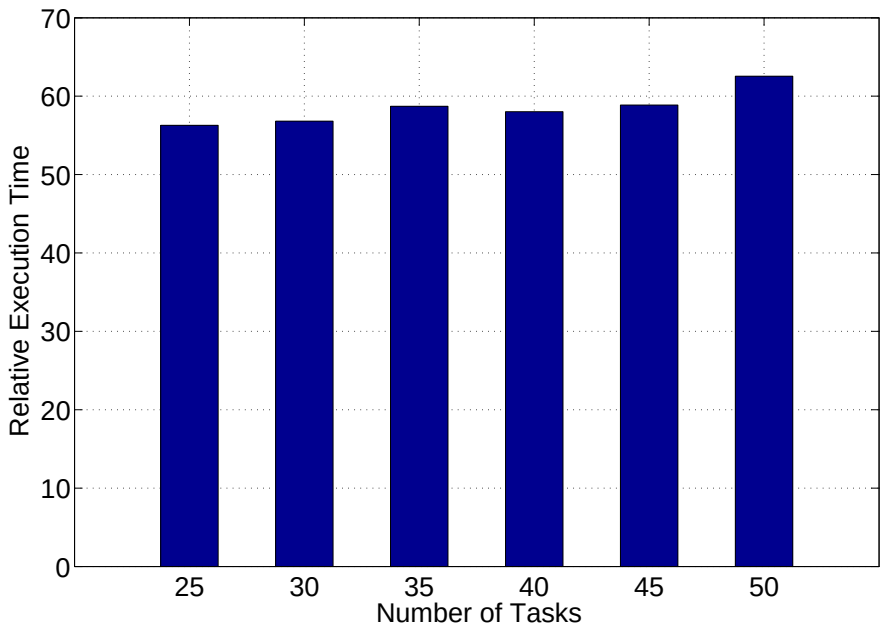


Figure 3.13: EcoMapS: The Execution Time Ratio of E-MinMin over E-CNPT

CHAPTER 4

REAL-TIME TASK MAPPING AND SCHEDULING IN SINGLE-HOP CLUSTERED WIRELESS SENSOR NETWORKS

In Chapter 3, we discuss task mapping and scheduling in single-hop clustered WSNs for energy-constrained applications. However, certain applications such as video surveillance have inherent real-time requirements. For such applications, information processing must be finished within certain deadlines. On the other hand, the stringent environment of WSNs requires energy-efficiency at all layers of WSNs. Thus, it is desirable to develop a solution providing deadline guarantees in an energy-efficient manner. Minimization of overall application energy consumption may lead to extensive execution loads of certain sensors than others. Even though out-of-battery sensors can be replaced in densely deployed WSNs, the consequent unbalanced lifetime of sensors may lead to frequent rescheduling and network topology changes. Thus, energy-balanced solutions are desirable in WSNs.

In this chapter, we propose localized cross-layer **R**eal-time **T**ask **Map**ping and **S**cheduling (RT-MapS) solutions for Dynamic Voltage Scaling (DVS) [47] enabled WSNs. We consider deadline-constrained applications executed in a single-hop cluster of a homogeneous WSN. To prolong network lifetime, the energy-balanced RT-MapS solution aims to *minimize the Maximum Energy Consumption per Node (MECpN) subject to application deadline constraints*. The design objective of RT-MapS can be formulated as finding a schedule

H^o that has the minimum MECpN under the deadline constraint:

$$Find\ H^o = arg\ \min MECpN(H), \tag{4.1}$$

$$where\ MECpN(H) = \max_k E_k, \tag{4.2}$$

$$\text{subject to } length(H) = \max_{i,k} f_{i,m_k} \leq DL, \tag{4.3}$$

where $length(H)$ and $MECpN(H)$ are the *schedule length* and *maximum energy consumption per node* of H, respectively, E_k is the energy consumption of node m_k, and DL is the deadline of the application. The Hyper-DAG and wireless channel model presented in Chapter 3.2, and the communication scheduling algorithm presented in Chapter 3.3 are implemented in RT-MapS. The communication scheduling algorithm is integrated as part of RT-MapS with the collision avoidance feature. The resulting start and finish times of communication events constitute the schedule used by the Medium Access Control (MAC). In RT-MapS, communication and computation are jointly scheduled in two phases: *Task mapping and scheduling phase* and *DVS phase*. In the *Task Mapping and Scheduling Phase*, two low-complexity task mapping and scheduling algorithms, the CNPT and Min-Min algorithm, are extended and implemented with the objective of minimizing MECpN subject to deadline constraints. A novel DVS algorithm is proposed and implemented in the *DVS phase* to further reduce energy consumption.

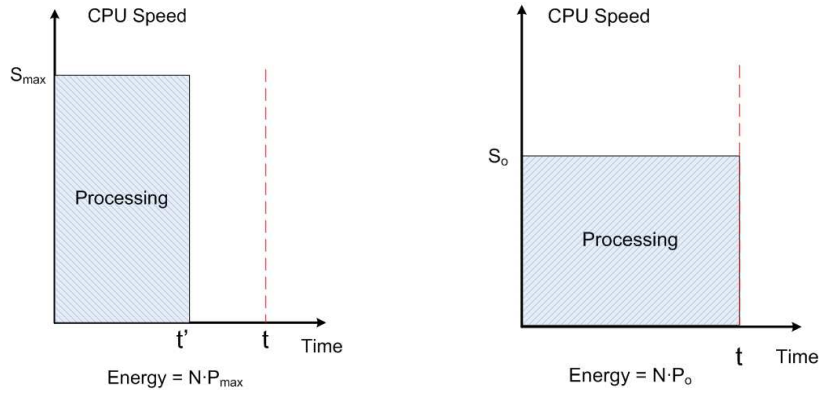
4.1 The Dynamic Voltage Scaling Problem

In this chapter, we consider sensors equipped with Dynamic Voltage Scaling (DVS) enabled processors such as StrongArm SA1100 [64]. We assume that such DVS-enabled processors have finite number of CPU speeds and supply voltage levels. The delay of speed and voltage adjustment for a DVS processor can be in the order of 10 - 100 μs [12] [45]

[47]. Such DVS adjustment overhead can be accounted into the adjusted task execution time. For the sake of simplicity, we assume the DVS adjustment overhead to be negligible throughout this dissertation.

Due to the discrete nature of task mapping and scheduling, a schedule that meets a deadline may do so with slack time before the deadline. The unbalanced load of sensors and communication scheduling also result 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 Equation 2.4 and 2.3, a decrease in CPU supply voltage leads to approximately proportional increase in execution time and approximately quadratic decrease in computation energy consumption. An example of DVS is shown in Fig. 4.1. The relationship of the CPU speed and the unit energy consumption is approximately shown in Fig. 2.2. Assuming that the execution load of task v_i is *N* clock cycles and its deadline is *t*, the execution time of v_i with speed S_{max} is t', and the corresponding energy consumption is $N \cdot P_{max}$. As demonstrated in Fig. 4.1(a), the CPU slack time is $t - t'$ before the deadline *t*. By adjusting the CPU speed to S_o, $v_i's$ execution time increases to *t* and its energy consumption decreases to $N \cdot P_o$ (Fig. 4.1(b)), which leads to energy savings of $N \cdot (P_{max} - P_o)$. Here, P_{max} and P_o stand for the original and the adjusted computation energy consumption per clock cycle of v_i with CPU speed of S_{max} and S_o, respectively.

Task scheduling for DVS-enabled systems is under active research [5], [16], [44]. In [5], a periodic real-time task scheduling mechanism is proposed for DVS-enabled systems with limited energy supplies. However, [5] only considers single-processor systems and does not address task scheduling in multiprocessor networks. Task scheduling in DVS multiprocessor systems is discussed in [16] based on shaping of battery discharging profiles. The discussed multiprocessor system is driven by a single battery, which is not applicable to



(a) CPU Speed = S_{max} (b) CPU Speed = S_o

Figure 4.1: DVS Scheduling Example (Task with *N* Clock Cycles)

WSNs. A DVS-based energy management scheme is presented in [44] for distributed real-time systems with consideration of communication and precedence constraints. However, [44] focuses on parallel processing systems and does not extend to wireless communication systems. The DVS technique is implemented in [69] to save energy in WSNs. The DVS algorithm is applied to a calculated schedule with the constraints of wireless communication events. To re-scale each task, application schedules need to be re-adjusted with the scheduling algorithm in [69], which leads to a high computation complexity. In this chapter, a low-complexity DVS algorithm is presented to optimize energy consumption in WSN environments.

4.2 Outline of the Proposed RT-MapS Solution

The proposed RT-MapS sloution is demonstrated with the flowchart in Fig. 4.2. RT-MapS has two phases: *Task Mapping and Scheduling Phase* and *DVS Phase*. In the *Task Mapping and Scheduling Phase*, communication and computation tasks are scheduled. Two

low-complexity task mapping and scheduling algorithms, CNPT [28] and Min-Min [11] [53] are extended and implemented with the objective of minimizing MECpN subject to the deadline constraint. Our proposed communication scheduling algorithm is then embedded in the execution of the extended CNPT and Min-Min algorithms to satisfy the *Dependency Constraint*. The energy consumption is further reduced in the *DVS Phase*. In the following sections, the main components of the RT-MapS solution, namely, Hyper-DAG based CNPT and Min-Min algorithms (referred to as H-CNPT and H-MinMin), and DVS algorithm, are presented in addition to the Hyper-DAG, wireless channel model, and communication scheduling algorithm presented in Chapter 3.

4.3 Task Mapping and Scheduling with H-CNPT and H-MinMin Algorithm

In the *Task Mapping and Scheduling Phase* of RT-MapS, the tasks of Hyper-DAGs are mapped and scheduled on sensors. To meet the *Dependency Constraint* during task mapping and scheduling, if a computation task depends on a communication task assigned on another sensor node, the *communication scheduling algorithm* will be executed to duplicate the absent communication task. With the Communication Scheduling Algorithm in Chapter 3.3, task mapping and scheduling in single-hop wireless networks can be tackled as a generic task mapping and scheduling problem with additional constraints. In this section, two task mapping and scheduling algorithms, H-CNPT algorithm and H-MinMin algorithm are presented with the objective of minimizing MECpN subject to deadline constraints. To guarantee deadlines, sensors are scheduled with the maximum CPU speed f_{cpu}^{max}. The H-CNPT and H-MinMin algorithms also employ the concept of *computing sensor* presented

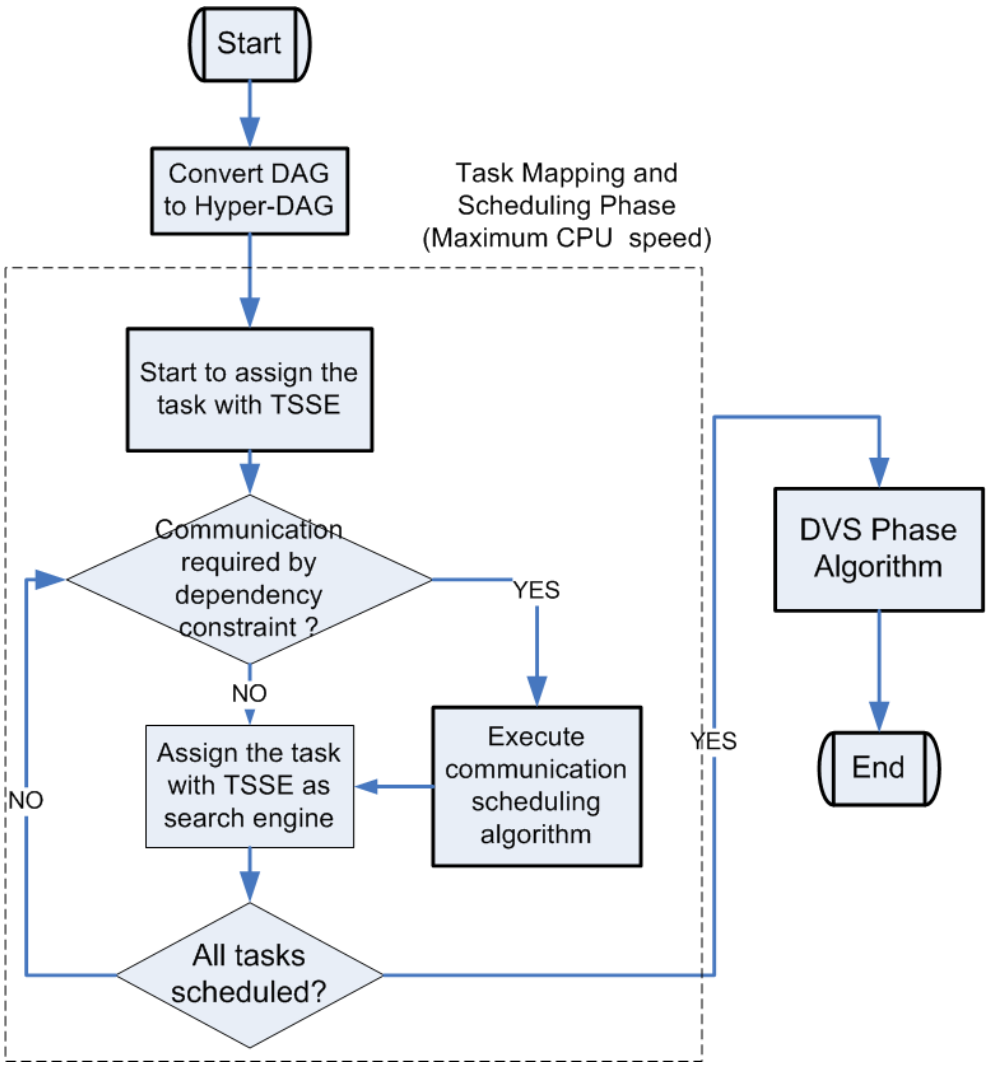


Figure 4.2: RT-MapS Flowchart

in Chapter 3.4. In RT-MapS, H-CNPT and H-MinMin will iteratively search for the optimal schedule with different number of computing sensors subject to deadline constraints.

4.3.1 H-CNPT Algorithm

H-CNPT's strategy is to assign the tasks along the most critical path first to the nodes with earliest execution start times. By adjusting the number of computing sensors in each scheduling iteration and choosing the schedule with the minimum MECpN under the deadline constraint, the design objective of H-CNPT is achieved. Similar to CNPT [28], H-CNPT also has two stages: *listing stage* and *sensor assignment stage*. In the *listing stage*, tasks are sequentialized into a queue L such that the most critical path comes first and a task is always enqueued after its immediate predecessors. In the *sensor assignment stage*, the tasks will be dequeued from L and assigned to the sensors with the minimum execution start time. Several scheduling iterations will be run in the sensor assignment stage with different number of computing sensors, and only the most optimal schedule is chosen. The *listing stage* and *sensor assignment stage* of H-CNPT are introduced individually as follows.

Listing Stage: The Listing Stage of H-CNPT is similar to that of CNPT [28] except that there are two types of tasks in H-CNPT: Computation tasks and communication tasks. Thus, the formulas to calculate the Earliest Start Time $EST(v_i)$ and the Latest Start Time $LST(v_i)$ of task v_i are different from those of CNPT, and are presented as follows:

$$EST(v_i) = \max_{v_m \in pred(v_i)} \{EST(v_m) + t_m\}, \tag{4.4}$$

$$LST(v_i) = \min_{v_m \in succ(v_i)} \{LST(v_m)\} - t_i, \tag{4.5}$$

Input: Task queue L; number of available sensors in the cluster p; deadline DL
Output: Schedule H^o of tasks in L with the MECpN under the deadline constraint
H-CNPT Algorithm:

---

1. $L_{min} \leftarrow \infty$ /*minimum schedule length*/
2. $MECpN^o \leftarrow \infty$ /*optimal MECpN*/
3. **FOR** q = 1 to p /*Search computing sensor space*/
4. H = SingleCNPT(L,q)
6. **IF** $length(H) < L_{min}$ /*shortest schedule*/
7. $L_{min} \leftarrow length(H)$
8. $H_{min} \leftarrow H$
9. **IF** $length(H) \leq DL$ and $MECpN(H) < MECpN^o$ /*optimal schedule*/
10. $MECpN^o \leftarrow MECpN(H)$
11. $H^o \leftarrow H$
12. **IF** $L_{min} \leq DL$
13. Return H^o
14. **ELSE**
15. Return H_{min}

Figure 4.3: RT-MapS: H-CNPT Algorithm

where t_i equals to the execution length on sensor nodes if $v_i \in V$ or to the execution length on $\mathcal{C}$ if $v_i \in R$. After the Listing Phase, the task graph is sequentialized into L and is ready for the Sensor Assignment Phase. The details of the Listing Stage can be found in [28].

Sensor Assignment Stage: In the Sensor Assignment Stage, H-CNPT will iteratively search the schedule space with different number of computing sensors. Among these schedules, the one with the minimum energy consumption under the deadline constraint is chosen as the solution. If no schedule meets the deadline constraint, the schedule with the minimum schedule length is chosen. The detailed description of the H-CNPT algorithm is presented in Fig. 4.3.

Input: task queue L; number of computing sensors q
Output: Schedule of tasks in L
SingleCNPT Algorithm:

---

while L is not empty
1. Dequeue v_i from L
2. **IF** $v_i \in R$ /* communication task */
3. Assign v_i to node $m(pred(v_i))$
4. **ELSE IF** $pred(v_i) = \emptyset$ /*entry-tasks*/
5. Assign v_i to node m_i^o with min $EAT(m_i^o)$
6. **ELSE** /* non-entry computation tasks*/
7. **FOR** computing sensors $\{m_k\}$
8. Calculate EEST(v_i, m_k) with a copy of current schedule:
9. **IF** $pred(v_i) \subseteq T(m_k)$ /*meet dependency constraint*/
10. EEST(v_i, m_k) $\leftarrow \max(EAT(m_k), f_{pred(v_i),m_k})$
11. **ELSE** /*schedule communication to meet dependency constraint*/
12. **FOR** $v_n \in pred(v_i) - T(m_k)$
13. CommTaskSchedule(v_n,$m(v_n)$,m_k)
14. EEST(v_i, m_k) $\leftarrow \max(EAT(m_k), f_{pred(v_i),m_k})$
15. Keep the schedule with minimum EEST(v_i, m^o)
16. Schedule v_i on m^o: $s_{v_i,m^o} \leftarrow EEST(v_i, m^o)$

Figure 4.4: RT-MapS: SingleCNPT Algorithm

In Fig. 4.3, SingleCNPT(L,q) is a single round of task scheduling that schedules the tasks in L with q computing sensors, where q is the total number of available computing sensors. The actual number of computing sensors in use can be smaller than q depending on the application and the scheduling algorithm. The core of SingleCNPT(L,q) is the extended CNPT *processor assignment algorithm*. The basic strategy of the algorithm is to assign tasks to the sensor with the minimum Earliest Execution Start Time (EEST). During task scheduling, *Dependency Constraint* must be satisfied via communication scheduling. SingleCNPT(L,q) is presented in Fig. 4.4.

In Fig. 4.4, $EAT(m_k)$ is the Earliest Available Time of node m_k, and EEST(v_i, m_k) is the Earliest Execution Start Time of v_i on sensor m_k. Different from EST, EEST represents the actual execution start time of a task if assigned on a sensor node.

4.3.2 H-MinMin Algorithm

Similar to H-CNPT, H-MinMin also searches for a schedule with optimal number of computing sensors that has the smallest MECpN subject to the deadline constraint. The H-MinMin's optimal number of computing sensors searching algorithm is the same as the **H-CNPT Algorithm** in Section 4.3.1, except that the input of the H-MinMin algorithm is the Hyper-DAG instead of the task queue L, and the core of the searching algorithm is the **SingleMinMin** instead of the **SingleCNPT**. In the following, we introduce the procedure SingleMinMin(Hyper-DAG,q) that schedules the tasks of the Hyper-DAG with q computing sensors.

The core of the SingleMinMin algorithm is the fitness function. For each task-node combination (v,m), the fitness function $fit(m, k, \alpha)$ indicates the combined cost in time and energy domain of assigning task v to node m, where α is the weight parameter trading off the time cost for the balanced energy consumption. To evaluate energy consumption balance, we define the Fairness Index (FI), which is a variation of Jain's Fairness Index [49], as follows:

$$FI = \frac{(\sum_{k=1}^{n} E_k)^2}{n \sum_{k=1}^{n} E_k^2}, \tag{4.6}$$

where n is the number of active sensors. The "active sensors" are the sensors that execute either entry-tasks or non-entry-tasks. FI varies in [0,1], and the closer of FI to 1 , the better the energy consumption balance of the schedule. At each step of the SingleMinMin 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 [53], the following notations are introduced first:

- $f_{v,m}$ is the scheduled finish time of v on m
- $FI(v,m)$ is the FI of the schedule after assigning v on m
- $NPT(v,m)$ is the normalized partial execution time of assigning v on m: $NPT(v,m) = f_{v,m}/DL$

Thus, the fitness of assigning v on m with α is defined as:

$$fit(v,m,\alpha) = \alpha \cdot NPT(v,m) + (1-\alpha) \cdot (1 - FI(v,m)). \tag{4.7}$$

The SingleMinMin Algorithm is presented in Fig. 4.5. In the description of SingleMinMin, a "mappable" task is either an entry-task or a task that has all immediate predecessors already been assigned, and the "mappable task list" is the list that contains currently mappable tasks of the Hyper-DAG. For each application, we compare its schedules with different α value ranging from 0 to 1 in 0.1 increments. The schedule with the minimum MECpN 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.

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

Input: Hyper-DAG; number of computing sensors: q
Output: Schedule H^o of tasks in Hyper-DAG
SingleMinMin Algorithm:

---

1. **FOR** $\alpha = 0$; $\alpha \leq 1.0$; $\alpha += 0.1$ /*scan different α value*/
2. **FOR** entry-tasks v_i /*first assign entry-tasks*/
3. Assign v_i on node m_i^o with min $EAT(m_i^o)$
4. Assign $succ(v_i)$ on m_i^o
5. Initialize the mappable task list *L*
6. **WHILE** L is not empty, with a copy of current schedule:
7. **FOR** task $v_i \in L$ /*scan all task-sensor combinations*/
8. **FOR** all computing sensor m_k
9. **IF** $pred(v_i) \not\subseteq T(m_k)$
10. **FOR** $v_n \in pred(v_i) - T(m_k)$
11. **CommTaskSchedule**(v_n,$m(v_n)$,m_k)
12. Assign v_i to m_k, calculate $fit(v_i, m_k, \alpha)$
13. Find m_i^o: $fit(v_i, m_i^o, \alpha) = \min$
14. Keep the schedule with (v,m): $fit(v, m, \alpha) = min$
15. Assign v to m, remove v from *L*
16. Assign $succ(v)$ on m /*assign communication task*/
17. Update *L* with any new unassigned mappable tasks
18. Among all schedules with different values of α
19. **IF** $\exists H : length(H) \leq DL$ with $\min MECpN(H)$
20. Return H /*optimal schedule*/
21. **ELSE**
22. Return $H : length(H) = \min$ /*best-effort schedule*/

Figure 4.5: RT-MapS: SingleMin Algorithm

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: Schedule Length Extension (SLE) Stage and Schedule Hole Elimination (SHE) Stage. In the SLE stage, the slack time between schedule length $length(H)$ and application deadline DL is eliminated by proportionally slowing down all sensors' CPU speed. Let β be defined as $\beta = \frac{length(H)}{DL} < 1$, and the re-scale 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 independent of the CPU operation as assumed in Section 2.2, 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. 4.6. 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). Assume that the original schedule length is 8 tu with CPU speed f_{cpu}^{max}, the deadline DL = 12 tu, and the calculated re-scale factor $\gamma^{-1} = 1.5$. In the SLE Stage of Fig. 4.6, 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

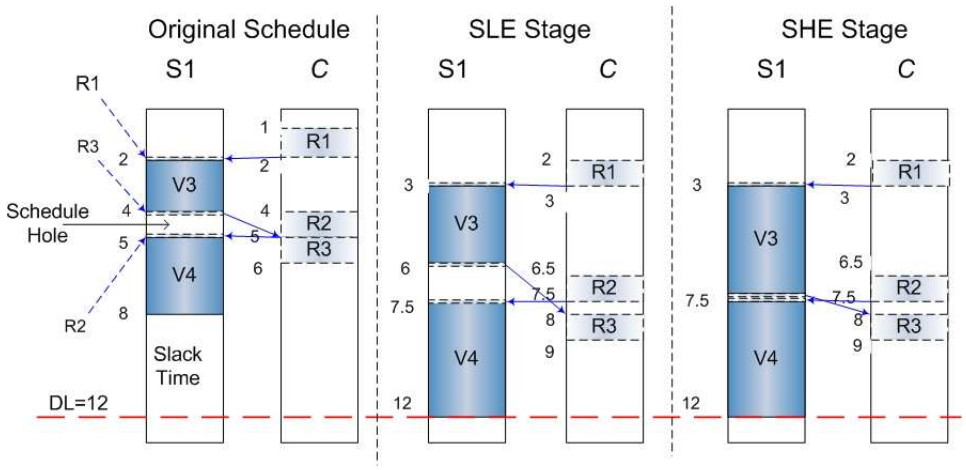


Figure 4.6: Demonstration of Partial DVS Adjustment

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. 4.7. The SHE algorithm iteratively scans each sensor's schedule to detect time intervals $[ds, df]$ that contain schedule holes. As demonstrated in Fig. 4.6, 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. ds equals the minimum of the

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. $ds \leftarrow 0$, $df \leftarrow \infty$ /*Initialization*/
3. Scan tasks $v_i \in T_{ds}^{\infty}(m_k)$ in increasing order of start time
4. **IF** $\exists$ a copy of $v_i \in R$: $v_i^c \in T(\mathcal{C})$ /*Transmitted communication task */
5. Find the computation task v_j following v_i
6. **IF** m_k is the sender of v_i^c
7. $df \leftarrow \min(s_{v_i^c,\mathcal{C}}, s_{v_j,m_k})$ /*Computation must finish before transmitting*/
8. **SpeedAdjust**(m_k,ds,df,$\gamma \cdot f_{cpu}^{max}$)
9. $ds \leftarrow df$
10. **ELSE** /*m_k is the receiver of v_i^c*/
11. $ds \leftarrow \max(ds, f_{v_i^c,m_k}, s_{v_j,m_k})$ /*Computation cannot start before reception*/
12. **ELSE IF** v_i is exit-task and $f_{v_i} < DL$ /*Adjustment bounded by deadline*/
13. **SpeedAdjust**(m_k,ds,DL,$\gamma \cdot f_{cpu}^{max}$)

Figure 4.7: Rt-MapS: DVS SHE Algorithm

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. 4.8. In $SpeedAdjust()$, the *CPU utility* η during a time interval $[ds, df]$ is defined as:

$$\eta = e_{ds}^{df}/(df - ds), \tag{4.8}$$

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 6-10, 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

Assume that the application T is represented as $T = (V, E)$, $|V| = v$, $|E| = e$, the number of entry-tasks is f, and the cluster has p sensor nodes. Thus, the Hyper-DAG is $T' = (V', E')$, where $|V'| = 2v$ and $|E'| = 2e$.

4.5.1 Computational Complexity of RT-MapS with H-CNPT

The time complexity of RT-MapS with H-CNPT is analyzed as follows:

- Listing Stage of H-CNPT: similar to CNPT [28], the complexity is $O(v + e)$.
- SingleCNPT: the communication tasks have complexity of $v \cdot O(1) = O(v)$, the entry-tasks have complexity of $f \cdot O(p) = O(fp)$, other non-entry computation tasks have complexity of $(v - f) \cdot O(p) \cdot O(e/v)$. Hence, the overall complexity of SingleCNPT is $O(v)+O(fp)+(v-f)\cdot O(p)\cdot O(e/v)$. For the worst case, $e = O(v^2)$ and $f = O(v)$, thus the complexity of SingleCNPT is $O(pv^2)$ for the worst case.

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

---

1. $e^{df}_{ds} \leftarrow 0$, $tt \leftarrow ds$ /*Initialization*/
2. **FOR** $v_i \in T^{ft}_{ds}(m_k)$ and $v_i \in V$ /*Calculate CPU execution time e^{df}_{ds} in [ds,df]*/
3. $\quad e^{df}_{ds} \leftarrow e^{df}_{ds} + t_{v_i,m_k}$
4. $\eta \leftarrow e^{df}_{ds}/(df - ds)$ /*CPU utility in [ds,df]*/
5. $f^o_{cpu} \leftarrow \lceil f_{cpu} \cdot \eta \rceil$ /*Adjusted CPU speed in [ds,df]*/
6. **FOR** $v_i \in T^{df}_{ds}(m_k)$ and $v_i \in V$ /*Adjust computation tasks*/
7. $\quad s_{v_i,m_k} \leftarrow tt$
8. $\quad t_{v_i,m_k} \leftarrow t_{v_i,m_k} \cdot \frac{f_{cpu}}{f^o_{cpu}}$
9. $\quad f_{v_i,m_k} \leftarrow s_{v_i,m_k} + t_{v_i,m_k}$
10. $\quad tt \leftarrow f_{v_i,m_k}$
11. **FOR** $v_i \in T^{ft}_{ds}(m_k)$ and $v_i \in R$ /*Locally assigned communication task*/
12. $\quad$ **IF** $pred(v_i) \in T(m_k)$
13. $\quad\quad s_{v_i,m_k} \leftarrow f_{pred(v_i),m_k}$, $f_{v_i,m_k} \leftarrow f_{pred(v_i),m_k}$
14. Update the energy consumption of m_k

Figure 4.8: RT-MapS: DVS Adjustment Algoithm for a Single Sensor in $[ds, df]$

- RT-MapS with H-CNPT: the SingleCNPT algorithm will be called $O(p)$ times. Thus, the complexity of the whole algorithm is $O(v+e)+O(p)\cdot O(v^2p)$ = $O(p^2v^2)$ for the worst case.

4.5.2 Computational Complexity of RT-MapS with H-MinMin

The time complexity of RT-MapS with H-MinMin is analyzed as follows:

- SingleMinMin: the complexity of SingleMinMin is dominated by the loop starting from Step 6, which is executed $O(v)$ times. Similarly to SingleCNPT, the complexity of the loop starting from Step 7 has the complexity of $O(v)\cdot O(p)\cdot O(e/v)$ = $O(pe)$. Thus, SingleMinMin has the complexity of $O(pv^3)$ for the worst case.

- RT-MapS with H-MinMin: Similar to the analysis of H-CNPT, the complexity is $O(p)\cdot O(pv^3)$ = $O(p^2v^3)$ for the worst case.

Regarding the DVS algorithm, the SLE stage needs to adjust all tasks once, thus has complexity of $O(v)$; the *SpeedAdjust Algorithm* of the SHE stage will only scan and adjust tasks assigned on each sensors, thus has a complexity of $O(\frac{v}{p})$. The SHE Algorithm will scan and adjust all unadjusted tasks at most once, thus has a complexity of $O(v\cdot\frac{v}{p})$ = $O(\frac{v^2}{p})$. Thus, the complexity of the DVS algorithm is $O(v+v^2/p)$.

4.6 Simulation Results

The performances of the RT-MapS with the H-CNPT algorithm and the RT-MapS with the H-MinMin algorithm are evaluated through simulations, and denoted as H-CNPT and H-MinMin in this section, respectively. The performance of DCA and EbTA is also evaluated as benchmarks. DCA is extended with our proposed communication scheduling algorithm to deliver the intermediate results of entry-tasks to the cluster head for further

processing. DCA algorithm is also implemented with DVS for fair comparison. We first simulate the video surveillance application described in Chapter 3.6.2. To further evaluate RT-MapS performance, simulations are run on arbitrary applications with randomly generated DAGs. Our simulations with random DAGs study the following scenarios:

- The effect of application deadline constraints
- The effect of number of tasks in applications
- The effect of inter-task dependency
- The effect of communication load

In these simulations, we observe schedule length, deadline missing ratio (DMR), MECpN, FI, and application energy consumption metrics. The schedule length is defined as the finish time of the exit-task of an application, the DMR is defined as the ratio of the number of the simulation runs whose schedule length is larger than the imposed deadline over the number of the overall simulation runs, and MECpN and FI are as defined in Equation 4.2 and Section 4.3.2, respectively. Application energy consumption includes computation and communication energy expenditure of all sensors.

4.6.1 Simulation Parameters

In our simulation study, the bandwidth of the channel is set to 1Mb/s and the transmission range to 10 meters. We assume that there are 10 sensors in a single-hop cluster. 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 Equation 2.1, 2.2, 2.3 are in coherence with [52], [64], [30] as follows: E_{elec} = 50 nJ/b, ε_{amp} = 10 pJ/b/m^2, V_T = 26 mV, C = 0.67 nF, I_o = 1.196 mA, n = 21.26, K = 239.28 MHz/V and c = 0.5 V.

DL	Metrics	H-CNPT	H-MinMin	EbTA	DCA
3 ms	Schedule Length (ms)	3.00	3.59	3.01	5.64
	MECpN (uJ)	585.2	320.2	842.8	1138.9
	Energy Consumption (uJ)	2178.1	2145.2	2262.7	2238.4
5 ms	Schedule Length (ms)	4.97	4.97	4.87	5.64
	MECpN (uJ)	344.2	237.9	509.4	1138.9
	Energy Consumption (uJ)	1278.8	1587.3	1816.8	2238.4
7 ms	Schedule Length (ms)	6.93	6.93	6.61	6.90
	MECpN (uJ)	267.2	177.7	384.8	834.5
	Energy Consumption (uJ)	993.6	1196.4	1934.0	1594.4

Table 4.1: RT-MapS: Simulation with the Visual Surveillance Example

4.6.2 Simulation with a Real-life Example: Distributed Visual Surveillance

In this section, we evaluate the performance of RT-MapS, DCA, and EbTA algorithms with the real-life example of distributed visual surveillance presented in Fig. 1.1. The application is as described in Section 3.6.2 and abstracted with the DAG in Fig. 3.6. In the simulation, we consider a single intruder, 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 $E_{48} - E_{9,10}$. As shown in Table 4.1, both RT-MapS algorithms have better capacity to meet deadlines than DCA when deadlines are small. In this specific application where all communications are unicast, EbTA outperforms H-MinMin in terms of meeting deadline constraints, while H-CNPT still performs the best. Regarding energy consumption, both RT-MapS algorithms have smaller MECpN than DCA and EbTA. Regarding the comparison of RT-MapS algorithms, H-MinMin achieves smaller MECpN with the cost of higher application energy consumption.

4.6.3 Simulation with Random DAGs

Simulations are run on randomly generated DAGs, which are created based on three parameters: The number of tasks *numTask*, the number of entry-tasks *numEntry*, and the maximum number of predecessors *maxPred*. The number of each non-entry task's predecessors, the computation load, and the communication data volume of a task are uniformly distributed over [1, *maxPred*], [300K CC, $\pm 10\%$], and [800 bits, $\pm 10\%$], respectively. The simulation results presented in this section correspond to the average of two hundred independent runs.

Effect of Application Deadlines

The effect of application deadlines and DVS adjustment are investigated with randomly generated DAGs as *numTask* = 30, *numEntry* = 10, and *maxPred* = 5. To evaluate the effect of DVS, the performance of DCA, EbTA, H-CNPT and H-MinMin before the voltage adjustment (denoted as DCA*, EbTA*, H-CNPT* and H-MinMin*, respectively) are also investigated.

As shown in Fig. 4.9(a) and Fig. 4.9(b), both RT-MapS algorithms have better capability to meet small deadlines than DCA and EbTA. Compared to DCA, RT-MapS can have multiple computing sensors in parallel according to deadline constraints, while DCA has only one sensor for high level computing. On the other hand, though EbTA also employs multiple sensors for computing, it does not exploit the broadcast nature of wireless communication like RT-MapS does. In EbTA, a task must send information individually to its immediate successors with larger overall communication time. Such multi-communication

feature also introduces higher dependency between tasks and weaken the parallelism between sensors, which leads to larger schedule lengths and lower energy balance level. Regarding the comparison of the RT-MapS algorithms themselves, H-CNPT outperforms H-MinMin in term of schedule lengths and DMR when deadlines are small. The scheduling criteria of H-CNPT is determined only by schedule lengths, while the fitness function of H-MinMin is a combination of schedule length and energy consumption. The tradeoff between schedule length and energy consumption degrades the schedule length performance of H-MinMin algorithm.

As shown in Fig. 4.10(a) and Fig. 4.10(b), the H-MinMin algorithm outperforms other algorithms in terms of energy consumption balance for most simulated scenarios. In DCA, most tasks are run on a single sensor while H-MinMin can evenly distribute tasks among multiple sensors to obtain energy-balanced schedules. The multi-communication feature of EbTA decreases the parallelism between sensors and leads to unbalanced energy consumption. Furthermore, the sensor with the most computational activities is burdened by higher communication energy consumption in EbTA, which further decreases energy consumption balance. On the other hand, the broadcast scheduling feature of RT-MapS conserves communication energy consumption and eases parallel scheduling between sensors. Regarding the comparison of H-CNPT and H-MinMin, H-MinMin still outperforms H-CNPT by taking energy consumption fairness into account when calculating fitness value and scheduling tasks.

Regarding application energy consumption, when deadlines are relatively large, computation tasks in RT-MapS are distributed among multiple sensors to achieve energy balance, which leads to higher application energy consumption due to more communication activities (Fig. 4.10(c)). However, after implementing the DVS algorithm, the application

energy consumption of both RT-MapS algorithms are smaller than DCA and EbTA due to the larger exploitable slack time before deadlines.

As we can see from Fig. 4.10(a) and Fig. 4.10(c), our DVS algorithm is an effective approach for energy efficient solutions. When deadlines are sufficiently large, the DVS adjustment results in about 50 % energy consumption savings by "pushing" the schedule length close to the deadline in RT-MapS. Even when deadlines are relatively small and there is little slack time before application deadlines, the DVS adjustment of RT-MapS can still save about 15% energy compared with the scenarios without the DVS adjustment. This energy saving stems from eliminating the schedule holes caused by the unbalanced load of sensors and communication scheduling. On the other hand, the DVS algorithm of EbTA cannot exploit the slack time of unbalanced sensors when schedule lengths are over deadlines. It should be noted that though the DVS adjustment may increase schedule lengths (Fig. 4.9(a)), the DMR is not affected (Fig. 4.9(b)) for any of the simulated deadline values.

Effect of Inter-task Dependency

The inter-task dependency is determined by the in/out degree of application DAGs. Simulations with sets of DAGs with $maxPred = 5$ and $maxPred = 10$ (*numTask = 30*, *numEntry* = 10) are executed.

According to the simulation results of Fig. 4.11 and 4.12, the inter-task dependency barely affects the performance of DCA due to the fact that DCA has most of the tasks executed on the cluster head, and therefore has the least need for communication. On the other hand, EbTA is significantly affected by the increment of inter-task dependency, as more communication events are needed to deliver a task's result to its immediate successors with higher inter-task dependency. Regarding the RT-MapS algorithms, increasing the

in/out degree of DAGs only leads to higher dependency between a communication task and its immediate successors without introducing new communication tasks, thus they are less affected than EbTA. But greater dependency degree between tasks leads to less parallelism between sensors and a larger number of communication tasks scheduled on $\mathcal{C}$, which leads to more energy consumption and longer schedules. Thus, the RT-MapS algorithms are affected more by inter-task dependency increase than DCA. Compared with H-CNPT, H-MinMin is affected more and has a higher possibility of missing deadlines when the communication load increase. However, in all simulated scenarios, both RT-MapS algorithms outperforms DCA and EbTA in terms of guarantee deadline constraints with minimum MECpN.

Effect of Number of Tasks

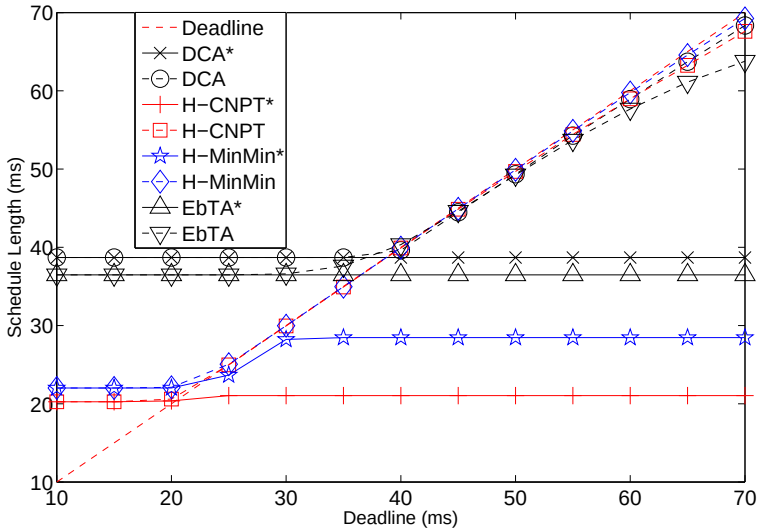
To investigate the effect of the number of tasks in applications, simulations are run on randomly generated DAGs with 25, 30, 35 tasks (*numEntry* = 10, *maxPred* = 5).

According to Fig. 4.13 and 4.14, the performance of all algorithms degrade with the increase of application scales, and energy consumption is dominated by the number of tasks. When the number of tasks increases, the application energy consumption and MECpN of all algorithms increase proportionally. However, the RT-MapS algorithms have smaller energy consumption than DCA and EbTA for all simulated scenarios. Regarding schedule lengths and DMR, though all algorithms are affected when the number of tasks increases, the RT-MapS algorithms are less affected due to their better capacity to meet deadline constraints by adjusting the number of computing sensors.

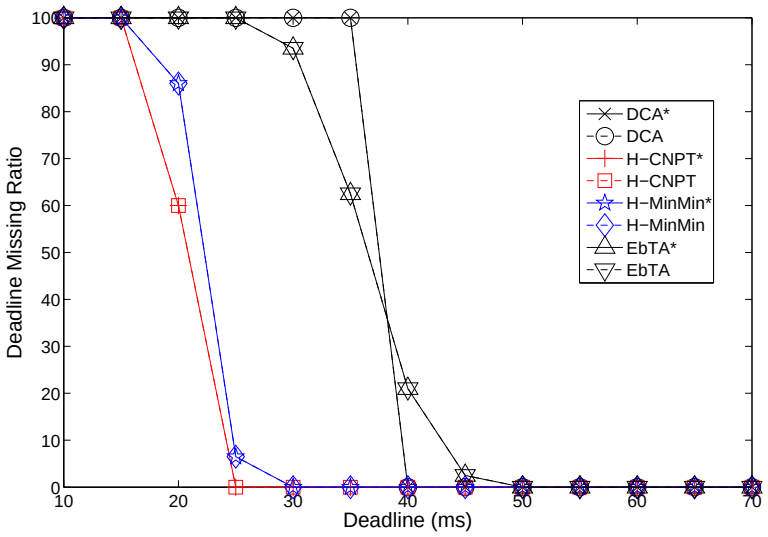
Effect of Communication Load

For task mapping and scheduling in wireless networks, the relationship between communication and computation load may affect the overall performance. This factor is evaluated by changing the average communication data volume with fixed average computation load. Simulations are run with randomly generated DAGs with *numTask* = 30, *numEntry* = 10, *maxPred* = 5. The two different settings of DAGs have communication data volume uniformly distributed in [800bit, $\pm 10\%$], and [1000bit, $\pm 10\%$] with task computation load uniformly distributed in [300KCC, $\pm 10\%$].

As shown in Fig. 4.15 and 4.16, the performance of all simulated algorithms are affected by the communication load increment. Among all algorithms, EbTA is affected the most by communication load increment in terms of schedule length, DMR, MECpN, and application energy consumption. Since DCA has most of its tasks executed on the cluster head, it has the least communication tasks scheduled on the channel. Thus, DCA is affected the least regarding energy consumption and MECp when communication load is increased. On the other hand, the RT-MapS algorithms assign tasks on different sensors to speed up execution, which leads to more communication tasks scheduled on $\mathcal{C}$. Thus, the RT-MapS algorithms are affected more by the communication load increment than DCA. On the other hand, due to their broadcast scheduling feature, the RT-MapS algorithms are less affected by communication load changes compared with EbTA. It is observed that, even when communication load increases, the RT-MapS algorithms still outperform DCA and EbTA with smaller MECpN subject to deadline constraints.

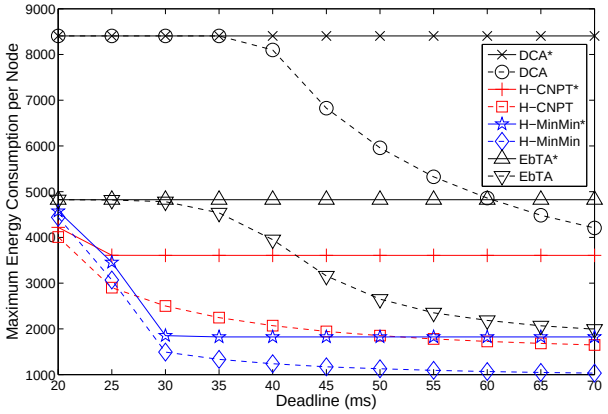


(a) Schedule Length

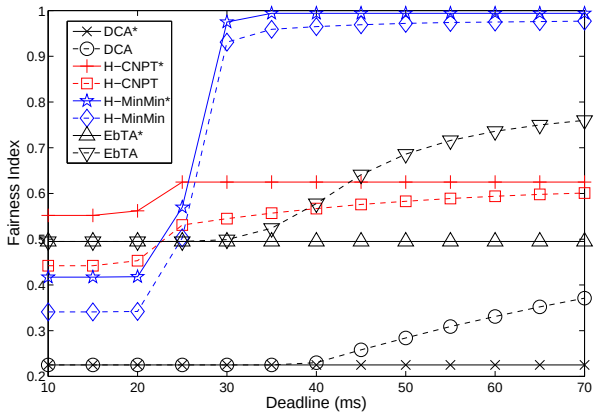


(b) Deadline Missing Ratio

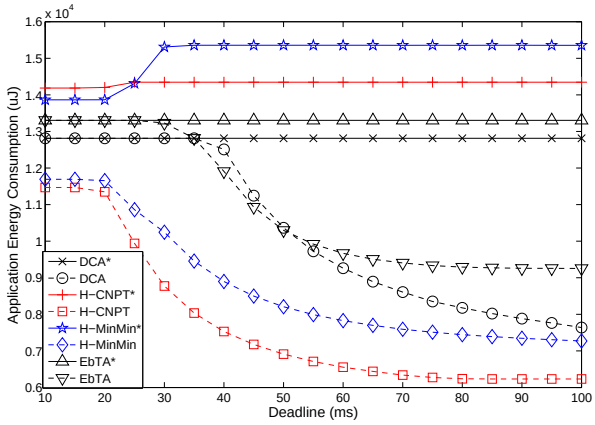
Figure 4.9: RT-MapS: Effect of Application Deadlines in the Time Domain



(a) MECpN

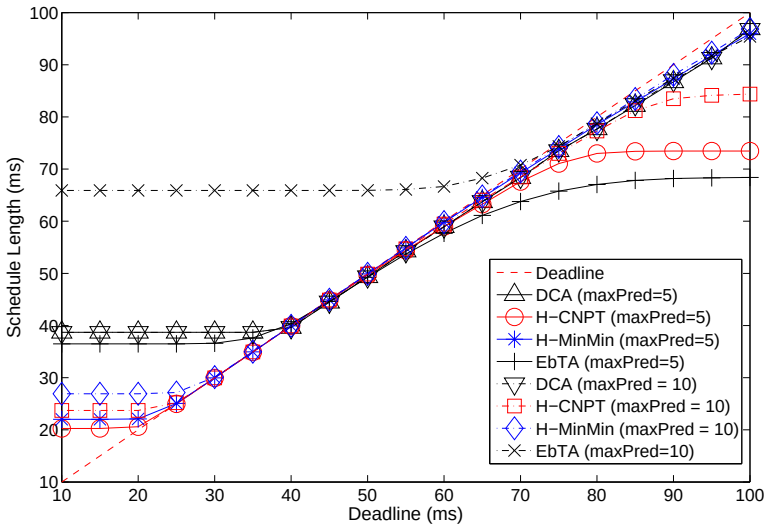


(b) Energy Fairness Index

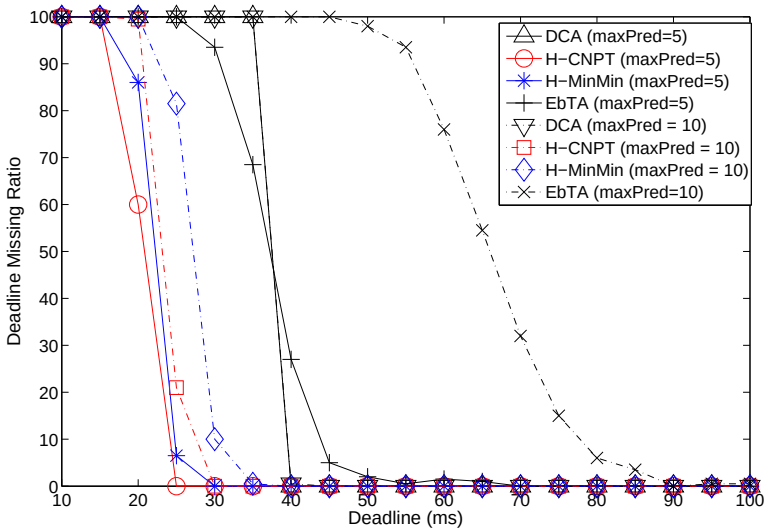


(c) Application Energy Consumption

Figure 4.10: RT-MapS: Effect of Application Deadlines in the Energy Domain

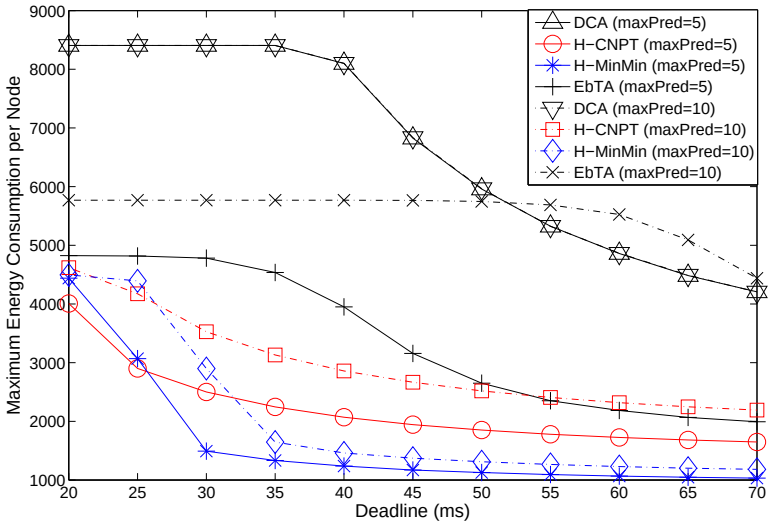


(a) Schedule Length

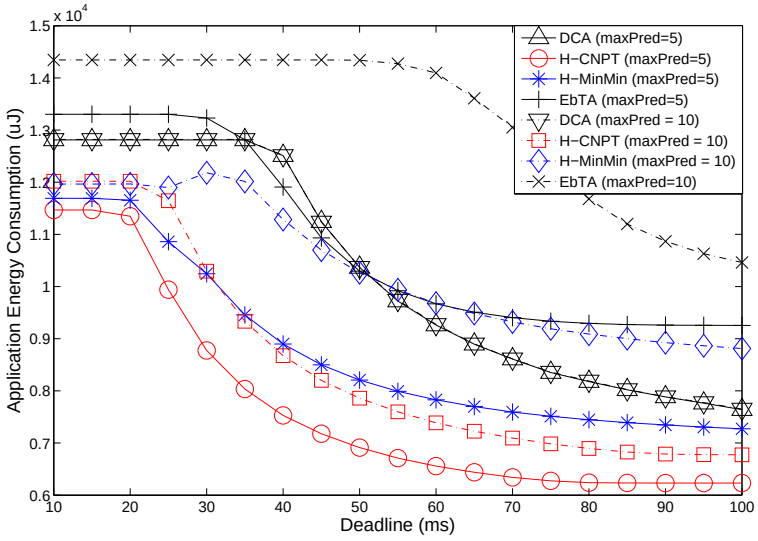


(b) Deadline Missing Ratio

Figure 4.11: RT-MapS: Effect of Inter-Task Dependency in the Time Domain

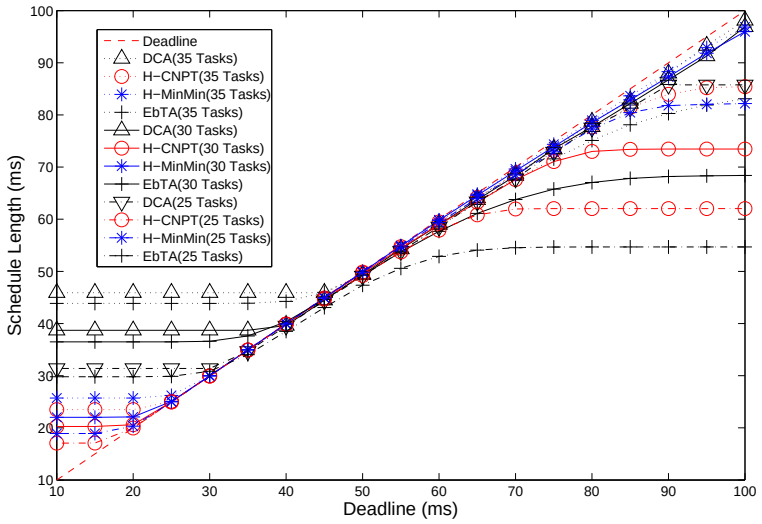


(a) MECpN

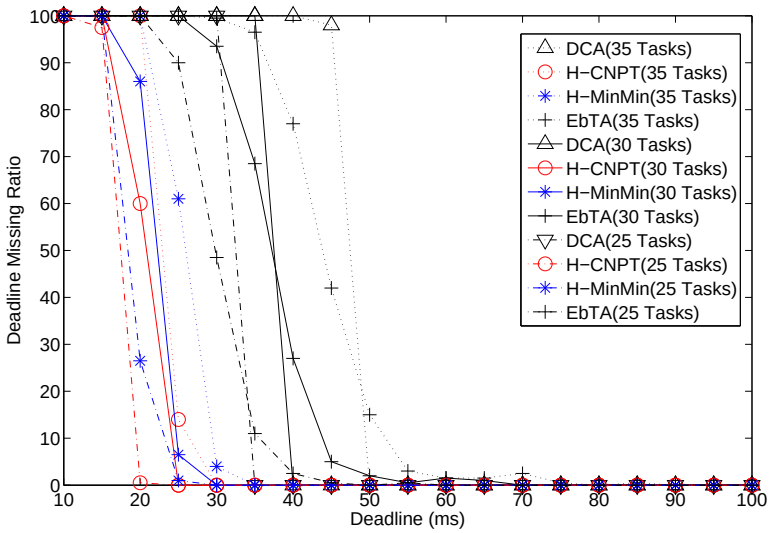


(b) Application Energy Consumption

Figure 4.12: RT-MapS: Effect of Inter-Task Dependency in the Energy Domain

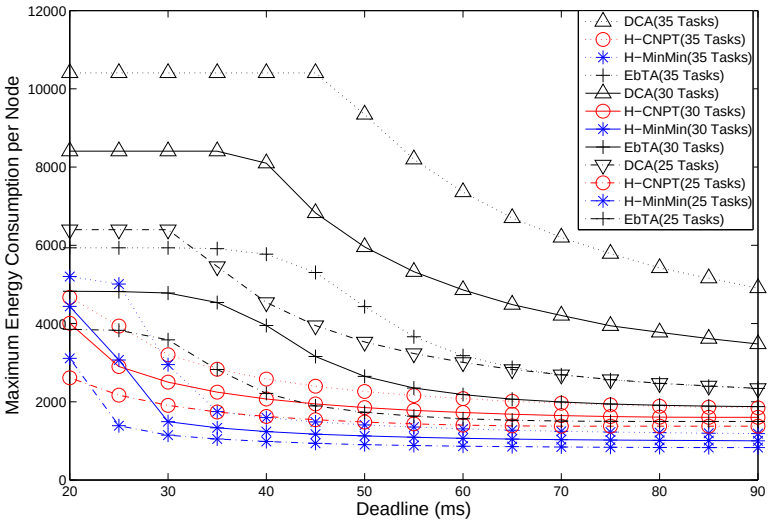


(a) Schedule Length

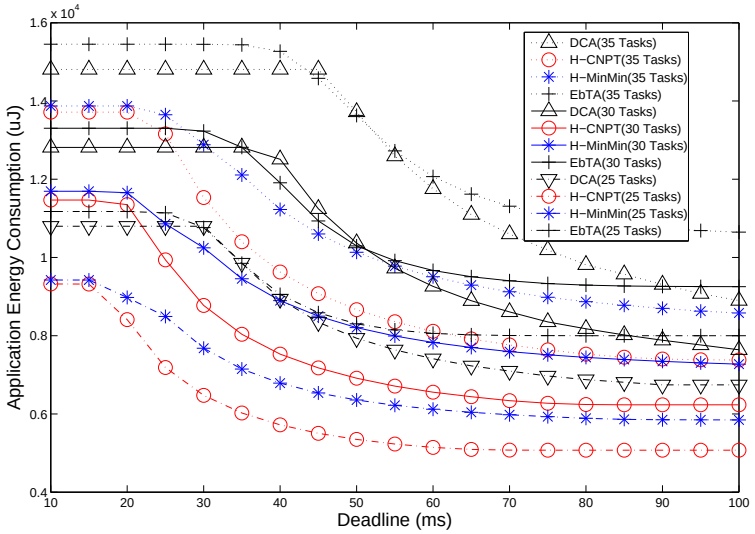


(b) Deadline Missing Ratio

Figure 4.13: RT-MapS: Effect of Number of Tasks in the Time Domain

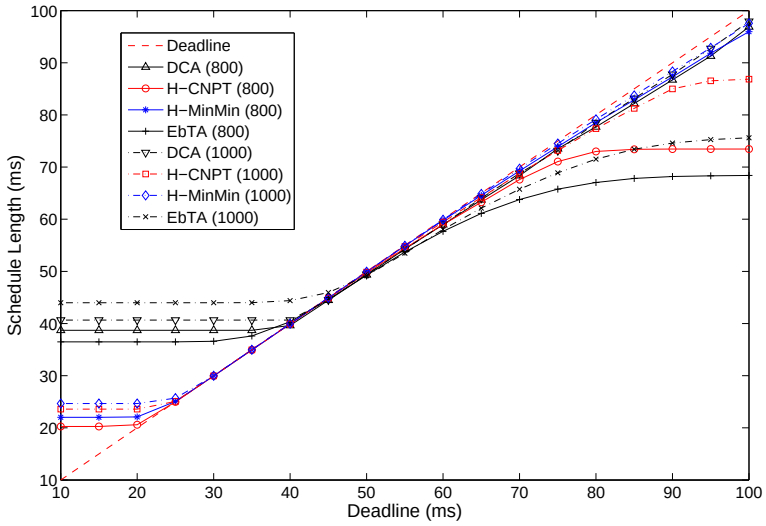


(a) MECpN

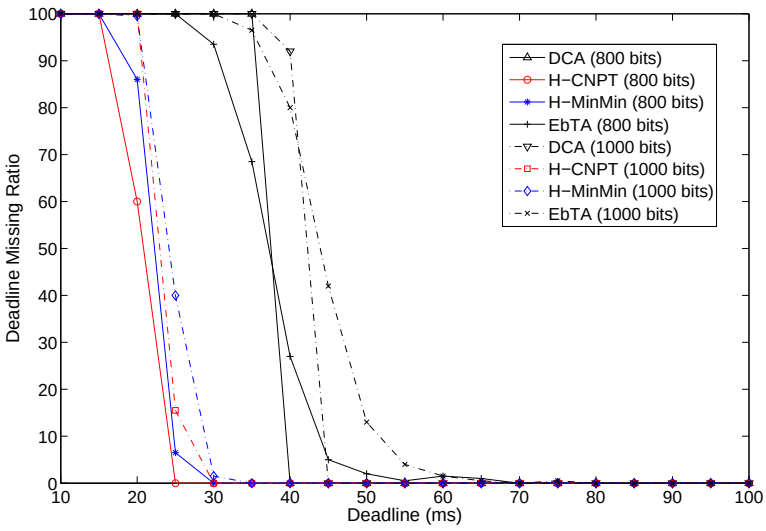


(b) Application Energy Consumption

Figure 4.14: RT-MapS: Effect of Number of Tasks in the Energy Domain

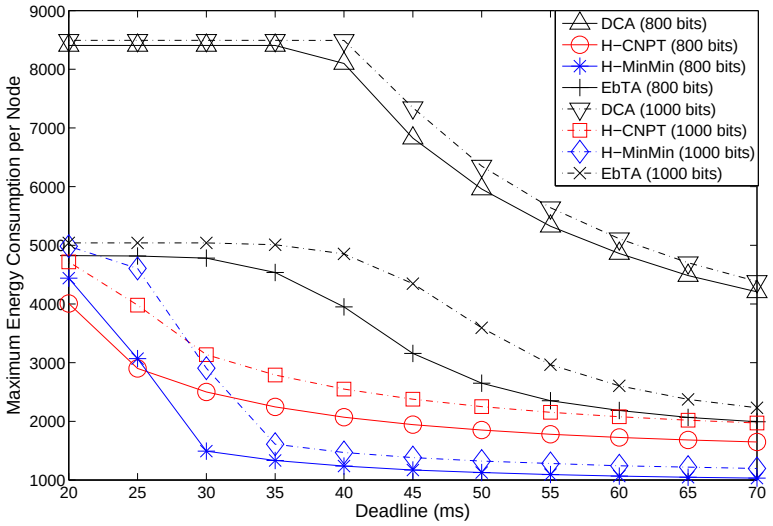


(a) Schedule Length

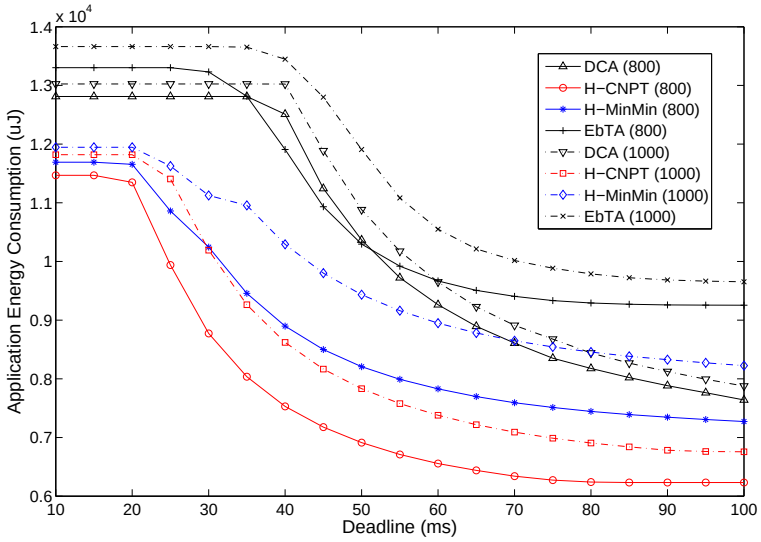


(b) Deadline Missing Ratio

Figure 4.15: RT-MapS: Effect of Communication Load in the Time Domain



(a) MECpN



(b) Application Energy Consumption

Figure 4.16: RT-MapS: Effect of Communication Load in the Energy Domain

CHAPTER 5

REAL-TIME TASK MAPPING AND SCHEDULING IN MULTI-HOP CLUSTERED WIRELESS SENSOR NETWORKS

In Chapter 3 and 4, task mapping and scheduling solutions are presented for single-hop clustered WSNs. However, clustering sensors into single-hop groups leads to a large number of clusters, and consequently comes with the cost of large communication and routing overhead [9] [68] in large-scale WSNs. Many multi-hop clustering algorithms have been proposed for large-scale WSNs [4] [9] [67], which provide better scalability and energy-efficiency. Thus, it is desirable to develop more general task mapping and scheduling solutions for multi-hop clustered WSNs.

In this chapter, we propose *Multi-Hop Task Mapping and Scheduling (MTMS)*, which provides the in-network computation capacity required by arbitrary real-time applications in multi-hop WSNs. The following network assumptions in addition to those in Chapter 2.2 are made for multi-hop WSNs discussed in this chapter:

- Homogeneous sensors are grouped into k-hop clusters with a clustering algorithm such as [4] [9] [67]. 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.

- Location information is locally available within clusters through localization algorithms such as [14] [51].

- Same as that in Chapter 4.1, we assume that sensors are equipped with DVS processors such as StrongARM SA-1100 [52], and DVS adjustment overhead is negligible.

MTMS aims to guarantee application deadlines with the minimum energy consumption. 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:

$$\text{Find } H^o = arg\min energy(H), \tag{5.1}$$

$$\text{where } energy(H) = \sum_{i,k} c_{v_i,m_k} + \sum_{k} CommEng(m_k), \tag{5.2}$$

$$\text{subject to } length(H) = \max_{i,k} f_{v_i,m_k} \leq DL, \tag{5.3}$$

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. MTMS not only maps and schedules *computation tasks* to sensors in parallel to accelerate execution, but also addresses *communication scheduling* among sensors in a *multi-hop* cluster of WSNs. The Hyper-DAG application model presented in Chapter 3.2 is employed in MTMS. A novel model is developed to abstract multi-hop wireless channels. Based on this channel model, multi-hop communication scheduling algorithm is integrated as part of MTMS with the collision avoidance feature. The resulting start and finish times of communication events constitute the schedule used by the MAC. 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.

5.1 Outline of the Proposed MTMS Solution

Similar to RT-MapS in Chapter 4, the proposed MTMS solution also has two phases: *Task Mapping and Scheduling Phase* and *DVS Phase*. In the *Task Mapping and Scheduling Phase*, communication and computation tasks are scheduled with the proposed task schedule search engine (TSSE) algorithms. To guarantee deadline constraints, computation tasks are scheduled with the highest CPU speed in the Task Mapping and scheduling Phase. Two low-complexity TSSE algorithms are developed with the objective to minimize application energy consumption subject to deadline constraints. One TSSE algorithm is an extended version of the Min-Min algorithm[11] [53] for multi-hop WSNs, Multi-hop Min-Min (MMM). Another TSSE algorithm is the Dynamic Critical-path Task Mapping and Scheduling (DCTMP). Our proposed communication scheduling algorithm is embedded in the execution of the MMM and DCTMP algorithms to satisfy the *Dependency Constraint*. The DVS algorithm presented in Chapter 4.4 is implemented in the *DVS phase* to further reduce the energy consumption. In the following sections, the main components of the MTMS solution, namely, communication scheduling algorithm, and the MMM and DCTMP algorithms, are presented.

5.2 Multi-hop Network Channel Modeling

To properly schedule communication events, we model the multi-hop channel as a virtual node $\mathcal{C}$ on which only communication tasks can be executed. Different from the virtual node model in [26] and Chapter 3 and 4, where only single-hop channels are considered, our multi-hop channel model takes potential interference between simultaneous communications into consideration.

Unlike in single-hop networks, there can be multiple simultaneous communications in multi-hop networks. Thus, the virtual node $\mathcal{C}$ in multi-hop channel model should be able to execute multiple communication tasks simultaneously. To avoid interference between scheduled communication tasks, a "*penalty function*" is introduced into the cost function of communication scheduling. Under 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:

$$P_{st}^{ft}(v) = \begin{cases} \infty, if\ \exists \gamma \in T_{st}^{ft}(\mathcal{C}) : S(\gamma) \in N(R(v))\ or\ R(\gamma) \in N(S(v)) \\ 0, otherwise, \end{cases} \tag{5.4}$$

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

- 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 Equation 2.1 and 2.2, 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 $\mathcal{C}$.

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 defer the discussion of alternative penalty functions to our future work.

5.3 Communication Scheduling Algorithm

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 algorithms described in Section 5.4.

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 3.2, 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 denotes the procedure of the data transmission. After v_i^c is finished by $\mathcal{C}$, v_i^c is duplicated to m_r, 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 saving 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. 5.1.

In Fig. 5.1, Steps 4-18 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 21-27. Compared with originating a new communication for each recipient, the broadcast reception method leads to energy saving of one data transmission for each additional data reception.

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. **IF** $v_i \in T(m_r)$ /*No need to communicate if v_i already on m_r*/
2. Return;
3. Find a copy of v_i: $v_i^c \in T(\mathcal{C})$, $S(v_i^c) = m_s$ /*m_s sent v_i before?*/
4. **IF** v_i^c does not exist /*No, unicast scheduling from scratch*/
5. Find $v_i \in T(m_s)$
6. Find time interval [st,ft]:
7. $cost(v_i, st, ft) = \min$ /*Find interval with minimum cost*/
8. $st \geq f_{v_i,m_s}$, $ft - st = t_{v_i,\mathcal{C}}$ /*Make sure v_i can be executed on $\mathcal{C}$*/
9. Schedule a copy of v_i to $\mathcal{C}$:
10. $s_{v_i^c,\mathcal{C}} \leftarrow st$, $f_{v_i^c,\mathcal{C}} \leftarrow ft$
11. $T(\mathcal{C}) \leftarrow T(\mathcal{C}) \cup \{v_i^c\}$
12. Update the energy consumption of m_s
13. Schedule a copy of v_i^c to m_r:
14. $s_{v_i^r,m_r} \leftarrow f_{v_i^c,\mathcal{C}}$
15. $f_{v_i^r,m_r} \leftarrow f_{v_i^c,\mathcal{C}}$ /*Communication tasks' execution time is zero on sensors*/
16. $T(m_r) \leftarrow T(m_r) \cup \{v_i^r\}$
17. Update the energy consumption of m_r
18. Return
19. **ELSE** /*Yes, try broadcast reception first*/
20. $st \leftarrow s_{v_i^c,\mathcal{C}}$, $ft \leftarrow f_{v_i^c,\mathcal{C}}$ /*Consider v_i^c's transmission duration*/
21. **IF** $\nexists \gamma \in T_{st}^{ft}(\mathcal{C}) : S(\gamma) \in N(m_r)$ /*Receiving v_i^c won't be interfered*/
22. Schedule a copy of v_i^c to m_r: /*Receive the broadcasted packet*/
23. $s_{v_i^r,m_r} \leftarrow f_{v_i^c,\mathcal{C}}$
24. $f_{v_i^r,m_r} \leftarrow f_{v_i^c,\mathcal{C}}$ /*Communication tasks' execution time is zero on sensors*/
25. $T(m_r) \leftarrow T(m_r) \cup \{v_i^r\}$, $R(v_i^c) \leftarrow R(v_i^c) \cup \{m_r\}$
26. Update the energy consumption of m_r
27. Return
28. **ELSE** /*Within transmission range of ongoing transmission's sender*/
29. Goto Step 5 /*Need to schedule another transmission of v_i from m_s*/

Figure 5.1: MTMS: Communication Task Scheduling Algorithm between One-hop Neighbors

In our multi-hop communication scheduling algorithm, a routing algorithm is used to obtain the $path = (m_1, ..., m_n)$ from sender m_s to receiver m_r, where $m_1 = m_s$ and $m_n = m_r$. In this paper, we employ the low complexity stateless geographic routing algorithm, GPSR [37]. 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 multi-hop communication scheduling is presented in Fig. 5.2.

5.4 Multi-Hop Task Mapping and Scheduling Algorithms

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

- A computation task can be assigned only on sensor nodes ie., $\forall \gamma_i \in V : t_{v_i,\mathcal{C}} = \infty, c_{v_i,\mathcal{C}} = \infty$
- A communication task can be assigned on sensors or $\mathcal{C}$

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

Figure 5.2: MTMS: Communication Task Scheduling Algorithm

- A communication task assigned on a sensor denotes data stored in node memory, and is ready for processing on the same node. Thus, its execution time and energy consumption are zero.

- A non-entry computation task assigned on a sensor must have all its immediate predecessors available before it can start 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*, *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 [25] and heuristic algorithms are needed to obtain practical solutions. Two task mapping and scheduling algorithms, Multi-hop MinMin (MMM) and Dynamic Critical-path Task Mapping and Scheduling (DCTMS), are presented in this section. To guarantee deadline constraints, both MMM and DCTMS schedule computation tasks with the highest CPU speed.

5.4.1 The MMM Algorithm

Due to its satisfactory performance at relatively low complexity, Min-Min algorithm [53] is modified and implemented in MTMS. The modified Min-Min algorithm is developed for multi-hop environments, and is referred to as the MMM algorithm.

The core of the MMM 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 MMM 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 [53], 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.
- $PE(v_i, m_k)$ 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 v_i's all immediate predecessor 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 tradeoff 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:

$$fit(v_i, m_k, \alpha) = \alpha \cdot NPT(v_i, m_k) + (1 - \alpha) \cdot NPE(v_i, m_k). \tag{5.5}$$

The MMM Algorithm is presented in Fig. 5.3. In the description of MMM, a "mappable" task is either an entry-task or a task that has all immediate predecessors already been

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 tradeoffs between scheduling cost in time and energy domains, the α value is kept unchanged in Steps 2-15. However, different applications may find optimal schedules with different α values.

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 MMM is needed to recover the functionality. To adjust the previous schedule is also a viable solution to quickly recover sensor failures, which will be part of our future work.

5.4.2 The DCTMS Algorithm

Our proposed Dynamic Critical-path Task Mapping and Scheduling (DCTMS) algorithm is composed by the following procedures:

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

The *DCEOTS* procedure calculates the critical-path of Hyper-DAGs, and finds the most critical task of the critical-path to be assigned in the OSSTA stage. Here, a "critical-path" is a set of tasks in a DAG, along which tasks potentially have the largest execution time and may determine schedule lengths. In the *OSSTA* procedure, the selected task will then be experimentally assigned to "active sensors". Among these task-sensor combinations,

Input: Hyper-DAG; sensor set: SS
Output: Schedule H^o of tasks in DAG with optimized energy consumption under deadline constraints
The MMM Algorithm:

---

1. **FOR** $\alpha = 0$; $\alpha \leq 1.0$; $\alpha += 0.1$
2. Assign entry-tasks with *Entry-task Assignment Constraint*
3. Initialize the mappable task list *L*
4. **WHILE** L is not empty /*Repeat until all tasks assigned*/
5. **FOR** task $v_i \in L$ /*Calculate with all (task,sensor) combinations*/
6. **FOR** all computing sensor m_k
7. **IF** $pred(v_i) \not\subseteq T(m_k)$ /*Ensure *communication dependency constraint**/
8. **FOR** $v_n \in pred(v_i) - T(m_k)$
9. **CommTaskSchedule**(v_n,$m(v_n)$,m_k)
10. Assign v_i to m_k, calculate $fit(v_i, m_k, \alpha)$
11. Find m_i^o: $fit(v_i, m_i^o, \alpha)$ is minimum
12. Find the task-sensor pair (v,m): $fit(v, m, \alpha)$ is minimum
13. Assign v to m, remove v from *L*
14. assign $succ(v)$ on m /*Locally assign communication task on sensor*/
15. Update *L* with any new unassigned mappable tasks
16. Among all schedules with different values of α
17. **IF** $\exists H : length(H) \leq DL$ with $\min energy(H)$
18. return H
19. **ELSE**
20. return $H : length(H) = \min$

Figure 5.3: MTMS: MMM Algorithm

the assignment that gives the shortest schedule length is chosen. Here, an "active sensor" is a sensor that either runs computation tasks or participates in communication activities by sending, receiving or forwarding communication tasks. The network topology is taken into consideration when calculating critical-paths in the DCTMS procedure. The communication scheduling algorithms presented in Chapter 5.3 are embedded into the execution of the DCEOTS procedure. Both of the DCTMS and DCEOTS procedures are iteratively executed until all tasks are assigned. The details of the DCTMS algorithm are described in the following sections.

The DCEOTS Procedure

The core of the DCTMS scheduling algorithm is the *DCEOTS Procedure* that dynamically evaluates critical paths. Unlike traditional dynamic critical path scheduling algorithms that have wired connections between processors with fixed communication latency, DCTMS is executed on Hyper-DAGs in multi-hop WSNs. Thus, the communication latency of a communication task is not only determined by data volume but the assignment of the communication task. Depending on locations of senders and receivers, communication tasks may travel various number of hops, which leads to various communication latency of a same communication task in a Hyper-DAG. However, communication latency is needed in evaluating critical paths. Since the selected task will be experimentally assigned to each active sensor, a natural estimation method of communication latency is to take its average across all active sensors. Let AVG_{hop} be the average hop-distance between all active sensors, the average communication latency of a communication task v_i between active sensors be $AVG_{hop} \cdot R_{v_i}/BW$, where R_{v_i} is the data volume of v_i, and BW be the channel bandwidth. Each time when an idle sensor is involved in computation or communication activities, it becomes an active sensor, and the AVG_{hop} is updated accordingly.

The details of the DCEOTS procedure is presented in Fig. 5.4. The *DCEOTS Procedure* dynamically calculates critical paths. Similar to the E-CNPT algorithm in [60], DCEOTS first iteratively calculates the earliest start time $EST(v_i)$ of task v_i by traversing down Hyper-DAGs. For tasks that have already been assigned, their EST equals their scheduled start time. Otherwise, their EST is given by:

$$EST(v_i) = \max_{v \in pred(v_i)} \{EST(v) + t_v\}, \tag{5.6}$$

$$\text{where } t_v = \begin{cases} C_v / f_{CPU}^{max}, v \in R \\ AVG_{hop} \cdot R_v / BW, v \in R, \end{cases} \tag{5.7}$$

where C_v is the the computation load of v.

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

$$LST(v_i) = \min_{v \in succ(v_i)} \{LST(v)\} - t_{v_i}, \tag{5.8}$$

where t_{v_i} has the same definition as t_v in Equation 5.7.

Starting from the exit-task, the path along which tasks have the same value of EST and LST is the critical-path. A task already been assigned is not considered when calculating critical-paths. Thus, a dynamic critical path ends when a task's immediate predecessors are scheduled tasks. Such an unscheduled "top" task that is closest to the scheduled tasks is called a primary critical-node (PC). A "mappable" PC will be passed to the OSSTA procedure for further processing. Here, a mappable task is either an entry-task or a task whose immediate predecessors are already scheduled. If the PC is not mappable, a *secondary critical-path* will be found: Starting from the PC, a task's immediate predecessor with the

Input: Hyper-DAG; Partial schedule on Sensor set SS
Output: Mappable PC or SC
The DCEOTS Procedure:

---

1. Traverse down Hyper-DAG, calculate EST for each task
2. Traverse up Hyper-DAG, calculate LST for each task
3. $v_{top} \leftarrow v_{exit}$ /*search for PC/SC starting from the exit task*/
4. **WHILE** none of $pred(v_{top})$ has been scheduled /*traverse up the primary critical-path*/
5. Find $v_i \in pred(v_{top})$:
6. $EST(v_i) = LST(v_i)$
7. $v_{top} \leftarrow v_i$
8. **IF** v_{top} is mappable /*mappable PC is found*/
9. Return v_{top}
10. **ELSE** /*search for mappable SC*/
11. **WHILE** v_{top} is not mappable /*traverse up the secondary critical-path*/
12. Find $v_i \in pred(v_{top})$:
13. $LST(v_i)$ is minimum
14. $v_{top} \leftarrow v_i$
15. Return v_{top}

Figure 5.4: MTMS: DCEOTS Procedure

minimum LST is iteratively added to the path until a mappable task is found. Such a mappable task on the secondary critical-path is called a secondary critical-node (SC), and is passed to the OSSTA procedure for further processing.

The OSSTA Procedure

The OSSTA procedure is presented in Fig. 5.5. In each iteration of Steps 3-14, the mappable PC or SC found in the DCEOTS procedure is scheduled. For each task v^o, we compare the schedules with v^o assigned to different active sensors. Among all candidate schedules, the schedule with the earliest finish time of v^o is chosen. When assigning a task v^o to m_k, if any immediate predecessor of v^o is not available on m_k, the communication

Input: Hyper-DAG; sensor set: SS
Output: Schedule of v^o
The OSSTA Procedure:

---

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

Figure 5.5: MTMS: OSSTA Procedure

scheduling algorithms are executed to duplicate a copy of the missing immediate predecessor onto m_k. This procedure repeats until all tasks are scheduled. During the scheduling, sensors are scheduled with full speed f_{cpu}^{max} to guarantee deadlines.

5.5 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 task, n communication tasks, and the average in-degree of computation tasks is e/n.

5.5.1 Computational Complexity of MTMS with MMM

The computational complexity of the MTMS solution with the MMM algorithm is analyzed as follows:

In the MMM Algorithm, the loop from Step 8 to Step 9 is executed in $O(\frac{e}{n})$ time, the loop from Step 6 to Step 10 is executed in $O(s)$ time, and the loops starting from Step 5 and Step 4 are both executed in $O(n)$ time. The communication scheduling algorithm has a complexity determined by the routing algorithm. The geographic algorithm GPSR has a complexity of $O(k)$. Thus, the complexity of the *task mapping and scheduling phase* is $O(\frac{e}{n} \cdot s \cdot n^2 \cdot k)$ = $O(ensk)$.

As discussed in Chapter 4.5, the complexity of the DVS algorithm is $O(n + n^2/s)$. Taking both of the *task mapping and scheduling phase* and the *DVS phase* into account, the overall complexity of MMM is $O(ensk+n+\frac{n^2}{s})$. As we assume that sensors are uniformly distributed in a network, we have $s = O(k^2)$ and the overall complexity is $O(enk^3 + \frac{n^2}{k^2})$. Since $e = O(n^2)$ in general, we get $O(n^3k^3 + \frac{n^2}{k^2})$.

5.5.2 Computational Complexity of MTMS with DCTMS

The computational complexity of the MTMS solution with the DCTMS algorithm is analyzed as follows.

The loop starting from Step 3 is executed $O(n)$ times. Similar to the listing stage of the listing stage of the E-CNPT algorithm in Chapter 3, DCEOTS has the computational complexity of $O(n + e)$. The loop between Step 8 and 12 is executed $O(s)$ times, the loop of Step 10 and 11 is executed $O(e/n)$ times. Since the communication scheduling algorithm has complexity of $O(k)$, the overall computational complexity of DCTMS is $O(n \cdot (O(n + e) + O(sek/n))$ = $O(nv + ne + sek)$.

Taking the DVS algorithm into account, the computational complexity of the MTMS solution with DCTMS is $O(n + n^2/s) + O(n^2 + ne + sek)$ = $O(n^2 + ne + sek + n^2/s)$. As we assume that sensors are uniformly distributed in a network, we have $s = O(k^2)$ and the overall complexity is $O(n^2 + ne + ek^3 + \frac{n^2}{k^2})$ = $O(n^3 + n^2k^3 + \frac{n^2}{k^2})$ ($e = O(n^2)$).

5.6 Simulation Results

The performances of the MTMS solution with the MMM algorithm and the DCTMS algorithm are evaluated through simulations, and denoted as MMM and DCTMS in this section, respectively. The performance of DCA and EbTA is also evaluated as benchmarks. DCA is extended with our proposed multi-hop communication scheduling algorithm to deliver the intermediate results of entry-tasks to the cluster head for further processing. DCA algorithm is also implemented with DVS for fair comparison. Simulations are first run on a real-life video surveillance application as a proof of concept. To further evaluate MTMS performance, simulations are run on arbitrary applications with randomly generated DAGs. Our simulations with random DAGs study the following scenarios:

- Effects of application deadline constraints and DVS adjustment
- Effect of number of tasks in applications
- Effect of cluster size
- Effect of communication load
- Comparison of algorithm execution times
- Comparison with EbTA [69] in single-hop clustered networks

In these simulations, we observe schedule length, deadline missing ratio (DMR), and application energy consumption metrics. The schedule length is defined as the finish time of the exit-task of an application, the DMR is defined as the ratio of the number of the simulation runs whose schedule length is larger than the imposed deadline over the number of the overall simulation runs, and application energy consumption includes computation and communication energy expenditure of all sensors.

5.6.1 Simulation Parameters

In our simulation study, the bandwidth of the channel is set to 1Mb/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 Equation 2.1 - 2.3 are in coherence with [52], [64], [30] as follows:
E_{elec} = 50 nJ/b, ε_{amp} = 10 pJ/b/m^2, V_T = 26 mV, C = 0.67 nF, I_o = 1.196 mA, n = 21.26, K = 239.28 MHz/V and c = 0.5 V. The sensors are assumed to be 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.

5.6.2 The Real-life Example of Distributed Visual Surveillance

A simple visual data processing application as shown in Fig. 1.1 is considered in this section. Camera sensor nodes work collaboratively to monitor various objects in a given area. Since information from neighboring camera sensor nodes is highly correlated, locally processing information will significantly decrease data volume to be transmitted. In a distributed visual object recognition scenario, neighboring sensor nodes pair up to exchange information for object recognition. The feature detection algorithms proposed in [59] and [65] for realtime object recognition is exploited. With this approach, features are extracted

locally, followed by the voting operation [65]. An edge detector is applied to extract interest points for each images. For each point of interest, all captured features from different images are fused. Voting is exercised for each interest points to classify features. The final result of detected features and their votes are aggregated and delivered back to base stations.

The collaborative visual object recognition application is abstracted as the DAG shown in Fig. 5.6. Tasks $v_1 - v_4$ are entry tasks which convert original images to binary images using edge detection and interest point detection [65]. 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 [72]. The object recognition in each image is done by "comparing the extraction of feature points and the interest points over edge detectors" [65]. The Hausdorff distances [33] are used as the criteria for image matching and voting. Object information from different cameras are fused to eliminate redundancy in $v_9 - v_{11}$.

We assume that data size generated at each camera sensor is 128 $\times$ 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 40000 KCC, the computation load for matching in $v_9 - v_1 1$ 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 MTMS algorithms, MMM and DCTMS, with the DCA and EbTA algorithm in Table 5.1. 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.

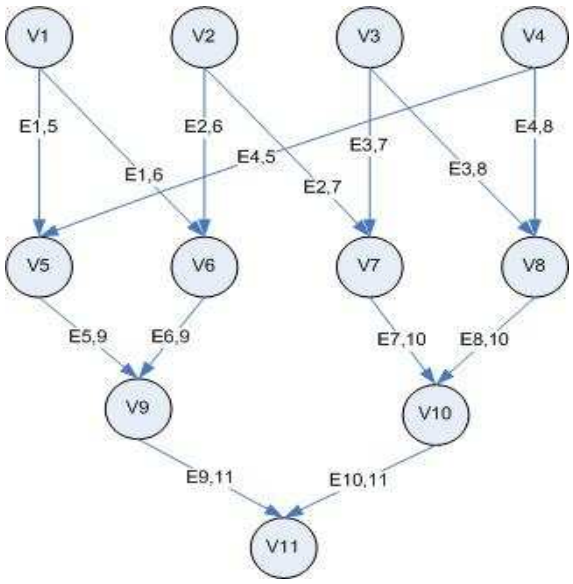


Figure 5.6: DAG for distributed feature extraction application

We compare the energy consumption and schedule length for all three algorithm under two different deadline conditions: $deadline = 0.4s$ and $deadline = 0.8s$. In both scenario, according to the simulation results, both MTMS and EbTA outperform DCA with smaller energy consumption and better capacity to meet deadlines. Regarding the comparison of EbTA, DCTMS and MMM, DCTMS performs the best while MMM delivers the poorest performance in the sense of minimizing energy consumption subject to deadline constraints.

In the example above, sending these four 16K Byte-images will consume about 51 mJ per hop. According to Table I, the energy consumption of processing these images with $deadline = 0.8s$ is about 73 mJ with DCTMS. 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

Deadline(s)	Metrics	DCA	EbTA	MMM	DCTMS
0.4	Energy consumption (mJ)	218.0	131.7	182.1	93.62
	Schedule length (s)	0.798	0.334	0.400	0.400
	Meet deadline	no	yes	yes	yes
0.8	Energy consumption (mJ)	218.0	92.6	97.3	72.7
	Schedule length (s)	0.798	0.703	0.800	0.751
	Meet deadline	yes	yes	yes	yes

Table 5.1: MTMS: Simulation with the Object Recognition Example

the results is smaller than directly delivering original images as long as clusters are more than two hops away from base stations, which is satisfied in most large-scale WSNs.

5.6.3 Simulation with Random DAGs

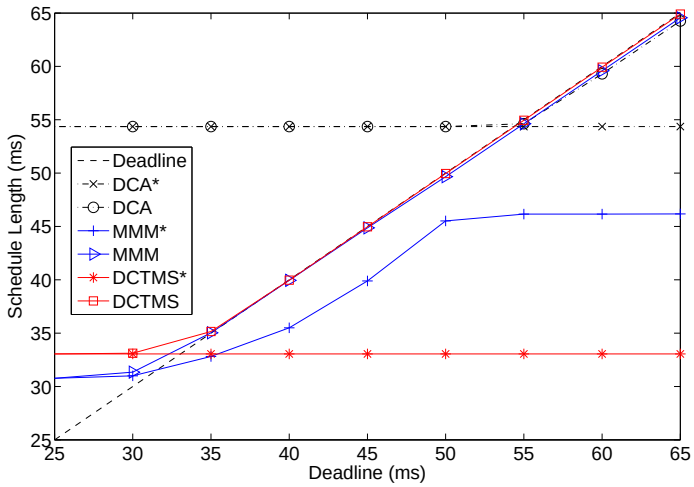
To evaluate MTMS performance for arbitrary applications, simulations are run on randomly generated DAGs which are scheduled on randomly created multi-hop clusters. Random DAGs are created based on three parameters: The number of tasks *numTask*, the number of entry-tasks *numEntry*, and the maximum number of predecessors *maxPred*. Unless specifically stated, the number of each non-entry 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 $\pm 10\%$], and [800 bits $\pm 10\%$], respectively. The sensors are uniformly distributed on disc of radius $k \cdot r$ to form a $k - hop$ connected 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.

Effect of Application Deadlines and DVS adjustment

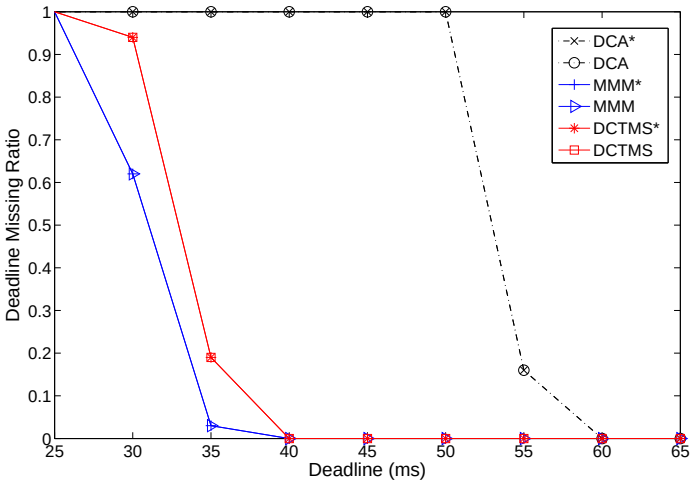
We investigate the effect of application deadlines and DVS adjustment with 250 pairs of randomly created DAGs and 3-hop clusters. The parameters of DAGs considered for this set of simulations are *numTask* = 40, *numEntry* = 10, and *maxPred* = 10. To evaluate the effect of DVS, the performance of DCA, MMM and DCTMS before the voltage adjustment (denoted as DCA* , MTMS*, DCTMS*, respectively) are also investigated. In this section, two more metrics evaluating energy consumption balance are investigated, namely, the Maximum Energy Consumption per Node (MECpN) and the Node Energy Consumption Fairness Index (FI) defined in Chapter 4.3.2.

The schedule length performance of investigated algorithms is shown in Fig. 5.7, while the energy consumption performance is demonstrated in Fig. 5.8.

As shown in Fig. 5.7(a) and Fig. 5.7(b), MTMS has better capability to meet small deadlines compared with DCA. When deadlines are very small, even though 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 superior performance of MTMS over DCA is due to the fact that is that MTMS can have more sensors involved in parallel to process information, while DCA has only one sensor for high-level data processing. Regarding the comparison of the MTMS algorithms themselves, MMM outperforms DCTMS in term of meeting deadlines and providing smaller schedule lengths and DMR when deadlines are small. In each iteration of task scheduling in DCTMS (Fig. 5.5), the schedule space with only one PC or SC task assigned to different active sensors is searched. On the other hand, MMM experiments all possible task-sensor combinations and chooses the task-sensor combination that delivers the minimum scheduling cost. Therefore, MMM searches



(a) Schedule Length



(b) Deadline Missing Ratio

Figure 5.7: MTMS: Effect of Application Deadlines in the Time Domain

a much larger schedule space than DCTMS, and more likely is able to find a feasible solution.

Regarding energy consumption balance, DCA performs the worst as most activities are loaded on cluster heads. Compared with DCTMS, MMM delivers more unbalanced

schedules as shown in Fig. 5.8(b) and Fig. 5.8(c), especially when deadline increases. In MMM*, energy consumption efficiency is part of the objective function. When deadlines constraints are relaxed, MMM* tends to schedule tasks on fewer sensors to conserve communication energy consumption, which leads to unbalanced energy consumption. On the other hand, DCTMS* primarily aims to deliver the shortest schedule by evenly distributing computation loads among sensors, which leads to more balanced schedules.

Regarding the DVS effect on energy consumption, DCA* has better energy consumption performance than MMM* and DCTMS* for most scenarios according to Fig. 5.8(a). However, by implementing DVS algorithm, the energy consumption of MMM and DCTMS are significantly reduced, and are smaller than that of DCA. Our DVS algorithm is an effective approach for energy efficient solutions, as shown in Fig. 5.8. 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 16-19% energy compared with the scenarios without the DVS adjustment. This energy saving stems from eliminating "schedule holes" caused by the unbalanced load of sensors and communication scheduling. When deadlines increases and are sufficiently large, the DVS adjustment results in about 40 % energy consumption savings by "pushing" the schedule length close to the deadline in MMM. Though the DVS adjustment may increase schedule lengths (Fig. 5.7(a)), the deadline missing ratio is not affected (Fig. 5.7(b)) for any of the simulated deadline values. Regarding the comparison of the MTMS algorithms themselves, MMM slightly outperforms DCTMS in terms of application energy consumption for most simulated scenarios. This performance stems from the factor that MMM* delivers shorter schedule lengths than DCTMS*, which enable more aggressive DVS adjustment and consequently better energy conservation in general. On the other hand, the unbalanced schedule of MMM may also compromise the

ability of the DVS adjustment, which leads to the point that the energy consumption of DCTMS is smaller than MMM in Fig. 5.8(a).

Effect of Number of Tasks

To investigate the effect of number of tasks in applications, simulations are run on randomly generated DAGs with 40, 45, 50 tasks (*numEntry* = 10, *maxPred* = 10). For a fair comparison, each set of 40, 45, 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. 5.9(b), 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 as shown in Fig. 5.9(c). This property is also reflected with schedule length presented in Fig. 5.9(a). Thus, MTMS has a better scalability compared with DCA regarding schedule length and deadline missing ratio. Regarding the comparison of MMM and DCTMS, they are equally affected by the task number increase. In most simulated scenarios, MTMS outperforms DCTMS with smaller energy consumption and DMR.

Effect of 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 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 Figure 5.10. As the cluster size increases, the performance of DCA degrades correspondingly, while the performance of MTMS is less affected. Regarding the comparison of the MTMS algorithms themselves, when deadlines are sufficiently larger, the schedule lengths of MMM and DCTMS are barely affected by cluster size increase. When deadlines are small, the schedule lengths and energy consumption of MMM and DCTMS increase when cluster size increases. In terms of energy consumption, MMM is affected more than DCTMS by cluster size as shown in Fig. 5.10(a).

Comparison of Heuristic Execution Time

Execution time is also an important factor to evaluate heuristic algorithms. As we have analyzed in Section 5.5, the number of tasks and cluster size both have effect over the computational complexity of MMM and DCTMS. In this section, the relative execution time of MMM over DCTMS is tested. We run simulation with random DAGs of different number of tasks (*numTask* = 40, 45, 50) over 2-hop, 3-hop and 4-hop clusters. Each result represents the average of 250 simulation runs. In both scenarios, we set *numEntry* = 10 and *maxPred* = 10. As shown in Table 5.2, the variation of the number of tasks and the cluster size have almost the effect on the computation time of MMM and DCTMS. For all investigated scenarios, DCTMS is about 24-25 times faster than MMM. Due to its better schedule length performance, MMM is suitable for WSNs with critical real-time requirements. For a WSN that updates its applications frequently, DCTMS can be favored over MMM due to DCTMS's shorter schedule computation time.

Cluster Size	Number of Tasks		
	40	45	50
2-Hop	24	24.1	24.4
3-Hop	24.2	24.4	24.7
4-Hop	23.9	24.1	24.3

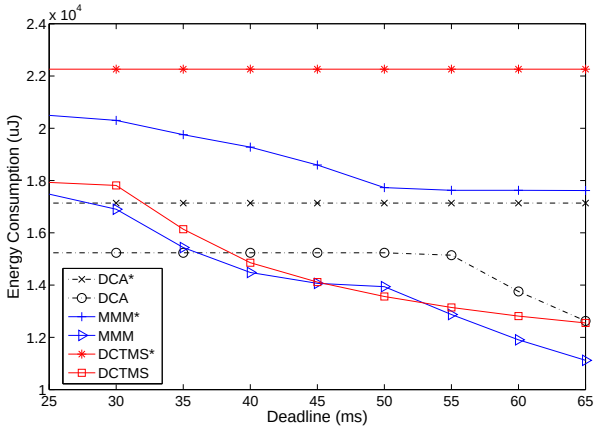
Table 5.2: MTMS: Execution Time Ratio of MMM to DCTMS

Comparison with EbTA [69]

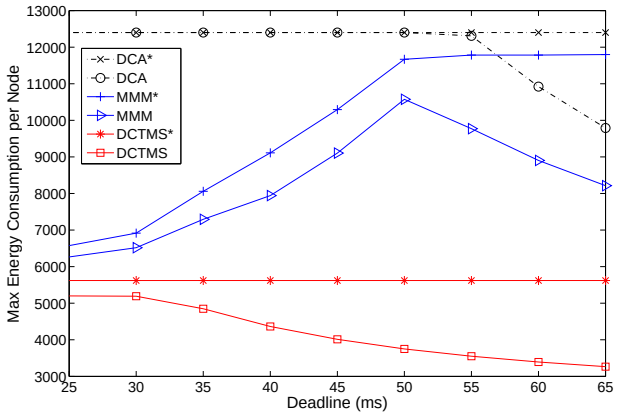
To further evaluate our proposed solution, we compare the performance of MTMS with EbTA [69]. Since EbTA is not designed for multi-hop networks, we run simulations for single-hop, single-channel clusters. Due to the small scale of a single-hop cluster (5 sensors as assumed in Section 5.6.1), performances are evaluated with applications of less computation load. The presented results are the average of 250 simulation runs of random DAGs with *numTask* = 20, *numEntry* = 5 and *maxPred* = 5. The metrics we observe are schedule length, application energy consumption, deadline missing ratio, and MECpN.

As shown in Fig. 5.11 and 5.12, 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

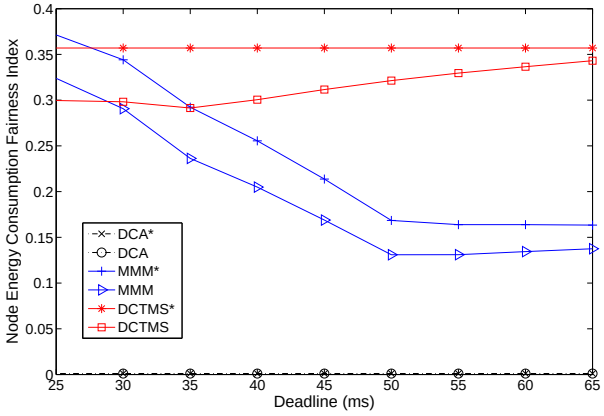
affected by the communication load between computation tasks of an application compared with EbTA.



(a) Application Energy Consumption

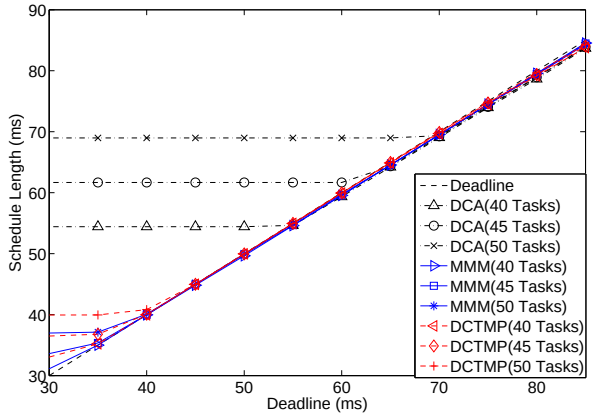


(b) Max Energy Consumption per Node (MECpN)

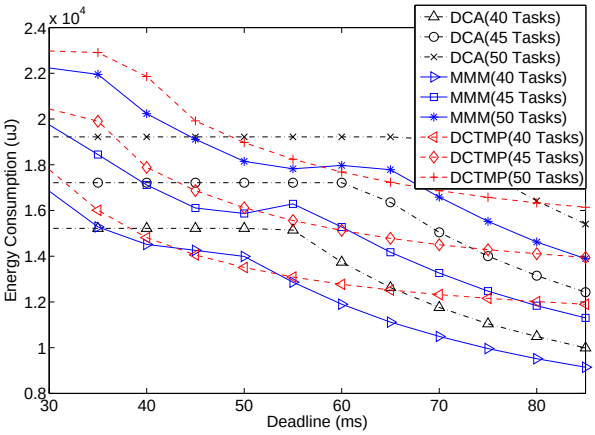


(c) Node Energy Consumption Fairness Index (FI)

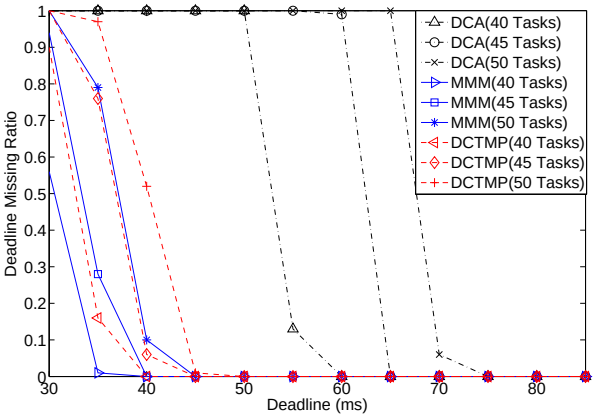
Figure 5.8: MTMS: Effect of Application Deadlines in the Energy Domain



(a) Schedule Length

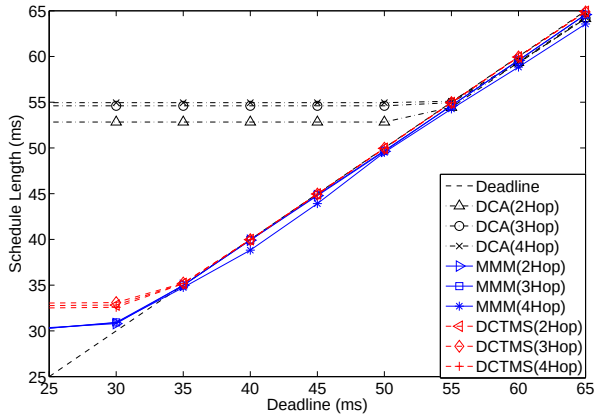


(b) Energy Consumption

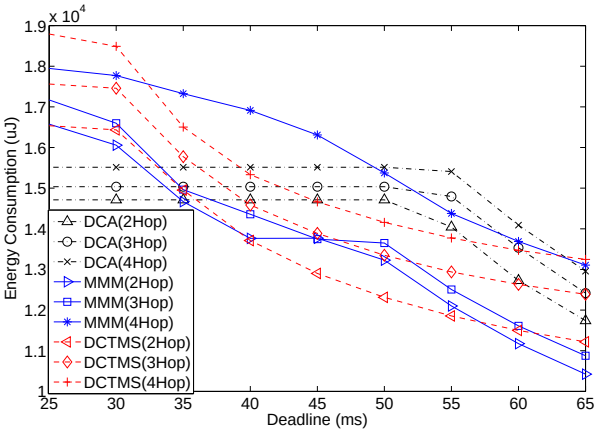


(c) Deadline Missing Ratio

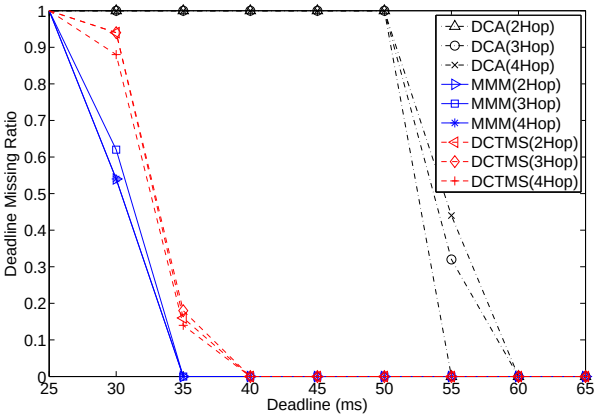
Figure 5.9: MTMS: Effect of Number of Tasks (40 tasks vs 45 tasks vs 50 tasks)



(a) Schedule Length

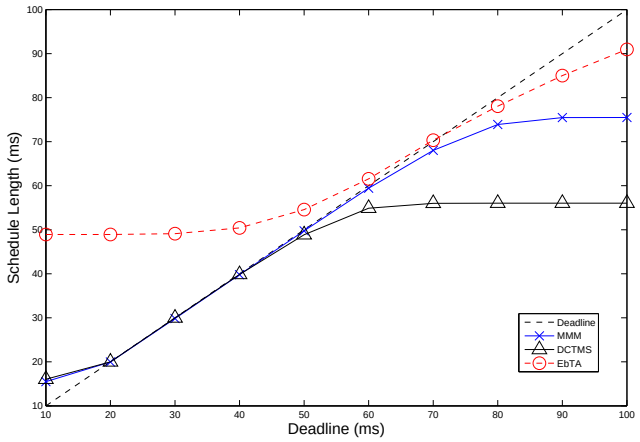


(b) Energy Consumption

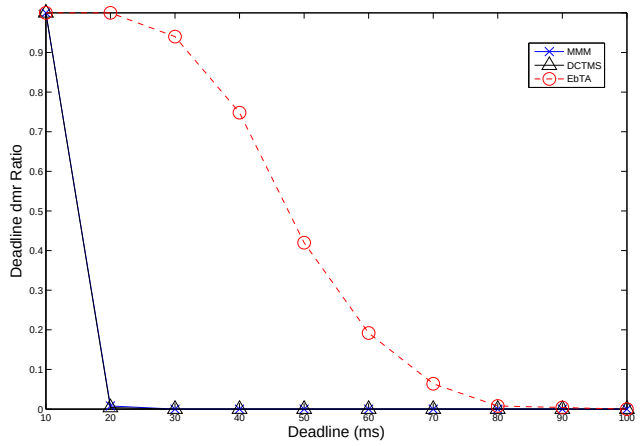


(c) Deadline Missing Ratio

Figure 5.10: MTMS: Effect of Cluster Size

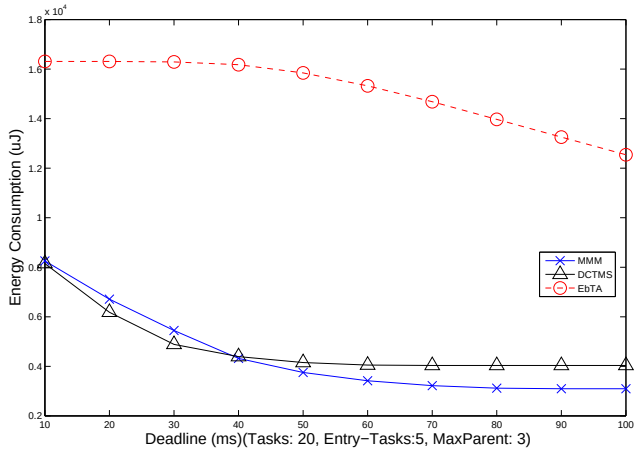


(a) Schedule Length

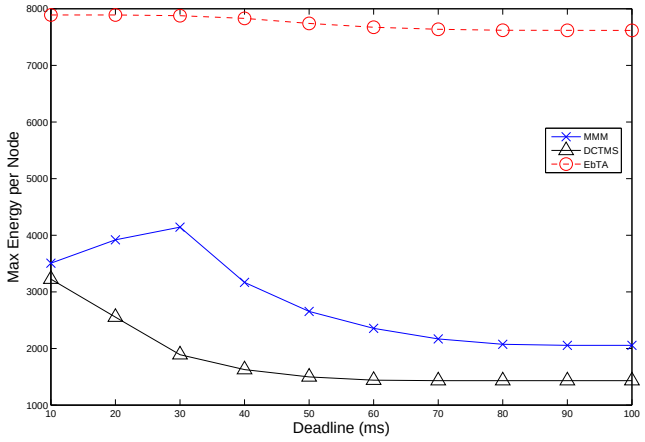


(b) Deadline Missing Ratio

Figure 5.11: MTMS: Performance Comparison with EbTA in the Time Domain



(a) Application Energy Consumption



(b) Max Energy Consumption per Node

Figure 5.12: MTMS: Performance Comparison with EbTA in the Energy Domain

CHAPTER 6

ADAPTIVE SENSOR FAILURE HANDLING

In WSNs, sensors are prone to failures. In case of sensor failures, the current application executing instance is stopped. Furthermore, the schedules previously created by the scheduling algorithms may not be feasible solutions. In such cases, the WSN's functionality needs to be recovered as soon as possible with a promptly generated schedule for the subsequent application executing instances. Instead of rescheduling from scratch, which can be time consuming, low-complexity recovery algorithms are preferred.

Since energy-constrained applications are considered in Chapter 3 for single-hop clustered WSNs, we first present a sensor failure recovery algorithm for the EcoMapS solution. In Chapter 4 and Chapter 5, real-time task mapping and scheduling solutions are presented. Thus, we also propose a sensor failure handling algorithm in its general form for real-time applications in multi-hop clusters. It should be noted that the prompt network functionality recovery may come at the cost of degraded performance. Therefore, we check the performance after sensor failure handling. If the performance degrades to a certain level, the scheduling algorithm should be executed to find a schedule with better performance.

6.1 Sensor Failure Handling for EcoMapS in Single-Hop Clustered WSNs

In single-hop clusters where sensors are within each other's transmission range, sensors are identical from the perspective of network topology. Thus, as long as spared sensors exist, sensor failure handling is a trivial problem, which can be solved by replacing failing sensors with functioning idle sensors. Therefore, we mainly focus on sensor failure handling when there is no idle sensors in this section.

6.1.1 The Proposed Single-Hop Sensor Failure Handling Algorithm

Let the original schedule be H^o, and the failing sensor be m_f. The strategy of the quick recovery algorithm is to merge the tasks of the m_f onto the sensor m^o that has the maximum idle time ratio $IR(m_k)$ to balance computation load among sensors. Here, the idle time ratio $IR(m_k) = e_0^{length(H)}(m_k)/length(H)$ is the ratio of m_k's CPU execution time $e_0^{length(H)}$ to the schedule length $length(H)$. If there are more than one sensor failures, the quick recovery algorithm is iteratively executed to handle the failures one by one. The rationale behind merging the tasks of the failing sensor onto another sensor instead of re-distributing the tasks among all of the working sensors is to guarantee the energy consumption constraint, as proved in *Theorem 1*. The quick recovery algorithm is shown in Fig. 6.1, where TH is the threshold of unacceptable schedule length degrade in quick recovery.

Theorem 1. *The recovered scheme H^s still meets the energy consumption budget constraint, that is, if $energy(H^o) \leq EB$, then $energy(H^s) \leq EB$.*

Proof. The energy consumption of a schedule H is composed of computation energy ($compEng(H)$) and communication energy ($commEng(H)$). Since $compEng(H)$ is fixed

Input: Failing sensor m_f, original sensor set SS, original schedule H^o
Output: Recovered schedule H^s
Single-Hop QuickRecovery Algorithm:

---

1. **IF** $\exists m_{idle} \in SS - \{m_f\} : T(m_{idle}) = \emptyset$ /*there are idle sensors*/
2. Reassign T(m_f) onto m_{idle}
3. **ELSE** /*no free sensor, have to merge the tasks*/
4. Find $m^o \in SS - \{m_f\}$: $IR(m^o)$ is maximum
5. $t \leftarrow 0$, $\Delta t \leftarrow 0$ /*initialize task adjustment parameters*/
6. **FOR** unadjusted task $v_i \in S = T(m_f) \cup T(m^o) : s_{v_i}$ is minimum
7. **IF** $v_i \in V$ /*computation task*/
8. Schedule v_i onto m^o
9. **ELSE** /*communication task*/
10. **IF** there is a duplicated copy of v_i in S
11. Remove the duplicated copy
12. Find the copy of v_i on $\mathcal{C}$: v_i^c
13. **IF** m_f/m^o are the only sender/receiver of v_i^c
14. Remove v_i^c from $\mathcal{C}$
15. **ELSE IF** $pred(v_i) \in S$ /*send result to other tasks*/
16. Schedule v_i right after $pred(v_i)$
17. **IF** $succ(v_i) \not\subseteq S$ /*affect tasks on other sensors*/
18. Find the copy of v_i on $\mathcal{C}$: v_i^c
19. **IF** $f_{v_i,m^o} > s_{v_i^c,\mathcal{C}}$
20. $\Delta t \leftarrow max(\Delta t, f_{v_i,m^o} - s_{v_i^c,\mathcal{C}})$
21. $t' \leftarrow f_{v_i^c,\mathcal{C}}$
22. **FOR** $m_l \in SS \cup \{\mathcal{C}\} - \{m_f, m^o\}$
23. Postpone unadjusted $\gamma_j \in T_t^{t'}(m_l)$ by Δt
24. t $\leftarrow t'$
25. **ELSE** /*receive result from other sensors*/
26. Find the copy of v_i on $\mathcal{C}$: v_i^c
27. $t' \leftarrow f_{v_i^c,\mathcal{C}}$
28. **FOR** $m_l \in SS \cup \{\mathcal{C}\} - \{m_f, m^o\}$
29. Postpone unadjusted $\gamma_j \in T_t^{t'}(m_l)$ by Δt
30. t $\leftarrow t'$
31. Postpone all unadjusted tasks by Δt
32. **IF** $\frac{length(H^s)}{length(H^o)} > TH$
33. *Run EcoMapS Scheduling Algorithm*

Figure 6.1: Quick Recovery Algorithm for EcoMapS in Single-Hop Clusters

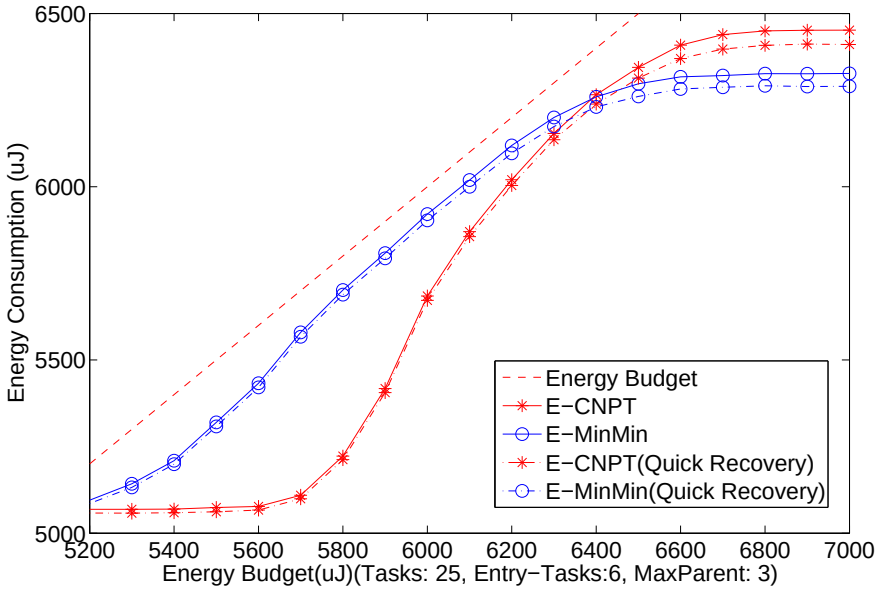
for an application in homogeneous WSNs, $compEng(H^s) = compEng(H^o)$ holds. $commEng(H)$ is determined by the communication tasks assigned on $\mathcal{C}$. According to Step 14, 23, and 29 of the *quickRecovery* algorithm, the only operations related with the communication tasks on $\mathcal{C}$ are task removals and task shifting in time domain. In other words, no new tasks are assigned to $\mathcal{C}$ and, therefore, no additional energy is consumed for communication. Hence, $commEng(H^s) \leq commEng(H^o)$ holds. If $energy(H^o) \leq EB$, then $energy(H^s) = compEng(H^s) + commEng(H^s) \leq compEng(H^o) + commEng(H^o) = energy(H^o) \leq EB$ holds, as well. □

6.1.2 Simulation Results

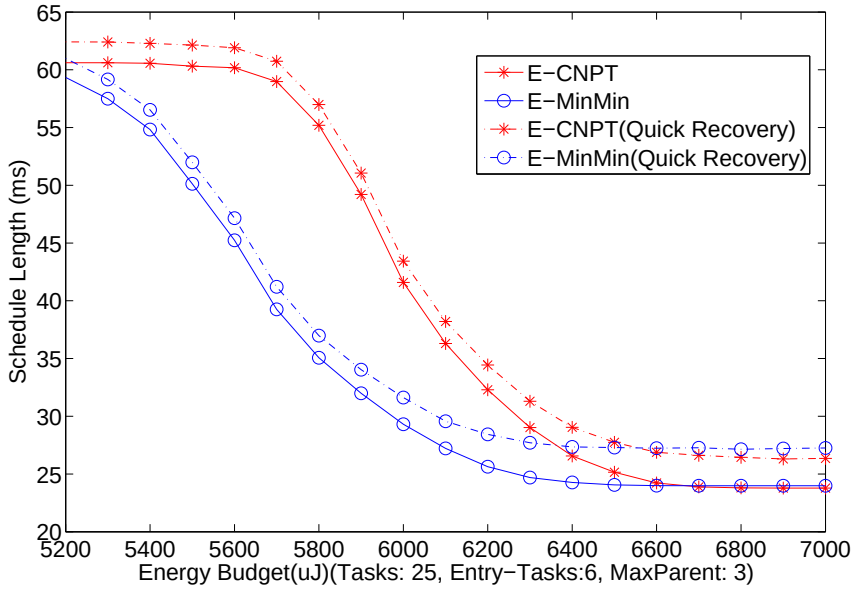
The Quick Recovery Algorithm for EcoMapS is evaluated in this section. Since the recovery mechanism with idle sensors as backup is trivial, the tested scenarios only consider task merging cases without idle sensors. The random DAGs considered in the simulations have the parameters of *numTask* = 25, *numEntry* = 6, and *maxPred* = 3. The simulated scenarios are generated by randomly selecting one failing sensor and merging its tasks onto other working sensors using the quick recovery algorithm presented in Fig. 6.1. From Fig. 6.2(a), it can be observed that as long as the original schedule meets energy consumption constraints, the recovered schedule satisfies the constraint as well. As we discussed in the proof of Theorem 1, task merging leads to less energy consumption at the cost of longer schedule lengths according to Fig. 6.2(b).

6.2 Sensor Failure Handling for Real-Time Applications in Multi-Hop Clustered WSNs

In Chapter 6.1, we present the sensor failure handling algorithm for EcoMapS in single-hops clustered WSNs. However, sensor failure handling is more complicated in multi-hop



(a) Energy Consumption



(b) Schedule Length

Figure 6.2: Performance of Quick Recovery for EcoMapS

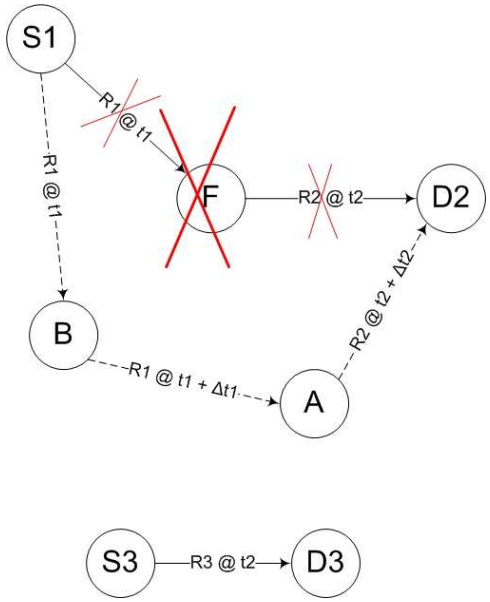


Figure 6.3: MTMS Recovery Demonstration

environments, where sensors have different locations with different neighbors. Replacing a failing sensor with an idle one is no longer a trivial problem in multi-hop clusters: Depending on the location of the failing and alternative sensors, a previously one-hop transmission may become a multi-hop communication, and a formerly collision-free data delivery may cause interference if not properly adjusted. A simple demonstrative example is shown in Fig. 6.3, where sensor F is the failing sensor, and sensor A is the alternative sensor of F. When F is replaced by A, the previous transmission $R1$ from $S1$ to F starting at $t1$ must be re-routed through sensor B as A is two hops away from $S1$. Such packet rerouting with a longer path is referred to as "path extension" in this section. For the delivery of communication task $R2$ to sensor $D2$, it will still be a single-hop communication since A is a single-hop neighbor of $D2$. However, A's 1-hop neighbor $D3$ is previously scheduled

to receive $R4$ starting at $t2$. Thus, the delivery of $R2$ by A must be adjusted to avoid interfering $D3$'s reception. Therefore, potential communication interference and path extension should be addressed when calculating alternative schedules in multi-hop clustered WSNs, as demonstrated in Fig. 6.3. When there is no idle sensor to replace the failing sensor, the tasks of the failing sensor should be merged to an alternative sensor in a similar procedure as presented in Section 6.1. During the task merging procedure, potential path extension and communication interference should also be handled.

6.2.1 The Proposed Multi-Hop Sensor Failure Handling Algorithm

When sensor failures occur, we select an idle sensor among the failing sensor's one-hop neighbors to replace its functionality if idle sensors exist among the failing sensor's single-hop neighbors. The tasks previously assigned to the failing sensor are then reassigned to the alternative sensor. If there is no idle sensor among the failing sensor's single-hop neighbors, an alternative sensor is selected to which the failing sensor's tasks are merged. When reassigning and merging tasks, interference avoidance and packet rerouting must be addressed. Since interference avoidance and packet rerouting may lead to larger schedule lengths, a reverse procedure of the DVS algorithm presented in Chapter 4.4 is developed, which is referred to as the DVS^{-1} algorithm. The DVS^{-1} algorithm is implemented before reassigning the failing sensor's tasks, which increases CPU speed to guarantee deadline constraints. After the tasks of the failing sensor is reassigned, the DVS algorithm in Chapter 4.4 is exercised to conserve energy consumption.

Optimal Alternative Sensor Selection

We first introduce the alternative sensor selection algorithm in Fig. 6.4 when there are idle sensors among the failing sensor's one-hop neighbors. As discussed above, reassigning failing sensors' tasks may lead to path extension with larger schedule length and energy consumption. Therefore, the primary objective of selecting alternative sensors is to minimize the number of path extension. To achieve this goal, we must select the alternative sensor m_a that has the maximum *"similarity factor (SF)"* with the failing sensor m_f. Here, SF is the measurement of the feasibility of replacing m_f by m_a: The higher SF is, the shorter path extension becomes, and the better performance is achieved by the alternative schedule. The similarity factor $SF(m_a, m_f)$ is defined as follows:

- Assume that the number of packets exchanged between m_f and its one-hop neighbor m_k is $d(m_f, m_k)$.
- m_f's overall communication degree is $DEG(m_k) = \sum d(m_f, m_k)$, where $m_k \in N(m_f)$, and $N(m_f)$ is the set of m_f's one-hop neighbors.
- A sensor m_k's connectivity function $c(m_k, m_l)$ equals 1 if sensor m_l is m_k's one-hop neighbor, and 0 otherwise.
- When replacing m_f by m_a, the number of packets that can still be exchanged within one hop between m_a and m_f's single-hop neighbors is $\sum d(m_f, m_k) \cdot c(m_a, m_k)$, where $m_k \in N(m_f)$.
- Let $SF(m_f, m_a)$ be defined as

$$SF(m_f, m_a) = \frac{\sum d(m_f, m_k) \cdot c(m_a, m_k)}{DEG(m_k)}, \tag{6.1}$$

Input: Failing sensor m_f, original sensor set SS, original schedule H^o
Output: Alternative sensor m_a to which $T(m_f)$ are reassigned
AlterSel()

---

1. $DEG \leftarrow 0$ /*Initialization*/
2. **FOR** sensors $m_k \in N(m_f)$ /*caculate m_f's overall communication degree*/
3. $\quad DEG \leftarrow DEG + d(m_f, m_k)$
4. **FOR** idle sensors $m_k \in N(m_f)$ /*candidate alternative sensors*/
5. $\quad SF(m_f, m_k) \leftarrow 0$
6. $\quad$ **FOR** sensor $m_l \in N(m_f)$, $m_l \neq m_k$ /*calculate SF of each candidate*/
7. $\quad\quad SF(m_f, m_k) \leftarrow SF(m_f, m_k) + \frac{d(m_f, m_k) \cdot c(m_k, m_l)}{DEG}$
8. Among these candidates, find sensor m_a:
9. $\quad SF(m_f, m_a)$ is maximum.

Figure 6.4: Alternative Sensor Selection Algorithm for MTMS in Multi-hop Clusters

where $m_k \in N(m_f)$. Here, $SF(m_f, m_a)$ indicates the ratio of m_f's communication that do not need path extension when m_f is replaced by m_a.

When there is no idle sensor among the failing sensor's one-hop neighbors, the failing sensor's tasks must be merged to one of its neighbors that already have tasks assigned. To balance workload among sensors, the sensor that has the minimum computation and communication activities is selected. When a tie occurs, the sensor with the highest similarity factor is favored to minimize path extension. The alternative merging sensor selection procedure is shown in Fig. 6.5.

The DVS^{-1} Algorithm

Since the task reassignment procedure may prolong schedule lengths, CPU speed should be increased to compensate the delay. Different from the DVS algorithm that is presented in Chapter 4.4, the DVS^{-1} algorithm aims to maximize the CPU speed of task execution

Input: Failing sensor m_f, original sensor set SS, original schedule H^o
Output: Alternative sensor m_a to which $T(m_f)$ are merged
MergeSel()

---

1. Among m_f's one-hop neighbors
2. Find S: E_{m_k} is minimum for $m_k \in S$ /*E_{m_k} is the energy consumption of m_k*/
3. **IF** $|S| > 1$
4. Find $m^o \in S$: $SF(m_f, m_k)$ is maximum
5. Return m^o
6. **ELSE**
7. Return m_a: $m_a \in S$

Figure 6.5: Merging Sensor Selection Algorithm for MTMS in Multi-hop Clusters

and minimize schedule lengths of a DVS-adjusted schedule. Similar to the DVS algorithm, the DVS^{-1} is composed of two procedures, the SHE^{-1} procedure and the SLE^{-1} procedure, which are the inverse procedures of SHE and SLE in Chapter 4.4, respectively. In our DVS^{-1} algorithm, we assume that the CPU speed after the SLE procedure is available as f_{cpu}^{SLE}. We further assume that in the original schedule, the CPU speed of executing a computation task v_i is known as $f_{cpu}(v_i)$.

Based on these assumptions, we first present the inverse procedure of the SHE procedure, SHE^{-1}, in Fig. 6.6. The SHE^{-1} procedure increases CPU speed to f_{cpu}^{SLE}, and adjusts task schedules. The SHE^{-1} procedure has the same structure as the SHE procedure, and the Steps 8-15 and 20-27 are the inverse procedure of the SpeedAdjust() algorithm in Fig. 4.8. The inverse procedure of the SLE procedure, SLE^{-1}, is presented in Fig. 6.7. The SLE^{-1} procedure increases the CPU speed to f_{cpu}^{max}, and adjust task schedule accordingly.

Input: original schedule H, CPU speed after SLE f_{cpu}^{SLE}, sensor set SS, application deadline DL
Output: Adjusted schedule H^s
SHE^{-1} **Algorithm**:

---

1. **FOR** sensor $m_k \in SS$
2. $ds \leftarrow 0$, $df \leftarrow \infty$ /*Initialization*/
3. Scan tasks $v_i \in T_{ds}^{\infty}(m_k)$ in increasing order of start time
4. **IF** $\exists$ a copy of $v_i \in R$: $v_i^c \in T(\mathcal{C})$ /*Transmitted communication task */
5. Find the computation task v_j following v_i
6. **IF** m_k is the sender of v_i^c
7. $df \leftarrow \min(s_{v_i^c,\mathcal{C}}, s_{v_j,m_k})$ /*Computation must finish before transmitting*/
8. **FOR** $v_j \in T_{ds}^{df}(m_k)$ /*Adjust schedule in $[ds, df] * /$
9. **IF** $v_j \in V$ /*Computation task*/
10. $t_{v_j,m_k} \leftarrow t_{v_j,m_k} \cdot \frac{f_{cpu}(v_i)}{f_{cpu}^{SLE}}$
11. $f_{v_j,m_k} \leftarrow s_{v_j,m_k} + t_{v_j,m_k}$
12. **ELSE** /*Communication task*/
13. $s_{v_j,m_k} \leftarrow f_{pred(v_j),m_k}$
14. $f_{v_j,m_k} \leftarrow s_{v_j,m_k}$
15. Update m_k's energy consumption
16. $ds \leftarrow df$
17. **ELSE** /*m_k is the receiver of v_i^c*/
18. $ds \leftarrow \max(ds, f_{v_i^c,m_k}, s_{v_j,m_k})$ /*Computation cannot start before reception*/
19. **ELSE IF** v_i is exit-task and $f_{v_i} < DL$ /*Adjustment bounded by deadline*/
20. **FOR** $v_j \in T_{ds}^{df}(m_k)$ /*Adjust schedule in $[ds, df]$
21. **IF** $v_j \in V$ /*Computation task*/
22. $t_{v_j,m_k} \leftarrow t_{v_j,m_k} \cdot \frac{f_{cpu}(v_i)}{f_{cpu}^{SLE}}$
23. $f_{v_j,m_k} \leftarrow s_{v_j,m_k} + t_{v_j,m_k}$
24. **ELSE** /*Communication task*/
25. $s_{v_j,m_k} \leftarrow f_{pred(v_j),m_k}$
26. $f_{v_j,m_k} \leftarrow s_{v_j,m_k}$
27. Update m_k's energy consumption

Figure 6.6: DVS^{-1}: The SHE^{-1} Algorithm

Input: original schedule H, CPU speed after SLE f_{cpu}^{SLE}, sensor set SS, application deadline DL
Output: Adjusted schedule H^s
SLE^{-1} **Algorithm**:

---

1. $\delta \leftarrow \frac{f_{cpu}^{SLE}}{f_{cpu}^{max}}$
1. **FOR** $m_k \in SS \cup \{\mathcal{C}\}$ /*Scan all sensors and $\mathcal{C}$*/
2. **FOR** $v_i \in T(m_k)$
3. **IF** $v_i \in V$ /*Computation task*/
4. $s_{v_i,m_k} \leftarrow \delta \cdot s_{v_i,m_k}$
5. $t_{v_i,m_k} \leftarrow \delta \cdot t_{v_i,m_k}$
6. $f_{v_i,m_k} \leftarrow \delta \cdot f_{v_i,m_k}$
7. **ELSE** /*Communication task*/
8. $f_{v_i,m_k} \leftarrow \delta \cdot f_{v_i,m_k}$
9. $s_{v_i,m_k} \leftarrow f_{v_i,m_k} - t_{v_i,m_k}$
10. Update m_k's energy consumption

Figure 6.7: DVS^{-1}: The SLE^{-1} Algorithm

Failing Sensor Task Reassignment

We first introduce the task reassigning algorithm when idle sensors exist among the failing sensor's one-hop neighbors. After the optimal alternative sensor m_a is chosen, the tasks originally assigned to m_f should be reassigned to m_a. Computation tasks and communication tasks that are not transmitted over the wireless channel can be directly re-assigned to m_a. However, communication tasks exchanged between m_f and its neighbors in original schedules should be carefully handled since they may cause path extension and interference as demonstrated in Fig. 6.3. The algorithm of the multi-hop failure handling algorithm with idle sensors for MTMS is presented in Fig. 6.8, where $SendPktHandler()$ and $RecvPktHandler()$ adjust each individual communication task sent and received by

Input: Failing sensor m_f, alternative sensor m_a, original sensor set SS, original schedule H^o
Output: Recovered schedule H^r
Multi-hop sensor failure handling algorithm with idle sensor replacement

---

1. $DVS^{-1}(H^o)$ /*first increase CPU speed to maximum*/
2. **FOR** task $v_i \in T(m_f)$
3. **IF** v_i is a computation task
4. Reassign v_i to m_a:
5. $s_{v_i,m_a} \leftarrow \max(f_{pred(v_i),m_a})$
6. $f_{v_i,m_a} \leftarrow s_{v_i,m_a} + t_{v_i,m_a}$
7. **ELSE** /*communication task*/
8. **IF** v_i is sent by m_f
9. $SendPktHandler(v_i, m_f, m_a, SS, H^o)$
10. **ELSE IF** v_i is received by m_f
11. $RecvPktHandler(v_i, m_f, m_a, SS, H^o)$
12. **ELSE IF** v_i is forwarded by m_f
13. $RecvPktHandler(v_i, m_f, m_a, SS, H^o)$
14. $SendPktHandler(v_i, m_f, m_a, SS, H^o)$
15. **ELSE** /*locally processed data*/
16. Reassign v_i to m_a:
17. $s_{v_i,m_a} \leftarrow \max(f_{pred(v_i),m_a})$
18. $f_{v_i,m_a} \leftarrow s_{v_i,m_a}$
19. $SS \leftarrow SS - \{m_f\}$ /*remove the failing sensor*/
20. Execute the DVS algorithm in Chapter 4.4

Figure 6.8: Multi-hop Sensor Failure Handling Algorithm with Idle Sensor Replacement

m_f, respectively. A forwarded packet in an original schedule is first received, then transmitted to the next-hop by m_f. Therefore, reassigning a forwarded packet is handled by consequent execution of $RecvPktHandler()$ and $SendPktHandler()$.

The $SendPktHandler()$ algorithm is presented in Fig. 6.9. When reassigning a communication task v_i originally sent by the failing sensor m_f, the receivers of v_i can either all be the one-hop neighbors of the alternative sensor m_a, or some of them be multiple hops away from m_a. For the former scenario, the reassigned communication is still a single-hop

communication event. However, the reassigned communication may interfere neighboring communications since the sender is changed from m_f to m_a: Within the transmission range of m_a, if there is a sensor previously scheduled to receive another packet v_j at the same time, the transmission of v_i collides with v_j. To avoid such interference, either v_i's or the neighboring communication task's schedule should be postponed. As the successive tasks of v_i on the receivers cannot start execution without receiving the data, the execution schedule of the successive tasks on the receiver must also be postponed accordingly. To keep relative execution order among tasks assigned to different sensors, we also shift tasks across all sensors and $\mathcal{C}$ that are originally scheduled after the inference occurrence time. Given the potential interference time interval $[st, ft]$, the task shifting algorithm is presented in Fig. 6.10. The task rescheduling procedure when all receivers of v_i are m_a's one-hop neighbors is shown in Steps 2-6 in Fig. 6.9. If some of the receivers of v_i are multiple hops away from m_a, multi-hop paths from m_a to the receivers are first obtained. v_i is then delivered hop-by-hop from m_a to the receivers. If a sensor along the path already has the data of v_i, no data deliver of v_i to this sensor is scheduled to avoid duplicated communication. Compared with the original single-hop communication, the available time of v_i on these receivers are delayed due to additional communication hops. To keep relative execution orders of tasks scheduled on all sensors, tasks originally scheduled after the transmission finish time of v_i are postponed by the prolonged communication time. If there are more than one path extensions, the task shifting offset is determined by the maximum of path extension length. The task adjustment procedure for path extension is shown in Steps 8-14 in Fig. 6.9. For each hop of communication scheduling, if interference may occur, it is handled in a similar way with the scenario where all receivers of v_i are m_a's one-hop neighbors.

Input: Communication task v_i, failing sensor m_f, alternative sensor m_a, original sensor set SS, original schedule H^o
Output: Partially adjusted schedule H with v_i reassigned
SendPktHandler()

---

1. Find the copy of v_i on $\mathcal{C}$: v_i^c is sent by m_f
2. **IF** $R(v_i^c) \subset N(m_a)$ /*No path extension is needed*/
3. **IF** interference exists in $[st, ft]$
4. $TaskShift(v_i^c, st, ft)$ /*shift tasks to avoid interference*/
5. Reassign v_i to m_a:
6. $s_{v_i,m_a} \leftarrow f_{v_i^c,\mathcal{C}}$, $f_{v_i,m_a} \leftarrow s_{v_i,m_a}$
7. **ELSE** /*path extension is needed*/
8. **FOR** $m_k \in R(v_i^c) - N(m_a)$ /*Calculate shifting offset*/
9. Find the path from m_a to m_k:
10. $path = (m_1, m_2, ..., m_n)$, $m_1 = m_a$, $m_n = m_k$
11. Find N: N is the maximum of all $path$'s length
12. **FOR** task $v_p \in T(m_l)$: $m_l \in SS \cup \{\mathcal{C}\}$, $s_{v_p,m_l} \geq f_{v_i^c,\mathcal{C}}$
13. $s_{v_p,m_l} \leftarrow s_{v_p,m_l} + (N-2)t_{v_i^c,\mathcal{C}}$
14. $f_{v_p,m_l} \leftarrow s_{v_p,m_l} + t_{v_i^c,m_l}$
15. **FOR** $m_k \in R(v_i^c) - N(m_a)$ /*reroute the packet delivery*/
16. Find the path from m_a to m_k:
17. $path = (m_1, m_2, ..., m_n)$, $m_1 = m_a$, $m_n = m_k$
18. **FOR** j := 1 to $n-1$
19. Schedule a copy of v_i^c from m_j to m_{j+1} if $v_i \notin T(m_{j+1})$
20. **IF** interference exists in $[st, ft]$
21. $TaskShift(v_i^c, st, ft)$ /*shift tasks to avoid interference*/

Figure 6.9: Adjust Packet Originally Sent by the Failing Sensor

Input: Communication task v_i on $\mathcal{C}$, interference time interval $[st, ft]$, original sensor set SS, original schedule H^o
Output: Adjusted schedule H after task shifting
$TaskShift(v_i, st, ft)$

---

1. **IF** $s_{v_i,\mathcal{C}} \leq st$ /*calculate shifting parameters*/
2. $ss \leftarrow st$
3. $\Delta \leftarrow f_{v_i,\mathcal{C}} - st$
4. **ELSE**
5. $ss \leftarrow s_{v_i,\mathcal{C}}$
6. $\Delta \leftarrow ft - s_{v_i,\mathcal{C}}$
7. **FOR** $m_k \in SS \cup \{\mathcal{C}\}$
8. **FOR** $v_p \in T(m_k)$: $s_{v_p,m_k} \geq ss$ /*tasks starting after ss*/
9. $s_{v_p,m_k} \leftarrow s_{v_p,m_k} + \Delta$ /* postpone tasks by Δ*/
10. $f_{v_p,m_k} \leftarrow s_{v_p,m_k} + t_{v_p,m_k}$

Figure 6.10: Task Shifting Algorithm to Avoid Interference

The $RecvPktHandler()$ algorithm is presented in Fig. 6.11. The $RecvPktHandler()$ algorithm is similar to the $SendPktHandler()$ algorithm except that there is only one receiver involved in the communication. If the alternative sensor m_a is a one-hop neighbor of the sender, the communication task v_i can be directly reassigned to m_a, as shown in Steps 2-6. Possible interference is handled in the same way as that in the $SendPktHandler()$ algorithm. If m_a is multiple hops away from the sender, an alternative path is found from the sender to m_a, and the prolonged communication time and potential interference are handled as shown in Steps 8-16.

When there are no idle sensors among the failing sensor's one-hop neighbors, the failing sensor's tasks must be merged to an alternative sensor selected with the algorithm presented in Fig. 6.5. The task merging algorithm has the similar structure with the algorithm presented in Fig. 6.1. When reassigning a communication task, path extension and

Input: Communication task v_i, failing sensor m_f, alternative sensor m_a, original sensor set SS, original schedule H^o
Output: Partially adjusted schedule H with v_i reassigned
RecvPktHandler()

---

1. Find the copy of v_i on $\mathcal{C}$: v_i^c is sent by m_f
2. **IF** $S(v_i^c) \subset N(m_a)$ /*No path extension is needed*/
3. **IF** interference exists in $[st, ft]$
4. $TaskShift(v_i^c, st, ft)$ /*shift tasks to avoid interference*/
5. Reassign v_i to m_a:
6. $s_{v_i,m_a} \leftarrow s_{v_i^c,\mathcal{C}}$, $f_{v_i,m_a} \leftarrow s_{v_i,m_a}$
7. **ELSE** /*path extension is needed*/
8. Find the path from $S(v_i^c)$ to m_a:
9. $path = (m_1, m_2, ..., m_n)$, $m_1 = S(v_i^c)$, $m_n = m_a$
10. **FOR** task $v_p \in T(m_l)$: $m_l \in SS \cup \{\mathcal{C}\}$, $s_{v_p,m_l} \geq f_{v_i^c,\mathcal{C}}$
11. $s_{v_p,m_l} \leftarrow s_{v_p,m_l} + (n-2)t_{v_i^c,\mathcal{C}}$
12. $f_{v_p,m_l} \leftarrow s_{v_p,m_l} + t_{v_i^c,m_l}$
13. **FOR** j := 1 to $n-1$
14. Schedule a copy of v_i^c from m_j to m_{j+1} if $v_i \notin T(m_{j+1})$
15. **IF** interference exists in $[st, ft]$
16. $TaskShift(v_i^c, st, ft)$ /*shift tasks to avoid interference*/

Figure 6.11: Adjust Packet Originally Received by the Failing Sensor

communication interference may occur. Therefore, the communication task adjustment algorithms presented in Fig. 6.9 6.11 are also implemented to handle path extension and avoid communication interference.

6.2.2 Simulation Result

The Sensor Failing Handling Algorithm for MTMS in multi-hop environments is evaluated in this section. The simulation parameter is in coherence with those in Chapter 5.6.1. We investigate the performance of sensor failure handling 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. The simulated scenarios are generated by randomly selecting one active sensor as a failing sensor, and reassigning its computation tasks and communication events using the algorithms presented in Chapter 6.2.1. In these simulations, schedule recovery for both MMM and DCTMS algorithms is evaluated.

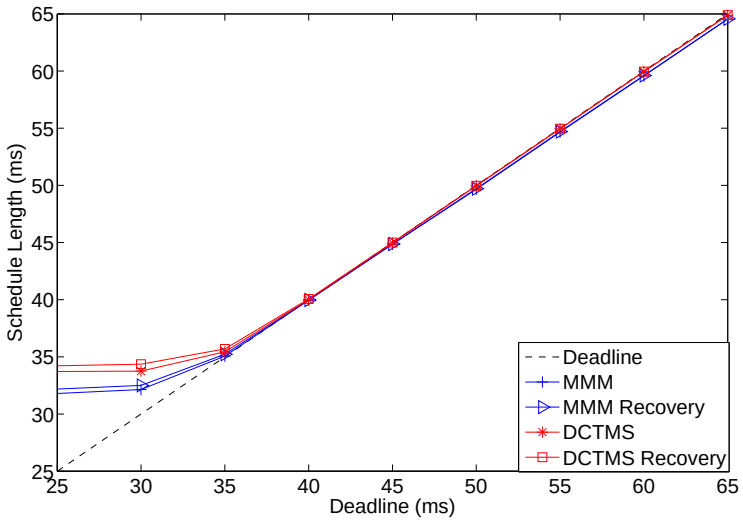
As shown in Fig. 6.13, the sensor failure handling algorithm delivers almost the same performance when the deadline is sufficiently large regarding schedule lengths and deadline missing ratio (DMR). With large deadlines, slack time exists after increasing CPU speed by executing the DVS^{-1} algorithm. Thus, the prolonged schedule length caused by reassigning tasks to the alternative sensor is compensated by the CPU speedup, which leads to the same level of DMR. The recovered schedule is then further adjusted by the DVS algorithm to conserve energy consumption, which produces the same level of performance in schedule lengths. When the deadline decreases, the CPU speed of the original schedules increases to meet deadline constraints. Therefore, the DVS^{-1} algorithm brings no much benefit in compensating the schedule length increase, which subsequently leads to larger

Input: Failing sensor m_f, merging sensor m_a, original sensor set SS, original schedule H^o
Output: Recovered schedule H^s
Multi-hop sensor failure handling algorithm with task merging

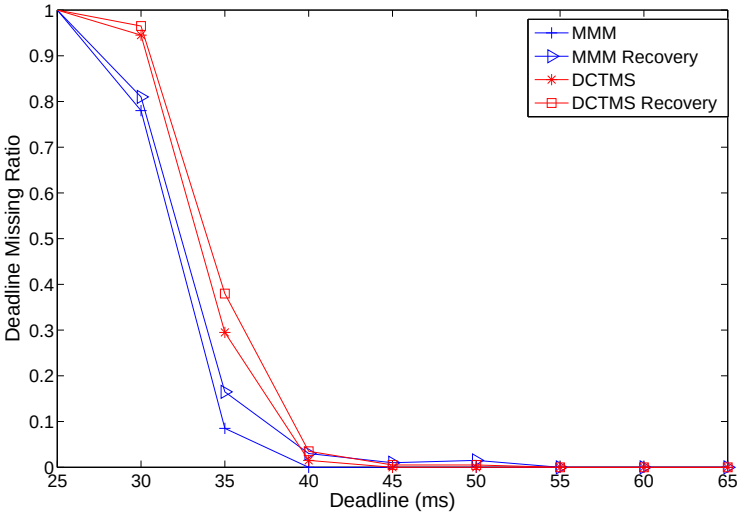
---

1. **FOR** unadjusted task $v_i \in S = T(m_f) \cup T(m_a) : s_{v_i,S}$ is minimum
2. **IF** $v_i \in V$ /*computation task*/
3. Schedule v_i onto m_a
4. $s_{v_i,m_a} \leftarrow \max(f_{pred(v_i),m_a})$
5. $f_{v_i,m_a} \leftarrow s_{v_i,m_a}$
6. **ELSE** /*communication task*/
7. **IF** there is a duplicated copy of v_i in S
8. Remove the duplicated copy
9. Find the copy of v_i on $\mathcal{C}$: v_i^c
10. **IF** m_f/m_a are the only sender/receiver of v_i^c
11. Remove v_i^c from $\mathcal{C}$
11. **ELSE IF** $pred(v_i) \in S$ /*send result to other tasks*/
12. $SendPktHandler(v_i, m_f, m_a, SS, H^o)$
13. **ELSE IF** v_i is received by m_f
14. $RecvPktHandler(v_i, m_f, m_a, SS, H^o)$
15. **ELSE IF** v_i is forwarded by m_f
16. $RecvPktHandler(v_i, m_f, m_a, SS, H^o)$
17. $SendPktHandler(v_i, m_f, m_a, SS, H^o)$
18. **ELSE** /*locally processed data*/
19. Reassign v_i to m_a:
20. $s_{v_i,m_a} \leftarrow \max(f_{pred(v_i),m_a})$
21. $f_{v_i,m_a} \leftarrow s_{v_i,m_a}$
22. $SS \leftarrow SS - \{m_f\}$ /*remove the failing sensor*/
23. Execute the DVS algorithm in Chapter 4.4

Figure 6.12: Multi-hop Sensor Failure Handling Algorithm with Task Merging



(a) Schedule Length

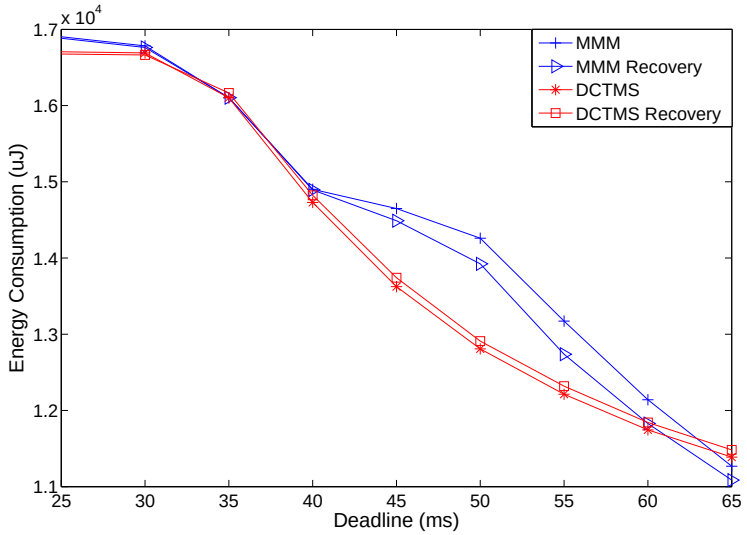


(b) Deadline Missing Ratio

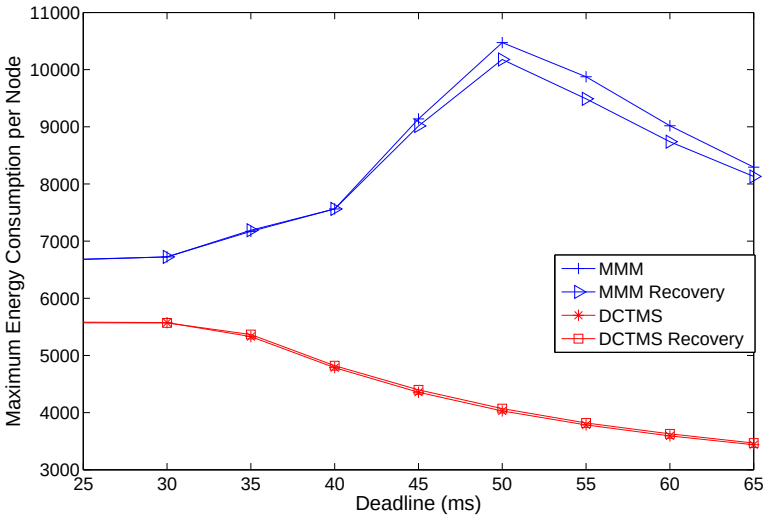
Figure 6.13: Performance of Sensor Failure Handling for MTMS in the Time Domain

schedule lengths and DMR of the recovered schedules. However, by optimizing alternative sensor selections, the performance only slightly degrades as shown in Fig. 6.13.

Regarding energy consumption, when deadlines are small, the recovered schedules have almost the same application energy consumption and MECpN, as shown in Fig. 6.14. This behavior stems from the factor that by optimizing alternative sensor selection, path extensions are minimized. Consequently, additional energy expenditure in communication is also minimized. When deadlines increase, application energy consumption is reduced by exploiting CPU idle time with the DVS algorithm. In recovered schedules, schedule lengths are increased due to the inference avoidance and path extension, which leads to smaller CPU slack time for the DVS implementation. Such CPU slack time decrease results in the slightly larger application energy consumption and MECpN of DCTMS. For the original schedules generated by the MMM algorithm, computation load converges to fewer number of sensors when deadline increases. Such unbalanced schedules lead to less number of schedule holes, which constrains the energy conservation of the DVS algorithm in the original schedules. However, the recovered schedules have more and larger schedule holes caused by task schedule shifting for interference avoidance and path extension. Therefore, the recovered schedules of MMM have smaller application energy consumption and MECpN when the deadlines increase, compared with their original schedules.



(a) Application Energy Consumption



(b) Max Energy Consumption per Node (MECpN)

Figure 6.14: Performance of Sensor Failure Handling for MTMS in the Energy Domain

CHAPTER 7

CONCLUSION AND FUTURE WORK

In this dissertation, we address the task mapping and scheduling problem to enable collaborative in-network processing in large-scale WSNs. We consider WSNs composed of homogeneous wireless sensors grouped into clusters, within which applications are iteratively executed. Since energy consumption efficiency is one of the most critical consideration for any WSN solution, our proposed solutions aim to achieve energy-efficiency from different aspects. To enhance information processing capacity in WSNs, schedule length optimization is also part of our design objectives. The contribution of this research can be summarized as follows.

First, our solutions are application-independent that simultaneously address computation and communication scheduling. Different from traditional task mapping and scheduling solutions for wired networks, we presented a Hyper-DAG application model that explicitly represents communication events between computation tasks. To reflect the wireless communication features, we first model a single-hop wireless channel as a virtual node that can only execute communication tasks. Based on this Hyper-DAG model and single-hop channel model, we propose the EcoMapS solution for single-hop clustered homogeneous WSN clusters in Chapter 3. EcoMapS aims to minimize schedule lengths of applications under energy consumption constraints. Two variations of EcoMapS, the E-MinMin based

EcoMapS and the E-CNPT based EcoMapS, are presented. Simulation results show that both EcoMapS algorithms deliver superior performance than existing work. Regarding comparison of the EcoMapS algorithms themselves, E-MinMin outperforms E-CNPT with respect to schedule lengths and energy consumption balance but has a larger computation overhead. Thus, the E-MinMin based EcoMapS algorithm is suitable for WSN applications that do not change frequently, while the E-CNPT based EcoMapS is preferable where application updates occur more frequently.

We then further investigate energy consumption optimization in WSNs with the promising DVS technology. In Chapter 4, the schedule length and energy consumption optimization problem is tackled with the objective of minimizing balanced energy consumption subject to deadline constraints. To minimize energy consumption, a novel DVS algorithm is developed in Chapter 4 that exploits CPU slack time by reducing CPU speed without violating deadline constraints. Different from existing DVS algorithms, wireless communication between sensors is considered in our proposed DVS algorithm. Based on the Hyper-DAG model and the single-hop channel model, the RT-MapS solution achieves its design objectives. As demonstrated with the simulation results, RT-MapS outperforms existing solutions with respect of minimizing energy consumption subject to deadline constraints. Among the RT-MapS algorithms, H-CNPT has a higher capacity to meet deadline constraints, and thus is suitable for applications with strict real-time requirements. H-MinMin on the other hand provides better energy-balanced schedules, and is more plausible for more energy-constrained applications with relatively lower real-time requirements.

Both Chapter 3 and 4 consider single-hop environments, which forms a basis for more general solutions in multi-hop clustered WSNs. In Chapter 5, we propose a task mapping and scheduling solution for multi-hop WSNs, MTMS. 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 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, the task scheduling algorithm schedules tasks with minimum energy consumption subject to deadline constraints. Two task mapping and scheduling algorithms, MMM and DCTMS, are developed as part of the MTMS solutions. Simulation results show significant performance improvements of MTMS compared with existing solutions in terms of minimizing energy consumption subject to delay constrains. Further analysis and simulation indicates the MMM based MTMS algorithm is suitable for WSN applications that have strict real-time requirements, while the DCTMS based MTMS is preferable where application updates occur frequently with relatively relaxed deadline constraints.

The final contribution of this dissertation relates to the sensor failure handling. In WSNs, sensors are prone to failures. In case of sensor failures, WSN functionality needs to be recovered as soon as possible since the previously calculated schedules may no longer be feasible solutions. Sensor failure handling is critical for reliable solutions in WSNs. In Chapter 6, two low-complexity sensor failure handling algorithms are proposed for single-hop and multi-hop clusters. We first present the sensor failure recovery algorithm for EcoMapS in single-hop clustered WSNs. Simulation results show that the recovered schedules provide the same level of energy consumption constraint guarantee as the original schedules, though the schedule lengths are slightly larger. We then develop the sensor failure handling algorithm for MTMS in multi-hop environments. By reassigning tasks of failing sensors to optimally selected alternative sensors, the recovered schedules deliver

slightly degraded but still satisfying performance as demonstrated with the simulation results.

In our future work, communication failure handling in WSNs should be further investigated. In Chapter 6, the sensor failure handling problem is solved, which enhance the robustness of our solutions for WSN applications. On the other hand, communication failure may also disturb application executions. Packet losses caused by interference are well handled with the exclusive channel access approach in Chapter 3 and 4, as well as the penalty function presented in Chapter 5. However, packet losses may also occur because of channel conditions. Such packet losses can be handled by retransmitting the erroneous packets. As packet retransmission may delay application finish time, sensors should compensate the retransmission time by speeding up the subsequent task executions.

In the present definition of the penalty function in Chapter 5, only interference avoidance is considered. However, the penalty function can be further extended with factors such as link quality and traffic load. With this approach, the scheduled communication will be able to adaptively choose reliable links, and balance communication load among cluster nodes, which will increase the communication reliability and the network lifetime.

Integrating the joint effort of sensor failure handling, communication failure handling, and adaptive communication scheduling with extended penalty functions, we expect to form a basis of developing a reliable adaptive task mapping and scheduling solution for collaborative in-network processing in WSNs. This adaptive solution should be able to deliver an optimal schedule in normal operation conditions. Upon network condition changes such as network topology changes, sensor failures, channel condition degrade, and new application arrivals, the system can dynamically adapt itself to achieve network functionality with satisfying performance.

WSNs in the future are envisioned to observe different types of events, which leads to different information processing simultaneously executed in WSNs. Therefore, a general solution to schedule multiple applications should be part of the future work. To map tasks of multiple applications and allocate network resources, a viable solution can be developed in an incremental method: We can first independently map and schedule each application using the proposed solutions in the dissertation, then merge these schedules with optimal network resource utilization.

Intra-sensor scheduling may also be a promising and challenging problem to investigate in the future. In this dissertation, we only consider computation and communication scheduling between sensors in WSNs. However, a wireless sensor itself has various resources, such as CPU time, memory space, wireless bandwidth, and battery lifetime, which need to be carefully managed. All scheduled computation and communication activities must be executed subject to these resource constraints, which is not explicitly considered in our present solutions. Furthermore, wireless sensor nodes may be equipped with multiple sensors detecting different events. Depending on applications, the detected events may occur in an aperiodic pattern. Therefore, a dynamic intra-sensor scheduling algorithm should be proposed to handle these events and efficiently allocate sensor resources.

BIBLIOGRAPHY

[1] *CC2420 Data Sheet*.

[2] *MICA2DOT Data Sheet*.

[3] I. F. Akyidiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks (Elsevier) Journal*, 38(4):393–422, March 2002.

[4] Alan D. Amis, Ravi Prakash, Dung Huynh, and Thai Vuong. Max-Min D-Cluster formation in wireless ad hoc networks. In *Proc. of the 19nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'00)*, pages 32–41, March 2000.

[5] Hakan Aydin, Rami Melhem, Daniel Mossé, and Pedro Mejia-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Transactions on Computers*, 53(5):584–600, May 2004.

[6] Cevdet Aykanat, Füsun Özgüner, Fikret Ercal, and Ponnuswamy Sadayappan. Iterative algorithms for solution of large sparse systems of linear equations on hypercubes. *IEEE Transactions on Computers*, 37(12):1554–1568, December 1988.

[7] Seung Jun Baek, Gustavo de Veciana, and Xun Su. Minimizing energy consumption in large-scale sensor networks through distributed data compression and hierarchical aggregation. *IEEE Journal on Selected Areas in Communications*, 22(6):1130–1140, August 2004.

[8] Paramvir Bahl, Ranveer Chandra, and John Dunagan. SSCH: Slotted seeded channel hopping for capacity improvement in IEEE 802.11 ad-hoc wireless networks. In *Proc. of ACM MobiCom'04*, pages 216–230, September/October 2004.

[9] Seema Bandyopadhyay and Edward J. Coyle. An energy efficient hierarchical clustering algorithm for wireless sensor networks. In *Proc. of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'03)*, volume 3, pages 1713–1723, March/April 2003.

[10] Doruk Bozdag, Füsun Özgüner, Eylem Ekici, and Umit Catalyurek. A task duplication based scheduling algorithm using partial schedules. In *Proc. of 2005 International Conference on parallel Processing (ICPP'05)*, pages 630–637, June 2005.

[11] Tracy D. Braun, Howard Jay Siegel, Noah Beck, Ladislau L. Boloni, Muthucumaru Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, Bin Yao, Debra Hensgen, and Richard F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, June 2001.

[12] Thomas D. Burd and Robert W. Brodersen. Design issues for dynamic voltage scaling. In *Proc. of the 2000 International Symposium on Low Power Electronics and Design (ISLPED'00)*, pages 9–14, July 2000.

[13] Yuanzhu Peter Chen, Arthur L. Liestman, and Jiangchuan Liu. A hierarchical energy-efficient framework for data aggregation in wireless sensor networks. *IEEE Transactions on Vehicular Technology*, 55(3):789–796, May 2006.

[14] Krishna Kant Chintalapudi, Amit Dhariwal, Ramesh Govindan, and Gaurav Sukhatme. Ad-hoc localization using ranging and sectoring. In *Proc. of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'04)*, volume 4, pages 2662–2672, March 2004.

[15] Jim Chou, Dragan Petrovic, and Kannan Ramachandran. A distributed and adaptive signal processing approach to reducing energy consumption in sensor networks. In *Proc. of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'03)*, volume 2, pages 1054–1062, April 2003.

[16] Princey Chowdhury and Chaitali Chakrabarti. Static task-scheduling algorithms for battery-powered DVS systems. *IEEE Transaction on Very Large Scale Integration Systems*, 13(2):226–237, February 2005.

[17] Robert T. Collins, Alan J. Lipton, Hironobu Fujivoshi, and Takeo Kanade. Algorithms for cooperative multisensor surveillance. *IEEE Proceedings*, 89(10):1456–1477, October 2001.

[18] Luis D. de Cerio, Miguel Valero-Garcia, and Antonio Gonzalez. Hypercube algorithms on mesh connected multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, 13(12):1247–1260, December 2002.

[19] Atakan Dogan. *Matching and Scheduling of Applications in Heterogeneous Computing Systems with Emphasis on High-Performance, Reliability, and DoS*. PhD thesis, The Ohio State University, 2001.

[20] Atakan Dogan and Füsun Özgüner. Optimal and suboptimal reliable scheduling of precedence-constrained tasks in heterogeneous distributed computing. In *Proc. of International Conference on Parallel Processing, Workshop on Network-Based Computing*, page 429, August 2000.

[21] Atakan Dogan and Füsun Özgüner. Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogenous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):308–323, March 2002.

[22] Atakan Dogan and Füsun Özgüner. Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems. *The Computer Journal*, 48(3):300–314, 2005.

[23] Emad Felemban, Chang-Gun Lee, Eylem Ekici, Ryan Boder, and Serdar Vural. Probabilistic QoS guarantee in reliability and timeliness domains in wireless sensor networks. In *Proc. of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'05)*, March 2005.

[24] Wu-Chi Feng, Ed Kaiser, Wu Chang Feng, and Mikael Le Baillif. Panoptes: Scalable low-power video sensor networking technologies. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 1(2):151–167, May 2005.

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

[26] Simon Giannecchini, Marco Caccamo, and Chi-Sheng Shih. Collaborative resource allocation in wireless sensor networks. In *Proc. of Euromicro Conference on Real-Time Systems (ECRTS'04)*, pages 35–44, June/July 2004.

[27] Kavitha Golconda, Füsun Özgüner, and Atakan Dogan. A comparison of static qos-based scheduling heuristics for a meta-task with multiple qos dimensions in heterogeneous computing. In *Proc. of the 13th Heterogeneous Computing Workshop, in conjunction with IPDPS'05*, April 2004.

[28] Tarek Hagras and Jan Janecek. A high performance, low complexity algorithm for compile-time job scheduling in homogeneous computing environments. In *Proc. of International Conference on Parallel Processing Workshops (ICPPW'03)*, pages 149–155, October 2003.

[29] Tian He, John A Stankovic, Chenyang Lu, and Tarek Abdelzaher. SPEED: A stateless protocol for real-time communication in sensor networks. In *Proc. of the International Conference on Distributed Computing Systems (ICDCS'03)*, pages 46–55, May 2003.

[30] Wendi B. Heinzelman, Anantha P. Chandrakasan, and Hari Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, 1(4):660–670, October 2002.

[31] Ting-Chao Hou, Ling-Fan Tsao, and Hsin-Chiao Liu. Analyzing the throughput of IEEE 802.11 DCF scheme with hidden nodes. In *Proc. of IEEE 58th Vehicular Technology Conference*, volume 5, pages 2870–2874, October 2003.

[32] An-Swol Hu and Sergio D. Servetto. On the scalability of cooperative time synchronization in pulse-connected networks. *IEEE Transactions on Information Theory*, 52(6):2725–2748, June 2006.

[33] Daniel P. Huttenlocher, Gregory A. Klanderman, and William J. Rucklidge. Comparing images using the hausdorff distance. 15:850–863, September 1993.

[34] Gwo-Jen Hwang and Shian-Shyong Tseng. A heuristic task assignment algorithm to maximize reliability of a distributed system. *IEEE Transactions on Reliability*, 42(3):408–415, September 1993.

[35] Baback Izadi and Füsun Özgüner. Reconfigurable k-ary tree multiprocessors. *International Journal of Parallel and Distributed Systems and Networks*, 3(4):227–234, 2000.

[36] Rajgopal Kannan and S. Sitharama Iyengar. Game-theoretic models for reliable path-length and energy-constrained routing with data aggregation in wireless sensor networks. *IEEE Journal on Selected Areas in Communications*, 22(6):1141–1150, August 2004.

[37] Brad Karp and H. T. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proc. of ACM MobiCom'00*, pages 243–254, August 2000.

[38] Purushottam Kulkarni, Deepak Ganesan, and Prashant Shenoy. Multimedia sensing: The case for multi-tier camera sensor networks. In *Proc. of the 15th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'05)*, pages 141–146, June 2005.

[39] Rajnish Kumar, Matthew Wolenetz, Bikash Agarwalla, JunSuk Shin, Phillip Hutto, Arnab Paul, and Umakishore Ramachandran. DFuse: A framework for distributed data fusion. In *Proc. of The ACM Conference on Embedded Networked Sensor Systems (SenSys'03)*, pages 114–125, November 2003.

[40] Ram Kumar, Vlasios Tsiatsis, and Mani B. Srivastava. Computation hierarchy for in-network processing. In *Proc. of the 2nd ACM international conference on Wireless Sensor Networks and Applications (WSNA'03)*, pages 68–77, September 2003.

[41] Seungjoon Lee, Bobby Bhattacharjee, and Suman Banerjee. Efficient geographic routing in mulitihop wireless networks. In *Proc. of ACM MobiHoc'05*, pages 230–241, May 2005.

[42] Juan Liu, James Reich, and Feng Zhao. Collaborative in-network processing for target tracking. *EURASIP Journal on Applied Signal Processing*, (4):378–391, March 2003.

[43] Chenyang Lu, Brian M. Blum, Tarek F. Abdelzaher, John A. Stankovic, and Tian He. RAP: A real-time communication architecture for large-scale wireless sensor networks. In *Proc. of IEEE Real-Time and Embedded Technology and Application Symposium (RTAS'02)*, pages 55–66, September 2002.

[44] Ramesh Mishra, Namrata Rastogi, Dakai Zhu, Daniel Mossé, and Rami Melhem. Energy aware scheduling for distributed real-time systems. In *Proc. of Parallel and Distributed Processing Symposium*, April 2003.

[45] Trevor Pering, Tom Burd, and Robert Brodersen. Dynamic voltage scaling and the design of a low-power microprocessor system. In *Proc. of Driven Microarchitecture Workshop*, pages 107–112, 1998.

[46] Paul Pop, Petru Eles, Zebo Peng, and Traian Pop. Scheduling and mapping in an incremental design methodology for distributed real-time embedded systems. *IEEE Transactions on Very Large Scale Integration*, 12(8):793–811, August 2004.

[47] Johan Pouwelse, Koen Langendoen, and Henk Sips. Dynamic voltage scaling on a low-power microprocessor. In *Proc. of the 7th ACM International Conference on Mobile Computing and Networking (Mobicom'01)*, pages 251–259, July 2001.

[48] Suresh Rai, Jerry L. Trahan, and Thomas Smailus. Processor allocation in hypercube multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 6(6):606–616, June 1995.

[49] R.Jain, D-M. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared conputer systems. Technical Report TR-301, DEC Research Report, September 1984.

[50] Danbing Seto, John P. Lehoczky, Lui Sha, and Kang G. Shin. On task schedulability in real-time control systems. In *Proc. of 17th IEEE Real-Time Systems Symposium (RTSS'96)*, page 13, December 1996.

[51] Yi Shang, Wheeler Ruml, Ying Zhang, and Markus Fromherz. Localization from connectivity in sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 15(11):961–974, November 2004.

[52] Eugene Shih, SeongHwan Cho, Nathan Ickes, Rex Min, Amit Sinha, Alice Wang, and Anantha Chandrakasan. Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks. In *Proc. of ACM MobiCom'01*, pages 272–286, July 2001.

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

[54] Mitali Singh and Viktor K. Prasanna. A hierarchical model for distributed collaborative computation in wireless sensor networks. In *Proc. of Parallel and Distributed Processing Symposium (IPDPS'03)*, April 2003.

[55] Fikret Sivrikaya and Bulent Yener. Time synchronization in sensor networks: A survey. *IEEE Network*, 18(4):45–50, July/August 2004.

[56] Jungmin So and Nitin H. Vaidya. Multi-channel MAC for ad hoc networks: Handling multi-channel hidden terminals using a single transceiver. In *Proc. of ACM MobiHoc'04*, pages 222–233, May 2004.

[57] Ivan Stojmenovic and Xu Lin. Power-aware localized routing in wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1122–1133, October 2001.

[58] Weilian Su and Ian F. Akyildiz. Time-diffusion synchronization protocol for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 13(2):384–397, April 2005.

[59] Yoshida Tatsuya, Kagesawa Masataka, and Ikeuchi Katsushi. Local-feature based vehicle recognition system using parallel vision board. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1828–1833, 1999.

[60] Yuan Tian, Eylem Ekici, and Füsun Özgüner. Energy-constrained task mapping and scheduling in wireless sensor networks. In *Proc. of the IEEE International Workshop on Resource Provisioning and Management in Sensor Networks (RPMSN'05), in conjunction with MASS'05*, pages 211–218, November 2005.

[61] Yih-Jia Tsai and Philip K. McKinley. An extended dominating node approach to collective communication in all-port wormhole-routed 2d meshes. In *Proc. of the Scalable High-Performance Computing Conference*, pages 199–206, May 1994.

[62] M. Valera and S. A. Velastin. Intelligent distributed surveillance systems: A review. *IEEE Proceedings on Vision, Image and Signal Processing*, 152(2):192–204, April 2005.

[63] Tom Vercauteren, Dong Guo, and Xiaodong Wang. Joint multiple target tracking and classification in collaborative sensor networks. *IEEE Journal on Selected Areas in Communication*, 23(4):714–723, April 2005.

[64] Alice Wang and Anantha Chandrakasan. Energy-efficient DSPs for wireless sensor networks. *IEEE Signal Processing Magazine*, pages 68–78, July 2002.

[65] J. You, P. Bhattaharya, and S. Hungenahally. Real-time object recognition: Hierarchical image matching in a parallel virtual machine environment. In *Proc. of 14th International Conference on Pattern Recognition*, volume 1, pages 275–277, 1998.

[66] Mohamed Younis, Kemal Akkaya, and Anugeetha Kunjithapatham. Optimization of task allocation in a cluster-based sensor network. In *Proc. of IEEE International Symposium on Computers and Communications (ISCC'03)*, pages 329–334, June/July 2003.

[67] Ossama Younis and Sonia Fahmy. HEED: A hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks. *IEEE Transactions on Mobile Computing*, 3(4):366–379, October-December 2004.

[68] Ossama Younis, Marwan Krunz, and Srinivasan Ramasubramanian. Node clustering in wireless sensor networks: Recent developments and deployment challenges. *IEEE Network*, 20(3):20–25, May/June 2006.

[69] Yang Yu and Viktor. K. Prasanna. Energy-balanced task allocation for collaborative processing in wireless sensor networks. *ACM/Kluwer Journal of Mobile Networks and Applications*, 10(1-2):115–131, February 2005.

[70] Yihong Zhou and Scott M. Nettles. Balancing the hidden and exposed node problems with power control in CSMA/CA-based wireless networks. In *Proc. of IEEE Wireless Communications and Networking Conference*, volume 2, pages 683–688, March 2005.

[71] Yahui Zhu and Mohan Ahuja. On job scheduling on a hypercube. *IEEE Transactions on Parallel and Distributed Systems*, 4(1):62–69, January 1993.

[72] Barbara Zitova and Jan Flusser. Image registration methods: A survey. *Elsevier Image and Vision Computing*, 21(11):977–1000, October 2003.