2009

System level energy management in networked real-time embedded systems

Sudha Anil Kumar Gathala
Iowa State University

System level energy management in networked real-time embedded systems

by

Gathala Sudha Anil Kumar

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Iowa State University

Ames, Iowa

2009

DEDICATION

To my beloved mom, Mano Rama Devi, and my dear dad, Jayaprakash Gathala.

TABLE OF CONTENTS

LIST OF TABLES vii

LIST OF FIGURES viii

ACKNOWLEDGEMENTS xi

ABSTRACT xiii

CHAPTER 1. Introduction 1

1.1 Networked Embedded System and Design Requirements 2

1.2 The System-level Energy-aware Resource Management Framework 5

1.3 Thesis Statement 7

1.4 Organization 8

CHAPTER 2. Energy management in computing system through cross-layer algorithms 10

2.1 Problem Statement 11

2.2 Related work and motivation 12

2.3 Exploiting intra-task structure through early execution 16

2.3.1 Illustrative example 17

2.3.2 Overheads 20

2.3.3 The Early execution criterion 23

2.4 The generic early execution algorithm 25

2.4.1 Schedulability Guarantees 27

2.5 The conservative early execution (CEE) algorithm 28

2.6 Simulation Studies 31

2.6.1 Sensor Application 32
2.6.2 Video Phone Application 34
2.6.3 Random Task Sets 37
2.7 Discussion 40
CHAPTER 3. Energy management in communication system using dynamic modulation scaling 41
3.1 Related work and motivation 42
3.2 System Model 44
3.3 Problem Statement and ILP Formulation 46
3.3.1 ILP Formulation 46
3.3.2 Problem complexity 48
3.4 Continuous Problem Formulation 50
3.5 Energy-Aware Heuristic Scheduling Algorithm 53
3.5.1 Movement Algorithm 55
3.6 Performance Evaluation 58
3.6.1 Results for uniform node distribution 59
3.6.2 Results for clustered node distribution 60
3.7 Discussion 62
CHAPTER 4. Energy management in communication system using power adaptation 64
4.1 Related work and motivation 64
4.2 Problem Statement 66
4.3 System Model and Problem Formulation 67
4.3.1 System Model 67
4.3.2 Problem Formulation 69
4.3.3 Problem complexity 71
4.4 ILP Formulation 72
4.5 Energy-aware Scheduling algorithms 73

4.5.1 Motivational example 74

4.5.2 Greedy Scheduling Algorithm 75

4.5.3 Movement Algorithm 78

4.6 Simulations studies 83

4.6.1 Results for uniform node distribution 84

4.6.2 Results for clustered node distribution 86

4.7 Discussion 88

CHAPTER 5. System level energy management in single hop networks . . 90

5.1 Related work and motivation 90

5.2 System Model and Problem Statement 92

5.3 Energy-aware scheduling Algorithms 96

5.3.1 System level energy-time tradeoffs 96

5.3.2 Gain based Static Scheduling (GSS) 98

5.3.3 Distributed Slack Propagation Algorithm (DSP)-Dynamic Slack 100

5.4 Performance evaluation 107

5.5 Discussion 110

CHAPTER 6. System level energy management in multi-hop networks . . 111

6.1 Related Work 112

6.2 System Model 115

6.3 Problem Formulation 119

6.3.1 Problem formulation: Ideal model 120

6.3.2 Problem Formulation: Practical Model 122

6.4 Energy-Aware Scheduling Algorithms 125

6.4.1 System Level Energy-Time Tradeoffs 126

6.4.2 Illustrative Example 129

6.4.3 Gain based Scheduling Algorithm (GSA) 130

6.4.4 Extended Gain based Scheduling Algorithm (EGSA) 132

6.5 Performance Evaluation 136

6.5.1 Results for low bandwidth conditions 137

6.5.2 Results for high bandwidth conditions 138

6.6 Discussion 140

CHAPTER 7. Conclusions and Future Work 141

BIBLIOGRAPHY 146

LIST OF TABLES

Table 2.1 Power specifications[82, 58] 16
Table 2.2 Overhead measurements (us: microseconds, ms: milliseconds) 20
Table 2.3 Estimated execution times @ 400 MHz 36

Table 4.1 Motivational example: normalized inter-node distances 75
Table 4.2 Parameter values [65] 75

Table 5.1 Tuple values after each DSP iteration 105

Table 6.1 Power specifications (from [82]) 117

LIST OF FIGURES

Figure 1.1 Architecture of the networked system level energy management 5

Figure 2.1 Computation energy management - related work summary 13
Figure 2.2 Illustrative example: inter-task schedule 18
Figure 2.3 Illustrative example: early execution schedule 19
Figure 2.4 Early execution - schedulability tests 24
Figure 2.5 Early execution - effective CFG modifications 28
Figure 2.6 Illustrative example: different schedules 29
Figure 2.7 Simulation setup 32
Figure 2.8 (a) Effect of utilization at $P_b = 0.90$ (b) Effect of P_b at $U = 0.80$ . . . 34
Figure 2.9 Video Phone, DVS settings: (a) Intel PXA255 (b) AMD Opteron . . . 36
Figure 2.10 Effect of utilization 37
Figure 2.11 Effect of ACET/WCET 38
Figure 2.12 Effect of the best path probability 39

Figure 3.1 Transmission energy: f_i, Total energy: $f_i + E_{tc} + E_{rc}$ 51
Figure 3.2 Illustrative example: schedules obtained by different algorithms 53
Figure 3.3 Energy gain function, $Gain_{ij}$ versus b_j 55
Figure 3.4 Working of the Movement Algorithm 56
Figure 3.5 Effect of utilization ($N = 10$) 60
Figure 3.6 Effect of number of modulation levels($U = 0.8$) 61
Figure 3.7 Effect of number of clusters ($N = 10, U = 0.8, r_c = 0.10$) 62
Figure 3.8 Effect of cluster radius ($N = 10, U = 0.8, v_c = 4$) 63

Figure 4.1 Illustrative example: direct hop strategy 74

Figure 4.2 Illustrative example: greedy strategy 76

Figure 4.3 Energy Gain function 80

Figure 4.4 Illustrative example - Movement Table 81

Figure 4.5 Illustrative example - Movement algorithm 82

Figure 4.6 Effect of utilization ($v = 100, \beta = 7$) 85

Figure 4.7 Effect of number of nodes ($U = 0.5, \beta = 7$) 86

Figure 4.8 Effect of the required reliability ($v = 100$,$U = 0.5$) 87

Figure 4.9 Effect of number of clusters ($v = 50, \beta = 7, U = 0.5, r_c = 0.10$) 87

Figure 4.10 Effect of cluster radius ($v = 100, \beta = 7, U = 0.5, n_c = 10$) 88

Figure 5.1 Task and CPU model 93

Figure 5.2 System level energy-time tradeoffs 97

Figure 5.3 Illustrative example: input schedule 99

Figure 5.4 Working of the GSS algorithm 99

Figure 5.5 Illustrative example: online schedule 103

Figure 5.6 Slack Propagation Tree (SPT) 103

Figure 5.7 Schedule after one traversal of SPT 105

Figure 5.8 Effect of channel bandwidth 108

Figure 5.9 Effect of $\frac{BCET}{WCET}$ at $W = 1KHz$ 108

Figure 5.10 Effect of $\frac{BCET}{WCET}$ at $W = 1000KHz$ 109

Figure 5.11 Effect of slack factor 110

Figure 6.1 Data Aggregation Tree Model 116

Figure 6.2 System level energy-time tradeoffs 127

Figure 6.3 Illustrative example: Data aggregation tree 129

Figure 6.4 Illustrative example: Input schedule (Total energy consumption = $18.99mJ$)130

Figure 6.5 Illustrative example: Greedy slack allocation schedule (Total energy consumption = $13.25mJ$) 131

Figure 6.6 Illustrative example: GSA schedule (Total energy consumption = $9.38mJ$) 133

Figure 6.7 EGSA: Motivational example 134

Figure 6.8 Effect of radius factor (R_f) 138

Figure 6.9 Effect of computational workload (C) 139

Figure 6.10 Effect of radius factor (R_f) 139

Figure 6.11 Effect of slack factor (S_f) 140

ACKNOWLEDGEMENTS

I submit all glory and honor to my god, Lord Jesus Christ, for he is the one who watched over me and presented me with this day. God blessed me with the company of an amazing set of people - my teachers, my friends, and my family. In the following, I would like to acknowledge some of these people who helped me to complete my graduate education successfully.

First and foremost, I would like to thank my advisor, Prof. Manimaran Govindarasu, for his excellent guidance during my stay at Iowa State University. It is a great opportunity to work with this extremely capable and blessed Professor.

Dr. Manimaran exhibited great patience and took utmost care in training me. During the initial phase of my PhD, he carefully observed and identified different areas in which I could improve to succeed as a researcher in the longer term. Subsequently, he provided me with ample opportunities that allowed me to shape myself as an independent researcher. In particular, he helped me to develop clear thinking and establish good technical writing and presentation skills. Throughout, he insisted in performing better, and encouraged me to reach the highest standards in everything I tried. It is always a pleasant experience to recall from where I started, and where I confidently stand today. If not for my advisor, I wouldn't be here. I express my deepest gratitude towards him. God bless Dr. Manimaran and his family.

I would also like to thank Prof. Zhengdao Wang for his collaboration in the research work. Dr. Wang has spent long hours with me in technical discussions, and presentations. He also helped me in reviewing my research papers and always gave very useful feedback. I am very grateful to him for all his help and encouragement.

I would further like to thank Dr. Arun Somani for giving me an opportunity to work with him. I am thankful to him for all his valuable suggestions. I also wish to thank Dr. Daji Qiao for being a very friendly and supportive Professor.

I thank all my past and present research lab members: Muthuprasanna Muthusrinivasan, Mohammad Fraiwan, Kavitha Balasubramanian, Basheer Al-Duwairi, Durga Kocherlakota, Suzhen Lin, Benazir Fateh, and Kayalvizhi Manimaran. It was great working with these people and I certainly learned something from each one of them.

I would like to thank my friends: Veerendra Allada, Pavana Vennapusa, Srikanth Komarina, Satya Saripalli, Prasad Avirneni, and Nishanth Gaddam. It is these people who made my stay at Iowa State University a fun filled journey. I am thankful to each one of them.

I would specifically like to thank my friend, Dinesh Rajan, for patiently attending all my practice presentations before my preliminary and PhD final examinations. He always encouraged me and gave very constructive feedback. I sincerely thank him for all his help.

Finally, I owe my warm gratitude to my beloved Mom, Mano Rama Devi, and my dear Dad, Jayaprakash Gathala, for believing in me with their unconditional love and support. I thank my brothers, Sudha Sunil and Sudha Kiran, and my sister, Padma Rajyam, for their constant encouragement and love.

ABSTRACT

Real-time embedded systems play a prominent role in a variety of applications ranging from medical sensors in human body to signaling sensors in war fields. The consumer domain of the embedded devices is large and ever increasing. A natural result of this trend coupled with those in sensor technologies and wireless communications have led to the rise of a new class of systems, called the *networked real-time embedded systems*. While the networked embedded systems enable newer and functionally richer applications, they pose major research challenges due to the complex requirements of satisfying temporal and reliability constraints in a resource limited distributed computing environment.

This dissertation develops a comprehensive algorithmic framework for system-level energy management in networked real-time embedded systems, with the goal of optimizing the energy consumption while satisfying application requirements (deadlines and precedence relations) and channel reliability constraints. The energy-management problem is decomposed into three levels and research contributions are made for each level: computing subsystem level, communication subsystem level, and the system level.

For the energy management at computing subsystem level, cross-layer energy-aware task scheduling algorithms are presented which employ the dynamic voltage scaling (DVS) power management technique to minimize the processor energy consumption while meeting all the task deadlines. Simulation studies show that the presented cross-layer algorithms yield enhanced processor energy savings compared to the existing algorithms for a variety of workload conditions. The communication energy management is addressed considering two different power management techniques namely, dynamic modulation scaling (DMS) and power adaptation. In each case, the energy-aware message scheduling problem is formulated tackling which both analytical and algorithmic solutions are presented. Performance results show that the proposed polynomial-time heuristic scheduling algorithms offer comparable energy savings to that of the analytically derived optimal solutions.

Thirdly, system-level energy management is addressed for a model where the individual nodes in the network support both DVS and DMS techniques. This work is the first of its kind that combines compute-level and communication-level energy management in an integrated manner. Tradeoffs between compute and communication energy consumption is established and cut-off region that favors one over the other is derived based on task/message and system characteristics - current processor frequency, current radio modulation level, task deadline, channel bandwidth, source-destination distance. Further, specific system-level energy-aware scheduling problems are formulated for single-hop and multi-hop networked embedded systems with deadline constraints. For the problems, based on the above system-level tradeoff analysis, novel algorithms for combined optimization of computation and communication energy are presented. Our performance results show that the proposed system-level energy-aware algorithms perform significantly better than the corresponding component-level algorithms.

In conclusion, this dissertation advances the state-of-the-art research in energy management in networked real-time embedded systems through an integrated framework and associated cross-layer algorithms and analyses. The system-level energy management formulation provides several avenues for further research, which include instantiating different power management techniques for computing and communication energy optimizations and studying their tradeoffs.

CHAPTER 1. Introduction

Real-time systems have undergone an evolution in the last few years in terms of their number and variety of applications as well as in complexity. A natural result of these advances coupled with those in sensor techniques and networking have led to the rise of a new class of application which fall into the distributed real-time embedded systems category [1, 2, 3]. Recent technological advancements in device scaling have been instrumental in enabling the mass production of such devices at reduced costs. As a result, applications with a number of internetworked embedded systems have become prominent. At the same time, there has been a need to move from stand-alone real-time unit into a network of units that collaborate to achieve a real-time functionality [4]. Extensive research has already been carried out to achieve real-time guarantees over a set of nodes distributed over wired networks [5]-[9]. However, there exist a number of real-time applications in domains such as industrial processing, military, robotics and tracking, which require the nodes to communicate over the wireless medium where the application dynamics prevent the existence of a wired communication infrastructure. These applications present challenges beyond those of traditional embedded or networked systems since they involve many heterogeneous nodes and links, shared and constrained resources, and are deployed in dynamic environments with changing participants where resource contention is dynamic [2]. Hence, resource management in embedded real-time networks requires efficient algorithms and strategies that achieve competing requirements such as time-sensitive energy-efficient reliable message delivery. In what follows, we discuss some applications in this category and discuss their requirements and challenges.

Safety-critical mobile applications running on resource-constrained embedded systems will play an increasingly important role in domains such as automotive systems, space, robotics and avionics [1]. The core controlling module in such mission critical applications is an embedded system consisting of a number of autonomous components. These components form a wireless ad hoc network for cooperatively communicating with each other to achieve the desired functionality. In these applications, a failure or violation of deadlines can be disastrous leading to loss of life, money or equipment. Hence, there arises a need to coordinate and operate within stringent timing constraints overcoming the limitations of the ad hoc wireless network. For example, robots used in urban search and rescue cooperate together and with humans in overlapping workspaces. For this working environment to remain safe and secure, not only must internal computations of robots meet their deadlines, but timely coordination of robots behavior is also required [4]. Other such medium-scale distributed real-time embedded applications include target tracking systems that perform surveillance, detection and tracking of time critical targets or a mobile robotics application where a team of autonomous robots cooperate in achieving a common goal such as using sensor feeds to locate trapped humans in a building on fire.

These applications need to meet certain real-time constraints in response to transient events, such as fast-moving targets where the time to detect and respond to events is shortened significantly. In surveillance systems, for example, communication delays within sensing and actuating loops directly affect the quality of tracking. While providing real-time guarantees is the primary requirement in these applications, mechanisms need to exist to meet other crucial system needs such as energy consumption and accuracy. In most cases, there are tradeoffs involved in balancing these competing requirements.

1.1 Networked Embedded System and Design Requirements

The typical architecture in a distributed real-time embedded system consists of several processor controlled nodes interconnected through one or more interconnection networks. The system software running on each node enables the execution of one or more concurrent tasks which is activated by the arrival of triggering events generated by the external environment, a timer or arrival of a message from another task. A response to an event generally involves several tasks to be executed on different nodes and several messages to be exchanged in the network. The tasks on the same node may share data and resources using the normal synchronization mechanisms present in shared memory systems and also interact with tasks on other nodes by exchanging messages using the services provided by the communication subsystem. For the proper functioning of the whole system, each individual task as well as all the messages exchanged need to complete before stringent deadlines.

The workload in majority of the distributed embedded real time applications is similar to those found in traditional real-time systems comprising of periodic and aperiodic tasks. Periodic tasks form the base load invoked at regular intervals while aperiodic tasks include the transient load generated in response to alarm or an external environment stimuli. However, one can expect stronger cooperation between the internetworked units in more dynamic and complex systems inducing richer communication patterns than simple periodic messages.

For distributed real-time embedded system, the primary requirement is that there is an end-to-end timing requirement that needs to be met. This implies that there exists a set of messages with complex precedence constraints that need to be exchanged between the networked nodes before some deadline. Hence, one needs to characterize the different message communications and computations that are possible and perform a pre-runtime analysis to guarantee apriori that all the task deadlines will be met after. Moreover, in a distributed real-time system, the ability to meet task deadlines largely depends on the underlying task allocation, and hence, we need a pre-runtime task allocation algorithm that takes into consideration the real-time constraints [5, 6, 8, 9]. Intertask communication significantly influences the response time of these distributed application and hence the design needs to account for the effect of delays imposed by the communication network and precedence constraints imposed by the communicating tasks during task allocation.

Since the inherent nature of many of the discussed applications precludes the use of wired networks, wireless networks are commonly used in such applications. The wireless medium is inherently unreliable due to characteristics such as fading and interference. Hence to guarantee that tasks should meet timing constraints, it becomes necessary to develop techniques that characterize the unreliability in the network channel characteristics and take them into account while making scheduling decisions.

Energy management is another crucial aspect for internetworked embedded devices. These devices contain not only radio and computer components, but also complete system functionalities, such as networking functions across all levels of the protocol stack. Energy savings and allocation among these modules will affect the life time of these battery-powered devices. Energy management also needs to be considered together with other constraints in size, real-time requirements, functionalities, and network connectivity.

In this thesis, we address the problem of energy-aware resource management in networked real-time embedded systems over a wireless network. This requires energy management at the computing subsystem, communication subsystem, and at the system level integrating the two. For the computing system, there are well known techniques, such as dynamic voltage scaling (DVS) in CMOS technology [10], that have been used by inter-task [11]-[17] and intra-task [18]-[31] scheduling algorithms for energy savings in embedded systems. Moreover, there are research efforts that focus on a combination of processor, memory [32], and I/O devices in an integrated manner [33, 34, 35] to improve the overall energy savings of stand-alone embedded systems. For the communication system, techniques such as dynamic modulation scaling (DMS) [36]-[76], power adaptation [40], and cross-layer designs [41]-[45] have been proposed for minimizing energy consumption in wireless and sensor networks. The objectives and workloads that are relevant to these wireless sensor networks are much different from that of the networked real-time embedded systems, and hence those techniques cannot be directly applied to the networked system considered in this dissertation.

Not only does there exist plenty of room to improve the energy savings offered by these task and message scheduling algorithms, but also there is a greater opportunity to look at the system as a whole balancing the tradeoff between energy savings in computing vs. communication so as to maximize the overall system energy efficiency. The research presented in this dissertation fills this void by developing energy-aware algorithms which would appropriately leverage the low power modes supported by different subsystems (both computation and communication subsystems) within each node of the network to obtain enhanced system-level energy savings.

1.2 The System-level Energy-aware Resource Management Framework

The major challenge in performing energy management in networked embedded systems lies in estimating the exact workload required by the application. As the exact workload determines the least power mode that the device can operate at while meeting the deadlines. In case of local computation, the workload refers to the task execution times which exhibit a wide variation from their worst case estimates. Most of the existing work speculate the task execution times. On the other hand, in the case of messages, the workload refers to the number of retransmissions required over a wireless link for a successful transmission. In our research, we consider both real-time constraints and channel conditions (reliability) while achieving energy efficiency of the networked embedded system. The proposed energy-aware resource management approach, shown in Figure 1.1, has the following three key components:

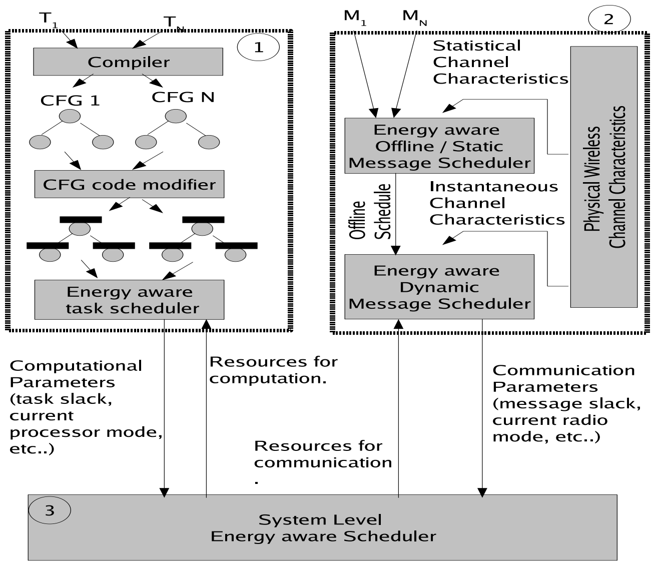

T_1
T_N
1
Compiler
CFG 1
CFG N
CFG code modifier
Energy aware task scheduler
M_1
M_N
2
Statistical Channel Characteristics
Energy aware Offline / Static Message Scheduler
Offline Schedule
Instantaneous Channel Characteristics
Energy aware Dynamic Message Scheduler
Physical Wireless Channel Characteristics
Computational Parameters (task slack, current processor mode, etc..)
Resources for computation.
Resources for communication .
Communication Parameters (message slack, current radio mode, etc..)
3
System Level Energy aware Scheduler

Figure 1.1 Architecture of the networked system level energy management

1. Energy Management at Computing Subsystem: This deals with the energy-aware real-time scheduling of tasks on a local node. Specifically, the goal is to minimize the processor energy consumption while meeting all the task deadlines. We present a cross-layer task scheduling algorithm that exploits intra-task information - such run-time branching information - provided at the granularity of control flow graph (CFG) of a task to minimize the processor energy consumption, employing DVS [10] technique. CFG represents the block level control flow structure of the program, with each node in the CFG denoting a basic block of computation and each edge in the CFG indicating a control dependency between two basic blocks. These algorithms can be employed in networked as well as stand-alone embedded systems.

2. Energy Management at Communication Subsystem: This deals with the energy-aware real-time scheduling of the inter-node messages over the wireless medium that is prone to phenomenon such as, fading, noise, and interference. Specifically, given a set of messages each with a source and a destination, the goal is to transmit them with the objective of minimizing the energy consumption of the communication subsystem while meeting all the message deadlines with a given probability of success. Due to the fading and noisy nature of the wireless channel, it is not feasible to guarantee 100% reliability.

3. Energy Management at the Networked System Level: In a typical networked embedded application, each node performs some local computation (task) and communicates the results (message) to a remote node in the network. Both task and message deadlines need to be met in order to provide end-to-end deadline guarantees. Further, to minimize the total energy consumption while guaranteeing the deadlines, the algorithm needs to optimally distribute the available slack among different tasks and messages. Task utilizes the slack to perform DVS while the message uses the slack to perform DMS or any other similar technique which tradeoffs time for energy. In general, the computation energy is much lesser than the communication energy. Therefore, giving as much slack as possible for communication sounds appealing on the surface. However, as can be seen from a more refined analysis, there is diminishing returns when the transmission time is increased beyond a certain threshold. Therefore, there should be a balance between the computing subsystem and the communication subsystem in slack distribution.

Since the grand problem of energy-aware real-time scheduling in networked embedded system is very complex (even its non-energy, non-real-time version itself is NP-complete [46]), we adopt a modular approach wherein the proposed generic algorithms/techniques build upon existing real-time scheduling algorithms to significantly improve their energy performance. Specifically, (1) for computing system, the proposed cross-layer design techniques work in conjunction with inter-task DVS algorithms (e.g., [11]-[17]); (2) for communication system, the proposed modulation and power-adaptation based techniques work in conjunction with medium access control protocols (e.g., [85, 86, 87]) in wireless networks; (3) for the networked system, the proposed tradeoff-based slack distribution algorithms assume the existence of underlying integrated task and message schedules produced by a distributed real-time scheduling algorithm (e.g., [5]-[9]).

1.3 Thesis Statement

This dissertation addresses the energy management problem in networked embedded real-time systems. We provide a comprehensive solution to this problem by addressing the energy management at computing and communication subsystems of individual nodes in the network. For computation energy management, we present a cross-layer approach which uses compiler generated control-flow information of different tasks at the operating-system level. Following this approach we are able to achieve enhanced processor energy savings when compared to the existing single layer (i.e., stand alone compiler-level or os-level) energy management techniques. For communication energy management, by considering the communication workload at different nodes in the network along with the wireless channel conditions, we are able to achieve significant communication energy savings while providing reliable and timely message delivery over the shared wireless network.

Furthermore, exploring a new dimension in the solution space of the energy management problem, this thesis establishes a novel system-level framework to perform integrated computation and communication energy management. This approach is significantly different from the existing component-level energy management techniques that are popular in the literature. The proposed system-level framework allows us to perform resource (i.e., time and energy) allocations across the computation and communication subsystems. We have applied the above approach to both single-hop and multi-hop networked embedded real-time systems considering specific applications in each case. Our results show that the proposed system-level approach yields much higher energy savings than the corresponding component-level algorithms.

1.4 Organization

The rest of the dissertation is organized as follows.

- In chapter 2, we address the energy-aware real-time task scheduling problem where the objective is to minimize the processor energy consumption while guaranteeing all the task deadlines. Firstly, we present a detailed review of the existing work in this context. We then present our generic cross-layer task scheduling algorithm that exploits intra-task information - such as run-time branching information - provided at the granularity of control flow graph (CFG) of a task to minimize the overall processor energy consumption. We present our simulation results towards the end of the chapter.

- Chapters 3 and 4 deal with the communication energy management. We address the energy-aware real-time message scheduling problem where the objective is to minimize the total communication energy consumption while meeting message dealine and reliability constraints. Chapter 3 employs Dynamic Modulation Scaling (DMS) as the basic underlying energy management technique while chapter 4 employs the technique of hop-by-hop transmissions. In each chapter, we prove that the corresponding problem is NP-Hard and provide ILP formulations to solve the problem optimally. Further, we also present polynomial time heuristic scheduling algorithms. Finally, we present our results.

- In chapter 5, we address the system-level energy managment problem involving both messages and tasks. we consider a single-hop networked real-time embedded system where each node supports both dynamic voltage scaling (DVS) and dynamic modulation scaling (DMS) power management techniques to tradeoff time for energy savings. In this model, we address the problem of scheduling periodic complex tasks where each task consists of several precedence constrained message passing sub-tasks. Our contributions towards this problem are two fold. First, we analyze the system level energy-time trade-offs considering both the computation and communication workloads by defining a novel energy gain metric. We then present static (centralized) and dynamic (distributed) energy gain based slack allocation algorithms which reduce the total energy consumption, while guaranteeing the ready time, deadline and precedence constraints. Towards the end of the chapter, we present our simulation results.

- In chapter 6, we address the system-level energy management problem for a particular mult-hop networked embedded system. Specifically, we tackle the problem of minimizing the total energy consumption of data aggregation with an end-to-end latency constraint while taking into account both the computational and communication workloads in the network. First, we present an analytical problem formulation for the ideal case where each node can scale its frequency and modulation continuously. Second, we present a Mixed Integer Linear Programming (MILP) formulation to obtain the optimal solution for the practical case where only few discrete frequency and modulation levels are supported by each node. Further, we present polynomial time heuristic algorithms, which employ the energy-gain metric established in chapter 5. Finally, we present our results and discussions.
- In chapter 7, we present our conclusions and identify several interesting energy-aware resrearch problems in the networked embedded real-time systems area.

CHAPTER 2. Energy management in computing system through cross-layer algorithms

This chapter deals with the energy management at computation subsystem level. Specifically, we concentrate on the minimization of processor's energy consumption in a uni-processor model. Most of the embedded processors are based on CMOS technology. CMOS based processors have both static and dynamic power dissipation [10]. The static power consumption is contributed by the leakage current and is present even when the circuit is not switching. On the other hand, the dynamic power dissipation is entirely due to the switching activity of the processor. Although there is a rising concern for the increased static power component in latest CMOS processors, we believe that the dynamic power is still the dominant component in the overall processor power consumption.

In this work, we concentrate on the dynamic power consumption of the processor which is given by [10]

$$P_{dyn} \propto V_{dd}^2 f \tag{2.1}$$

where V_{dd} is the supply voltage and f is the processor switching frequency. As per the above equation, the energy dissipated per cycle is directly proportional to the square of the supply voltage,V_{dd}. A widely used technique that exploits this characteristic is the Dynamic Voltage Scaling (DVS), whose goal is to choose the supply voltage and operating frequency as per the performance level required by the tasks. DVS allows to tradeoff the processor's speed for the energy savings. Since the energy consumption is directly proportional to the square of the supply voltage, DVS can make significant energy savings. Several energy aware real-time DVS algorithms have been proposed addressing a wide range of scheduling problems for a variety of task models.

2.1 Problem Statement

We address the following energy aware scheduling problem: *Given a set of periodic real-time tasks, the objective is to minimize the processor's energy consumption while meeting all the task deadlines.*

This problem can be optimally solved if the exact execution times of all the tasks are known apriori [18]. However, the task execution times often vary from the worst-case estimation and it is this non-determinism which makes the problem harder to solve. The existing solutions to this problem can be broadly classified into three classes based on the execution time models they follow: In the first class of algorithms [47, 48, 49, 50], the execution time of each task is assumed to follow a probabilistic distribution with a known worst-case bound. In the second class of algorithms [78, 52], the worst case execution time of each task is expressed as a function of parameters like number of memory accesses or the number of loop iterations with the aim of achieving a tighter worst-case bound on the execution time. Finally, in the third class of algorithms [11, 79, 14], the actual execution time is bounded by the worst-case execution time and no further assumptions are made with respect to the actual execution times. Therefore, in this class of algorithms the worst-case estimate is very conservative and hence exploiting the ample dynamic slack generated becomes very important for energy savings. Although the early execution technique presented here can be easily adopted to all the three models described above, in the rest of the chapter, we follow the third model due to its generality.

The existing real-time DVS algorithms can be further classified into intra-task and inter-task DVS algorithms based on the granularity at which the voltage scaling is performed. The intra-task voltage scaling algorithms [54, 21, 20, 19, 27, 55, 28, 31] adjust the supply voltage within a task boundary at a finer granularity for example, at the basic block level. The inter-task voltage scaling algorithms [11, 79, 14] perform voltage scaling on a task by task basis. An inter-task DVS scheme works with a set of tasks where as an intra-task scheme works with a single task. The early execution algorithms presented in this chapter work with a set of periodic tasks and perform both basic block level and task level voltage scaling.

2.2 Related work and motivation

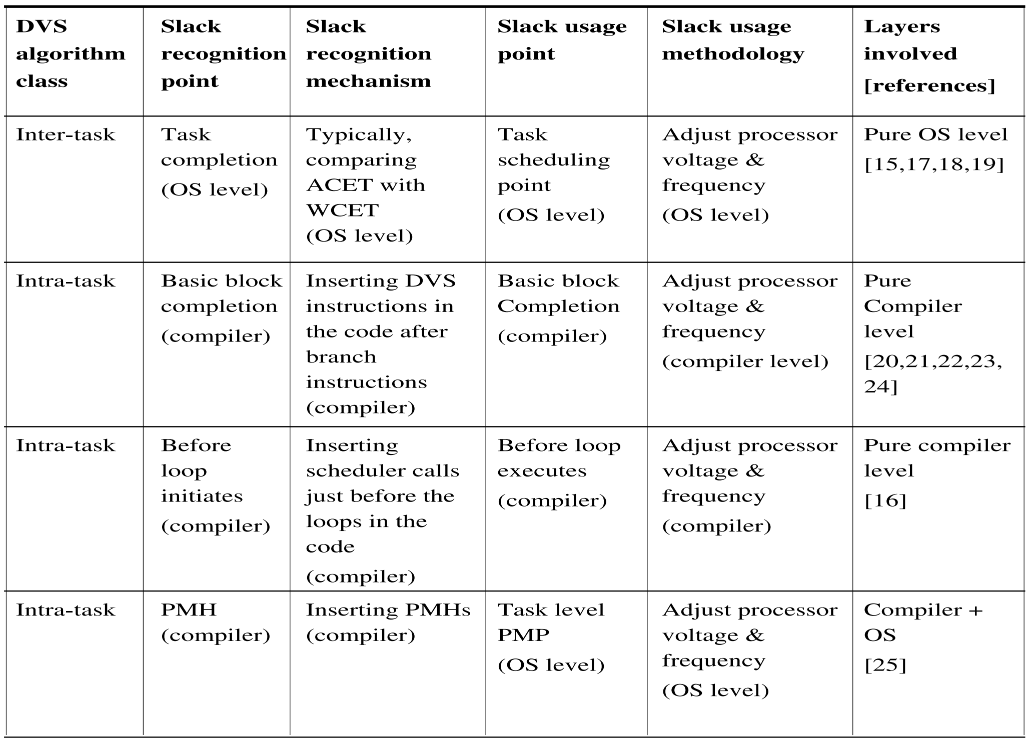
A typical energy aware real-time DVS algorithm consists of the following four components:

- *Slack Recognition Points:* It refers to the point or the granularity at which the algorithm recognizes the dynamic slack generated in the schedule. For example, in [55] the slack is recognized by the PMHs (Power Management Hints) at the code segment level.
- *Slack Recognition Mechanism:* It refers to the mechanism employed by the DVS algorithm to obtain this information. For example, in [55], insertion of the PMHs into the application code is the slack recognition mechanism and this is performed at the compiler level.
- *Slack Usage Point:* It is the point in schedule, when the recognized slack is exploited in some algorithm dependent way. In [55], the PMPs (Power Management Points) are the slack usage points.
- *Slack Usage Mechanism:* It refers to the mechanism employed by the DVS algorithm to use the slack for energy savings which is typically the processor's frequency and voltage adjustment.

The DVS algorithms differ from each other while being different in one or more of the four components. For example, conventional intra-task algorithms [54, 20] have the slack recognition points and the slack usage points to be the same. On the other hand in [55], they have been decoupled as PMHs and PMPs. Figure 2.1 shows the four components for different DVS algorithms.

Interestingly, most of the existing real-time DVS algorithms including [52, 11, 79, 55] work in the following two step framework. At each slack usage point:

1. Obtain the maximum usable/safe slack (S_{max}) as updated at the latest slack recognition point.



DVS algorithm class	**Slack recognition point**	**Slack recognition mechanism**	**Slack usage point**	**Slack usage methodology**	**Layers involved [references]**
Inter-task	Task completion (OS level)	Typically, comparing ACET with WCET (OS level)	Task scheduling point (OS level)	Adjust processor voltage & frequency (OS level)	Pure OS level [15,17,18,19]
Intra-task	Basic block completion (compiler)	Inserting DVS instructions in the code after branch instructions (compiler)	Basic block Completion (compiler)	Adjust processor voltage & frequency (compiler level)	Pure Compiler level [20,21,22,23, 24]
Intra-task	Before loop initiates (compiler)	Inserting scheduler calls just before the loops in the code (compiler)	Before loop executes (compiler)	Adjust processor voltage & frequency (compiler)	Pure compiler level [16]
Intra-task	PMH (compiler)	Inserting PMHs (compiler)	Task level PMP (OS level)	Adjust processor voltage & frequency (OS level)	Compiler + OS [25]

Figure 2.1 Computation energy management - related work summary

2. Adjust the operating frequency of the processor to use S_a amount of slack based on the slack usage mechanism. Where, $S_a \leq S_{max}$.

For example, the intra-task algorithm presented in [55], at each power management point obtains the dynamic slack notified by the latest PMH (step 1) and adjusts the operating frequency accordingly (step 2). Similarly, most of the other RT-DVS algorithms follow the above framework.

Depending on the exact slack usage mechanism S_a is either equal to S_{max} or strictly less than S_{max}. This decision is typically made by speculating the actual execution times of the ready tasks which are yet to complete. For example, the look-ahead EDF [11] speculates the best case execution times for the future tasks and hence aggressively reduces the operating frequency of the current task. On the other hand, DRA (Dynamic Reclaiming Algorithm)

[79] and cycle conserving EDF [11] implicitly assume a worst case work load for the future task instances. The AGR (Aggressive Speed Adjustment) [11] performs DVS speculating an average workload rather than best or worst case workloads. In [30], the actual execution times are estimated using feedback control theory and this amount of computation is operated at least possible frequency while the rest is operated at the maximum frequency. This kind of speculation or estimation based frequency adjustment cannot reduce the energy consumption effectively when the speculation/estimation goes wrong. With this motivation, in this chapter, we present a technique which uses a part of the available slack for reducing the non-determinism in task execution times and hence the need for speculation. The rest of the slack plus the additional slack generated due to the reduction in non-determinism is used for reducing the operating frequency. Thus our work in this chapter, differs from the existing RT-DVS algorithms in the slack usage mechanism.

The **basic idea of our early execution technique** which is targeted for a multi-task setup is as follows. At the slack usage point, whenever some non-zero slack (S_{max}) is available the early execution technique tries to execute the branch instructions of the other ready tasks using this slack which will otherwise be executed later. This early execution of the branch instructions is done with the aim of reducing the non determinism involved in the actual execution times of the future tasks as early as possible. Since dealing at instruction level incurs a lot of overhead, in this chapter we work at the basic block level. More specifically, we work with the control flow graph (CFG) of each task and perform early execution of the basic branch blocks. Typical intra-task DVS algorithms assume a block size of the order of $10 * 10^6$ CPU execution cycles [20]. We followed a similar model in this chapter. In cases where the basic block size is much lesser, we assume a sequence of blocks (a sub-graph in the CFG) are composed into a large enough basic block. The early execution technique is independent of the task code model and can be used in other non-CFG models as well, however, due to the clarity that the CFG model offers we have chosen to use this model here.

The work which is closely related to our research in terms of the motivation is presented in [50, 56]. In [56], the authors find the best order of the tasks for execution in a frame based setup with the aim of reducing the biggest non-determinism as early as possible. The basic idea there is to execute the task which has the largest variation between the worst case execution time and expected execution time first. In [50], the authors came up with a priority metric based on which the task's execution order can be chosen. Although the need for reducing the non-determinism was well motivated and expressed in [56], their work is very restricted. In [50, 56], they consider only the frame based tasks and the presented solutions cannot be directly applied to a more general setup. While the early execution technique presented in this chapter can very well be adopted to a more general setup of periodic tasks with different periods and deadlines. In [56], the authors extended their work from frame based setup to handle more general cases. For the unique period and different deadline case, they propose to use the static priority algorithms like DMS (Deadline Monotonic Scheduling) and try to order the tasks which have the same deadlines which is a very restrictive case. No further mention was made regarding the task ordering for the different period and different deadline case. Unlike the work presented in [56], the early execution technique presented here can very well be used with dynamic priority algorithms like EDF as well. Also, our early execution technique works at a much fine grained level i.e. at the basic block level and hence can make much better energy savings.

Technically, the early execution concept presented in this chapter can be employed along with any of the above existing multi-task DVS algorithms to further enhance the energy savings. However, the deadline guarantees and other properties provided by the underlying DVS algorithm need to be reconsidered while employing the early execution technique. Working out the details for employing early execution technique for each of the above algorithms would be an exhaustive exercise. Therefore, in this chapter, we restrict ourselves to the inter-task algorithms and present a generic early execution algorithm which can applied over many of the existing inter-task algorithms. We also prove that our generic early execution algorithm does not violate any of the deadline guarantees provided by the underlying inter-task algorithms.

The rest of the chapter is organized as follows: in section 2.3, we present the basic idea of the proposed early execution algorithm along with an illustrative example. We also outline the different overheads that the early execution algorithm would incur and provide a series of tests that need to be checked to guarantee deadline constraints. In section 2.4, we present our generic early execution algorithm and estimate the cost of the algorithm in terms of the measured overhead parameters. In section 2.5, we introduce negative early executions and extend the generic early execution algorithm to avoid such energy inefficient early executions. We present our simulation results in section 2.6 and final discussions in section 2.7.

Symbol	AMD Mode	AMD Power	PXA255 Mode	PXA255 Power
f_{high}	2000 MHz	89 W	400 MHz	411 mW
f_{mid}	1800 MHz	66 W	300 MHz	283 mW
f_{low}	1000 MHz	22 W	200 MHz	175 mW
Idle	StopGrant	4.1 W	33MHz@Idle	45 mW

Table 2.1 Power specifications[82, 58]

2.3 Exploiting intra-task structure through early execution

The proposed early execution technique aims at minimizing the total energy consumption by reducing the non-determinism in the workload, this is achieved by exploiting the intra-task structure (i.e, CFGs) of the individual tasks at the inter-task level. The early execution algorithm uses intra-task structure in the following two ways:

- *Early execution of basic branch blocks:* Whenever the current task generates slack at run-time due to the shorter branch execution, the early execution algorithm uses this slack to execute the basic branch blocks of the other ready tasks rather than using the entire available slack for slowing down the processor.

- *Fine grained estimation of average utilization:* Even after performing early execution, there remains some non-determinism in the workload. In order to handle it, our algorithm also estimates the average utilization and update it after the completion of every basic branch block. The estimated average utilization is used to limit the aggressiveness of the underlying inter-task algorithm so that the speed adjustment is justified by the average workload. We underline that our approach of estimating the average utilization is more powerful than the existing inter-task level approach of estimating the average utilization [79]. This is because our intra-task level estimate of the average utilization allows a quicker adaption to the exact execution times within the task as opposed to after the task.

In the following subsection, we present a working example to demonstrate the effectiveness of our early execution concept and compare it with the existing inter-task algorithms.

2.3.1 Illustrative example

Consider a simple task set with two tasks: T_1 & T_2. The individual block sizes, worst case computation (WC_i) and the deadline of each task are expressed as the number of CPU execution cycles. The absolute ready time (R_i), period (P_i) and deadline (D_i) can be obtained by dividing them with f_{high}, the highest frequency supported by the processor. Consider the task parameters of $T1$ & $T2$:

- $R_1 = 0$, $WC_1 = 100 * 10^6$, $P_1 = D_1 = 225 * 10^6$
- $R_2 = 0$, $WC_2 = 100 * 10^6$, $P_2 = D_2 = 225 * 10^6$

In the following, we will work with the AMD Opteron power specifications (see table 1). The absolute period and deadline times for the above tasks are: $P_1 = D_1 = (225*10^6/2000MHz) = 112.5ms$. Similarly, $P_2 = D_2 = 112.5ms$. The CFGs of the two tasks are shown in figure 2.2. The highlighted paths (T_1: $B11, B12$ & T_2: $B21, B22$) execute more often due to program path locality. Since both the tasks have equal deadlines and ready times, we assume the EDF* scheduling policy [79] which schedules the lowest index task first among the equal priority tasks.

For the given task set, most of the inter-task DVS algorithms including the Look-ahead EDF, AGR, DRA, OTE (One Task Extension) produce the same frequency settings as shown in figure 2.2. Since the inter-task schemes have no idea of what the actual execution times

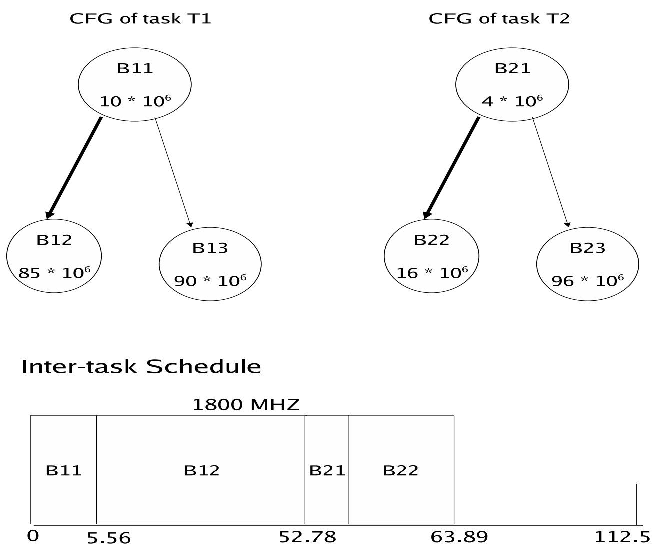


Figure 2.2 Illustrative example: inter-task schedule

are going to be, they start out with the 1800 MHz frequency (which is the minimum safe frequency to meet all the deadlines) *assuming a worst case workload.* At the completion of task T_1 (time = 95/1800 = 52.78 ms), the inter-task algorithms see a slack of $5 * 10^6$ cycles (or 5/1800 = 2.7 ms), but cannot reduce the frequency any further, once again assuming the worst case computation for task T_2. At time $63.89ms$ ($= 52.78 + 20/1800$ ms), the inter-task algorithms see that the actual workload is much less than the worst case workload. However, there is no computation left to utilize the available slack (from time 63.89 ms to time 112.5 ms) and hence the slack is left unused. The energy consumption of the task set as per the inter-task schedule is $E_{inter} = 63.89 * 66 + (112.5 - 63.89) * 4.1 = 4416.0$mJ (milli Joules). Where the power consumption from time 0 to 63.89 ms is chosen as 66 W and for the rest of the time an idle power of 4.1 W is used as per table 2.1.

Now, consider the schedule obtained by performing early execution in figure 2.3. At time 0, the task T_1 starts with the 1800 MHz frequency and completes the basic block $B11$ at time = 5.56 ms and notifies the slack generated as $5 * 10^6$ cycles (or 2.7 ms) to the early-execution scheduler. The early-execution scheduler schedules the T_2's branching block ($B21$) which can be accommodated in the generated slack. By doing this, at time 7.78 ms, the early-

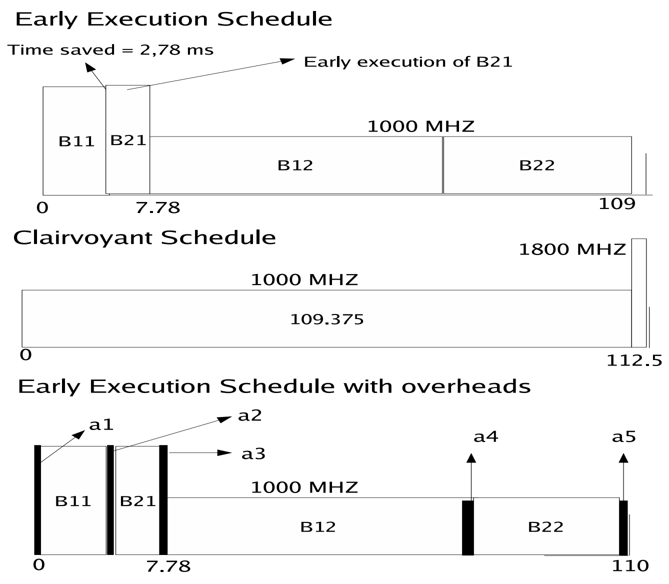


Figure 2.3 Illustrative example: early execution schedule

execution scheduler knows the exact workload till time 112.5 ms and hence it uses most of the available slack to schedule the rest of the computations at a much lower frequency of 1000 MHz (obtained as, $101 * 10^6/(112.5 - 7.78) = 964MHz < 1000MHz$) as shown in the figure. The energy consumption of the early execution schedule is $E_{early} = 7.78*66+(108.78-7.78)*22+(112.5-108.78)*4.1 = 2750.73$ mJ. Clearly, by breaking the non-determinism due to the execution of basic blocks of different tasks at a much earlier time, the early execution scheduler is operating a large amount of computation at a much lower frequency while still guaranteeing the deadlines. In the rest of the chapter, we call the task which offers the early execution time as the **host task** and the task which executes its decision basic blocks as the **guest task**. In this example, $T1$ is the host task and $T2$ is the guest task.

Figure 2.3 also shows the schedule produced by the clairvoyant scheduler which by definition knows the exact workload (the computation times) at time 0 and hence operates at the optimal speed(s) ($115/112.5 = 1022MHz$, therefore operates at 1000MHz and 1800MHz, as $1000 < 1022 < 1800$) as per the actual workload. The energy consumption incurred by the clairvoyant algorithm is $E_{Clair} = (109.375) * 22 + (112.5 - 109) * 66 = 2612.5$ mJ. In this example, the proposed early execution technique achieves about 37.70% of energy savings over the inter-task schemes and it incurs about 5.02% more energy than the clairvoyant algorithm. Similar working with the PXA255 settings for this example results an improvement of 20.5% over the inter-task algorithms and early execution algorithm incurs about 2.56% more energy over the clairvoyant algorithm.

In order to realize the significant energy gains offered by the early execution technique in a safe manner, we need to make sure that the early execution does not disturb any non-host and non-guest tasks. This can be guaranteed by executing the guest task at the host task's priority during the early execution. This way other tasks are oblivious of the early execution and no intermediate task can preempt the guest task leading to a possible indefinite blocking of the host task; further, no higher priority task undergoes a priority inversion due to early execution.

2.3.2 Overheads

Like any fine grained approach the proposed early execution algorithm also incurs overheads. We have measured the different overheads that would be incurred by the early execution algorithm and calculated the energy associated with it. Our measurements have shown that the overheads are very low both in terms of time and energy. In the following, we provide these details.

	AMD Time taken	AMD # CPU cycles	PXA255 Time taken	PXA255 # CPU cycles
α_1	0.5 us	1000	6us	2400
α_2	6 us	12000	32 us	12800
α_3	10 us	20000	45 us	18000
α_4	1.5 ms	$3 * 10^6$	7.5 ms	$3 * 10^6$
α_5	1 ms	$2 * 10^6$	5 ms	$2 * 10^6$

Table 2.2 Overhead measurements (us: microseconds, ms: milliseconds)

Implementing the early execution algorithm involves two phases: an offline phase and an online phase. The offline phase constructs the CFG of the application and adds the necessary system calls at appropriate points in the code. While the online phase simply responds to these system calls and it does not require to maintain the task CFGs. The online phase is basically carried out by the early execution scheduler and for each task it needs to know just two parameters namely: time saved, ts and cycles left, cl. The first parameter ts is the time saved by the task due to a shorter branch execution (required when the task acts as a host) and cl is the remaining number of execution cycles left to complete the next basic branching block (required when the task acts as a guest). The early execution scheduler will need to incorporate these two new parameters into the operating system kernel's task structure which will be updated regularly by the task's code. The offline phase calculates these parameters for each basic block and adds the necessary function calls to the task's code. The task parameters are updated through the standard POSIX interfaces like *pthread_set* function calls in RTLinuxPro [59] and other Linux based real-time operating systems. Similar interfaces can be created for the above two parameters as well (something like *pthread_set_ts* and *pthread_set_cl*).

We consider the different overheads involved in the online phase and ignore the offline overheads. The following is a list of the various online overheads:

- *Parameter update time* (α_1): The time taken to update a single task parameter which is typically made by a *pthread_set* function call in RTLinuxPro. This is also the time incurred in changing a task's priority, which is again a task parameter.

- *Context switch time* (α_2): This is the time taken to switch between two tasks after the scheduler has chosen the next task to run.

- *Scheduling overhead* (α_3): This is the worst case time taken to choose the next task based on one of the task parameters. Specifically, it is the time taken to scan the entire task list.

- *Frequency change* (α_4): This is the amount of time taken to change the operating frequency. In our measurements, different transitions took different amounts of time and α_4 represents the largest of all.

- *Idle mode transition time* (α_5): The amount of time taken to enter and exit the processor idle state.

We measured the above different overheads in RTLinuxPro on the AMD Opteron and Intel XScale PXA255 processors using the standard measurement utility programs that are shipped with the operating system. The system was running at f_{high} by default with hundred real-time runnable tasks (see table 2.2). Some of the overheads depend on the number of runnable tasks, but in a typical real-time embedded system the number of tasks is much lesser than hundred. Therefore, the above measurements serve as the upper bounds for the typical case.

The working of the above example considering the overheads is shown in figure 2.3. The different overheads ($a1$ to $a5$) can be calculated as follows:

- $a1 = \alpha_1/1800 = 0.56$ us, this is incurred when $B11$ updates its cl.

- $a2 = (\alpha_3+\alpha_1+\alpha_2+\alpha_1)/1800 = 18.89$ us. The first term is incurred in choosing the guest task, the second term is incurred in changing the guest task's priority to host priority. The third is incurred in switching from the host task to guest task and the final term is incurred by the guest task in updating its cl. Clearly, $a2$ and $B21$ can be accommodated in time saved 2.7 ms.

- $a3 = (\alpha_1 + \alpha_3 + \alpha_2 + \alpha_1 + \alpha_4)/1800 = 1.69$ ms. The first term represents the time incurred in changing back the guest task's priority to its original value. The second and third terms is incurred in choosing the next task (T_1) to run and switching to it. The fourth term represents the term incurred in updating the new task's cl. The final term represents the time incurred in changing the operating frequency.

- $a4 = 2 * \alpha_1/1000 = 2$ us, this is incurred when $B22$ updates its cl and when task T_1 updates its cl for the next instance as $10 * 10^6$. This is done at the end of each task instance to allow the early execution scheduler to consider its next instance for an early execution as and when it arrives and is runnable.

- $a5 = (\alpha_1 + \alpha_5)/1000 = 2.001$ ms, the first term represents the time incurred when task

T_2 updates its cl as $4*10^6$ for its next instance and the second represents the time taken to enter and exit the idle state.

The total overhead is 3.7 ms ($= a1 + a2 + a3 + a4 + a5$) and the energy incurred is 157 mJ ($= (a1 + a2 + a3) * 66 + (a4 + a5) * 22$). The total energy consumption of the early execution schedule with the overheads is 2892.562 mJ (obtained as, $7.78*66+(108.78-7.78)*22+(112.5-108.78-3.7)*4.1+157$) and the energy savings over inter-task schedule is about 34.49%. Now, the early execution algorithm incurs about 9.68% more energy than the clairvoyant algorithm.

With the PXA255 settings the total overhead is 17.72 ms and it incurs 4.91 mJ of energy. The early execution schedule now gives an improvement of 18.73% over the inter-task schedule and incurs about 4.65% more energy than the clairvoyant algorithm.

2.3.3 The Early execution criterion

In order to guarantee the real-time constraints while performing early execution and accommodate the associated overheads, we have developed the following three tests that need to be verified at different stages of implementation. The total early execution overheads involve two components namely, the time incurred in updating the task's parameters and the time incurred in running the early execution scheduler which uses these updated parameters. The first component is always encountered while the second is incurred only when the early execution scheduler is invoked. Therefore, it is possible to estimate the worst case value of the first component for each task and test if this overhead can be accommodated without violating deadlines offline. Once this is guaranteed, the worst case estimate of the second component needs to be accommodated within the dynamic slack generated due to shorter branch execution.

1. **Offline schedulability test**: This test is performed offline and checks for the task parameter update overheads. Every basic block updates both the ts and cl parameters in the worst case. Further, at the end of each task instance the cl for the next instance is notified so that early execution can be performed when the task is just ready. If a task T_i has b_i number of basic blocks in the path with largest number of blocks, then overhead

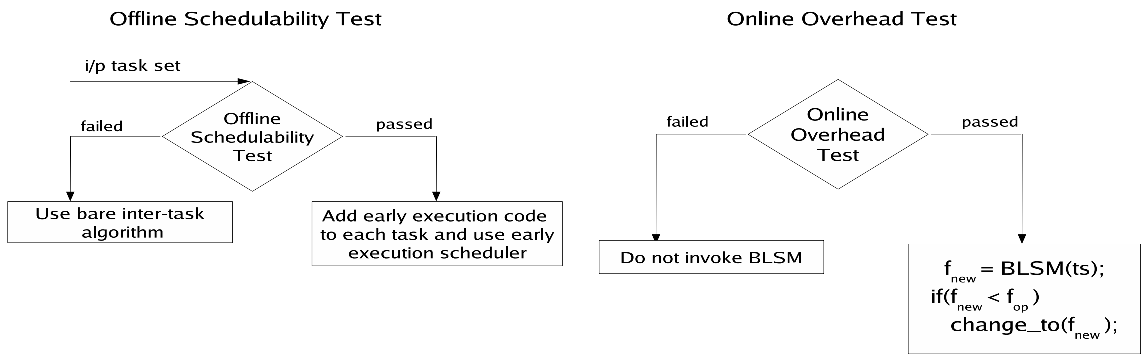

Offline Schedulability Test
i/p task set
failed
Offline Schedulability Test
passed
Use bare inter-task algorithm
Add early execution code to each task and use early execution scheduler
Online Overhead Test
failed
Online Overhead Test
passed
Do not invoke BLSM
f_new = BLSM(ts);
if(f_new < f_op)
change_to(f_new);

Figure 2.4 Early execution - schedulability tests

due to these updates is $b_i * 2 * \alpha_1 + \alpha_1$. For a given task set with n tasks, the worst-case computation time of each task T_i should now be considered as $C_i + (2b_i + 1)\alpha_1$ and the corresponding schedulability needs to be re-verified with the updated computation times. In case of the look-ahead EDF, the schedulability test now becomes:

$$\sum_{i=0}^{n} \left(\frac{C_i + (2b_i + 1)\alpha_1}{P_i}\right) \leq 1 \tag{2.2}$$

2. **Online overhead test**: This online test checks for the second component of overheads which include cost/time of the early execution scheduler (say, ϕ) and the time incurred in making a change to the new frequency (α_4). Therefore, before invoking the early execution scheduler on a shorter branch execution, it is mandatory to verify that the dynamic slack generated is greater than $((\phi + \alpha_4)/f_{op})$. Further, early execution algorithm is invoked only when the current operating frequency is greater than the minimum processor frequency. Mathematically,

$$ts > \frac{(\phi + \alpha_4)}{f_{op}} \tag{2.3}$$

$$f_{op} > f_{low} \tag{2.4}$$

where f_{op} is the current operating frequency. We derive a concrete expression for ϕ in the next section.

3. **Online early execution test**: The above two tests check for the overheads. Figure 2.4, shows how these tests are used at different stages. Now, within the scheduler, we

need to verify if the basic block of the guest task can be accommodated in the leftover dynamic slack ($ts - (\phi + \alpha_4)/f_{op}$) generated due to the shorter branch. More specifically, we need to verify the following condition before choosing any task as an eligible guest task in order to guarantee the deadlines.

$$(ts - \frac{(\phi + \alpha_4)}{f_{op}}) \geq \frac{cl}{f_{op}} \tag{2.5}$$

2.4 The generic early execution algorithm

We now present our generic early execution algorithm which reduces the energy consumption by performing safe early executions while meeting all the task deadlines. The generic early execution scheduler consists of two modules namely, Block Level Slack Manager (BLSM) and Task Level Slack Manager (TLSM) built over an inter-task DVS algorithm.

```
   Input: time saved due to shorter branch execution, $ts$
   Output: new operating frequency, $f_{new}$
1  $ts = ts - (\phi + \alpha_4)/f_{op}$
2  while $ts > 0$ do
3      Pick the next task ($T_j$)from the ready queue;
4      If(($T_j.cl/f_{op}$) $\leq ts$)
5      Add ($T_j$,$T_j.cl$) to the guest list;
6      $ts = ts - (T_j.cl/f_{op})$;
7      If(ready queue has no more tasks left) break;
8  end
9  foreach task $T_j$ in the guest list do
10     Execute $T_j$ at the host's priority for $T_j.cl/f_{op}$ amount of time at frequency = $f_{op}$;
11 end
12 Update the average utilization ($U_{avg}$) based on the early executions performed;
13 Update the appropriate inter-task algorithm parameters;
14 Set $f_{new}$ = TLSM($U_{avg}$);
15 return $f_{new}$
```

Algorithm 1: Block Level Slack Manager (BLSM)

Whenever a (host) task executes a basic branch block, it informs the amount of time saved to the BLSM. The pseudocode of the BLSM is shown in Algorithm 1 and it works as follows. In step 1, the worst case scheduler overhead and the frequency change overhead are deducted from ts. In step 2, the BLSM scans through the ready queue and picks the next task (T_j).

```
  Input: current average utilization, $U_{avg}$
  Output: new operating frequency, $f_{new}$
1 Calculate the $S_{max}$ as per the underlying inter-task DVS algorithm;
2 Calculate $f_{min}$, the operating frequency at which all the $S_{max}$ is used.;
3 Set $f_{new} = Max(f_{min}, U_{avg})$;
4 Set $f_{new}$ to the smallest discrete frequency level that is greater than $f_{new}$;
5 return $f_{new}$;
```

Algorithm 2: Task Level Slack Manager (TLSM)

In step 3, the BLSM checks if the task T_j passes the online early execution test outlined in the previous section. If the task T_j passes, the BLSM adds it to the guest list and deducts an appropriate amount from the ts. In steps 9 to 10, the BLSM executes each of the guest tasks for an appropriate amount of time at the host task's priority. The time incurred for the steps 1 to 9 can be approximated to two scans of the entire task list which is $2\alpha_3$ and step 10 includes the context switch time incurred in performing the early executions. In the worst case, all $n-1$ tasks can be eligible guest tasks in which results in a cost of $(n-1)\alpha_2$ for step 10. After all early executions, the scheduler performs a context switch back to the host task or any higher priority task, therefore, the total time incurred in all the context switches is $n\alpha_2$. In addition, each context switch involves increasing the guest task's priority to that of the host task and later back to its own. Hence, these priority changes incur a time of $n * 2\alpha_1$.

The BLSM updates the average utilization and the inter-task parameters in steps 12 and 13 respectively. The inter-task parameters for example, in the case of Look-ahead EDF, include the rem_cc_i of both the host and guest tasks. In case of AGR scheme the α-queue is updated with the actual execution times of the host and guest tasks. Therefore, the time incurred by step 13 can be approximated to $2\alpha_1$ and the worst case time of step 12 is α_3. Finally in step 14, the BLSM obtains the operating frequency by invoking the TLSM and returns the same.

The detailed pseudocode of the TLSM is presented in Algorithm 2 and it works as follows. In step 1, the TLSM calculates the maximum slack that is currently available as per the underlying inter-task DVS algorithm. In the worst case, a typical inter-task algorithm scans through the whole task list, hence this step costs about α_3. In step 2, the lowest operating frequency at which the whole of the maximum slack is utilized is obtained. In step 3, the operating frequency is chosen as the maximum of the average utilization and the minimum frequency. Finally, in step 4, the chosen operating frequency is rounded to the nearest discrete frequency level supported by the processor. We ignore the costs of steps 2,3 and 4 as they are only a few instructions. It is important to note that the TLSM is invoked by the BLSM after every early execution and it is also invoked after every task completion in a regular inter-task manner.

The worst case total cost of a single BLSM invocation is given by:

$$\phi = 4\alpha_3 + n(\alpha_2 + 2 * \alpha_1) + 2\alpha_1 \tag{2.6}$$

This corresponds to 0.7 ms at 2000MHz and 4.6 ms at 400MHz on AMD Opteron and Intel XScale PXA255 respectively. A typical basic block of size $10 * 10^6$ execution cycles [20] takes about 5 ms and 25 ms on AMD Opteron and PXA255 respectively. Therefore a call to the early execution algorithm would cost about $\frac{1}{5}$ th of basic block time at the most. The asymptotic run-time of both the TLSM and BLSM algorithms is $\bigcirc(n)$.

2.4.1 Schedulability Guarantees

The early execution scheduler preserves all the deadline guarantees provided by the underlying inter-task algorithm. In the following, we provide an informal proof for the above statement.

The early execution technique guides the underlying inter-task algorithm by showing the slack in the schedule at an earlier point of time. Without the early execution, inter-task algorithms would see the same slack at a later point, possibly when the slack cannot be utilized. This early information is achieved by executing a basic branch block of the guest task in the time slice of the host task. The guest task is executed at the host task's priority during the early execution. In this process, the rest of the tasks are left unaffected. As a result, we can consider an early execution to be modifying the CFGs (and the actual execution times) of the guest and host tasks.

Consider an early execution scenario with a host task (T_i) and a guest task (T_j) whose CFGs are shown in figure 2.5. Blocks $B2_i, B3_i, B2_j, B3_j$ here represent the whole sub-trees

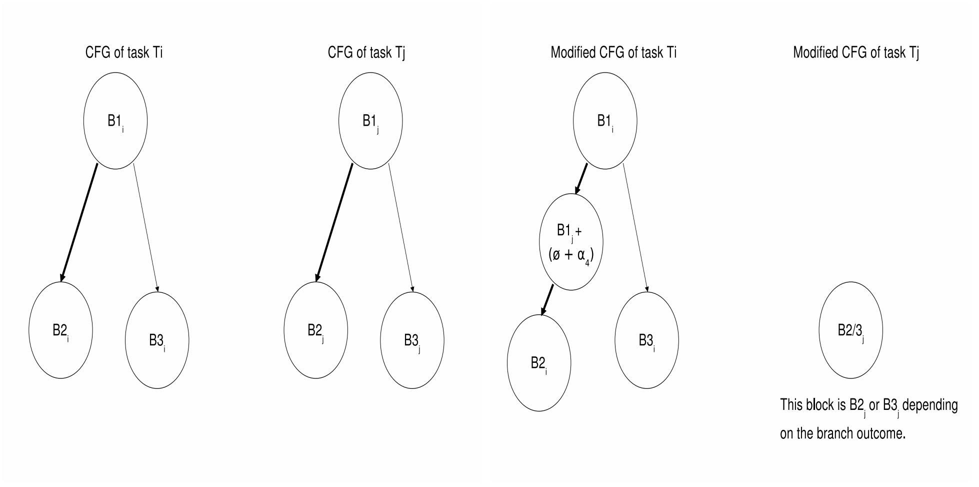


Figure 2.5 Early execution - effective CFG modifications

below them. In the degenerate case, they are simply basic blocks. The paths: $(B1_i, B3_i)$ and $(B1_j, B3_j)$ represent the corresponding worst-case (and largest) paths i.e, $B3_i > B2_i$ and $B3_j > B2_j$.

During the early execution, effectively, the block $B1_j$ is transferred from the guest task T_j to T_i as shown in figure 2.5. Further, the overheads $(\phi + \alpha_4)$ are accounted by including it in the transferred block. Such a CFG transformation will preserve the schedulability guarantees provided by the underlying inter-task algorithm as long as it does not increase the worst-case execution times of the host task. Clearly, the early execution scheme does not increase worst computation of any task. This because the early execution is performed only if $(B3_i - B2_i) > (B1_j + (\phi + \alpha_4))$ as ensured by the online early execution test. Further, the early execution algorithm always operates at a frequency greater than or equal to the minimum frequency depicted by the underlying inter-task algorithm (step 3 of TLSM). Consequently, the early execution algorithm preserves the deadline guarantees provided by the underlying inter-task algorithm.

2.5 The conservative early execution (CEE) algorithm

Although very rare in practice, sometimes an early execution can result in higher energy consumption than otherwise. We call such early executions as **negative early executions**.

In this section, we demonstrate the negative early executions with an illustrative example and extend the generic early execution algorithm presented above to avoid such negative early executions. Consider two simple tasks T_1 & T_2 with the following parameters:

- $R_1 = 0$, $WC_1 = 100 * 10^6$, $P_1 = D_1 = 225 * 10^6$
- $R_2 = 0$, $WC_2 = 100 * 10^6$, $P_2 = D_2 = 225 * 10^6$

Using the AMD Opteron power specifications, the absolute period and deadline times for the above tasks are: $P_1 = D_1 = P_2 = D_2 = (225 * 10^6/2000MHz) = 112.5ms$. Let the CFGs of T_1 & T_2 be as shown in figure 2.2 with the following modified basic block sizes: $B_{11} = 5 * 10^6$, $B_{12} = 5 * 10^6$, $B_{13} = 95 * 10^6$, $B_{21} = 50 * 10^6$, $B_{22} = 45 * 10^6$ and $B_{23} = 50 * 10^6$. The inter-task and the early execution schedules are shown in figure 2.6. At time 2.78, the above early generic early execution algorithm executes the branching block B_{21} of task T_2 using the time saved (50 ms) by the task T_1 due to shorter branch execution. The amounts of energy consumption incurred by the early execution and the inter-task schedules are as follows: $E_{early} = 30.58{*}66{+}50{*}22{+}31.92{*}4.1 = 3249.152W$ and $E_{inter} = 5.56{*}66{+}95{*}22{+}11.94{*}4.1 = 2505.914W$. Clearly, $E_{early} > E_{inter}$.

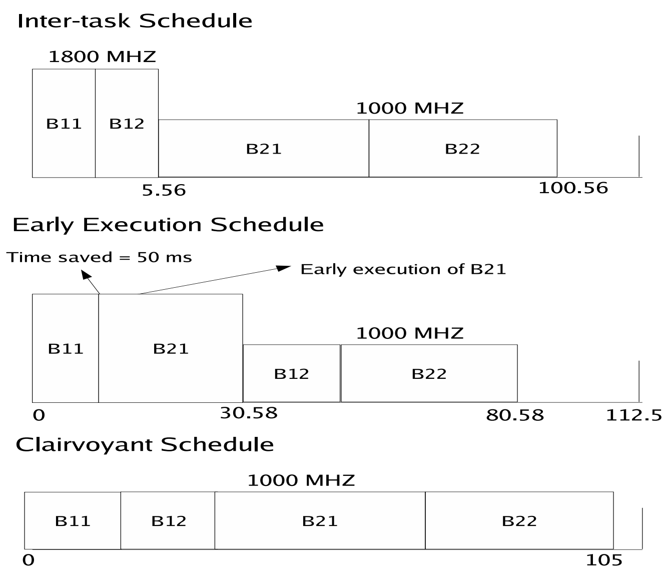


Figure 2.6 Illustrative example: different schedules

The reason behind the above negative early execution is as follows: In any early execution the branching block of the guest task is generally executed in a more non-deterministic environment and hence at a higher operating frequency than otherwise. For example, block B_{21} of task T_2 is executed at time 2.78 at a frequency of $1800MHz$, on the other hand it is executed at a frequency of $1000MHz$ otherwise, as shown in the inter-task schedule. Therefore, it is very important that the decision block is small enough to avoid the negative early executions. Furthermore, the amount of reduction in the average utilization that the early execution brings also dictates the benefits of the early execution. In this example, the best path execution length is not much different from the worst case execution path (i.e. the difference is only $B_{23} - B_{22} = 5 * 10^6$ cycles). Therefore, due to the above factors in this example the eligible early execution manifests as a negative early execution.

Theoretically one can calculate the net energy gains of an early execution and avoid it if the energy gain is negative. The energy gain obtained by an early execution is given by:

$$E_{gain} = \sum_{\forall i,j} (\frac{B_{ij}}{f_{xij}} * P_{xij} - \frac{B_{ij}}{f_{yij}} * P_{yij}) - (\frac{\phi + \alpha_4}{f_{op}}) * P_{op} \tag{2.7}$$

Where f_{yij} is the operating frequency of B_{ij} if the early execution is performed and f_{xij} is the operating frequency without the early execution. The terms P_{xij} and P_{yij} represent the corresponding power consumption values. The last term above represents the energy incurred by the early execution overheads.

Calculating the E_{gain} before performing an early execution according to the above formula however, is not practical in the absence of future knowledge as we cannot determine all the basic blocks that execute and their corresponding operating frequencies. Therefore, in this chapter, we make a conservative estimate of the E_{gain} as follows:

$$E_{cgain} = (\frac{C_{rem}}{f_{op}} * P_{op} - \frac{C_{rem}}{f_y} * P_y) + (\frac{cl}{f_{avg}} * P_{avg} - \frac{cl}{f_{op}} * P_{op}) \tag{2.8}$$

Where C_{rem} is the remaining cycles of the host task's best execution path, cl is the number of cycles that the guest task needs to execute in order to complete its next branch block, f_{op} is the current operating frequency and f_{avg} is the smallest frequency which is greater than the current average utilization. Here f_y, the operating frequency after the early execution is approximated as the smallest frequency which is greater than the updated average utilization assuming the average case path in the guest task will execute.

The first term in the above equation represents the amount of energy gained due to early execution while the second term represents the energy lost in executing the branching block (as it is executed earlier and at a higher frequency typically) of the guest task. The above estimate of the energy gains of an early execution is conservative in the sense that it considers the energy savings of the host task alone. On the other hand, the early execution can potentially reduce the operating frequency of a lot more tasks. In order to avoid the negative early executions associated with the average case execution path, we augment the online early execution test with the following additional condition: $E_{cgain} > 0$. We call this variation of the early execution algorithm as the conservative early execution (CEE) algorithm. Since the generic early execution algorithm does not perform any kind of check for negative early executions we refer to it as aggressive early execution (AEE) algorithm in the rest of the chapter.

2.6 Simulation Studies

We performed simulation studies to compare the relative performance of the following algorithms: (1) Look-ahead EDF, (2) Intra Look-ahead EDF (Look-ahead EDF applied at the basic block level) (3) AEE (4) CEE and (5) Clairvoyant algorithm, the theoretical lower bound of the energy consumption. Both the AEE and CEE algorithms assume Look-ahead EDF as the underlying inter-task algorithm. The performance metric for all our simulations is the normalized energy consumption i.e. the total energy consumption normalized with respect to the DVS unaware scheduler. We conducted three sets of simulations each with a different input task set. We used the following real applications in the first two simulation sets: *sensor application* and *video phone application.* In the third set, we used *randomly generated task sets.*

Figure 2.7.(a) outlines our general simulation methodology. The simulation setup consists of a discrete event simulator which accepts task set information, processor DVS settings (table

2.1) and the different overheads (table 2.2) as input. It evaluates all the above schemes and outputs the corresponding normalized energy consumption values as shown in the figure. The same simulator is used across the three simulation sets. However, the task set information is generated in a different manner for each simulation set. In the following, we present these details along with the results.

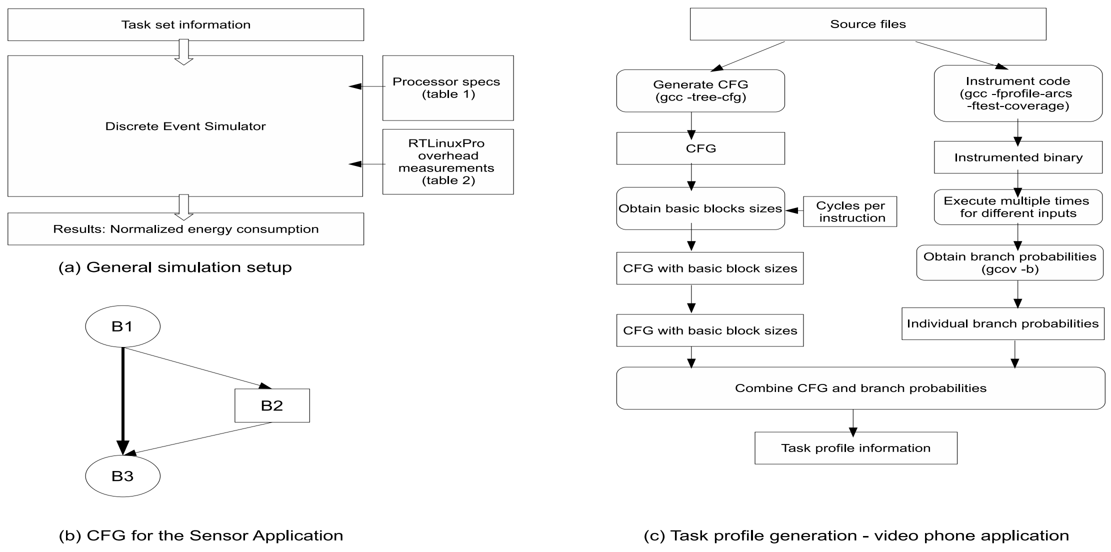

Task set information
Discrete Event Simulator
Processor specs (table 1)
RTLinuxPro overhead measurements (table 2)
Results: Normalized energy consumption
(a) General simulation setup
B1
B2
B3
(b) CFG for the Sensor Application
Source files
Generate CFG (gcc -tree-cfg)
Instrument code (gcc -fprofile-arcs -ftest-coverage)
CFG
Instrumented binary
Obtain basic blocks sizes
Cycles per instruction
Execute multiple times for different inputs
CFG with basic block sizes
Obtain branch probabilities (gcov -b)
CFG with basic block sizes
Individual branch probabilities
Combine CFG and branch probabilities
Task profile information
(c) Task profile generation - video phone application

Figure 2.7 Simulation setup

2.6.1 Sensor Application

Application Description: Computational demands on the individual sensor nodes are gradually increasing particularly due to the rising security concerns [22, 23]. Sensor nodes are now expected to perform computationally expensive message encryptions and decryptions [23]. In the following, we consider a representative sensor application involving a set of periodic tasks running on a single sensor node. Each task (hereafter referred as *sense-filter-send* task) periodically obtains a new sample of a specific phenomenon and determines if the obtained sample is significantly different from the previously communicated sample. If the sample is new, it is encrypted and communicated via the wireless medium to the appropriate destination; otherwise, the sample is ignored. In this chapter, we focus on the computational workloads of each task and ignore the communication aspects.

Workload details: The CFG of a single sense-filter-send task is shown in figure 2.7.(b). The block B_1 represents the initial processing on the obtained sample and the block B_2 denotes the number of computation cycles involved in performing the message encryption. Finally, the block B_3 represents the computations involved in things like clearing the message buffer, updating the latest communicated message, etc. In this chapter, we assumed $B_1 = 50$, $B_2 = 10^6$ and $B_3 = 10$ cycles. We obtained the message encryption time in CPU cycles (i.e, size of B_2) from [23]. In the CFG, the path (B_1, B_3) denotes the case where the new sample is not very different from the previously communicated sample in which case the number of CPU execution cycles to very few. We refer to this path as the best path and the probability of execution for this path depends on the specific phenomenon and the overall application. We considered five periodic tasks whose deadline was chosen equal to their period. In practice each sensor node is capable of sensing as high as ten different phenomenon simultaneously.

Simulation parameters: In this simulation set, we used Intel PXA255 processor settings. Further, we varied the following two parameters: system utilization (U) and the best path probabilities (P_b). A specific value of system utilization is achieved by choosing the task periods accordingly. The actual execution path for each task is chosen independently and uniformly at random with a probability P_b.

Effect of utilization (U): Figure 2.8.(a) compares the normalized energy consumption of the above algorithms with varying worst-case utilization. With the increasing worst-case utilization, the individual task periods are chosen small as a result the room for performing DVS is reduced. Consequently, the energy consumption of all the algorithms increase with increasing utilization. Throughout the range, both the early execution algorithms perform better than the basic underlying look-ahead EDF. At lower worst-case utilization all the schemes could operate at the minimum operating voltage. As the worst-case utilization increases, both variations of look-ahead EDF operate at higher processor frequencies while the early execution algorithms perform better by gaining a better idea of the exact work-load with the help of early executions. AEE shows an improvement of about 16% and 21% over look-ahead EDF at

$U = 0.80$ and $U = 0.90$ respectively. Throughout, AEE incurs as less as 2% or less additional energy than the clairvoyant algorithm.

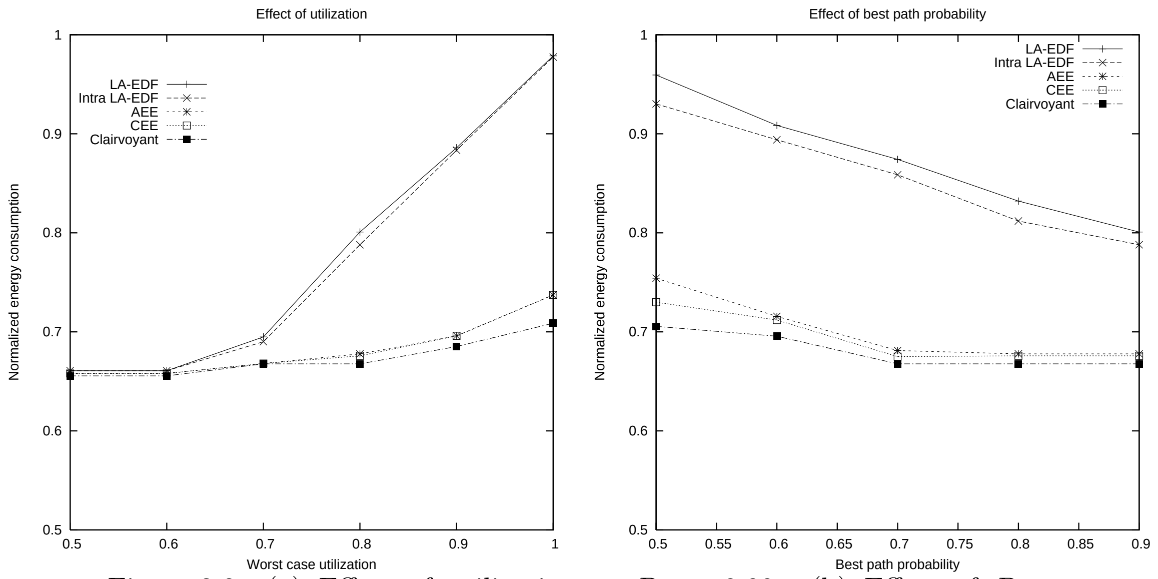


Figure 2.8 (a) Effect of utilization at $P_b = 0.90$ (b) Effect of P_b at $U = 0.80$

Effect of best path probability (P_b): Figure 2.8.(b) shows the normalized energy consumption of the above schemes with varying best case execution path probability. Higher the probability, higher is the number of instances which take the best case path in the corresponding CFGs. As we increase P_b, the amount of computation performed by each task decreases on an average. Therefore, the energy consumption of all the algorithms decreases with increasing probability. Also with increasing probability, the savings achieved by the early execution algorithms increase because each early execution will result in lowering the operating frequency with a higher likelihood. At $P_b = 0.70$, AEE shows an improvement of about 20% and 18% over look-ahead EDF and Intra-look-ahead EDF, respectively. Further, AEE incurs about 5% more energy than the clairvoyant algorithm at $P_b = 0.70$.

2.6.2 Video Phone Application

Application Description: In this simulation set, we considered a video phone application that consists of four periodic tasks as shown in table 2.3. The four tasks together perform video, audio decoding and encoding to obtain the functionality of a video phone. We would like to note that a similar application has been used for performance evaluation in [20]. The individual tasks are chosen from media-2 benchmark program suite [24].

Workload details and simulation parameters: For each task, starting with its source code we obtained its control flow graph, worst-case execution time and individual branch probabilities. Figure 2.7.(c) outlines the different steps we followed in obtaining this information. The CFGs are obtained using the GCC compiler with appropriate options as shown in the figure and later each basic block size is obtained by counting the number of assembly instructions that constitute that block. For simplicity, we assumed each instruction takes one CPU cycle. This flow is shown as the left branch in figure 2.7.(c).

Based on the CFG generated, we have noticed that at the basic block sizes varied significantly. In order to avoid excessive overheads, we have enabled early execution code only at the branches where significant slack would be generated (that is where the best and worst case branches differ in their sizes significantly). Specifically, early execution is performed only when the slack generated by the host task is at least twice as the early execution overheads.

Further for each task, we obtained the individual branch probabilities in the CFG using GNU profiler. The source code is compiled with required instrumentation and later run for different inputs to obtain the individual branch probabilities. For those branches where we could not determine the probabilities, we assumed all unknown branches are equi-probable. This flow is shown as the right branch in figure 2.7.(c). Table 2.3 shows both the WCET and BCET (best-case execution time) values at $400MHz$ for different tasks. The WCET and BCET values are estimated by calculating the execution times for longest and most probable paths respectively in the corresponding CFGs. In the following, we present our results with the video phone application. We varied the system utilization and studied the relative performance of different algorithms.

Effect of Utilization: The plots shown in figure 2.9 compare the normalized energy consumption of the above algorithms with varying worst-case utilization. With the increasing worst-case utilization, the individual task periods are chosen small as a result the room for performing DVS

	MPEG4 video encoding	MPEG4 audio decoding	GSM speech encoding	GSM speech decoding
WCET (msec)	55.7	10.86	2.83	1.43
ACET (msec)	14.50	1.62	1.97	0.59

Table 2.3 Estimated execution times @ 400 MHz

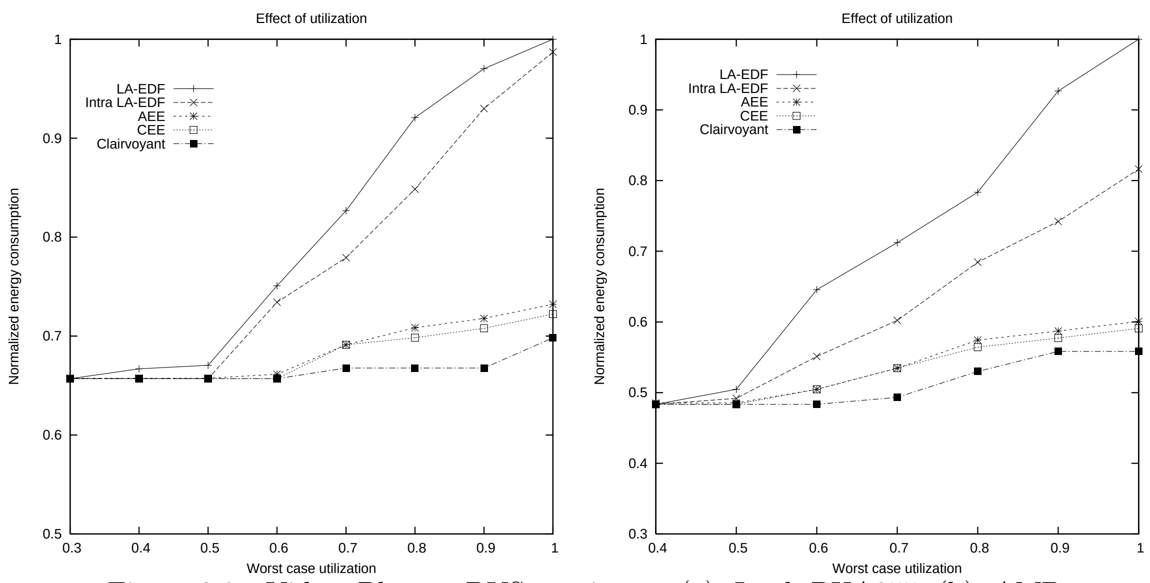


Figure 2.9 Video Phone, DVS settings: (a) Intel PXA255 (b) AMD Opteron

is reduced. Consequently, the energy consumption of all the algorithms increase with increasing utilization. Throughout the range, both the early execution algorithms perform better than the basic underlying look-ahead EDF. At lower worst-case utilization all the schemes could operate at the minimum operating voltage. As the worst-case utilization increases, both variations of look-ahead EDF operate at higher processor frequencies while the early execution algorithms perform better by gaining a better idea of the exact work-load with the help of early executions.

For PXA255 DVS settings (see figure 2.9.(a)), at $U = 0.60$ AEE shows an improvement of 11% over Look-ahead EDF and about 9% over Intra Look-ahead EDF scheme. At $U = 0.90$ AEE shows an improvement of 26% and 22% over Look-ahead EDF and Intra Look-ahead EDF schemes respectively. Similarly, for AMD Opteron (see figure 2.9.(b)), at $U = 0.60$ AEE shows an improvement of 21% over Look-ahead EDF and about 8% over Intra Look-ahead EDF scheme. At $U = 0.90$ AEE shows an improvement of 34% and 20% over Look-ahead EDF and Intra Look-ahead EDF schemes respectively.

2.6.3 Random Task Sets

In this simulation set, we have studied the performance of the above algorithms on randomly generated task sets. For each task, the number of basic blocks per task is chosen randomly from $[50, 100]$. The basic block size (number of CPU execution cycles) is chosen from $[10^6, 10^7]$. For each simulation run, we generated 30 tasks whose period is randomly chosen from the interval $[100, 4000]$. We randomly generated the control flow graphs for each of the 30 tasks with the appropriate worst-case and best-case execution paths.

We studied the effect of the following parameters: worst case utilization (U), ratio of the average case execution time to worst case execution time (ACET/WCET) and probability of the best case execution path (Pr). The default values for the above parameters are chosen as : $U = 0.98$, $ACET/WCET = 0.5$, $Pr = 0.8$.

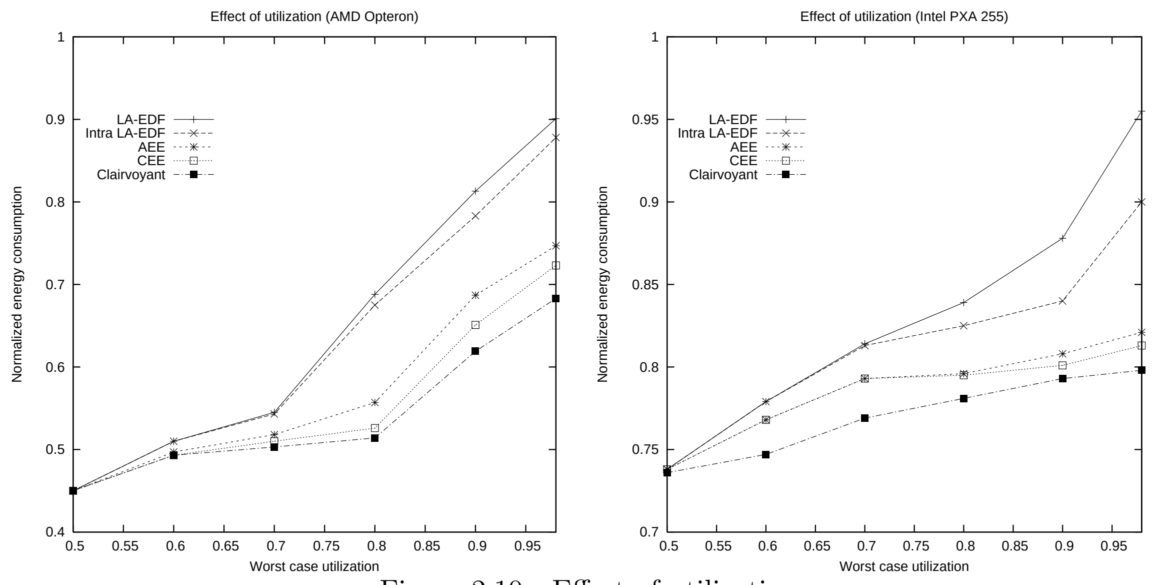


Figure 2.10 Effect of utilization

Effect of Utilization: The plots shown in figure 2.10 compare the normalized energy consumption of the above algorithms with varying worst-case utilization. With the increasing worst-case

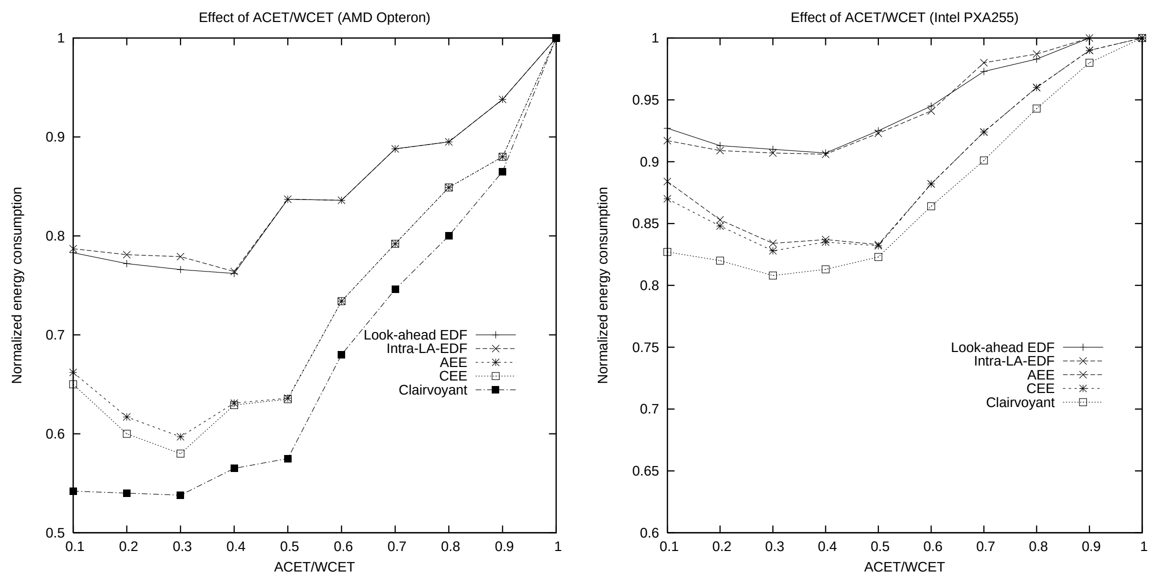


Figure 2.11 Effect of ACET/WCET

utilization, for a given ACET/WCET, the actual computation performed by each task also increases. Consequently, the energy consumption of all the schemes also increase. Throughout the range, both the early execution algorithms perform better than the basic underlying look-ahead EDF. At lower worst-case utilization all the schemes could operate at the minimum operating voltage. As the worst-case utilization increases, the look-ahead EDF operates at higher operating frequencies while the early execution algorithms perform better by gaining a better idea of the exact work-load with the help of early executions. At $U = 0.98$, the AEE shows an improvement of 24% (10%, for PXA255) over the look-ahead EDF and incurs about 6% (2% for PXA255) more energy than the clairvoyant algorithm. There is almost no difference between CEE and AEE indicating there are very few negative early executions for the chosen parameter values.

Effect of ACET/WCET: Figure 2.11 shows the normalized energy consumption of the above schemes with varying ACET/WCET. With increasing ACET/WCET, for a given worst case utilization, the amount of actual computation performed by each task also increases. Consequently, the energy consumption of all the algorithms also increase with increasing ACET/WCET. Throughout the range, the early execution algorithms perform significantly better than the un-

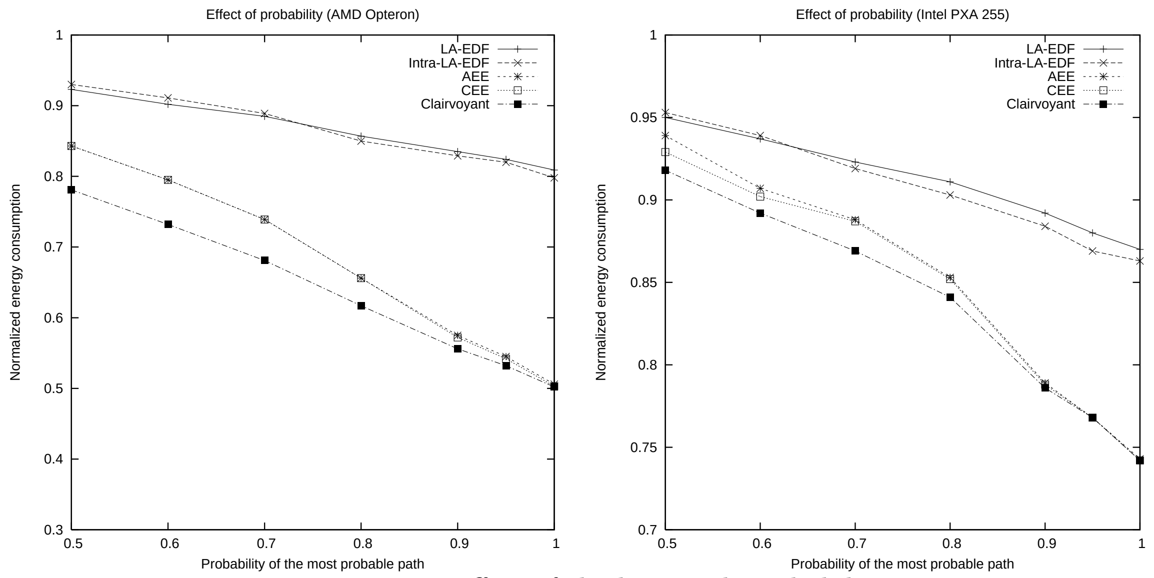


Figure 2.12 Effect of the best path probability

derlying look-ahead EDF algorithm. At lower ACET/WCET ratio, there are a good number of early executions performed and both the early execution algorithms perform better than the underlying look-ahead EDF. Further, the lot of room created by the low ACET/WCET results in a few negative early executions and CEE shows a slight improvement over the AEE. At higher ACET/WCET, there is relatively lesser room for early executions consequently both AEE and CEE show decreasing improvements over the look-ahead EDF. AEE shows an improvement of 17% (7.5% for PXA255) and 6% (3% for PXA255) at $ACET/WCET = 0.5$ and $ACET/WCET = 0.9$ respectively.

Effect of best path probability: Figure 2.12 shows the normalized energy consumption of the above schemes with varying best case execution path probability. Higher the probability, higher is the number of instances which take the best case path in the corresponding CFGs. For a given ACET/WCET (less than one), as we increase the probability the amount of computation performed by each task decreases on an average. Therefore, the energy consumption of all the algorithms decreases with increasing probability. Also with increasing probability, the savings achieved by the early execution algorithms increase because each early execution will result in lowering the operating frequency with a higher likelihood. AEE shows an improvement of 35% (13.5% for PXA255) over the look-ahead EDF and incurs about 7.5% (1.8% for PXA255) more energy over the clairvoyant algorithm at probability 0.9.

2.7 Discussion

In this chapter, we proposed a novel generic DVS technique for energy-aware scheduling of periodic tasks that can be adapted to many of the existing inter-task algorithms. By utilizing the available slack for early execution of the basic branch blocks of the other ready tasks, the algorithm attains a more accurate picture of the actual workload than the underlying inter-task algorithm and adapts accordingly. We have measured the different overheads that would be incurred by the proposed algorithm on AMD Opteron and Intel XScale PXA255 processors in RTLinuxPro and incorporated them in our simulations. We have evaluated the proposed algorithms for a variety of scenarios by conducting three sets of simulations. In the first two sets we considered real applications namely, a sensor application and a video phone application. In the third set, we considered randomly generated task sets. Our results show that the early execution technique yields significant energy gains over the look-ahead EDF. In our future work, we plan to extend the proposed early execution concepts to periodic task system with shared resources. The early execution principle in such a system requires taking into account not only the scheduler's operation, but also the rules of the resource access control protocol.

CHAPTER 3. Energy management in communication system using dynamic modulation scaling

In this chapter, we address the communication energy management problem by employing Dynamic Modulation Scaling (DMS) as the underlying energy management mechanism.

Timely and reliable message delivery is of paramount importance in a variety of application domains involving battery-driven embedded devices that communicate over the wireless network. Providing real-time and reliability guarantees for these applications is extremely challenging due to the severe energy limitations experienced by the individual devices which are further overburdened by the time-varying nature of the wireless medium. Trying to provide such stringent guarantees can demand exceedingly high energy resources from different nodes in the network. Therefore, energy management is a primary concern in the design and operation of these time-sensitive applications.

Current state-of-the art embedded devices support a variety of power management techniques which can be exploited by the higher layer protocols for effective energy management. These techniques include: *sleep-wakeup* [60, 61], *power adaptation* [62, 63] and *performance scaling* [64]. In this chapter we focus on performance scaling.

Performance scaling refers to the technique of lowering the transmission power and transmitting the packet for a longer period of time to reduce the transmission energy consumption. This technique allows to tradeoff transmission latency for communication energy savings. Such an energy-time tradeoff can be achieved either by varying the modulation level or error correcting codes or both simultaneously in a message transmission. These specific variations are termed as Dynamic Modulation Scaling (DMS), Dynamic Code Scaling (DCS) and Dynamic Modulation-Code Scaling (DMCS) respectively [65]. Although the specific variables or control knobs might vary across the three techniques, the fundamental tradeoffs they offer are similar. In the rest of the chapter, we work with the DMS technique.

3.1 Related work and motivation

Several algorithms and protocols have been developed to exploit the energy-time tradeoffs presented by different performance scaling techniques to achieve impressive energy gains for a variety of network setups. The different performance-scaling based scheduling algorithms that are proposed in the literature can be categorized as follows based on their network model.

- *Single Sender, Single Receiver*: The network setup involving a single sender and receiver has been considered in [64, 66, 67]. In [66], the problem of scheduling a set of finite (non-periodic) messages with a common deadline is addressed where the objective is to minimize the energy consumption of the sender node. An optimal algorithm has been presented along with an online scheduling algorithm. Recently, the work presented in [67] addressed a similar problem with individual packet deadlines rather than a common deadline for all packets. For this model, the authors presented an optimal offline scheduling algorithm which is further analyzed and extended in [64]. The problem of scheduling a set of periodic messages has been addressed in [68] where the objective is to minimize the total energy consumption of the schedule subject to individual message deadlines. All the above proposed scheduling algorithms exploit the convex nature of the energy function and are based on the assumption that energy consumption is a continuous and monotonically decreasing function.

- *Single Sender, Multiple Receivers*: The down-link problem of scheduling a set of packets with a common deadline from a single sender to multiple receivers is considered in [69]. The authors presented an optimal scheduling algorithm and further extended it to handle the case where packets have individual deadlines. The work in [69] also addresses the up-link scheduling problem involving *multiple senders and a single receiver*. Both offline and online algorithms have been developed for this problem.

Majority of the above mentioned work addresses different scheduling problems involving either a single sender or a single receiver and the proposed solutions cannot be directly extended to scenarios involving multiple senders and receivers. In this chapter, we consider a more general network model involving multiple senders and receivers which share the common wireless medium.

Further, as mentioned earlier, most existing work is based on the assumption that the transmission energy consumption is a continuous and monotonically decreasing function of transmission latency. While the monotonicity does not completely hold in the presence of reliability requirements as will be discussed in the subsequent sections, the continuity assumption itself does not hold in practice. In practical systems only a few discrete performance levels are allowed and each transmission can be performed at one of these levels. As a result the solutions developed assuming a continuous energy function merely represent lower bounds to the practical problem setups. Several chapters suggest a simple rounding mechanism (referred as *rounding algorithm* in the rest of this chapter) where the continuous solutions are rounded to the nearest discrete solution taking a conservative approach [70]. However, such rounding mechanism is not the optimal approach and hence cannot effectively reduce the energy consumption particularly, when the number of discrete performance levels are few which is typically the case in practical systems.

Unlike most of the existing work, our primary focus is on the discrete problem formulation where only a few discrete modulation levels are allowed. We however, also address the continuous problem formulation considering energy consumption is a continuous function for two important reasons. Firstly, obtaining an optimal solution for the discrete problem is difficult as the problem in this case becomes NP-Hard. Therefore, we use the continuous problem formulation to obtain lower and upper bounds for the optimal solution of the discrete problem. Secondly, we would like to compare the performance of the proposed heuristic scheduling algorithms developed for the discrete problem with that of the rounding algorithm.

In addition to the above surveyed work, several other network level problems have been addressed in conjunction with the DMS technique involving multiple senders and receivers [70].

However, the specific problems they address are fundamentally different from the problem we focus here.

The rest of the chapter is organized as follows: We present our system model in section 3.2. In section 3.3 we present the problem statement along with the ILP formulation. We also prove that the problem is NP-Hard. In section 3.4, we present the continuous problem formulation along with its solution. In section 3.5, we present our heuristic scheduling algorithm. We present our simulation results and conclusions in sections 3.6 and 3.7 respectively.

3.2 System Model

Network and Communication Model We consider a wireless network with v nodes which share the common wireless medium that is accessed in an exclusive manner. As in most of the existing work [70, 66, 69], we assume all nodes in the network are time synchronized.

As for the communication model, we assume that the modulation is Quadrature Amplitude Modulation (QAM). The channels are modeled as frequency-flat Rayleigh fading. Let b denote the modulation level i.e, number of bits per modulation constellation symbol. The constellation size is calculated as $M = 2^b$. Let E_b denote the received energy per bit, $N_0/2$ denote the channel noise power spectral density, E_s denote the received energy per constellation symbol, d^2_{min} the minimum average squared Euclidean distance between two constellation symbol at the receiver, and BER denote the bit error rate. We have the following relationships:

$$b = \log_2 M \tag{3.1}$$

$$\text{BER} \approx N_0/d^2_{min} \tag{3.2}$$

$$d^2_{min} = \frac{6}{M-1}E_s \tag{3.3}$$

$$E_s = bE_b \tag{3.4}$$

The approximation in (3.2) is valid for high signal-to-noise ratio (SNR). Combining these, we have

$$E_b \approx \frac{2^b - 1}{6b} \cdot \frac{N_0}{\text{BER}}. \tag{3.5}$$

If a message contains L bits, then the total necessary received energy will be $E_L = L \cdot E_b$. We assume that the propagation loss follows a polynomial model: the power decays in the α-th order of the distance:

$$E_{ts} = \left(\frac{d}{d_0}\right)^{\alpha} E_L \tag{3.6}$$

where E_{ts} is the necessary transmitted energy to achieve a received energy of E_L, d is the distance between the transmitter and the receiver, d_0 is a normalizing constant that depends on the wavelength. Usually α is between 2 and 5. As a result, the total transmitted radio energy can be expressed as

$$E_{ts} = \left(\frac{d}{d_0}\right)^{\alpha} \frac{L(2^b - 1)}{6b} \cdot \frac{N_0}{\mathrm{BER}} \tag{3.7}$$

Further, the energy consumption of the electronic circuitry during a message transmission and reception is given by, $E_{tc} = \frac{LC_t}{b}$ and $E_{rc} = \frac{LC_r}{b}$ respectively [65]. Here, C_t and C_r are implementation dependent constants.

The time taken for transmitting a L-bit message in the shared wireless medium is given by:

$$T = \frac{L}{Wb} \tag{3.8}$$

Here, W denotes the bandwidth (symbols per second) of the channel in hertz and the bandwidth in bits per second can be calculated as Wb. In the rest of the chapter, we assume $N_0 = 4 * 10^{-13}$, $\alpha = 2$, $C_t = 75nJ$, $C_r = 100nJ$, $L = 1024$ and $W = 1KHz$ as the default values.

Message Reliability Model: Individual message reliabilities can be provided by either performing retransmissions whenever necessary or by performing a single transmission with high enough energy. Each one of the two options presents a different kind of a energy-time tradeoff and they both require different scheduling approaches. In this chapter, we follow the single transmission model. In such a model, the message reliability Rel_{ij} defined as the probability that the message m_i reaches the destination without errors in a single transmission made at a modulation level b_j can be calculated as:

$$Rel_{ij} = (1 - SER)^{\frac{l_i}{b_j}} \tag{3.9}$$

where SER is the symbol-error-rate.

Scheduling Model: We use the earliest deadline first (EDF) [71] as the basic scheduling mechanism. EDF prioritizes messages based on their deadline and schedules the message with earliest deadline first. EDF is an optimal preemptive scheduling algorithm which can achieve 100% utilization while meeting all the deadlines. With EDF as the underlying scheduling policy we follow a two stage (offline and online) preemptive scheduling model. In the offline phase, the proposed scheduling algorithms assign modulation levels to individual messages ensuring that the message set remains schedulable under EDF. In the online phase, an EDF scheduler is employed which schedules the messages in the channel considering the modified message latencies due to updated modulation levels. We assume that the input message set is schedulable when each message is transmitted at the highest modulation level.

3.3 Problem Statement and ILP Formulation

Consider a set of periodic messages $\{m_1, m_2, ..., m_n\}$ where each message m_i has a period T_i equal to its deadline. Each message is associated with a unique source and destination node pair in the network. Let $\{b_1, b_2 ....b_N\}$ denote the different possible modulation levels where $b_i < b_{i+1}$. Each message m_i is of size l_i bits. The objective of the scheduling problem is to choose one of the N modulation levels for each message m_i which minimizes the total energy consumption while guaranteeing both the reliability and deadline constraints. To the best of our knowledge, this problem has not been addressed in conjunction with the DMS technique.

For simplicity, we assume all instances of a given periodic message are transmitted at the same modulation level. When dealing with periodic messages, the total energy consumption is defined with respect to a time window, say $[0, T_o]$, where T_o is an application specific parameter. In the following, we present an ILP formulation for the above stated scheduling problem for a given T_o.

3.3.1 ILP Formulation

We use the following symbols in the formulation.

- n: number of input messages.
- N: number of discrete modulation levels.
- d_i: normalized source destination distance of m_i.
- $E_{i,j}$: Energy consumption of a single transmission of message m_i at a modulation level b_j. This includes the corresponding reception energy and it is calculated as,

$$E_{ij} = d_i^2 \frac{l_i(2^{b_j} - 1)}{6b_j} \cdot \frac{N_0}{\text{BER}} + E_{tc} + E_{rc} \tag{3.10}$$

- R: minimum required reliability per message transmission that must be guaranteed.
- $x_{i,j}$: binary integer variable which indicates if the modulation level b_j is chosen for message m_i.
- T_i: period (equal to deadline) of message m_i.
- $\delta_{i,j}$: transmission latency of m_i at modulation level b_j and it is calculated as,

$$\delta_{i,j} = \frac{l_i}{Wb_j} \tag{3.11}$$

The following formulation accepts a set of periodic messages and outputs a modulation level for each message, with the objective of minimizing the overall energy consumption while satisfying the deadline and reliability constraints:

Minimize

$$\sum_{i=1}^{n} \sum_{j=0}^{N} \lceil \frac{T_o}{T_i} \rceil E_{ij} x_{ij} \tag{3.12}$$

Subject to:

$$\sum_{i=1}^{n} \frac{\sum_{j=0}^{N} \delta_{ij} x_{ij}}{T_i} \leq 1 \tag{3.13}$$

$$\sum_{j=0}^{N} Rel_{ij} x_{ij} \geq R, \forall i \in [1, n] \tag{3.14}$$

$$\sum_{j=0}^{N} x_{ij} = 1, \forall i \in [1, n] \tag{3.15}$$

$$x_{ij} \in \{0, 1\}, \forall i, j \tag{3.16}$$

The objective function in equation (3.12) represents the total energy consumed by the message set in a time window of $[0, T_o]$. Here, the term $\lceil \frac{T_o}{T_i} \rceil$ denotes the number of instances of message m_i that arrive in the window $[0, T_o]$. The constraint in equation (3.13) ensures that the message set remains schedulable under EDF. Equation (3.14) represents the reliability constraint of the individual messages. The constraint in equation (3.15) ensures only one modulation level is chosen for each message m_i.

In the above problem formulation with single transmission reliability model, the reliability constraint in equation 3.14 can be absorbed into the objective function. The following discussion presents the details. In order to guarantee the required reliability R for each message transmitted at a modulation level b_j, we should have $Rel_{ij} \geq R$. Further, an optimal energy minimization strategy would limit Rel_{ij} to R as providing higher reliability is not necessary and would incur more energy. Therefore in the optimal solution we have, $Rel_{ij} = R$. Consequently from equation 3.9 we have, $SER = 1 - R^{\frac{b_j}{l_i}}$. Further, assuming $SER \approx BER$, the reliability constraint in equation 3.14 can be accommodated into the problem statement as follows:

$$E_{ij} = (d_i)^2 \frac{l_i(2_j^b - 1)}{6b_j} \cdot \frac{N_0}{1 - R^{\frac{b_j}{l_i}}} + E_{tc} + E_{rc} \tag{3.17}$$

Since the reliability constraint is accounted into the objective function of the ILP formulation, equation 3.14 can be ignored while using equation 3.17 for E_{ij}.

3.3.2 Problem complexity

The scheduling problem with deadline and reliability constraints is NP-Hard. In this section, we prove this by reducing the well-known Multiple choice knapsack problem [72] to the problem at hand.

Multiple Choice Knapsack Problem (MCKP): Given λ classes $N_1, ...N_\lambda$ of items to pack in a knapsack of capacity c. Each item $O_{ij} \in N_i$ has a profit p_{ij} and weight w_{ij}. The problem is to choose one item from each class such that the profit sum is maximized without having the weight sum to exceed c.

The MCKP can be mathematically expressed as follows:

Maximize

$$\sum_{i=1}^{\lambda} \sum_{j \in N_i} p_{ij} x_{ij}$$

Subject to:

$$\sum_{i=1}^{\lambda} \sum_{j \in N_i} w_{ij} x_{ij} \leq c$$

$$\sum_{j \in N_i} x_{ij} = 1$$

$$x_{ij} \in \{0, 1\}$$

Theorem: The message scheduling problem with deadline and reliability constraints is NP-hard.

Proof: For the convenience of reduction, we rewrite the maximization objective of the MCKP as the following minimization objective.

$$\sum_{i=1}^{\lambda} \sum_{j \in N_i} (1 - p_{ij}) x_{ij}$$

Now, the MCKP has a one to one correspondence with the scheduling problem defined in equations 3.12-3.16 and can be reduced to an instance where all messages have a single period (and deadline) equal to T_o. Create a message m_i for each of the classes N_i. For each object O_{ij} in N_i create a modulation level b_j. Further, assign an energy consumption of $E_{ij} = (1 - p_{ij})$ and a latency $\delta_{ij} = w_{ij}$ to message m_i when transmitted at modulation level b_j. Finally, set $T_0 = c$. This reduction takes $O(\sum_{i=1}^{\lambda}(\mid N_i \mid)k)$ amount of time which is clearly a polynomial of the problem size. This implies that if the scheduling problem can be solved in polynomial time, then the MCKP can also be solved in polynomial time. However, it is known that the MCKP problem is NP-hard [72]. Therefore, the scheduling problem is also NP-hard.

3.4 Continuous Problem Formulation

Although the ILP formulation presented in the previous section provides an optimal solution, often times solving an ILP can demand excessively high processing and memory resources which makes it impractical to use for reasonably large message sets. Therefore, in this section we derive upper and lower bounds on the energy consumption of the optimal solution which can be obtained quickly. The basic idea is to formulate the scheduling problem as a convex optimization problem assuming the modulation levels of the individual messages can be varied continuously. A solution to this problem can be obtained in polynomial time and this represents a lower bound on the optimal solution of the original problem with discrete modulation levels. In the following, we drop the expressions E_{tc} and E_{rc} from discussion for better readability. Please note that the analysis however can be carried out with them in place.

The continuous case of the scheduling problem can be written as,

Minimize:

$$f(\vec{b}) = \sum_{i=1}^{n} \lceil \frac{T_o}{T_i} \rceil (\frac{2^{b_i} - 1}{b_i} \frac{N_0 d_i^2}{(1 - R^{\frac{b_i}{L}})}) \frac{l_i}{6} \tag{3.18}$$

Subject to:

$$g(\vec{b}) = \sum_{i=1}^{n} \frac{L}{WT_i} \frac{1}{b_i} \leq 1 \tag{3.19}$$

$$b_{min} \leq b_i, \forall i \in [1, n] \tag{3.20}$$

$$b_i \leq b_{max}, \forall i \in [1, n] \tag{3.21}$$

Here, b_i is the variable denoting modulation level of message m_i. The problem is to determine the values for each b_i with the objective of minimizing the total energy consumption while meeting all the deadlines. Where b_i varies continuously in the range $[b_{min}, b_{max}]$. We define the following symbols, $k_i = \lceil \frac{T_o}{T_i} \rceil N_0 d_i^2 \frac{l_i}{6}$ and $k_i' = \frac{L}{WT_i}$.

Now, the objective function can be written as, $f(\vec{b}) = \sum_{i=1}^{n} f_i$ and the constraint can be written as $g(\vec{b}) = \sum_{i=1}^{n} g_i < 1$ where $f_i = k_i(\frac{2^{b_i}-1}{b_i}\frac{1}{(1-R^{\frac{b_j}{l_i}})})$ and $g_i = \frac{k_i'}{b_i}$. Figure 3.1 plots f_i as a function of b_i. Clearly, f_i is a strict convex function of b_i. Therefore, $f(\vec{b})$ which is sum of n identical strict convex functions is also a strict convex function. Further, each g_i is a strict convex function of b_i as $\frac{\partial^2 g_i}{\partial b_i^2} > 0$. Therefore, $g(\vec{b})$ is also a strict convex function. This makes the problem defined in equations 3.18 to 3.20 a convex optimization problem which can be solved using standard Lagrangian multipliers technique.

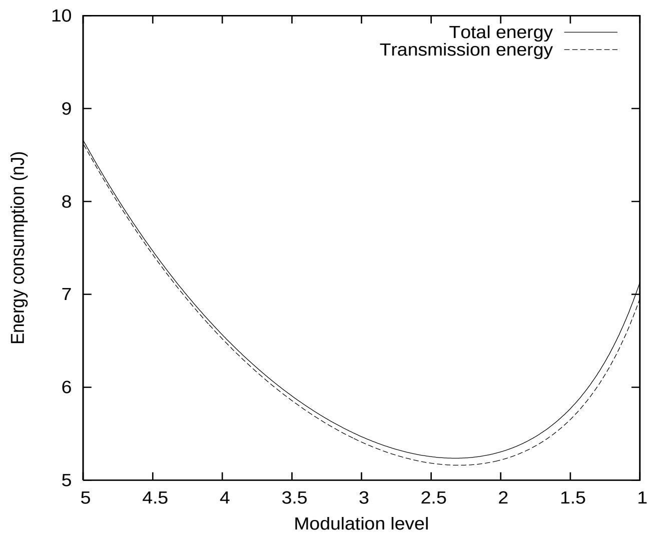


Figure 3.1 Transmission energy: f_i, Total energy: $f_i + E_{tc} + E_{rc}$

Optimal Solution for the continuous Problem: The dual of the above defined problem can be written as:

$$F(\vec{b}) = \sum_{i=1}^{n} f_i + \lambda(\sum_{i=1}^{n} g_i - 1) + \sum_{i=1}^{n} \mu_i(b_i - b_{max}) + \sum_{i=1}^{n} \gamma_i(b_{min} - b_i) \qquad (3.22)$$

where $\lambda \geq 0$, $\mu_i \geq 0$ and $\gamma_i \geq 0$.

An optimal solution to the above dual problem need to satisfy the following *K.K.T conditions* [73]:

- $\frac{\partial F}{\partial b_i} = 0, \forall i \in [1, n]$
- $\lambda(\sum_{i=1}^{n} g_i - 1) = 0$

- $\mu_i(b_i - b_{max}) = 0, \forall i \in [1, n]$
- $\gamma_i(b_{min} - b_i) = 0, \forall i \in [1, n]$

In order to evaluate the first K.K.T condition we perform the following simplification. Let $\overline{R} = (1 - R)$. Then we have, $f_i = k_i(\frac{2^{b_i}-1}{b_i}\frac{1}{1-(1-\overline{R})^{\frac{b_i}{l_i}}})$ which can be simplified to $f_i = k_i(\frac{2^{b_i}-1}{b_i}\frac{l_i}{\overline{R}b_i})$. With this simplification the first K.K.T condition transforms into the following condition.

$$\frac{[b_i^2(2^{b_i}ln2) - (2^{b_i} - 1)2b_i]}{b_i^2} + \frac{(\mu_i - \gamma_i)b_i^2\overline{R}}{k_i l_i} = \frac{\lambda k_i' \overline{R}}{k_i l_i} \tag{3.23}$$

In addition to the above K.K.T conditions, we use the following condition which is satisfied by the optimal solution for most of the problem instances.

$$\sum_{i=1}^{n} \frac{k_i'}{b_i} = 1 \tag{3.24}$$

This is because, if $\sum_{i=1}^{n} \frac{k_i'}{b_i} < 1$ there is more room in the schedule to perform further modulation scaling and hence an optimal solution would always satisfy equation 3.24. However, when the initial system utilization is extremely low leaving a huge room for modulation scaling (for example say, every message can be transmitted at lowest modulation level) reducing modulation levels of each message beyond a certain level (level 2 in figure 3.1) is not beneficial. In such scenarios, the optimal solution might have $\sum_{i=1}^{n} \frac{k_i'}{b_i} < 1$. In this analysis, we ignore those cases for simplicity and proceed with condition in equation 3.24.

Obtaining a closed form for b_i from the above set of equations is rather difficult. However, b_i can be solved numerically using standard numerical analysis techniques. In this chapter, we applied the Newton Raphson method to numerically solve for b_i for each given problem instance. The details of the numerical algorithm are omitted due to space limitations.

In the rest of the chapter, we refer to the above solution methodology as *continuous algorithm* and denote the obtained solution as $\{lb_1, ..lb_n\}$ where lb_i denotes the modulation level determined by the continuous algorithm for message m_i.

Now the above obtained solution can be approximated to obtain a discrete solution following the *rounding algorithm* as follows, $ub_i = discrete(lb_i)$ where the function *discrete* returns the smallest supported discrete modulation level which is greater than lb_i. The solution obtained from the rounding algorithm is not optimal for the discrete version of the problem however, it represents an upper bound of the optimal solution.

3.5 Energy-Aware Heuristic Scheduling Algorithm

In this section we present an efficient heuristic scheduling algorithm for the discrete version of the scheduling problem. Consider a simple example with two periodic messages m_1 and m_2 with periods (equal to deadlines) $T_1 = 256ms$ and $T_2 = 512ms$. Let the normalized source-destination distances of the two messages be $d_1 = 0.8$ and $d_2 = 1.0$. Further, let the time window of interest be $[0, 512]$ i.e, $T_o = 512ms$ and assume $R = 0.99$. In this example, we assume all integers in the interval $[5, 10]$ are valid modulation levels.

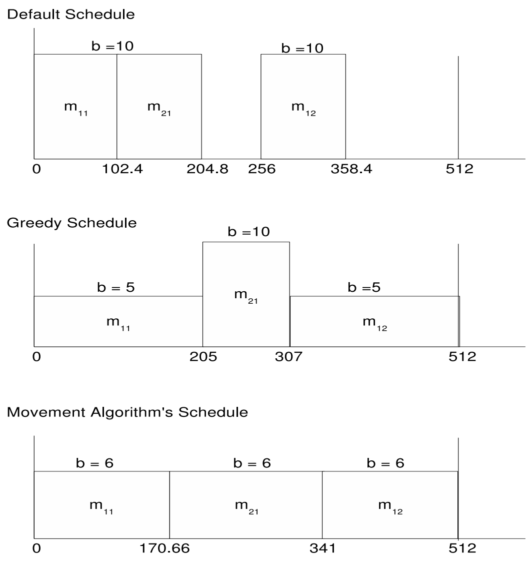


Figure 3.2 Illustrative example: schedules obtained by different algorithms

The default schedule of the two messages when transmitted at the highest modulation level $b = 10$ is shown in figure 3.2. The first and second instances of the message m_1 are denoted as m_{11} and m_{12}, respectively. The energy consumption values of the two messages can be calculated using equation 3.17 to obtain $E_1 = 91.08\mu J$ and $E_2 = 71.16\mu J$ considering all the instances upto time $512ms$. Therefore, the total energy consumption of this schedule is $162.24\mu J$. As shown in the default schedule, a lot of unused time is left (slack) in the schedule which can be utilized to reduce the modulation levels of different messages. The objective of the energy-aware scheduling algorithm is to allocate such available slack to different messages to effectively reduce the total energy consumption of the schedule without violating the deadline constraints.

Now, consider a *greedy scheduling* strategy where all the available slack is assigned to the highest energy consuming message. Applying such a strategy to the example at hand results in the greedy schedule shown in figure 3.2 where all the slack is allocated to m_1. The energy consumption of m_1 in the greedy schedule is, $E_1 = 11.040\mu J$ resulting in a total energy consumption of $82.20\mu J$ which corresponds to an improvement of about 50% over the default approach.

Although the above greedy approach results in impressive energy savings for the example at hand, it is in general not the best slack allocation approach. This is due to the following reason. For each message, as we increase its slack allocation the energy consumption of the message transmission reduces as it can now be transmitted at a lower modulation level. However, as we further increase its slack allocation the subsequent energy reductions offered by that message decrease due to convex relation between energy consumption and the modulation level. In order to succinctly represent this characteristic, we define a metric called *Energy Gain* for each message, m_i as follows:

$$Gain_{ij} = \lceil \frac{T_o}{T_i} \rceil (E_{ij} - E_{ij-1}) \tag{3.25}$$

The energy gain, $Gain_{ij}$ denotes the amount of energy reduction that would be obtained by transmitting the message m_i at a modulation level b_{j-1} as opposed to b_j. Figure 3.3 plots the energy gain values for the messages m_1 and m_2 as a function of the modulation levels, b_j. The following two observations can be made from this graph. First, as we decrease the modulation level of a message the corresponding energy gain decreases. For example, $Gain_{1,10} > Gain_{1,9}$

(in figure 3.3,$G_1 > G_3$) . Second, the energy gains obtained by the largest energy consuming message are not always the highest. For example, $Gain_{1,9} < Gain_{2,10}$ (in figure 3.3,$G_3 < G_2$). This observation proves that the above greedy scheduling algorithm is not the best strategy.

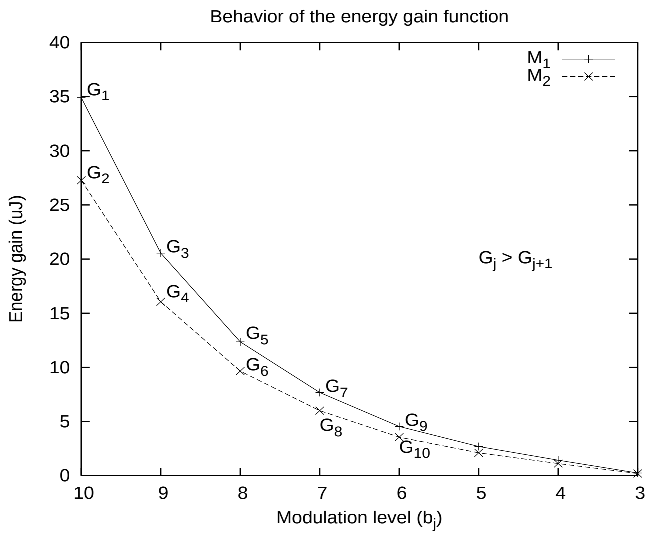


Figure 3.3 Energy gain function, $Gain_{ij}$ versus b_j

3.5.1 Movement Algorithm

Based on the above intuition, we now present our movement algorithm which allocates slack to different messages incrementally, keeping track of their decreasing energy gain values. The algorithm uses a simple data structure called the *movement table* which varies the modulation levels across columns and contains different messages along the rows. Further, each message is associated with a row in the table. As a result, the table has N columns and n rows. At any given point of time, each message occupies a unique cell in its row. If a message is placed in the column labeled b_j it means that it has been assigned a modulation level of b_j. The movement table for the example is shown in figure 3.4.

When the movement algorithm begins, each message is placed in a cell corresponding to its initial modulation level which is an input to the algorithm. By default, each messages is assigned an initial modulation level of b_{max} which corresponds to column one (leftmost column) in the movement table. Therefore, all the messages are placed in column one by default and

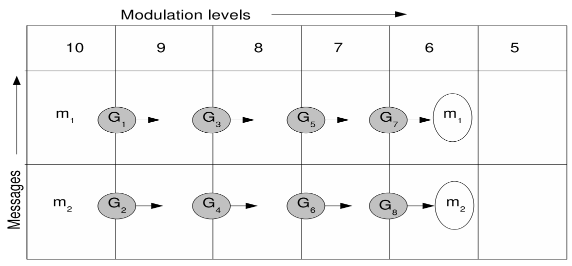


Figure 3.4 Working of the Movement Algorithm

the rest of the entries are left empty in the table.

The objective of the movement algorithm is to *move* as many messages as possible to the right without violating the deadline constraints. Each movement in the table corresponds to an incremental slack allocation i.e, by moving a message m_i from column b_j to b_{j-1}, the movement algorithm allocates enough slack to m_i such that it can now be transmitted at a modulation level of b_{j-1} instead of b_j. In order to ensure that such movement based slack allocation preserves deadline guarantees, the schedulability test is verified before every move in the algorithm. The pseudocode is presented in Algorithm 3. In the pseudocode, α_i denotes the current modulation level of message m_i and $next(\alpha_i)$ denotes the highest modulation level that satisfies $\alpha_i > next(\alpha_i)$.

The movement algorithm works as follows. In steps 1-7, the algorithm initializes the movement table and different parameters. The algorithm also maintains a set (Q) consisting of all the potential messages which can utilize more slack without violating the deadline constraints. In steps $8 - 20$, the algorithm iterates over the messages in Q and attempts to move each message one column to the right in the movement table. In step 9, the highest energy gain yielding message is selected. The following step ensures that the selected message has a positive energy gain. In step 12, the schedulability test is verified to check if reducing the modulation level of m_i would lead to any deadline violations. If the above assignment violates the schedulability test, it is removed from the set Q. On the other hand if the test is satisfied, the modulation level of the message is actually reduced by one level in step 16 and the message is moved one column to the right in the movement table in the following step. Step 18 updates the utilization accordingly. Finally, if the message is assigned the minimum modulation level in this iteration it is removed from Q in step 19 as its modulation level cannot be further reduced.

The computational complexity of the movement algorithm can be analyzed with the help of the movement table. Each message can move at most N columns to the right and for each column movement, the maximum gain yielding message is determined in linear time. Therefore, the time taken by a single message is at most $O(nN)$ and the total time taken by all the messages in the worst case is $O(n^2N)$.

```
Input: set of n messages with their modulation levels, $init\_b_i$
Output: Slack allocation for each message
1  Set $U = 0$;
2  for $i = 1; i \leq n; i++$ do
3      Set $\alpha_i = init\_b_i$;
4      Place $m_i$ in the column labeled $\alpha_i$ in the movement table;
5      Add $m_i$ to set $Q$;
6      Set $U = U + \frac{L}{W\alpha_i T_i}$
7  end
8  while $Q \neq \emptyset$ do
9      Pick up the message $m_i$ with highest $Gain_{i,next(\alpha_i)}$ from $Q$;
10     if($Gain_{i,next(\alpha_i)} \leq 0$) break;
11     Set $U' = U - \frac{L}{W\alpha_i T_i} + \frac{L}{W next(\alpha_i) T_i}$;
12     if $U' > 1$ then
13         Remove $m_i$ from $Q$ ;
14         Continue;
15     end
16     Set $\alpha_i = next(\alpha_i)$;
17     Move message $m_i$ one column to the right;
18     Set $U = U'$;
19     if($\alpha_i == b_{min}$) Remove $m_i$ from $Q$;
20 end
```

Algorithm 3: Movement Algorithm

Applying the movement algorithm to the example (see figure 3.4), all the messages are initially placed in column one of the movement table. In the first iteration, message m_1 is selected as it yields the maximum energy gain, G_1 (see figure 3.3) and is moved one column to the right. In the next iteration, m_2 is moved right by one column as $G_2 > G_3$. Following similar procedure, both the messages finally reach the column labeled 6. The intermediate movements are shown as arrows in the figure 3.4 these movements are performed in the following order: $\{G_1, G_2, G_3, G_4, G_5, G_6, G_7, G_8\}$. The algorithm terminates at this point as no more slack is available. Figures 3.2 shows the resulting schedule where both m_1 and m_2 operate at $b = 6$ incurring a total energy consumption of $27.75\mu J$. This corresponds to an improvement of 82% over the default schedule and 66% over the greedy schedule.

3.6 Performance Evaluation

We performed comprehensive performance evaluation studies to compare the relative performance of the following scheduling algorithms:

- ILP
- Continuous algorithm (labeled as *Lower bound*)
- Rounding algorithm (labeled as *Upper bound*)
- Movement algorithm with default modulation levels as input (labeled as *Mov-def*).
- Movement algorithm with the output of the rounding algorithm as the input (labeled as *Mov-ub*).

The performance metric is the normalized energy consumption where all energies are normalized with respect to the non-DMS aware strategy which transmits each message at the highest modulation level, b_{max}. We solved the ILP using the ILOG CPLEX 10.100 software [74].

We conducted two sets of simulation studies. In each simulation set, we considered a square region of size $500 * 500$ square meters where the nodes are randomly distributed following a specific distribution. We used the following parameters: $R = 0.99, N_0 = 4 * 10^{-13}, c_t = 75nJ, c_r = 100nJ, W = 1MHz, l_i = 1024, \forall i$. For each simulation run, we generated 30 periodic messages with randomly chosen source and destination nodes. The total energy consumption of each schedule was calculated in the time window $[0, LCM]$, where LCM is defined as the least common multiple of the message periods. In our simulations, for each parameter set of interest, we performed 20 different runs each with a different random number seed and the obtained average is plotted as a single point in the graph.

In simulation set one, we followed the uniform distribution to randomly generate node locations in the network. We varied the following parameters: system utilization, $U = \sum_{i=1}^{n} \frac{l_i}{Wb_{max}T_i}$ and N, the number of discrete modulation levels allowed to compare the performance of the proposed schemes. The N modulation levels are equally spaced integers (as much as possible) in the interval $[1, 10]$. We considered a network size of 50 nodes.

In the set two, we followed a clustered node distribution. The entire network consisted of v_c clusters each with a radius r_c. The cluster locations are randomly chosen following a uniform distribution. The total number of nodes are equally divided among the v_c clusters. Within each cluster, the nodes assigned to it are randomly distributed following a uniform distribution. In our simulation studies we varied the v_c and r_c parameters. Please note that the communication under this model is still single-hop in nature and we have chosen the above clustered distribution to study the effect of non-uniformity in node locations in the network.

3.6.1 Results for uniform node distribution

3.6.1.1 Effect of utilization(U)

Figure 3.5, shows the relative performance of the above schemes. As the utilization increases, the total energy consumption also increases due to the increased workload. All the schemes show this trend. At low values of U, there is ample slack available and all most all the messages can be transmitted at low modulation levels. As a result the gap between the schemes is less in the graph. Similarly, at $U = 1.0$ there is no slack available to perform any dynamic modulation scaling, as a result all the schemes behave like the default non-DMS-aware scheme. Throughout the range, both *Mov-def* and *Mov-ub* perform significantly better than the upper-bound and moreover, both of them show a performance very close to that achieved by the ILP. At $U = 0.6$, *Mov-ub* shows an improvement of 13% over the upper-bound and incurs as less as 1.5% and 5% energy over the ILP and lower-bound, respectively. Similarly, at $U = 0.90$, it shows an improvement of 16% over the upper-bound and incurs as less as 1.2%

energy over the ILP.

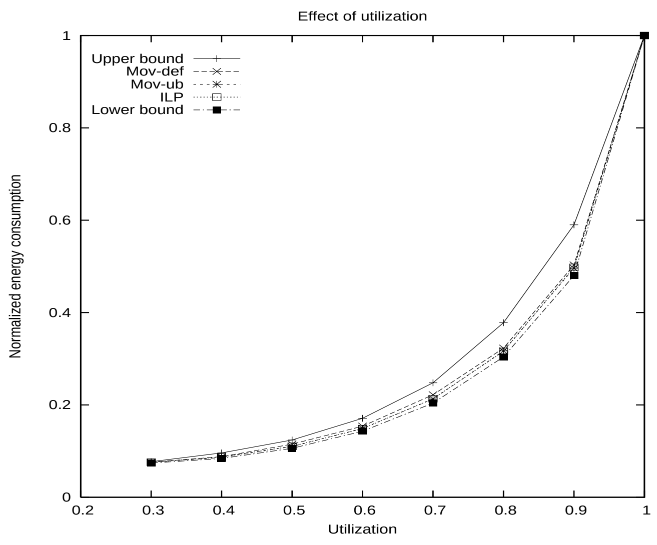


Figure 3.5 Effect of utilization ($N = 10$)

3.6.1.2 Effect of number of modulation levels (N)

Figure 3.6 compares the relative performance of the schemes by varying the number of discrete modulation levels supported by the communication hardware. As N increases, all the schemes get closer and closer to the lower-bound (i.e, the solution to the continuous problem). At low N, the upper-bound turns out to be a very course grained approximation of the continuous solution leaving a lot of room for improvement. As a result at $N = 3$, both *Mov-def* and *Mov-ub* show significant improvements over the upper-bound while at $N = 10$, they get closer to the upper bound. Specifically, at $N = 3$, *Mov-ub* shows an improvement of 45% over the upper-bound while incurring less than 1% energy over the ILP. Similarly at $N = 7$, it shows an improvement of 22% over the upper-bound while incurring as less as 1.5% over the ILP.

3.6.2 Results for clustered node distribution

3.6.2.1 Effect of number of clusters(v_c)

Figure 3.7 compares the performance of the above schemes by varying the number of clusters in the network (v_c). As the v_c increases, there are more clusters in the network and hence the

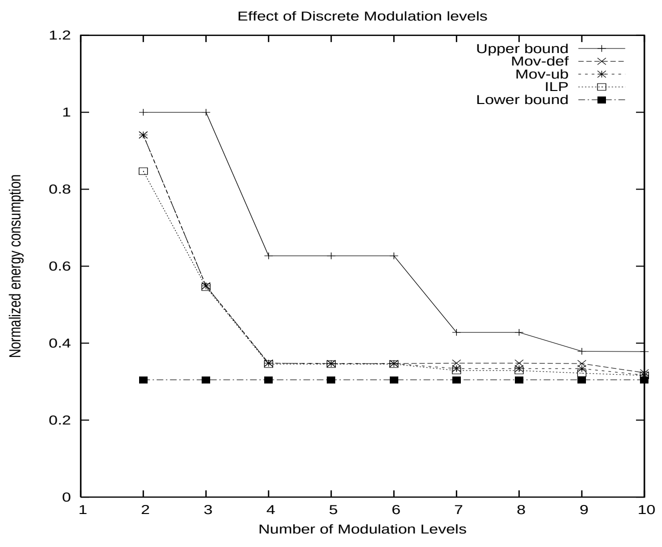


Figure 3.6 Effect of number of modulation levels($U = 0.8$)

nodes are more widely spread increasing the average distance of each randomly chosen source-destination pair. Consequently, the energy of all the schemes increase with v_c. However, as we further increase v_c, the spreading effect decreases consequently the energy consumption values stabilize for all the schemes. Throughout, *Mov-ub* performs better than upper-bound showing an average improvement of 15% and on an average the incurs as less as 1% energy compared to the ILP. The *Mov-def* also shows similar energy savings incurring slightly higher energy than *Mov-ub*.

3.6.2.2 Effect of cluster radius(r_c)

Figure 3.8 compares the performance of the above schemes by varying the normalized cluster radius(r_c). Where the cluster radii are normalized with respect to the diagonal (longest possible source-destination distance in the network) of the square region. As the r_c increases, each cluster expands and its nodes become more widely spread increasing the average distance of each randomly chosen source-destination pair. Consequently, the energy of all the schemes increase with r_c. Throughout, both the movement algorithms perform better than upper-bound showing an average improvement of 15% and on an average they incur as less as 3% energy compared to the ILP.

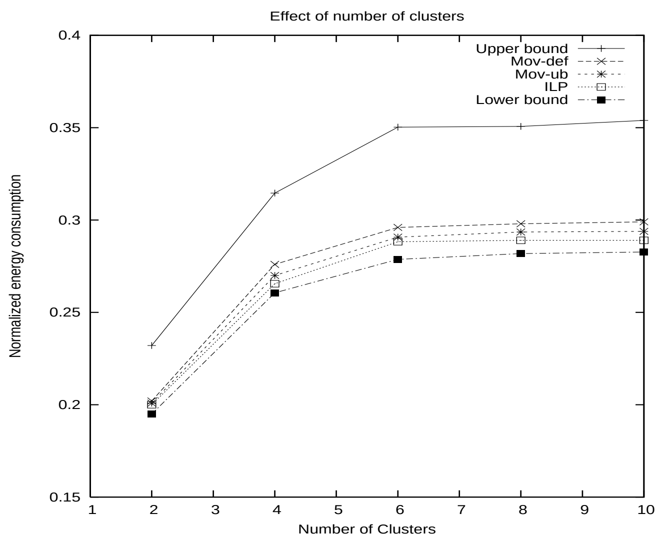


Figure 3.7 Effect of number of clusters ($N = 10, U = 0.8, r_c = 0.10$)

3.7 Discussion

In this chapter, we addressed the problem of scheduling a set of periodic real-time messages in a shared medium wireless network with the objective of minimizing the total energy consumption while meeting the required deadline and reliability constraints. We employed the dynamic modulation scaling (DMS) technique as the basic underlying power management mechanism. We first presented an ILP formulation for the discrete version of the problem. Since solving an ILP can take exponential amount of time, we derived lower and upper bounds by solving the continuous version of the problem using Lagrangian Multiplier technique. Finally, we presented an efficient polynomial time heuristic scheduling algorithm. Our simulation studies show that the proposed heuristic algorithm achieves significant improvements over the derived upper-bound and it offers comparable energy savings to that obtained by solving the ILP. In our future work, we plan to address the same problem for a different reliability model where the required message reliability is achieved via retransmissions instead of a single message transmission with high transmission power. Further, we plan to extend our work presented in this chapter to a multi-hop network setup.

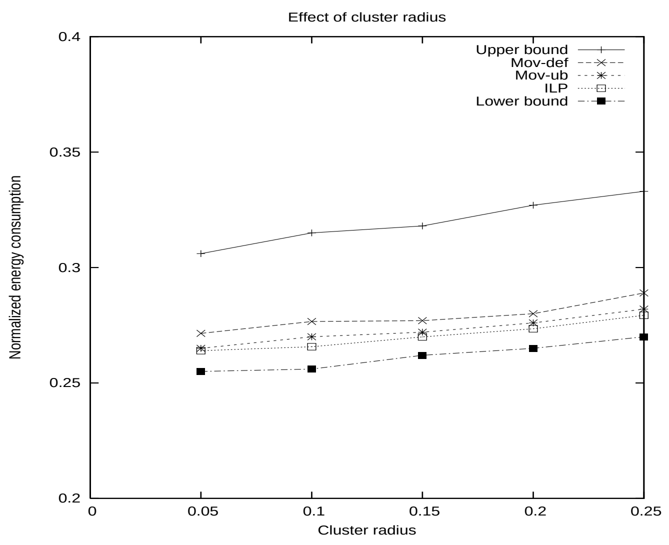


Figure 3.8 Effect of cluster radius ($N = 10, U = 0.8, v_c = 4$)

CHAPTER 4. Energy management in communication system using power adaptation

In this chapter, we address the communication energy management problem by employing power adaptation technique as the underlying energy management mechanism. First, we quickly review the related work and then present the problem statement, proposed solutions, and results.

4.1 Related work and motivation

Several energy-aware protocols have been proposed to address the challenge of providing deterministic and reliable real-time services over the wireless networks. These protocols can be broadly classified into two classes based on the kind of service they provide:

- *Throughput oriented*: This class of protocols exploit the tradeoff between energy consumption and network throughput. In [83], a protocol framework has been proposed for multi-hop wireless networks to maximize the throughput while keeping the energy costs low. These protocols primarily target different non-real-time applications where timely message delivery is not of paramount importance.

- *Soft real-time*: This category of protocols try to minimize either the latency or deadline violations in a best effort manner. The protocols which aim at minimizing the latency [84, 85] cannot provide any kind of guarantees and are unsuitable for real-time applications. On the other hand, the existing protocols which try to minimize the deadline violations either ignore the time varying nature of the wireless channel or try to minimize the deadline violations in a best effort manner. In [87], a centralized real-time MAC (Medium

Access Control) protocol is presented which works over the Hybrid Coordination Function (HCF) protocol of IEEE 802.11 with the aim of facilitating long sleep durations for the individual nodes. The authors proposed a non-preemptive EDF (Earliest Deadline First) based algorithm for message scheduling in the MAC layer assuming a non-time-varying channel. In [88], an energy aware data gathering protocol is presented for wireless sensor networks which uses dynamic modulation scaling technique to minimize the energy consumption while meeting the deadlines. Both the above protocols do not provide any per message guarantees and when applied to a typical time varying channel, the proposed algorithms might lead to unpredictable number of deadline violations.

Some protocols provide more precise guarantees like minimum deadline success ratio which is defined as the fraction of the input messages that are guaranteed to meet the deadline. In [89], the problem of scheduling multiple video streams has been considered where each stream is guaranteed a minimum deadline success ratio while minimizing the total energy consumption. These protocols are suitable for multimedia applications where occasional deadline violations are tolerable; however, they cannot provide per message guarantees and hence cannot be directly applied to real-time applications.

Providing hard deadline guarantees in a wireless environment is practically infeasible due to the time varying and lossy nature of the wireless channel [92]. In other words, in a wireless environment, it is not possible to guarantee that a given message can be successfully transmitted before a specified deadline with 100% probability. Therefore, allocating certain amount of resources (e.g. time slots) to a deadline constrained message ignoring the channel conditions may not always result in successful and timely message delivery. Moreover, the probability of timely message delivery increases with the amount of resources allocated to the message. Therefore, resource allocation should be done both based on the required probability of meeting the message deadlines as well as considering the nature of the channel.

In this chapter, we target hard real-time applications like industrial automation and military surveillance where each message needs to meet its deadline with a very high probability e.g, $(1 - 10^{-5})$. Providing such high success probabilities establishes a tremendous degree of determinism in meeting the message deadlines which is critical for the targeted applications.

With this motivation, for every message, we consider a minimum success probability constraint along with the conventional deadline (latency) constraint. In the rest of the chapter, we refer to the probability of meeting the deadline as reliability. Therefore, if a periodic message, m_i has a reliability constraint of Rel_t and a deadline of T_i, this means that every instance of m_i should reach the destination before the deadline T_i with a minimum probability of Rel_t.

Energy consumption becomes a critical factor when trying to provide such hard guarantees on a per message basis. Over-allocating resources to attain high reliabilities can result in intolerably high energy consumptions which can subsequently bring the network down. Therefore, the three parameters: deadline, reliability and energy consumption should be considered together in order to provide hard guarantees over a wireless network, which is the focus of the chapter. The protocols presented in [85] consider the three parameters together. However, they provide latency and reliability guarantees in a best effort manner and hence are unsuitable for hard real-time applications.

4.2 Problem Statement

Our basic approach is as follows: Instead of transmitting a message directly from the source to destination in one hop, it is possible to reduce the energy consumption if the message is transmitted via multiple hops with each hop being smaller than the direct hop. Most of the existing work [93, 94] which uses such hop-by-hop transmissions primarily addressed energy aware throughput optimization problems which cannot be directly used for the scheduling of messages with deadline and reliability constraints.

The strategy of transmitting messages via multiple smaller hops would reduce the energy consumption while incurring more time as each hop communication would occupy an entire time slot in the shared wireless medium. Further, such a strategy would also reduce the message reliability.

Problem Statement: Considering these tradeoffs, the objective of this chapter is to design efficient message scheduling algorithms which determine a hop-by-hop path for each input mes-

sage with the goal of minimizing the overall energy consumption while providing per message deadline and reliability guarantees.

The rest of the chapter is organized as follows: In section 4.3 we present our system model followed by the problem statement. We also prove that the problem is NP-Hard. In section 4.4, we present an ILP formulation for the scheduling problem. In section 4.5, we present a motivational example followed two heuristic scheduling algorithms. We present our simulation results and conclusions in sections 4.6 and 4.7 respectively.

4.3 System Model and Problem Formulation

4.3.1 System Model

4.3.1.1 Network Model

We consider a wireless network with v nodes which share the common wireless medium that is accessed in an exclusive manner. A source can communicate with a destination either directly or by making several hop-by-hop transmissions through intermediate nodes. We assume that the shared wireless medium is slotted in time with a fixed slot size. All messages are of equal length, each consisting of L bits and incur a single time slot for a transmission irrespective of the source and destination.

4.3.1.2 Channel Model

Although all the nodes in the network are in the same interference range and share the wireless medium, each channel connecting a pair of nodes behaves independent of the other. In this chapter, we assume that each channel independently follows the Rayleigh fading model and has a bandwidth of W Hertz.

Assuming a Quadrature Amplitude Modulation (QAM) scheme, the SNR (signal-to-noise ratio) achieved at the receiver when receiving a message of L bits with energy E_{recv} can be calculated as:

$$SNR = \frac{log_2 M E_{recv}}{L N_0} \tag{4.1}$$

where M and N_0 denote the modulation level and the noise power respectively. For a Rayleigh fading channel with high SNR the BER (bit error rate) can be estimated as:

$$BER \approx \frac{2(M-1)}{3} \frac{1}{SNR} \tag{4.2}$$

In order to support a minimum SNR (or maximum BER) at the receiver the signal should be transmitted with an energy equal to E_{ts} given by:

$$E_{ts} = E_{recv} d^2 \tag{4.3}$$

where d denotes the normalized source destination distance. Using equations (4.1-4.3) the expression for E_{ts} can be derived as follows:

$$E_{ts} = \frac{2L}{3} \frac{M-1}{log_2 M} \frac{N_0}{BER} d^2 = C d^2 \tag{4.4}$$

where, C is defined as: $C = \frac{2L}{3} \frac{M-1}{log_2 M} \frac{N_0}{BER}$

Similarly, the time taken for transmitting a L-bit message (which is also equal to the slot size) over a channel is given by:

$$T = \frac{L}{W log_2 M}$$

In the rest of the chapter, we assume that the parameters: L, W, N_0, BER and M (and hence C) remain fixed across different channels and transmissions.

4.3.1.3 Energy Model

- *Transmission energy:* Energy consumed in transmitting a message of L bits consists of the following two components: the energy provided by the power amplifier to the electromagnetic signals (E_{ts}, in eq (4.4)) carrying the message and the energy consumed by the electronic circuitry (E_{tc})[65]. Therefore, the total transmission energy E_t is given by:

$$E_t = E_{ts} + E_{tc} \tag{4.5}$$

where,

$$E_{tc} = \frac{LC_t}{log_2 M} \tag{4.6}$$

- *Reception energy:* The total energy consumed in receiving a message of L bits is equal to the energy consumed by the electronic circuitry (E_{rc}) and is given by [65]:

$$E_{rc} = \frac{LC_r}{log_2 M} \tag{4.7}$$

Here, C_t and C_r are implementation dependent constants. In the rest of the chapter, we will primarily use equations 4.4-4.7.

4.3.2 Problem Formulation

Consider a set of periodic messages $(m_1, m_2, ..., m_n)$ where each message m_i has a period T_i equal to its deadline. Let $path_{i1}, path_{i2}...path_{ij}...path_{iN_i}$ denote the different possible paths between the source and destination nodes of m_i. The objective of the scheduling problem is to choose one of the N_i paths for each message m_i which minimizes the total energy consumption while guaranteeing both the reliability and deadline constraints. When dealing with periodic messages, the total energy consumption is defined with respect to a time window, say $[0, T_o]$, where T_o is an application specific parameter. For a given T_o, the problem can be mathematically stated as follows:

Minimize

$$\sum_{i=1}^{n}\sum_{j=0}^{N_i}\lceil\frac{T_o}{T_i}\rceil E_{ij}x_{ij} \tag{4.8}$$

Subject to:

$$\sum_{i=1}^{n}\frac{\sum_{j=0}^{N_i}\delta_{ij}x_{ij}}{T_i} \leq 1 \tag{4.9}$$

$$\sum_{j=0}^{N_i} Rel_{ij}x_{ij} \geq R, \forall i \in [0, M] \tag{4.10}$$

$$\sum_{j=0}^{N_i} x_{ij} = 1, \forall i \in [0, M] \tag{4.11}$$

$$x_{ij} \in \{0, 1\}, , \forall i, j \tag{4.12}$$

where, E_{ij} is defined as the total energy consumption incurred by the transmission of m_i over $path_{ij}$; δ_{ij} and Rel_{ij} denote the latency incurred and reliability offered by $path_{ij}$ respectively. The objective function in equation (4.8) represents the total energy consumed by the message set in a time window of $[0, T_o]$. The constraint in equation (4.9) ensures that the message set remains schedulable under a work conserving scheduler like EDF (earliest deadline first). Equation (4.10) represents the per message reliability constraint. The binary variable x_{ij} indicates the selected path for m_i and the constraint in equation (4.11) ensures only one of the N_i paths is chosen for each message m_i.

Message reliabilities can be provided by either performing retransmissions whenever necessary or by performing a single transmission with high enough energy. Each one of the two options presents a different kind of a energy-time tradeoff and they both require different scheduling approaches to solve the above formulated problem. In this chapter, we follow the single transmission model where each message is transmitted just once with high enough energy.

For the single transmission model, the path reliabilities can be calculated from the constituent link reliabilities as follows. Assuming a constant BER across different channels, the reliability of each channel/link is given by:

$$Rel = (1 - BER)^{\frac{L}{log_2 M}}$$

where, channel reliability is defined as the probability that a L-bit message can be successfully transmitted over the channel in one time slot. The reliability of a path consisting of h hops is defined as the probability that a message can be successfully transmitted over the path in h time slots. It can be calculated as: Rel^h. Consequently, the reliability constraint in equation (4.10) gets reduced to a hop constraint as follows:

$$\sum_{j=0}^{N_i} h_{ij} x_{ij} \leq \beta \tag{4.13}$$

where h_{ij} denotes the number of hops in $path_{ij}$ and β denotes the hop constraint obtained as the largest integer h satisfying the relation: $Rel^h \geq Rel_t$.

4.3.3 Problem complexity

The scheduling problem with deadline and reliability constraints is NP-Hard. In this section, we prove this by reducing the well-known Multiple choice knapsack problem [72] to the problem at hand. Please refer to the definition of the MCKP problem presented in section 3.3.2 of chapter 3.

Theorem: The message scheduling problem with deadline and reliability constraints is NP-hard.

Proof: For the convenience of reduction, we rewrite the maximization objective of the MCKP as the following minimization objective.

$$\sum_{i=1}^{\lambda} \sum_{j \in N_i} (1 - p_{ij}) x_{ij}$$

Now, the MCKP has a one to one correspondence with the scheduling problem and can be reduced to an instance where all messages have a single period (and deadline) equal to T_o. Create a message m_i for each of the classes N_i. For each object O_{ij} in N_i create a path $path_{ij}$ with exactly β hops between the source and destination nodes of the message m_i. Further, assign an energy consumption of $E_{ij} = (1 - p_{ij})$ and a latency $\delta_{ij} = w_{ij}$ to $path_{ij}$. This reduction takes $O(\sum_{i=1}^{\lambda}(\mid N_i \mid)k)$ amount of time which is clearly a polynomial of the problem size. This implies that if the scheduling problem can be solved in polynomial time, then the MCKP can also be solved in polynomial time. However, it is known that the MCKP problem is NP-hard [72]. Therefore, the scheduling problem is also NP-hard.

4.4 ILP Formulation

In this section, we present an ILP formulation for the scheduling problem. We use the following symbols in the formulation:

- A: set of edges/hops in the network with v nodes. Since it is a broadcast medium, there are $v^2 - v$ edges in A.
- n: number of input messages.
- $d_{i,j}$: distance between nodes i and j in the network.
- C: constant in equation 4.4.
- T_o: Time window during which the total energy consumption of the message set should be optimized.
- $y_{i,j,k}$: binary integer variable which indicates if the hop (i,j) in the network is chosen as a part of path for message m_k.
- S_k: the source node of message m_k.
- D_k: the destination node of message m_k.
- T_k: period (equal to deadline) of message m_k.
- β: Maximum number of hops allowed per path by the reliability constraint.

The following formulation accepts a set of periodic messages and outputs a path for each message which minimize the overall energy consumption while guaranteeing the deadline and reliability constraints:

Minimize:

$$\sum_{k=1}^{n} \sum_{(i,j)\in A} \lceil \frac{T_o}{T_k} \rceil (C d_{i,j}^2 + E_{tc} + E_{rc}) y_{i,j,k} \tag{4.14}$$

Subject to:

$$\forall k, \sum_{(q,i)\in A} y_{q,i,k} - \sum_{(i,j)\in A} y_{i,j,k} = \begin{cases} \text{-1} & \text{if } i = S_k \\ 0 & \text{if } S_k \neq i \neq D_k \\ 1 & \text{if } i = D_k \end{cases} \tag{4.15}$$

$$\sum_{k=1}^{N} \frac{\sum_{(i,j)\in A} y_{i,j,k}}{T_k} \leq 1 \tag{4.16}$$

$$\sum_{k=1}^{N} \sum_{(i,j)\in A} y_{i,j,k} \leq \beta \tag{4.17}$$

where,

$$y_{i,j,k} \in \{0,1\}, \forall i,j,k \tag{4.18}$$

The objective function in equation 4.14 denotes the total energy consumption of the message set in the time window $[0, T_o]$ and is similar to equation 4.8. The constraint in equation 4.15 ensures a connected path is selected for each message. Equations 4.16 and 4.17 represent the schedulability constraint and the reliability constraint respectively. Equation 4.18 specifies that $y_{i,j,k}$ as a binary integer variable.

Although the ILP model is computationally-intensive to be used for large network sizes and message sets, it is helpful in determining a lower bound on the amount of energy consumed by a given message set.

4.5 Energy-aware Scheduling algorithms

Solving an ILP can demand excessively high computing and memory resources for reasonably large problem sizes. Therefore, in this section we present heuristic scheduling algorithms which provide effective energy reductions in polynomial time.

The proposed energy-aware algorithms employ the EDF (earliest deadline first) as the underlying scheduling policy. Given a set of n periodic messages $(m_1, m_2 ... m_n)$ each having a path latency of t_i and a period of T_i, the task set is schedulable under EDF if and only if the following condition satisfies:

$$U = \sum_{i=1}^{n} \frac{t_i}{T_i} \leq 1 \tag{4.19}$$

where U denotes the total system utilization. We assume all scheduling decisions (message arrivals, preemptions, transmissions etc.) are made at the slot boundaries.

The proposed scheduling algorithms select a path for each input message while ensuring the resulting message latencies do not violate the EDF schedulability and reliability constraints.

In the following, we present a motivational example demonstrating the working of the default direct hop strategy. In the subsequent sections, we use the same example to demonstrate the working of other energy-aware scheduling algorithms.

4.5.1 Motivational example

Consider three periodic messages $m_1(source = A, destination = F)$, $m_2(J, K)$ and $m_3(G, H)$ which start at time zero and have a common period (equal to deadline) of 7 time slots. Further assume that each message needs to meet its deadline with a minimum probability of $(1 - 10^{-5})$ i.e, $Rel_t = (1 - 10^{-15})$. In each time slot, only one message can be transmitted and each message transmission takes an entire slot irrespective of the source and destination. Figure 4.1 shows the network. Table 4.1 lists the normalized inter-node distances of interest.

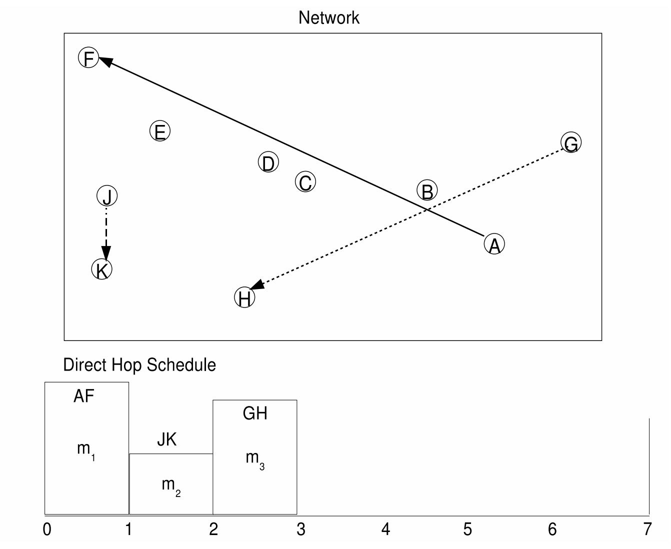


Figure 4.1 Illustrative example: direct hop strategy

Hop (i, j)	**Distance** (d_{ij})	**Hop** (i, j)	**Distance** (d_{ij})
(A, F)	0.755	(A, D)	0.380
(G, H)	0.600	(D, F)	0.376
(J, K)	0.082	(A, C)	0.335
(A, B)	0.123	(B, D)	0.257
(B, C)	0.212	(G, B)	0.213
(C, D)	0.046	(B, C)	0.212
(D, E)	0.217	(B, H)	0.389
(E, F)	0.169	(C, H)	0.216
(C, E)	0.262	(G, A)	0.113

Table 4.1 Motivational example: normalized inter-node distances

L(**bits**)	M	BER	N_o(**nJ**)	C_t(**nJ**)	C_r(**nJ**)
1024	256	10^{-8}	10^{-4}	75	100

Table 4.2 Parameter values [65]

Using the parameter values listed in table 4.2 we get $\beta \leq 7$, $C = 0.21760J$, $E_{tc} = 9.6\mu J$ and $E_{rc} = 12.8\mu J$.

Now consider the direct hop strategy where all messages are scheduled via the direct hops between the source destination nodes. Figure 4.1 depicts this scenario. The corresponding schedule is also shown in the same figure where y-axis denotes the power consumption (proportional to distance square) and the x-axis denotes the time. In the schedule shown, the individual hops used in each time slot are marked against it. Using equations 4.4-4.7, the energy consumption of the direct hop strategy can be calculated as: $E_1 = (C * (d_{AF})^2 + E_{tc} + E_{rc}) + (C * (d_{JK})^2 + E_{tc} + E_{rc}) + (C * (d_{GH})^2 + E_{tc} + E_{rc}) = 203.904mJ$.

4.5.2 Greedy Scheduling Algorithm

The direct hop strategy described above transmits each message directly in one single hop leaving the slack (extra time slots) unused. In the above example, the time slots 3 to 7 are left unused. Such leftover slack can be used to transmit some of the messages via less

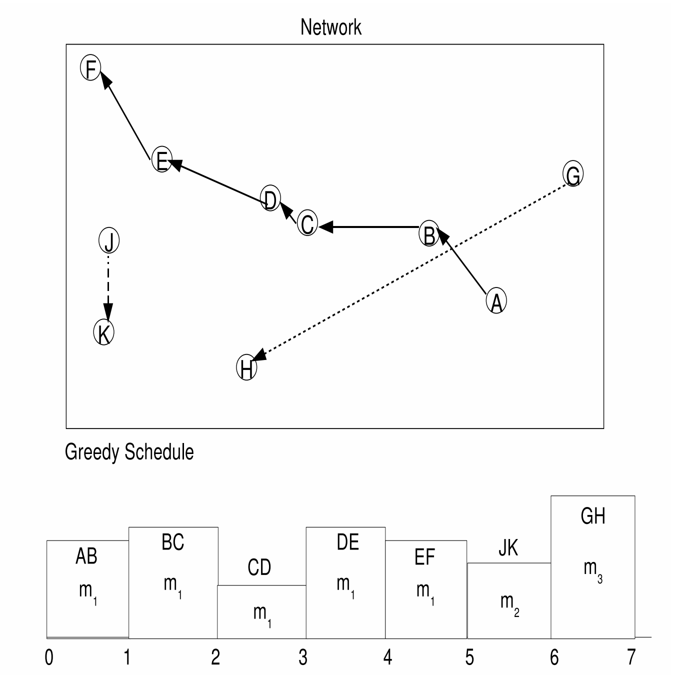


Figure 4.2 Illustrative example: greedy strategy

energy consuming multiple shorter hops instead of one long hop. The goal of an energy-aware scheduling algorithm is to distribute the available slack (which is often scarce) among the input messages with the objective of reducing the total energy consumption while guaranteeing the deadline and reliability constraints.

The greedy algorithm utilizes the available slack to perform hop-by-hop transmissions for as many messages as possible. The basic idea of the algorithm is as follows: it calculates the maximum available slack and assigns it to the highest energy consuming message. Using this slack a new path with shorter hops is selected for that message.

The pseudo-code for the greedy algorithm is presented in Algorithm 4. The algorithm works on an input schedule. By default, the direct hop schedule is assumed as the input schedule. The number of slots allocated to each message m_i is denoted by t_i. It also denotes the number of hops that the message can take in its source-destination path. In steps 3 to 12, the algorithm iterates over all messages choosing the highest energy consuming message first (step 6). In step 7, the total available slots for the chosen message are calculated. In step 9, the available slots are adjusted to ensure that reliability constraints are not violated. In

```
   Input: input schedule $H_i$, hop constraint $\beta$
   Output: Output Schedule, $H_o$
1  Set $i = 0$ and $H_o = H_i$;
2  Set $U = \sum_{j=1}^{N} \frac{t_j}{T_j}$ ;
3  while $i \leq n$ do
4      Set $i = i + 1$;
5      Calculate $s = 1 - U$;
6      Pick up the next message (say $m_i$) with highest energy consumption in
       $H_o$;
7      Calculate available slots $p = \lfloor sT_i \rfloor$;
8      if($p < 1$) continue;
9      Set $p = min(\beta, p + t_i)$;
10     Find the shortest path for $m_i$ with at most $p$ hops; denote the number
       of hops in the shortest path with $q$, where $q \leq p$;
11     Set $U = U - \frac{t_i}{T_i} + \frac{q}{T_i}$;
12     Set $t_i = q$ and update schedule $H_o$ accordingly;
13 end
```

Algorithm 4: Greedy(H_i,β), Greedy Scheduling Algorithm

step 10, the shortest path with at most p hops is selected. During this step, the edge weights are chosen as the corresponding energy consumption values considering all the components as discussed in section 4.3. Finally, the utilization and the schedule are updated in steps 11 and 12, respectively.

Considering the example discussed earlier, the greedy algorithm would choose the message $m_1(A, F)$ and assign the total available slack (four slots) to it which results in a five hop path for m_1 as shown in figure 4.2. The corresponding schedule is also shown in the same figure. The total energy consumption for this schedule is $109.949mJ$ which corresponds to an improvement of 46% over the direct hop strategy.

The computational complexity of the greedy algorithm is dominated by the shortest path calculation step. In this chapter, we assume the all-pairs shortest path algorithm presented in [95] is executed prior to running the greedy algorithm and the results are stored in a table appropriately. With this arrangement, the worst-case run times of the greedy algorithm is $O(v^3 log_2 v + n^2)$ where, v is the number of nodes in the network and n denotes the number of input messages.

4.5.3 Movement Algorithm

The greedy scheduling algorithm assigns all the available slack to the next highest energy consuming message in a greedy fashion. However, such an approach cannot reduce the total energy consumption effectively.

As we increase the slack allocation for a particular message the energy consumption of the message transmission reduces as each additional slot can potentially replace a long hop with two smaller hops in the path. However, as we further increase its slack allocation the subsequent energy reductions offered by that message decrease due to the convex relation between energy consumption and the hop distance (equation 4.4). In order to succinctly represent this characteristic, we define a metric called *Energy Gain* for each message, m_k as follows:

$$Gain(m_k, j) = \lceil \frac{T_o}{T_k} \rceil (E(m_k, j-1) - E(m_k, j)) \tag{4.20}$$

where, $E(m_k, j)$ denotes the energy consumption of the message m_k when transmitted over the least energy consuming path between its source and destination with at most j hops. Clearly, the energy gain is defined for $2 \leq j \leq \beta$.

The energy gain, $Gain(m_k, j)$ denotes the amount of energy reduction that would be obtained by allowing message m_k to take j hops as opposed to $j-1$ hops. In other words, it represents the energy reduction that would be obtained by allocating one more time slot to message m_k which is currently using $j-1$ hops. For effective energy savings, a good slack allocation strategy would allocate slack to the highest energy gain yielding message.

For the example at hand, figure 4.3 shows the variation of the energy gain for each of the three messages with increasing number of hops. Two important observations can be made from this graph. First, as we increase the number of hops per message the energy gain decreases. For example, $Gain(m_1, 2) > Gain(m_1, 3)$ (referring to the graph, $G1 > G3 > G5 > G6$). Second, the energy gains obtained by the largest energy consuming message are not always the highest. For example, $Gain(m_1, 3) < Gain(m_3, 2)$ (referring to the graph, $G3 < G2$). Therefore, slack allocation should performed keeping track of the decreasing energy gain values of the individual

```
Input: set of $n$ messages
Output: Slack allocation for each message
1  Set $U = 0$;
2  for $i = 1;i \leq n;i++$ do
3      Place $m_i$ in the first column of the movement table;
4      Set $h_i = 1$, number of hops of $m_i$ ;
5      Add $m_i$ to set $Q$;
6      Set $U = U + \frac{h_i}{T_i}$
7  end
8  while $Q \neq \emptyset$ do
9      Pick up the message with highest $Gain(m_i, h_i + 1)$ from $Q$;
10     if $h_i + 1 > \beta$ then
11         Remove $m_i$ from $Q$ ;
12         Continue;
13     end
14     Set $U' = U - \frac{h_i}{T_i} + \frac{h_i+1}{T_i}$;
15     if $U' > 1$ then
16         Remove $m_i$ from $Q$ ;
17         Continue;
18     end
19     Move message $m_i$ one column to the right;
20     Set $h_i = h_i + 1$;
21     Set $U = U'$;
22 end
```

Algorithm 5: Movement Algorithm

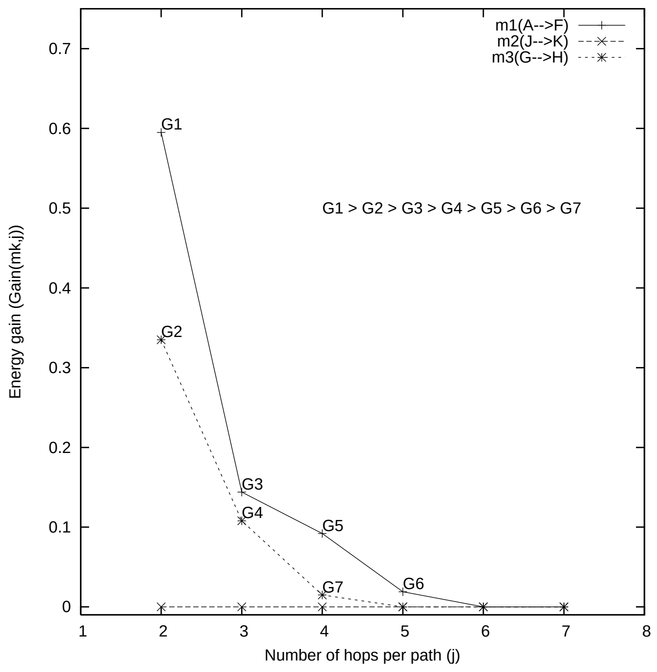


Figure 4.3 Energy Gain function

messages in the schedule.

We now present our movement algorithm which allocates slack to messages incrementally (one slot/hop at a time) based on the respective energy gain values. The algorithm uses a novel data structure called *movement table* (see figure 4.4).

The movement table varies the number of hops across columns and each message is associated with a row in the table. Further, at any given point of time, each message occupies a unique cell in that row. If a message is placed in the j^{th} column it means that it has been allocated j time slots and it takes the least energy consuming path with j hops. The movement table for the example is shown in figure 4.4.

When the movement algorithm begins, all the messages are placed in column one (labeled Hop 1) corresponding to a single hop transmission and the rest of the entries are left empty. In figure 4.4, we however filled in other entries for the illustration purpose. In each entry, the corresponding least energy consuming path is denoted. For example, the least energy consuming path for message m_1 with five hops is A, B, C, D, E, F as denoted in its column five. There exists no six hop path for message m_1 that consumes lesser energy than its five

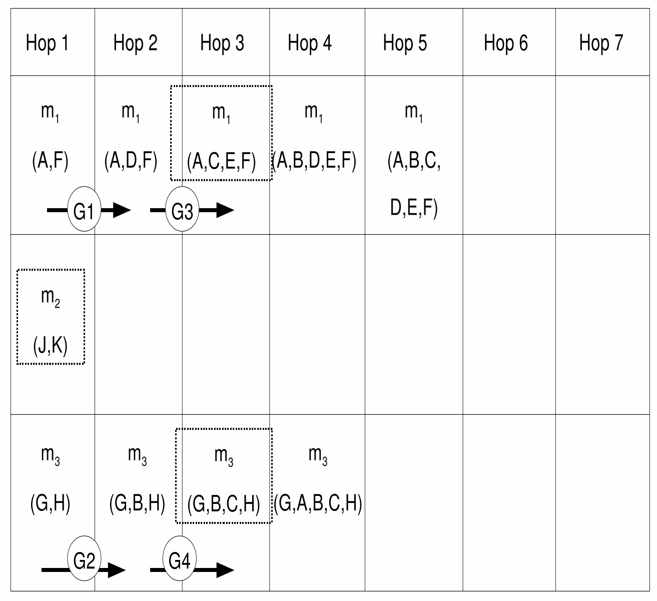


Figure 4.4 Illustrative example - Movement Table

hop path hence that entry is left empty.

The objective of the movement algorithm is to *move* as many messages as possible to the right without violating the reliability and schedulability constraints. The number of columns in the table denote the maximum number of hops allowed per message. The reliability guarantee can be provided by limiting the number of columns to β. The schedulability test is verified before every move in the algorithm. The pseudo-code for the movement algorithm is presented in Algorithm 5.

In steps 1-6, the algorithm initializes the movement table and different parameters. The algorithm also maintains a set (Q) consisting of all the potential messages which can utilize more slack without violating any constraints. In steps $8 - 22$, the algorithm iterates over the messages in Q and attempts to move each message one column to the right in the movement table. In step 9, the highest energy gain yielding message is selected. In the following step, it is verified if the message can take one more time slot without violating the reliability constraint. Similarly, in step 15, the schedulability test is verified to check if this slack allocation to m_i would lead to any deadline violations. If the selected message violates anyone of the above

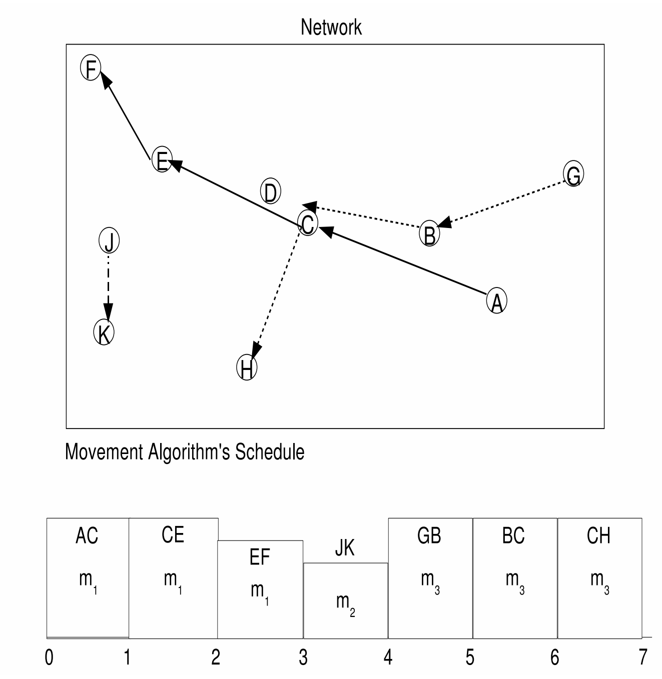


Figure 4.5 Illustrative example - Movement algorithm

tests, it is removed from the set Q. On the other hand, if all the constraints are satisfied, the message is actually moved to the right in the movement table in step 19 and other parameters are appropriately updated in steps $20-21$ before proceeding to the next iteration.

Applying the movement algorithm to the example (see figure 4.4), all the messages are initially placed in column one of the movement table. In the first iteration, message m_1 is selected as it yields the maximum energy gain, $G1$ (see figure 4.3) and is moved one column to the right. The corresponding paths for each message is shown in brackets in the movement table. In the subsequent iterations, messages m_2 (gain $= G_2$), m_1 (gain $= G_3$) and m_2 (gain $= G_4$) are selected in that order and allocated one slot for each selection. After four iterations, there is no slack available and hence the set Q becomes empty and the movement algorithm terminates. The final positions of the messages in the movement table are highlighted in figure 4.4. The corresponding paths and the schedule are shown in figure 4.5. In this example, the movement algorithm incurs an energy consumption of $76.99mJ$ and shows an improvement of 62% and 30% over the direct hop and greedy strategies respectively.

The computational complexity of the movement algorithm can be analyzed with the help of the movement table. Each message can move at-most β columns to the right and for each column movement, the maximum gain yielding message is determined in linear time. Therefore, the time taken by a single message is at most $O(n\beta)$ and the total time taken in the worst case is $O(n^2\beta)$. The movement algorithm also assumes the shortest paths for each message are calculated apriori therefore, the total worst case complexity of the movement algorithm is $O(v^3 log_2 v + n^2\beta)$.

In each iteration of the movement algorithm, the highest energy gain yielding message is allocated an additional time slot. In other words, for each additional time slot available the movement algorithm takes a locally optimal decision of assigning it to the highest energy yielding message. In general, such a locally optimal strategy cannot guarantee a global optimum for discrete problem formulations (here time can only be varied discretely). However, for this particular problem, we believe such an approach can result in a solution which is arbitrarily close to the globally optimum solution. In our simulation results we noticed that this is indeed the case.

4.6 Simulations studies

We performed two sets of simulation studies. In each set, we compared the performance of the proposed scheduling algorithms against the solution of the ILP formulated in section 4.4. We solved the ILP using the ILOG CPLEX 10.100 software [74].

In each simulation set, we considered a square region of size $500 * 500$ square meters where the nodes are randomly distributed following a certain distribution. We used the communication model parameters listed in table 4.2. For each simulation run, we generated 20 periodic messages with randomly chosen source and destination nodes. The performance metric was the normalized energy consumption where all energies are normalized with respect to the direct hop strategy. The total energy consumption of each schedule was calculated in the time window $[0, LCM]$, where LCM is defined as the least common multiple of the message periods. In our simulations, for each parameter set of interest, we performed ten different runs each with a different random number seed and the obtained average is plotted as a single point in the graph.

In the simulation set one, we followed the uniform distribution to randomly generate node locations in the network. We varied the following three parameters: U (system utilization), v (the number of nodes) and β (hop constraint) to evaluate the performance of different schemes.

In the set two, we followed a clustered node distribution. The entire network consisted of v_c clusters each with a radius r_c. The cluster locations are randomly chosen following a uniform distribution. The total number of nodes are equally divided among the v_c clusters. Within each cluster, the nodes assigned to it are randomly distributed following a uniform distribution. We believe such clustered node distributions are typical in applications like industrial automation. In our simulation studies we varied the v_c and r_c parameters and compared the performance of different schemes.

4.6.1 Results for uniform node distribution

4.6.1.1 Effect of utilization(U)

Figure 4.6, shows the relative performance of the above schemes. As the utilization increases, the total energy consumption also increases due to the increased workload. All the three schemes show this trend. At low values of U, there is ample slack available and all most all the messages can be transmitted via multiple hops easily. As a result the gap between the schemes is less in the graph. Similarly, at high values of U, there is almost no slack to perform any hop-by-hop transmissions as a result all the schemes default to direct transmission. Throughout the range, the movement algorithm performs significantly better than the greedy algorithm and moreover, it shows a performance very close to that achieved by the ILP. At $U = 0.5$, the movement algorithm shows an improvement of 31% over the Greedy algorithm. It incurs as less as 1.24% energy over the ILP.

4.6.1.2 Effect of number of nodes(v)

Figure 4.7 compares the relative performance of the schemes by varying the number of nodes in the network. As v increases the likelihood of finding intermediate nodes for multi-

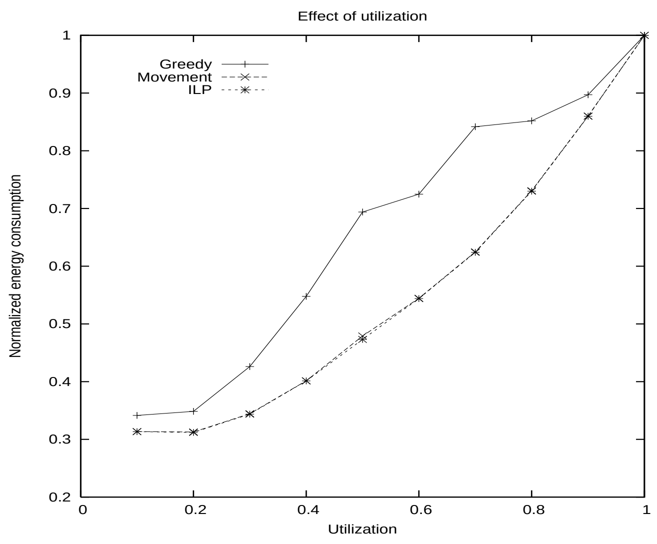


Figure 4.6 Effect of utilization ($v = 100, \beta = 7$)

hopping increases; however, after a certain point further multi-hopping will not be possible due to reliability and deadline constraints of the individual messages. Consequently, energy consumption values remain fairly constant for all the schemes after a point. The Movement algorithm shows an improvement of 29% and 31% over the Greedy scheme at $v = 80$ and $v = 200$ respectively.

Interestingly, when v is increased from 20 to 40, the greedy scheduling algorithm actually incurs more energy. This is because, with increased number of nodes in the network more slack can be assigned to a single message as it can now find intermediate nodes more easily. As a result, the greedy scheme allocates most of its slack to a subset of messages thereby missing out other messages which could potentially yield better energy gains. However, as we further increase v, this tendency is restricted by the deadline and reliability constraints and the greedy scheme behaves as expected.

The Movement algorithm incurred as less as 1.4% and 1.24% more energy than the ILP at $v = 80$ and $v = 200$ respectively. When v is increased beyond 300, the ILP solver ran out of memory and could not find the optimal solution. For these cases, we extrapolated the values obtained at $v = 200$ in the graph shown.

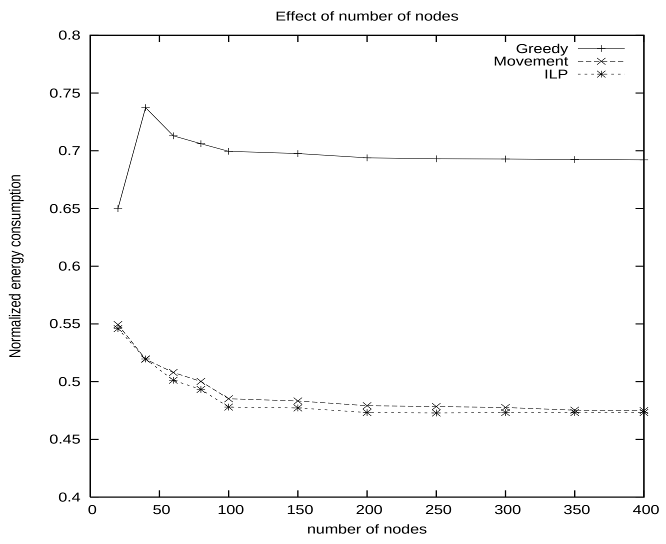


Figure 4.7 Effect of number of nodes ($U = 0.5, \beta = 7$)

4.6.1.3 Effect of required reliability(β)

Figure 4.8 shows the effect of required reliability (Rel_t) on the three schemes by varying the hop constraint, β. Higher the value of β lower is the required reliability. With increasing β, each message is allowed to take more smaller hops and hence the total energy consumption should decrease in general. However, due to the greedy nature of the Greedy algorithm, most of the slack is allocated only to a subset of messages thereby missing out other messages which could potentially yield better energy savings. As a result, the Greedy incurs higher energies as we increase β. The Movement algorithm on the other hand, performs slack allocation incrementally allocating each time slot to the next highest gain yielding message. At $\beta = 5$, it shows an improvement of 18% over the greedy algorithm and incurs as less as 1.24% energy more than the ILP.

4.6.2 Results for clustered node distribution

4.6.2.1 Effect of number of clusters(v_c)

Figure 4.9, compares the performance of the three schemes by varying the number of clusters in the network (v_c). As the v_c increases, there are more clusters in the network and

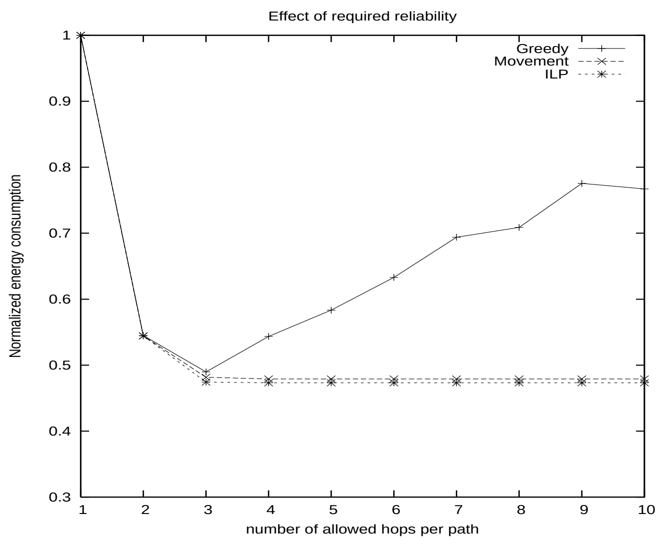


Figure 4.8 Effect of the required reliability ($v = 100$,$U = 0.5$)

hence the nodes are more widely spread. As a result, it is more likely to find intermediate nodes for the messages to perform hop-by-hop transmissions. Due to this phenomenon, the total energy consumption reduces with increasing v_c. All the three schemes show this trend. Throughout the range, the movement algorithm performs better than the greedy algorithm and offers a performance very close to that of the ILP. At $v_c = 6$, it shows an improvement of 18% over the greedy algorithm and incurs as less as 1% energy more than the ILP.

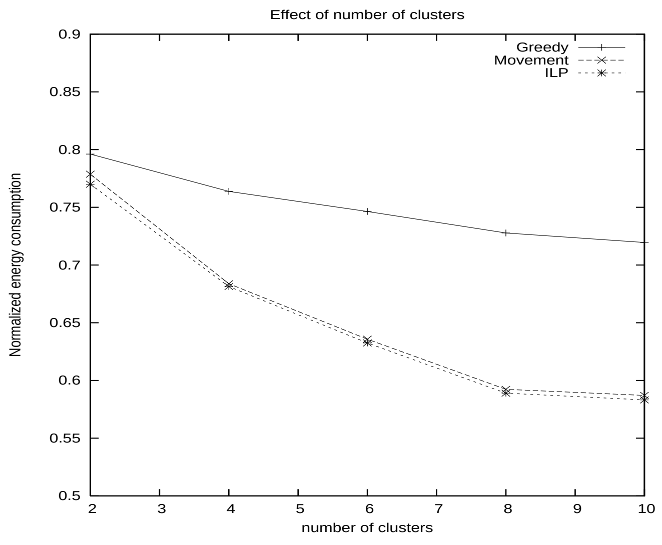


Figure 4.9 Effect of number of clusters ($v = 50, \beta = 7, U = 0.5, r_c = 0.10$)

4.6.2.2 Effect of cluster radius(r_c)

Figure 4.10, compares the performance of the three schemes by varying the normalized cluster radius(r_c). Where the cluster radii are normalized with respect to the diagonal (longest possible source-destination distance in the network) of the square region. As the r_c increases, each cluster expands and its nodes become more widely spread. As mentioned earlier, it is more likely to find intermediate nodes for the messages to perform hop-by-hop transmissions. Due to this phenomenon, the total energy consumption reduces with increasing r_c. All the three schemes show this trend. Throughout the range, the movement algorithm performs better than the greedy algorithm and offers a performance very close to that of the ILP. At $r_c = 0.15$, it shows an improvement of 20% over the greedy algorithm and incurs as less as 1% energy more than the ILP.

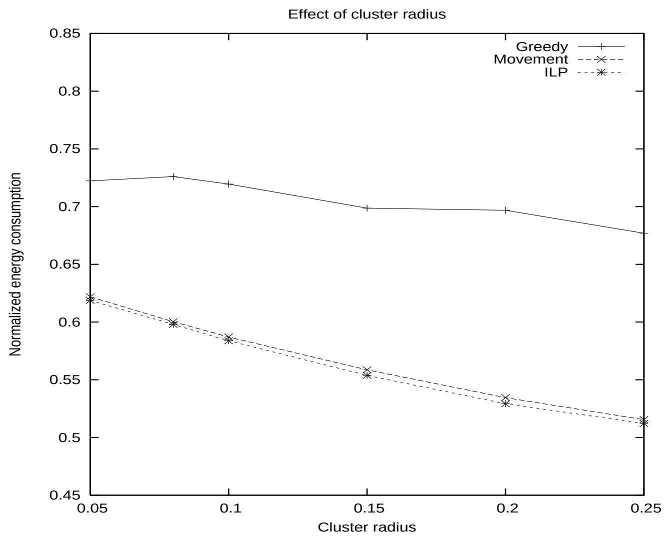


Figure 4.10 Effect of cluster radius ($v = 100, \beta = 7, U = 0.5, n_c = 10$)

4.7 Discussion

In this chapter, we addressed the problem of scheduling real-time messages over a shared-medium wireless network with the objective of minimizing the total energy consumption while providing the stringent probabilistic deadline guarantees. We employed the technique of using less energy consuming hop-by-hop transmissions instead of high energy incurring direct hop transmissions as a basis to solve this problem. After proving that this problem is NP-hard we proposed two energy-aware scheduling algorithms and compare them against an ILP solution of the scheduling problem. By means of our simulation results, we show that the proposed algorithms' performance is comparable to the ILP solution. In our future work, we plan to extend the presented scheduling algorithms to a multi-hop wireless network setup where parallelism offered by non-interfering transmission can be exploited.

CHAPTER 5. System level energy management in single hop networks

This chapter formulates the system-level energy management problem involving messages and tasks in a single-hop networked embedded systems. Here, we establish a novel framework that allows to perform integrated communication and computation energy management to achieve effective system-level energy savings.

The typical architecture in a networked real-time embedded system consists of several processor controlled nodes interconnected via the wireless network. The system software running on each node enables the execution of one or more concurrent tasks which are activated by the arrival of triggering events generated by the external environment, a timer or arrival of a message from another task. A response to an event generally involves several tasks to be executed on different nodes and several messages to be exchanged in the network. For the proper functioning of the whole system, each individual task as well as all the messages exchanged need to complete before stringent deadlines while incurring as less energy as possible. An effective energy-aware strategy for such a system would appropriately leverage the low power modes supported by different components within each node of the network under a unified system-level perspective.

5.1 Related work and motivation

Majority of the existing research work in the area of real-time energy-aware systems focused on reducing the energy consumption of the processor employing the well known dynamic voltage scaling (DVS) technique [79]. DVS refers to the technique of simultaneously varying the processor voltage and frequency as per the performance level required by the tasks. DVS allows to trade off processor's speed for energy savings. Several energy aware real-time DVS algorithms have been proposed addressing a wide range of task scheduling problems for a variety of system models [79, 78].

Recently, the focus of the research community has shifted from processor level energy management to system-level energy management, wherein the objective is to minimize the total energy consumption of the entire system as opposed to minimizing the processor energy consumption alone. More specifically, the interplay between the DVS technique employed to reduce the processor energy consumption and the rest of the system is being actively studied. In [80], an optimal processor frequency has been analytically derived to minimize the system energy consumption considering both the on-chip and off-chip workloads. The proposed solutions are effective for computation intensive real-time applications and do not address the communication aspects of the system.

The communication energy consumption of the system is considered in [81]. In their system model, the authors consider a DVS enabled processor along with a wireless card that independently employs a dynamic power management strategy wherein the card transitions between an active and several low power states as per the out-going traffic. Depending on the current state of the network card, the processor frequency is varied to control the finish times of the tasks which are same as the release times of the corresponding messages. This is done with the aim of maximizing the sleep durations for the network card and minimizing the sleep-to-active and active-to-sleep transition overheads. This work however does not consider message deadlines making it unsuitable for real-time applications where timely message delivery is of paramount importance.

From a different perspective, more effective power management techniques [77] can be employed to reduce the communication energy consumption as compared to low-power sleep modes that are assumed in the above work. Some such techniques include, power adaptation where transmission power is adapted based on the source-destination distance and dynamic modulation scaling (DMS) where the modulation level of the out-going message can be reduced to achieve transmission energy reductions [76].

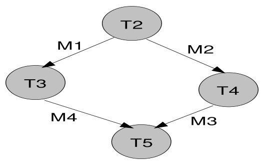
Similar to the DVS technique employed in processor energy management, DMS allows to tradeoff message transmission times for energy savings. The dynamic changes in the modulation level of the outgoing packets are communicated to the destination via the packet header and hence, the overheads of modulation scaling are almost negligible. Several energy aware DMS based schemes have been developed for non-real-time network applications [75] which cannot be directly extended to handle real-time workloads. In [76], the energy-aware problem of scheduling real-time messages at a single node is considered and a DMS based scheme has been presented. However, presented scheme does not consider the local computation issues and works at a node level ignoring the workload in the rest of the network. To the best of our knowledge, ours is the first work which addresses the system-level energy management problem considering both the computation and communication workloads in the network with real-time constraints.

5.2 System Model and Problem Statement

We consider a single-hop wireless networked embedded system with n nodes which share the common wireless medium that is accessed in an exclusive manner. We assume all nodes in the network are time synchronized. Each node supports DVS with k_t discrete frequency levels and DMS with k_m discrete modulation levels. In the examples and simulation studies we use the PXA255 processor's power specifications [82] shown in figure 5.1.(b) to model the processor energy consumption.

Task Model: We consider m periodic complex real-time tasks whose deadline is same as the period. Each complex task consists of several message exchanging sub-tasks where each sub-task has precedence constraints with other sub-tasks. For simplicity, in the rest of the chapter we refer to sub-tasks as tasks. For each given complex task, all its tasks and messages need to complete their execution before the deadline, respecting the precedence constraints. Figure 5.1.(a) shows the precedence graph of an example complex real-time task where nodes denote the tasks and edges denote the messages. We use the following notation in the rest of the chapter. For each task T_i, $desc(T_i)$ denotes the set of messages which are produced by it. For example, $desc(T_2) = \{M_1, M_2\}$. Similarly, $pred(T_i)$ denotes the predecessor messages of

T_i. For example, $pred(T_5) = \{M_3, M_4\}$. The workload for the tasks is specified as the number of CPU cycles (denoted as CC) and for the messages it is specified as the number of bits (denoted as L). We assume tasks and messages are non-preemptible.



(a) Precedence graph

Frequency	Power
f_3 = 400 MHz	p_3 = 411 mW
f_2 = 300 MHz	p_2 = 283 mW
f_1 = 200 MHz	p_1 = 175 mW

(b) Xscale PXA255 processor specifications

Figure 5.1   Task and CPU model

Problem Statement: The following issues need to be handled while scheduling such complex tasks in a networked embedded system with several embedded devices: (1) Task allocation - mapping individual tasks to nodes, (2) Scheduling - scheduling local tasks at each node and messages in the shared wireless medium, (3) Assigning frequency and modulation levels to the individual tasks and messages, respectively. Rich literature exists addressing the first two issues [96]. Therefore, in this chapter we assume that task allocation and scheduling have been performed apriori to obtain a feasible schedule. *Our primary focus is on assigning frequency and modulation levels to different tasks and messages respectively, to reduce the total energy consumption of the input schedule. This involves performing slack allocation across tasks and messages without violating the deadline and precedence constraints.*

In the rest of the chapter, we use the generic term *performance level* whenever we intend to refer to either a task's frequency level or a message's modulation level. Similarly, we use the generic term *entity* to refer to either a task or a message in the schedule. In these terms, our goal is to assign appropriate performance levels to different entities in the input schedule with the objective of minimizing the total energy consumption while not violating the deadline and precedence guarantees provided by the input schedule.

Communication Model: We assume that the modulation is Quadrature Amplitude Modulation (QAM). The channels are modeled as frequency-flat Rayleigh fading. Let b denote the number of bits per modulation constellation symbol. The constellation size is denoted as $M = 2^b$. Let E_b denote the received energy per bit, $N_0/2$ denote the channel noise power spectral density, E_s denote the received energy per constellation symbol, d_{min}^2 the minimum average squared Euclidean distance between two constellation symbol at the receiver, and BER denote the bit error rate. We have the following relationships:

$$b = \log_2 M \tag{5.1}$$

$$\text{BER} \approx N_0/d_{min}^2 \tag{5.2}$$

$$d_{min}^2 = \frac{6}{M-1}E_s \tag{5.3}$$

$$E_s = bE_b \tag{5.4}$$

The approximation in (5.2) is valid for high signal-to-noise ratio (SNR). Combining these, we have

$$E_b \approx \frac{2^b-1}{6b} \cdot \frac{N_0}{\text{BER}}. \tag{5.5}$$

If a message contains L bits, then the total necessary received energy will be $E_L = L \cdot E_b$. We assume that the propagation loss follows a polynomial model: the power decays in the α-th order of the distance:

$$E_{ts} = \left(\frac{d}{d_0}\right)^{\alpha} E_L \tag{5.6}$$

where E_{ts} is the necessary transmitted energy to achieve a received energy of E_L, d is the distance between the transmitter and the receiver, d_0 is a normalizing constant that depends on the wavelength. Usually α is between 2 and 5. As a result, the total transmitted radio energy can be expressed as

$$E_{ts} = \left(\frac{d}{d_0}\right)^{\alpha} \frac{L(2^b-1)}{6b} \cdot \frac{N_0}{\text{BER}} \tag{5.7}$$

In addition to the E_{ts}, some energy is consumed by the circuitry during the transmission which is given by: $E_{tc} = \frac{LC_t}{log_2 M}$. Similarly, the energy consumed in receiving a message is given by: $E_{rc} = \frac{LC_r}{log_2 M}$. Here C_t and C_r are implementation dependent constants[76]. In the rest of the chapter, we assume $N_0 = 4 * 10^{-13}$, $BER = 10^{-6}$, $C_t = 75nJ$, $C_r = 100nJ$, $L = 1024$ and $W = 1KHz$ as the default values.

The time taken for transmitting a L-bit message in the shared wireless medium is given by:

$$T = \frac{L}{Wb} \tag{5.8}$$

Here, W denotes the bandwidth (symbols per second) of the channel in hertz and the bandwidth in bits per second can be calculated as Wb. We assume each node supports DMS technique where modulation level (b) can be varied to tradeoff transmission latency for energy savings. Further, we assume that the BER is designed low enough to provide the necessary message reliabilities.

Notations: In the rest of the chapter, we use the following notation for each entity e_i in the schedule.

- $R(e_i)$: Ready time of the complex task to which e_i belongs to.
- $D(e_i)$: Deadline of the complex task to which e_i belongs to.
- $st(e_i)$: start time of e_i in the schedule.
- $ft(e_i)$: finish time of e_i in the schedule.
- $T(e_i, p_\alpha)$: processing time of entity e_i when operated at the performance level, p_α. For a task, it is calculated as $\frac{CC}{p_\alpha}$ where p_α refers to the operating frequency. Similarly, for a message it is calculated as $\frac{L}{Wp_\alpha}$ where p_α is its modulation level.
- $E(e_i, p_\alpha)$: energy consumption of entity e_i when operated at the performance level, p_α.
- $est(e_i)$: denotes earliest start time by which e_i can be scheduled without violating any of the constraints. It is calculated as, $est(e_i) = Max\{prev\_ft(e_i), pred\_ft(e_i), R(e_i)\}$. The expressions $prev\_ft(e_i)$ and $pred\_ft(e_i)$ are defined below.
- $lst(e_i)$: denotes latest start time at which e_i can be scheduled without violating any of the constraints. It is calculated as, $lst(e_i) = Min\{next\_st(e_i), desc\_st(e_i), D(e_i)\} - T(e_i, p_\alpha)$. The expressions $next\_st(e_i)$ and $desc\_st(e_i)$ are defined below.
- $host(e_i)$: denotes the processor on which e_i is scheduled. If e_i is a message, then it refers to the common shared wireless medium. In the schedule shown in figure 5.3, $host(T_2) = host(T_1) = P_2$ and $host(M_2) = host(M_1) = channel$.

- $prev(e_i)$: entity which is schedule right before e_i on the same processor or channel. In the schedule shown in figure 5.3, $prev(T_2) = T_1$ and $prev(M_2) = M_1$.
- $next(e_i)$: entity which is schedule right after e_i on the same processor or channel. In the schedule shown in figure 5.3, $next(T_1) = T_2$ and $next(M_1) = M_2$.
- $pred\_ft(e_i)$: earliest time by the which all the predecessor entities of e_i will finish. Mathematically, $pred\_ft(e_i) = Max_{\forall M_j \in pred(e_i)}\{est(e_j) + T(e_j, p_\alpha)\}$.
- $prev\_ft(e_i)$: earliest time by which the $prev(e_i)$ completes and is calculated as $est(prev(e_i)) + T(prev(e_i), p_\alpha)$.
- $desc\_st(e_i)$: the latest time by which at least one of the descendant entities of e_i will start in the schedule. It is calculated as, $desc\_st(e_i) = Min_{\forall e_j \in desc(e_i)}\{lst(e_j)\}$.
- $next\_st(e_i)$: the latest time by which $next(e_i)$ will start in the schedule.

5.3 Energy-aware scheduling Algorithms

In this section, first we analyze the system-level energy-time tradeoffs by defining a novel metric called normalized energy gain. We then present an offline static scheduling algorithm followed by a dynamic distributed scheduling algorithm.

5.3.1 System level energy-time tradeoffs

In order to effectively reduce the total energy consumption of the input schedule the available slack should be allocated across tasks and messages in the schedule. A good slack allocation strategy would allocate slack to the entity which results in maximum energy reduction for every additional unit of slack allocated to it. To determine the best entity for slack allocation, we define the following metric called *normalized energy gain* for each entity as follows:

$$G(i, i-1) = \frac{E_i - E_{i-1}}{T_{i-1} - T_i} \tag{5.9}$$

Where E_i and T_i respectively denote the energy consumption and time incurred by the entity when operated at i^{th} performance level. For each entity e_j which is currently assigned the i^{th} performance level, $G(i, i-1)$ represents the energy reduction that would be obtained by reducing its performance level to $i-1$ normalized with respect to the additional time incurred for operating it at performance level $i-1$ instead of level i. The energy gain metric succinctly captures how effectively a given amount of slack is utilized by each entity and ideally slack should be allocated to the entity which offers highest normalized energy gain. Figure 5.2

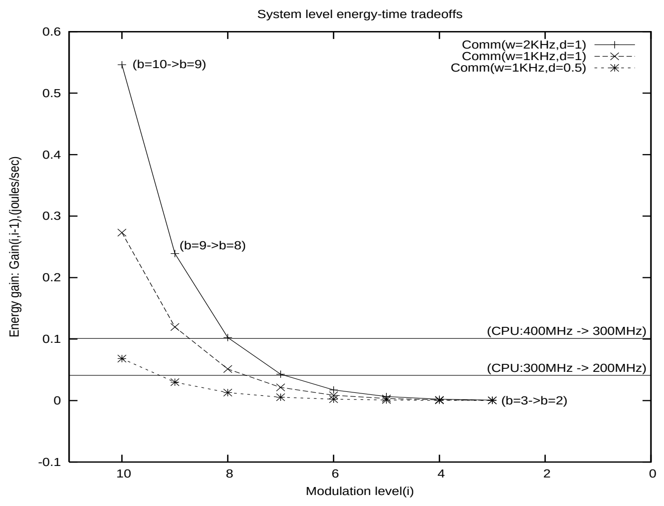


Figure 5.2 System level energy-time tradeoffs

shows the normalized energy gain for the messages as a function of i (modulation level) for different values of W and d with message size, $L = 1$. It also shows the DVS energy gains (horizontal lines) offered by a task of size $CC = 1$ for the PXA255 processor settings. The following two important observations can be made from the figure.

1. As we decrease the performance level, the subsequent energy gains obtained are decreasing both for the messages and tasks as shown in the figure. For example, the energy gain obtained by reducing the CPU frequency from $300MHz$ to $200MHz$ is lower than that obtained by reducing the frequency from $400MHz$ to $300MHz$. Similarly, for a given W and d, we have $G(i+1,i) > G(i,i-1)$. This trend suggests that we should allocate slack incrementally across messages and tasks keeping track of the decreasing energy gain values.
2. *The energy consumption of message transmission is in general higher than that of computation and it may appear on the surface that the slack should be entirely allocated to the messages. However, from the figure 5.2 we can see that the exact energy gains depend*

upon several parameters like W, d, the current modulation level b of the message and the current frequency level of the task. For example, in the figure considering the curve with $W = 1KHz$ and $d = 1$, we have $G(i = 8, i = 7) < G(400MHz, 300MHz)$ where as $G(i = 9, i = 8) > G(400MHz, 300MHz)$. This suggests that there are situations (specific values of W and d) where slack must be allocated to tasks rather than messages. For the cases, where the message energy gains are higher than the task energy gains, the energy gain metric helps comparing the energy reductions offered by each message and guides the slack allocation across different messages.

In order to ensure a safe slack allocation mechanism, we estimate the maximum safe slack for each entity prior to allocating any slack to it. For a given input schedule several task movement techniques like *sliding*, *passing* and *task migration* can be applied to determine the maximum safe slack for each entity. In this chapter, we use the sliding technique due to its simplicity. The proposed algorithms can be easily extended to use the other techniques. In sliding, no entity e_i is allowed to start before $prev(e_i)$ finishes. Further, each entity e_i should complete before its $next(e_i)$ starts execution in the schedule and $host(e_i)$ should not be modified at any point of time. Following this, the maximum available slack for each entity e_i can be calculated as $lst(e_i) - est(e_i) - T(e_i, p_\alpha)$ where p_α is the currently assigned performance level of e_i.

5.3.2 Gain based Static Scheduling (GSS)

We now present our offline gain based scheduling algorithm which assigns performance levels to each entity in the input schedule. The GSS algorithm proceeds in iterations allocating slack in an incrementally fashion. In each iteration, the highest energy gain yielding entity is chosen and it is allocated just enough slack to reduce its performance mode by one level. The rest of the slack is allocated similarly to the remaining entities in the schedule. At the end of each iteration, the individual energy gain values are updated.

In order to keep track of the messages which can utilize more slack without affecting any constraints, the GSS algorithm maintains a set Q and updates it after each iteration. The algorithm terminates once the set Q becomes empty. The following step by step procedure presents the GSS algorithm.

The worst-case computational complexity of the GSS algorithm can be derived as $O((n+m)(nk_t+mk_m))$ where n and m respectively denote the number of tasks and messages in the schedule.

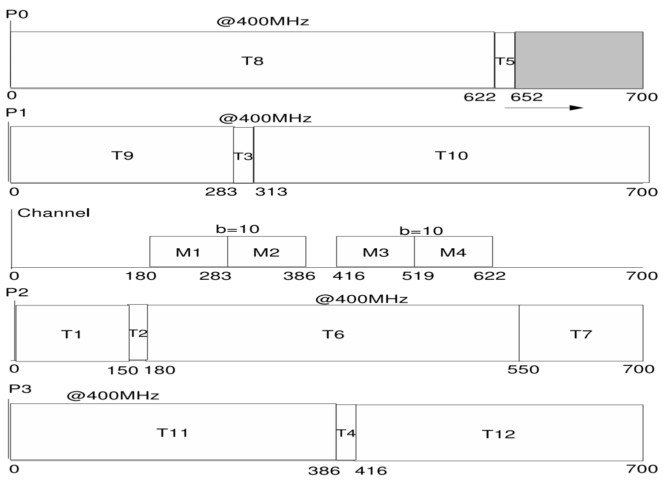


Figure 5.3 Illustrative example: input schedule

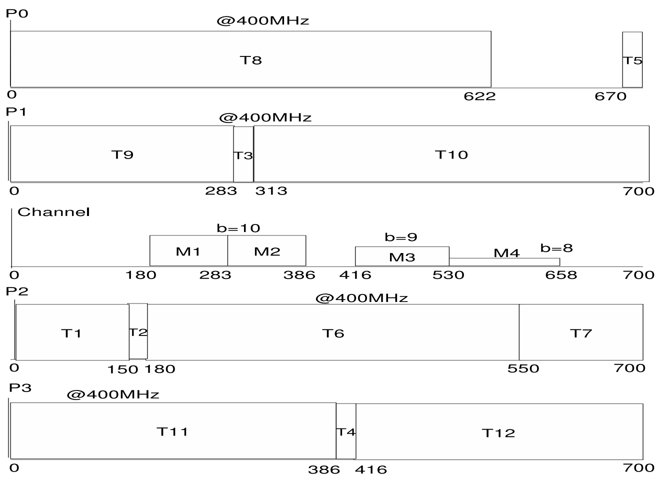


Figure 5.4 Working of the GSS algorithm

The illustrative example shown in figures 5.3 and 5.4 demonstrates the working of the GSS algorithm. Figure 5.3 represents the input schedule. We are primarily interested in tasks T_2, T_3, T_4, T_5 and messages M_1, M_2, M_3, M_4 whose precedence relationship is shown in figure 5.1.(a). For simplicity, we assume the other tasks in the schedule are independent; further, we assume a unit normalized source-destination distance for all the messages in the schedule. Our algorithms however work for the general cases.

The GSS algorithm will first pickup the highest energy gain yielding entity, M_4 which offers an energy gain of $3107uJ$ $(= \frac{N_0}{6BER} L(\frac{2^{10}-1}{10} - \frac{2^9-1}{9}))$ using $11ms$ (obtained as $\frac{L}{9W} - \frac{L}{10W}$) of slack. In fact, M_1, M_2, M_3 and M_4 have equal energy gains (i.e, $3107/11$ uJ/ms) at this point. However M_1 and M_2 have zero maximum slack and the tie between M_3 and M_4 is arbitrarily broken. Now, M_4's maximum available slack is determined by sliding T_5 towards the right as shown in the figures. Since $M_4 \in pred(T_5)$, it will now have $48ms$ of slack to lower its modulation level. Hence, the GSS algorithm reduces the modulation level of M_4 to $b = 9$ using $11ms$ of slack. Before allocating the remaining slack, the M_4's energy gain is updated to $1700/14uJ/ms$ which is now lower than that of M_3 which is $3107/11$ uJ/ms. As a result, M_3 is now given a slack of $11ms$ to lower its modulation level to $b = 9$. At this point, M_3 and M_4 have the same energy gains, breaking ties arbitrarily, GSS allocates more slack to M_4 leaving $12ms$ of slack which is unusable by any of the entities in the schedule. The resulting schedule is shown in figure in figure 5.4. The energy reduction obtained for message M_4 can be calculated as $4807uJ(= 3107+1700)$ and that of M_3 is $3107uJ$. Therefore, the GSS algorithms has reduced the energy consumption of the schedule by $7914uJ(= 4807 + 3107)$.

5.3.3 Distributed Slack Propagation Algorithm (DSP)-Dynamic Slack

The actual execution times of tasks exhibit a large degree of variation and are often much lesser than the worst-case estimates used in offline scheduling. As a result, in addition to the static slack that is available offline, some dynamic slack is generated at run-time as the schedule progresses. Such dynamic slack can be utilized to further scale-down the performance levels of the tasks and messages.

Performing global dynamic slack allocation considering all the entities in the schedule requires message passing across nodes via the wireless network which can incur exceedingly high energy and time overheads. With this motivation, we have developed a light weight distributed scheduling algorithm wherein each node independently follows the schedule generated by the

```
Input: Input Schedule, $H_i$
Output: Output Schedule, $H_o$
1  Set $H_o = H_i$;
2  Add all tasks and messages in $H_i$ into the set $Q$;
3  while $Q \neq \emptyset$ do
4      Pick up the highest energy gain yielding entity $e_i$ from $Q$;
5      Determine the maximum slack, $S_{max}$ available for $e_i$;
6      Let $s$ be the time required to operate $e_i$ at its next lower
       performance level;
7      if $s \leq S_max$ then
8          Decrement the performance level of $e_i$ by one level;
9          Update the schedule $H_o$;
10     end
11     else
12         Remove $e_i$ from $Q$;
13     end
14 end
```

Algorithm 6: Gain based Static Scheduling Algorithm

GSS algorithm; when the dynamic slack is generated on a processor it is utilized for local tasks and locally originating messages in an independent manner requiring no additional communication across the processors. Such a distributed slack utilization policy can potentially lead to deadline and precedence constraint violations if the independent decisions taken by each processor are inconsistent. In this section, we first present our DSP scheduling algorithm and then argue its correctness.

Whenever a task T_i completes its execution incurring less time than its worst-case execution time, the generated dynamic slack can be utilized in one of the two ways. The next task, $next(T_i)$ can utilize all of the dynamic slack to reduce its own frequency or it can simply slide left and allow its following task, $next(next(T_i))$ to use the slack. In other words, the task $next(T_i)$ can either use the slack for itself or propagate its slack to the following task $next(next(T_i))$. Similarly, the task $next(next(T_i))$ can choose to propagate the slack to its following task which is $next(next(next(T_i)))$ and so on. In figure 5.5, task T_1 completes at time $100ms$ leaving $50ms$ of slack. Now this slack can be used to either reduce the frequency of T_2 or simply to slide T_2 left so that the frequency of the following tasks' T_6 and/or T_7 can be decreased instead. Choosing the second option has an additional advantage because if T_2 propagates the slack by sliding left, the messages M_1 and M_2 which are produced by T_2 also get the slack (see fig 5.1.(a)). In the rest of the chapter, for each task, we denote the option of using the slack for itself as its *option zero* and the option of propagating the slack further as its *option one*. To decide whether T_2 should choose option zero or one, we need to evaluate the absolute energy reductions (or energy gains) that would be obtained by each of the two options.

In order to capture these different options and estimate their energy gains, we introduce a novel data structure called the *slack propagation tree (SPT)* where each message and task is represented as a separate node in the tree.

5.3.3.1 Slack Propagation Tree (SPT) Construction

Whenever some dynamic slack is generated due to the early completion of a task T_i on a processor P_j, the processor constructs the SPT with $next(T_i)$ as the root and other local tasks and locally originating messages that are yet to complete as the non-root tree nodes. Consider the SPT shown in figure 5.6. This SPT is constructed to decide how to use the slack generated by the early completion of task T_1 in figure 5.5. Since the slack is generated on P_2, the SPT is constructed with all the tasks on P_2 that are yet to complete (i.e, T_2,T_6,T_7) and their corresponding descendant messages that originate at P_2 (i.e, M_1, M_2).

Two different kinds of edges are used in the SPT. The edges drawn with a label 0 denote the option that the performance of the corresponding entity (from which the edge is drawn) is scaled down and slack cannot be propagated further. Hence the tree terminates here along that branch. On the other hand, edges drawn with a label 1 denote the option that the corresponding entity (from which the edge is drawn) will propagate the slack further to its following entities. In figure 5.6, As the slack can potentially be used to reduce the performance level of any of the above tasks and messages after appropriate sliding, a 0-labeled edge is drawn from each one of them to their respective leaf nodes. Whenever slack can be propagated from one task to another and from one message to another, a 1-labeled edge is created as shown in the figure. For example, by sliding T_2 left the slack can be propagated to task T_6 and message

M_1. Hence, 1-labeled edges are from T_2 to T_6 and T_2 to M_1. In the following, we denote the leaf node of e_k as $\phi_0(e_k)$ and the set of its 1-label child nodes as $\phi_1(e_k)$.

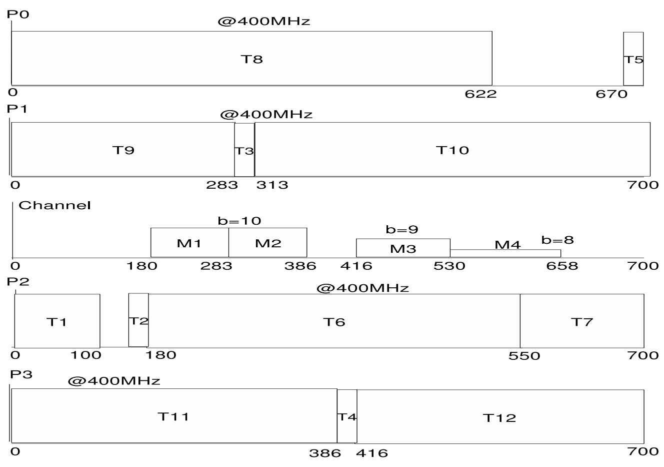


Figure 5.5 Illustrative example: online schedule

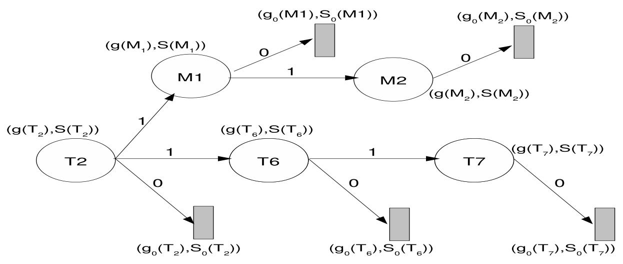


Figure 5.6 Slack Propagation Tree (SPT)

The SPT represents different possibilities via which the generated dynamic slack can be utilized. A complete schedule can be obtained by choosing either option zero or option one for each non-leaf internal node in the SPT. In order to make this choice at each node, we need to evaluate the energy gains that would be obtained by each of the two possibilities. To this end, we associate a tuple with each node in the SPT which consists of two elements namely, *energy gain* and *shift value*. Energy gain of a node denotes the amount of energy reduction that would be obtained by choosing the best option (either zero or one) at that node. The slack required to achieve this amount of energy gain at that node is stored in the shift value of its tuple. We denote the energy gain of a task T_k (message M_j) as $g(T_k)$ (as $g(M_j)$) and the amount of necessary shift as $S(T_k)$ (as $S(M_j)$). For notational convenience, we denote $S(C_0(T_k))$ as

$S_0(T_k)$. Similarly, we denote $S(C_0(M_j))$ as $S_0(M_k)$. In figure 5.6, the tuples for each node are shown by their side. In the following, we present the tuple value calculation details for each node in the SPT.

Tuple value calculations for leaf nodes: The gain value for a leaf node $C_0(e_k)$ denotes the amount of energy reduction that would be obtained by reducing the performance mode of e_k by one level. Mathematically, $g_0(e_k) = E(e_k, p_\alpha) - E(e_k, p_{\alpha-1})$ where p_α denotes the current performance level of e_k. Accordingly, its shift value is calculated as, $S_0(e_k) = T(e_k, p_{\alpha-1}) - T(e_k, p_\alpha)$. If slack is propagated to e_k and it chooses the option zero, its start time is shifted to $st(e_k) - S_0(e_k)$ and its performance is reduced to $p_{\alpha-1}$ in the output schedule while leaving its finish time unchanged. However, if $est(e_k) > st(e_k) - S_0(e_k)$ both $g_0(e_k)$ and $S_0(e_k)$ are set to zero and e_k's performance is not reduced in the output schedule.

Tuple value calculations for internal nodes: Consider the SPT in figure 5.6, to decide whether T_2 should choose option zero or one, we compare their corresponding energy gains, $g_0(T_2)$ and $(g(M_1) + g(T_6))$. The option which yields the maximum energy gain is chosen for T_2. In general the gain value of e_k is calculated as: $g(e_k) = Max(g_0(e_k), \sum_{\forall e_j \in \phi_1(e_k)} g(e_j))$. If $g_0(e_k) \geq \sum_{\forall e_j \in \phi_1(e_k)} g(e_j)$, $S(e_k)$ is set to $S_0(e_k)$ and e_k chooses option zero. Otherwise, $S(e_k)$ is set to $Max_{\forall e_j \in \phi_1(e_k)}(S(e_j))$ and e_k chooses option one propagating slack further.

In either case, if the start time of entity e_k cannot be shifted by $S(e_k)$ i.e, if e_k fails the conditions: $est(e_k) \leq st(e_k) - S(e_k)$ its gain and shift values are set to zero and its schedule is left unchanged. If e_k passes this test, its start time and finish time are shifted to $st(e_k) - S(e_k)$ and $ft(e_k) - S(e_k)$, respectively. We note that the above schedule modifications ensure that the new finish times of the entities do not exceed their respective finish times in the input schedule.

5.3.3.2 The DSP Algorithm

The DSP algorithm uses the SPT data structure to reduce the performance levels of the tasks and messages using the dynamic slack. It proceeds in iterations updating the tuple values of the SPT nodes in each iteration. The following step-by-step procedure outlines the DSP

	Iter-1	Iter-2	Iter-3	Iter-4	Iter-5
$\phi_0(M_1)$	$3107, 11$	$1700, 14$	$1700, 14$	$0, 0$	$0, 0$
$\phi_0(M_2)$	$3107, 11$	$3107, 11$	$1700, 14$	$1700, 14$	$0, 0$
$\phi_0(T_7)$	$5050, 50$	$0, 0$	$0, 0$	$0, 0$	$0, 0$
$\phi_0(T_6)$	$0, 0$	$0, 0$	$0, 0$	$0, 0$	$0, 0$
$\phi_0(T_2)$	$1830, 10$	$0, 0$	$0, 0$	$0, 0$	$0, 0$
T_6	$5050, 50$	$0, 0$	$0, 0$	$0, 0$	$0, 0$
M_1	$3107, 11$	$3107, 11$	$1700, 14$	$1700, 14$	$0, 0$
T_2	$8157, 50$	$0, 0$	$0, 0$	$0, 0$	$0, 0$

Table 5.1 Tuple values after each DSP iteration

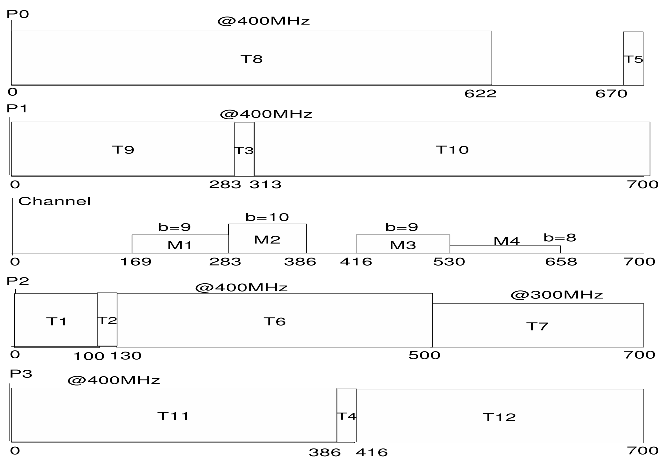


Figure 5.7 Schedule after one traversal of SPT

algorithm which is run by each processor independently after recognizing some dynamic slack in the schedule.

In figure 5.5, when T_2 completes its execution at time $100ms$ generating a dynamic slack of $s = 50ms$, processor P_2 constructs the SPT shown in figure 5.6 and updates the *est*s as follows: $est(T_2) = 100, est(T_6) = 130$ and $est(T_7) = 500$ while their start times in the current schedule are: $st(T_2) = 150, st(T_6) = 180$ and $st(T_7) = 550$. Table 1 shows the tuple values of different SPT nodes across different iterations. The tuple values for the first iteration are shown in column 1. For example, the gain value of $\phi_0(T_7)$ is calculated as

```
**Input**: Input Schedule, $H_i$
**Output**: Output Schedule, $H_o$
1 Construct the SPT including local tasks that are yet to
  complete and the messages these tasks produce;
2 Update the earliest start times of all the tasks and messages in
  the SPT;
3 Calculate the tuple values and determine the best option for
  each node in the SPT;
4 If there is any entity which yields non-zero energy gains proceed
  to the next step, otherwise exit the DSP algorithm;
5 Adjust the schedule according to the options chosen and go to
  step 3;
```

Algorithm 7: Gain based Static Scheduling Algorithm

$E(T_7, 400MHz) - E(T_7, 300MHz) = 411 * 150 - 283 * 200 = 5050uJ$ and the shift value is calculated as, $T(T_7, 300) - T(T_7, 400) = \frac{400}{300}150 - 150 = 50ms$. The tuple values of the other SPT nodes are calculated in a similar fashion.

The resultant schedule after the first iteration is shown in figure 5.7. As there is more slack for M_1 and M_2, at the end of iteration one they show non-zero energy gain values. As a result, DSP proceeds for the next iteration and it terminates after four iterations when the energy gain values of all the nodes reduce to zero. The tuple values for all the iterations are shown in table 5.1. In the final schedule, M_1 and M_2 are assigned a modulation level of $b = 8$ while rest of the schedule remains same as in figure 5.7. The energy reduction obtained by the DSP algorithm can be easily calculated from the table 5.1, by adding the unique gain values for each leaf node across several iterations which is $14664uJ$ (obtained as $2 * 3107 + 2 * 1700 + 5050$).

The run-time of the DSP algorithm can be estimated as $O(v^2(k_t + k_m))$ where v denote the number of tasks and messages allocated to the processor of interest. Further, as DSP ensures that each entity in the schedule completes prior to is finish time in the input schedule, each node can safely assume that remote tasks and incoming messages would complete latest by their finish times in the input schedule and thereby, schedule the local tasks and outgoing messages without affecting any precedence constraints. As the ready time and deadline constraints are globally known and fixed parameters they are taken care trivially.

5.4 Performance evaluation

We performed simulation studies to evaluate the relative performance of the proposed static and dynamic scheduling algorithms. We considered a unit circular region with 15 nodes where node locations are randomly chosen following an uniform distribution. The input schedule was generated using a simplified version of the scheduling algorithm presented in [96] to obtain a feasible schedule for randomly generated input task graphs. For each simulation run, we generated 30 complex periodic tasks and the simulation was performed in the time window of $[0, 50 * LCM]$ where LCM is the least common multiple of the task periods. We varied the following parameters: Channel bandwidth(W), Slack factor (s_f) of a complex task defined as $\frac{deadline-wcet}{T_{trans}}$ where $wcet$ denotes the worst-case execution time of the entire complex task in the input schedule, $\frac{BCET}{WCET}$: ratio of the best case sub-task execution time($BCET$) to worst case sub-task execution time ($WCET$). The actual execution time of each sub-task is chosen randomly in the interval $[BCET, WCET]$.

The performance metric in our simulation studies is the normalized total energy consumption of the schedule i.e, the total energy consumption of the schedule normalized with respect to the energy consumption of the input schedule. We compared the following four algorithms in our simulations: (1) Gain based computation only (comp-only) - variation of GSS where only tasks are considered, (2) Gain based communication only (comm-only) - variation of GSS where only messages are considered, (3) GSS, (4) DSP. Where the normalized energy consumption denotes the total energy consumption of the schedule normalized with respect to that of the input schedule.

Effect of channel bandwidth(W): Figure 5.8 shows the relative performance of the four schemes with varying channel bandwidth. The other parameters are chosen as: $s_f = 10$, $\frac{BCET}{WCET} = 0.5$. At high values of W, messages have significantly high energy gains i.e., reducing the modulation level of a message takes very less additional slack and yields high energy reductions. As a result, GSS behaves very much like the comm-only scheme where slack is given only to the messages. At $W = 1000KHz$, GSS shows an improvement of 70% over the comp-only scheme. At low bandwidths like $W = 1KHz$, the message energy gains are comparable to that of the tasks and are sometimes even lower than that of the tasks. Hence the comp-only scheme performs better than the comm-only scheme. Since GSS performs a system level slack allocation considering both the tasks and messages it performs better than both the above schemes yielding about 8% improvement over the comm-only scheme. Throughout, DSP performs better than all the above schemes utilizing the dynamic slack in the schedule. At $W = 1000KHz$, it shows an improvement of 15% over the GSS algorithm.

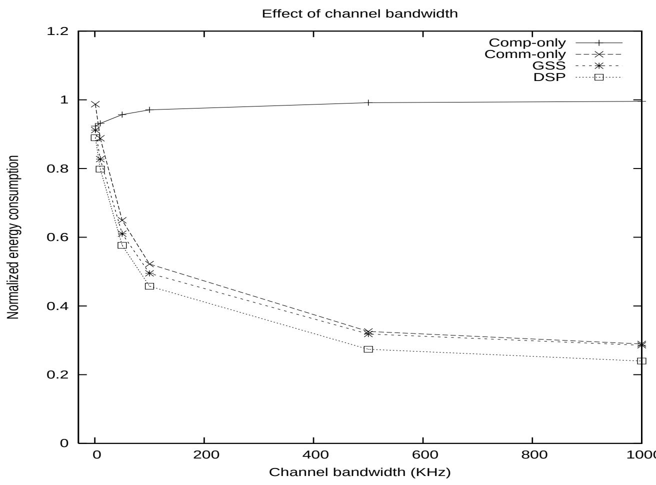


Figure 5.8 Effect of channel bandwidth

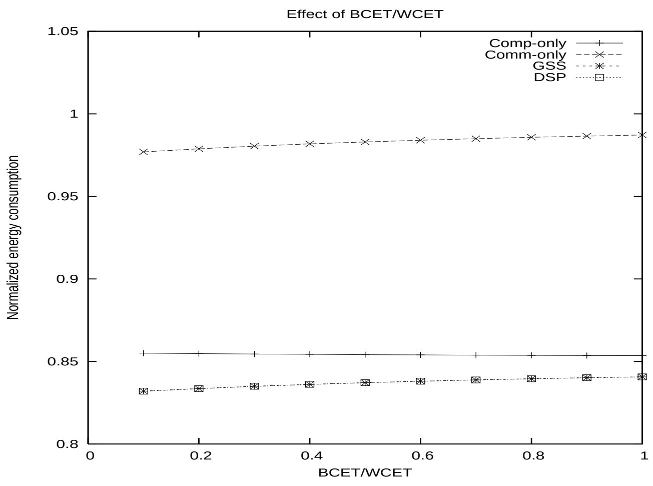


Figure 5.9 Effect of $\frac{BCET}{WCET}$ at $W = 1KHz$

Effect of $\frac{BCET}{WCET}$: Figures 5.9 and 5.10 show the relative performance of the above schemes with varying $\frac{BCET}{WCET}$ at $W = 1KHz$ ($s_f = 100$) and $W = 1000KHz$ ($s_f = 0.5$) respectively.

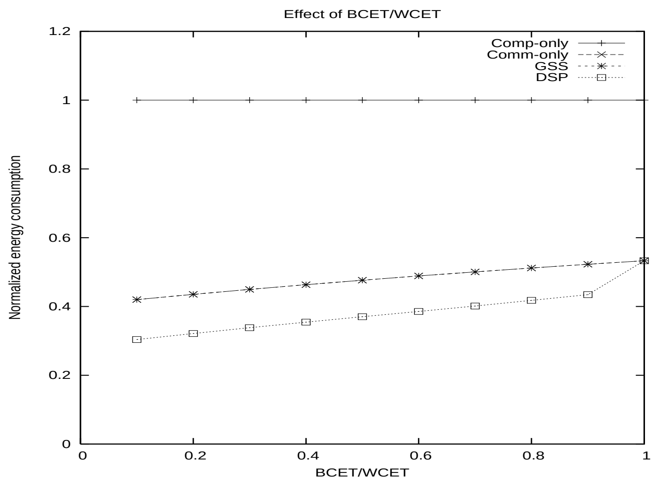


Figure 5.10 Effect of $\frac{BCET}{WCET}$ at $W = 1000KHz$

In figure 5.9 with $W = 1KHz$, tasks offer energy gains comparable to that of messages and sometimes offer even better energy gains. As a result, comp-only scheme performs better than the comm-only scheme. The situation is exactly reversed at high bandwidths as shown in figure 5.10. In all the above scenarios, GSS allocates slack to the highest energy gain yielding entity in the schedule and hence performs better than both the baseline algorithms. In figure 5.9, it shows an average improvement of 15% over comm-only scheme while in figure 5.10, it shows an average improvement of 45% over the comp-only scheme. DSP yields further energy savings as shown in figure 5.10. It shows an improvement of 27% and 19% over the GSS algorithm at $\frac{BCET}{WCET} = 0.2$ and $\frac{BCET}{WCET} = 0.8$, respectively.

Effect of slack factor (s_f): Figure 5.11 shows the relative performance of the above scheme with varying slack factor with $W = 1000KHz$ and $BCET/WCET = 0.5$. At low s_f, the available slack is insufficient to perform any dynamic voltage scaling and hence comp-only scheme does not show any improvements over the input schedule. On the other hand, as the s_f is increased the available slack is utilized for the tasks and hence comp-only scheme shows an improvement of 6% over the input schedule at $s_f = 200$. Throughout, GSS performs better than both the baseline schemes and shows an improvement of about 50% and 10% over the comp-only and comm-only schemes respectively. DSP uses the ample dynamic slack to reduce the total energy consumption showing an additional improvement of about 15% over

GSS at $s_f = 1.0$. As the slack factor increases, most of the tasks and messages operate at low performance levels subsequently, the energy reductions brought by the DSP diminish due to the convex nature of the energy function.

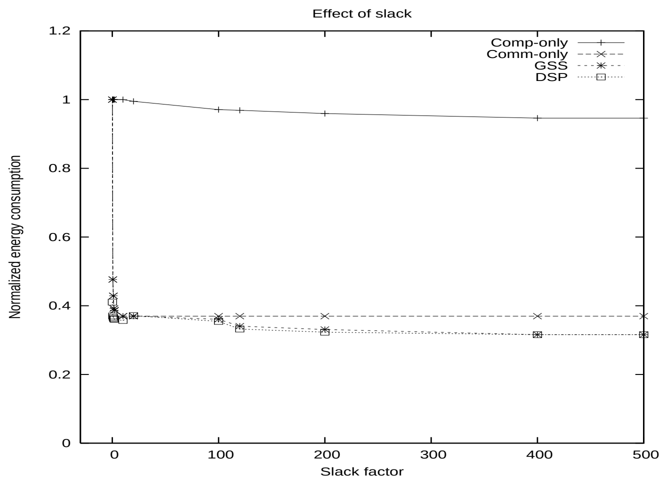


Figure 5.11 Effect of slack factor

5.5 Discussion

Recent technological advancements have opened up a plethora of real-time distributed applications where battery-driven embedded devices with computation and wireless communication capabilities provide time-critical services. Energy management is the key issue in the design and operation of such networked real-time embedded systems. In this chapter, we addressed the problem of scheduling complex periodic tasks in a single-hop wireless networked embedded system, where each node supports both DVS and DMS power management techniques. We presented novel centralized static and distributed dynamic scheduling algorithms to solve this problem. Starting with an input feasible schedule, the proposed algorithms perform slack allocation across tasks and messages with a system-level perspective. We performed simulations to study the effectiveness of the proposed schemes. Our results show the proposed schemes yield significant energy savings over the input schedule.

CHAPTER 6. System level energy management in multi-hop networks

In this chapter, we consider a specific multi-hop networked embedded system and address the end-to-end energy management problem taking into account both the computation and communication workloads in the system.

Wireless sensor network is a well-known multi-hop networked embedded system where data is monitored in a distributed manner and individual nodes collaborate to perform in-network data aggregation by exchanging messages across several hops. In this chapter, we focus on the end-to-end energy management of data aggregation in distributed wireless sensor networks.

Data aggregation is a fundamental task in sensor networks where energy management is a primary concern. Majority of the existing work on data aggregation aims at minimizing the communication energy consumption and ignores the computational workloads at the individual nodes. With the growing complexity in the targeted applications, the computational demands on the individual nodes are gradually increasing. For example, consider a military surveillance application where the data has to be encrypted and communicated in a secure manner. In such a scenario, each aggregating node need to perform data decryption and encryption to provide the required security. The work in [98] presents different secure data aggregation schemes where the primary focus is on the security issues.

On a different front, it is envisioned that the individual nodes in wireless sensor networks support machine learning capabilities in the future to adapt to the environmental variations where the individual nodes will be expected to execute heavier computational tasks. The work in [99, 100] presents different machine learning schemes for the wireless sensor nodes.

With this impending trend, it becomes inevitable to consider both the computational and communication energy consumptions at the individual nodes in order to perform effective end-to-end energy management for the data aggregation applications. In this chapter, we consider both the processor energy consumption incurred due to the computational workloads along with the communication energy consumption of each node in the network.

6.1 Related Work

Several power management techniques exist for processor and communication subsystems which can be exploited while performing data aggregation for effective energy minimization. For the processor, some of these techniques include: *sleep-wakeup* and *performance scaling*. For the communication subsystem, these techniques include: *sleep-wakeup* [61], *power adaptation* [62] and *performance scaling* [64]. In this chapter we focus on performance scaling techniques.

Performance scaling in processor energy management is often achieved by employing the well-known dynamic voltage scaling (DVS) technique [10]. DVS is the technique of simultaneously varying the processor voltage and frequency as per the performance level required by the tasks. It allows to tradeoff processor speed for energy savings. DVS has been extensively applied to address a wide variety of task scheduling problems targeting stand-alone embedded applications [80]. Majority of the proposed solutions in this context do not address the communication aspects and hence are not directly applicable to the data aggregation problem we addressed in this chapter. In the rest of the chapter, we refer to the processor frequency simply as frequency which is different from the communication RF frequency.

Similar to the DVS in processor energy management, performance scaling in communication energy management refers to the technique of lowering the transmission power and transmitting the packet for a longer period of time to reduce the transmission energy consumption. This technique allows to tradeoff transmission latency for communication energy savings. Such an energy-time tradeoff can be achieved either by varying the modulation level or error correcting codes or both simultaneously in a message transmission. These specific variations are termed as Dynamic Modulation Scaling (DMS), Dynamic Code Scaling (DCS) and Dynamic Modulation-Code Scaling (DMCS) respectively [65]. Although the specific variables or control knobs might vary across the three techniques, the fundamental tradeoffs they offer are similar. In the rest of the chapter, we work with the DMS technique where modulation level of each message can be varied to tradeoff transmission latency for energy savings.

Performance scaling for communication energy management has been applied in several contexts including the download problem involving a single sender and multiple receivers and the corresponding upload problem with single receiver and multiple senders [64, 67, 69, 66]. Majority of the above mentioned work applies communication performance scaling to different single-hop scheduling problems which cannot be directly extended to the data aggregation application which primarily is a multi-hop setup.

Further, most of the above existing work is based on the assumption that the transmission energy consumption is a continuous function of transmission latency. However, in practical systems only a few discrete performance levels are allowed and each transmission can be performed at one of these levels. As a result, the solutions developed assuming a continuous energy function merely represent lower bounds to the practical problem setups. Several chapters suggest a simple rounding mechanism where the continuous solutions are rounded to the nearest discrete solution taking a conservative approach [70]. However, such rounding mechanism is not the optimal approach and hence cannot effectively reduce the energy consumption particularly, when the number of discrete performance levels are few which is typically the case in practical systems.

The most relevant work for the problem addressed in this chapter is presented in [70]. The authors consider the energy minimization problem for a latency constrained data aggregation application with DMS technique as the underlying power management mechanism. Their model assumes that energy consumption is a continuous function of message modulation level (or correspondingly message transmission latency). Further, the authors assume a collision-free medium access which can be achieved either by employing multiple RF frequencies for communication or other techniques like multiple packet reception (MPR) [101].

For the above ideal model where modulation can be varied continuously, the authors develop numerical algorithms to solve the offline problem of scheduling messages at the individual nodes with the aim of minimizing the total energy consumption while meeting the end-to-end latency constraint. The proposed solution exploits the continuous convex nature of the problem and assigns modulation levels to different messages in the network. They also proposed a dynamic programming based solution for the online version of the problem assuming a continuous function for the transmission energy consumption. However, the presented solutions are not effective for the practical scenarios where each node supports discrete modulation levels. Moreover, as the authors pointed out in [70], the presented solutions in their work cannot be extended to more general medium access channel setups like TDMA (Time Division Multiple Access) or other contention-based protocols.

Our work differs from the existing work in the following ways.

- Unlike majority of the existing work which applies communication performance scaling to single-hop setups [64, 67, 69, 66], we address the network-level problem of end-to-end energy management considering the entire network with multiple hops.

- We consider a general communication model where all communications in the network are performed using the same RF frequency. In such a model, different conflicting message transmissions need to be scheduled.

- More importantly, we consider both computational and communication workloads at the individual nodes in the network. Further, we consider performance scaling for both processor and communication subsystems.

- Unlike the existing work, our primary focus is on the practical problem formulation wherein each node supports discrete performance levels.

To the best of our knowledge, ours is the first work which applies performance scaling both at the processor and communication level to the data aggregation problem considering the end-to-end latency constraint.

The rest of the chapter is organized as follows: We present our system model in section 6.2. In section 6.3 we present analytical problem formulations for both the ideal and practical performance scaling models. Further, we also prove that problem is NP-Hard for the practical model. In section 6.4, we first analyze system-level energy tradeoffs considering both computation and communication workloads at the individual nodes. We then present our scheduling algorithms which apply the analyzed system-level tradeoffs. In sections 6.5 and 6.6, we present our performance evaluation results and conclusions respectively.

6.2 System Model

Tree Model: We consider a data aggregation tree with V nodes. The root node is denoted as v_0 and the set of leaf nodes is denoted as V_l. Figure 6.1 outlines our model where each leaf node performs local sensing, computation (e.g, data encryption) and communication (e.g, sends the encrypted message to its parent). Each internal node performs local computation such as data decryption, aggregation and encryption. It further communicates the aggregated message to its parent. Similarly, the root node performs both local computation and communication to a base station which is possibly connected to a wired network. We denote the local computation at node v_i as the task T_i and its outgoing message as M_i (see figure 6.1). The task execution time and message sizes at node v_i are denoted as C_i computation cycles and L_i bits respectively. For each node v_i, we denote its set of child nodes as $V_c(i)$. This set is empty for the leaf nodes.
Precedence constraints: Each node starts its local computation only after receiving all the messages from its children. Similarly, each node begins its communication only after the local task completes its execution and the message is ready for transmission.
End-to-end latency constraint: Each message that is generated at a leaf node should reach the sink within a specified deadline, D. In other words, each leaf to root path has an end-to-end latency constraint of D.
Performance Scaling Model: Each node supports DVS with λ_t discrete frequency levels and DMS with λ_m discrete modulation levels. In the examples and simulation studies we use the PXA255 processor's power specifications [82] shown in table 6.1 to model the processor energy consumption. For the ideal case, where we assume the processor can vary its frequency continuously, we use the following expression to compute the processor energy consumption of a task with C computation cycles when operated at a frequency f: $\beta_c C f^2$ [10]. Here β_c is a

constant.

Data Correlation Model: For each non-leaf node, we model the correlations among the data collected from different child nodes as follows. Assuming each child node, v_i sends a packet of L_i bits to its parent node v_j, the outgoing message size at node v_j is given by:

$$L_j = \frac{\sum_{\forall v_i \in V_c(j)} L_i}{n_j\beta_f - \beta_f + 1} \tag{6.1}$$

where β_f is the correlation factor which varies in the range $[0,1]$ and n_j denotes the number of child nodes of v_j. We note that $\beta_f = 0$ corresponds to the case where there are no correlations in the data. On the other hand, $\beta_f = 1$ corresponds to the case of maximum data correlation. We assume the task execution times are not affected by the data correlations.

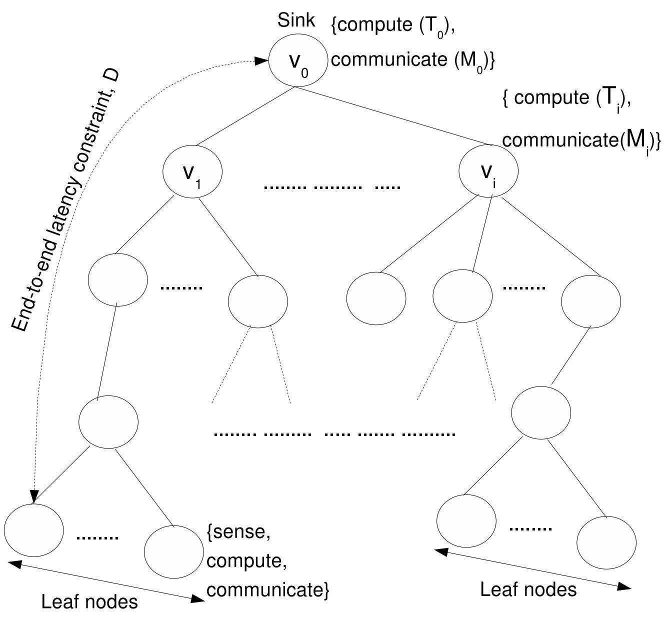

Sink
{compute (T_0), communicate (M_0)}
v_0
{ compute (T_i), communicate(M_i)}
v_1
v_i
End-to-end latency constraint, D
{sense, compute, communicate}
Leaf nodes
Leaf nodes

Figure 6.1 Data Aggregation Tree Model

Scheduling Model: For each internal node, the set of corresponding child nodes share the common wireless medium and their messages need to be scheduled in a non-conflicting manner. The shared wireless medium is slotted in time and is accessed in an exclusive manner following a TDMA protocol. We assume a feasible schedule is available where different messages and tasks operate at the maximum performance levels. We refer to this schedule as the input/default

Symbol	Mode	Power
f_{high}	400 MHz	411 mW
f_{mid}	300 MHz	283 mW
f_{low}	200 MHz	175 mW
Idle	33MHz@Idle	45 mW

Table 6.1 Power specifications (from [82])

schedule.

Given such a feasible input schedule that satisfies precedence and end-to-end latency constraints, *our primary focus is on assigning frequency and modulation levels to different tasks and messages in the data aggregation tree, with the objective of reducing the total energy consumption of the input schedule. This involves performing slack allocation across tasks and messages without violating the precedence and end-to-end latency constraints.*

In the rest of the chapter, we use the generic term *performance level* whenever we intend to refer to either a task's frequency level or a message's modulation level. Similarly, we use the generic term *entity* to refer to either a task or a message in the schedule. In these terms, our goal is to assign appropriate performance levels to different entities in the input schedule with the objective of minimizing the total energy consumption while not violating the deadline and precedence guarantees provided by the input schedule.

Communication Model: We assume that the modulation is Quadrature Amplitude Modulation (QAM). The channels are modeled as frequency-flat Rayleigh fading. Let b denote the number of bits per modulation constellation symbol. The constellation size is denoted as $M = 2^b$. Let E_b denote the received energy per bit, $N_0/2$ denote the channel noise power spectral density, E_s denote the received energy per constellation symbol, d^2_{min} is the minimum average squared Euclidean distance between two constellation symbols at the receiver, and

BER denote the bit error rate. We have the following relationships [97]:

$$b = \log_2 M \tag{6.2}$$

$$\text{BER} \approx N_0/d_{min}^2 \tag{6.3}$$

$$d_{min}^2 = \frac{6}{M-1}E_s \tag{6.4}$$

$$E_s = bE_b \tag{6.5}$$

The approximation in (6.3) is valid for high signal-to-noise ratio (SNR). Combining these, we have

$$E_b \approx \frac{2^b-1}{6b} \cdot \frac{N_0}{\text{BER}}. \tag{6.6}$$

If a message contains L bits, then the total necessary received energy will be $E_L = L \cdot E_b$. We assume that the propagation loss follows a polynomial model: the power decays in the α-th order of the distance:

$$E_{ts} = \left(\frac{d}{d_0}\right)^{\alpha} E_L \tag{6.7}$$

where E_{ts} is the necessary transmitted energy to achieve a received energy of E_L, d is the distance between the transmitter and the receiver, d_0 is a normalizing constant that depends on the wavelength. Usually α is between 2 and 5. As a result, the total transmitted radio energy can be expressed as

$$E_{ts} = \left(\frac{d}{d_0}\right)^{\alpha} \frac{L(2^b-1)}{6b} \cdot \frac{N_0}{\text{BER}} \tag{6.8}$$

In addition to the E_{ts}, some energy is consumed by the circuitry during the transmission and reception which are given by: $E_{tc} = \frac{L\beta_t}{b}$ and $E_{rc} = \frac{L\beta_r}{b}$, respectively; where$\beta_t$ and β_r are implementation dependent constants[65]. In the rest of the chapter, we assume $N_0 = 4{*}10^{-13}J$, BER $= 10^{-6}$, $\beta_t = 75nJ$, $\beta_r = 100nJ$, $L = 1024$ as the default values. The channel bandwidth in hertz is denoted with the symbol, W. The corresponding channel transmission rate in bits/sec can be calculated as Wb. We assume that the BER is designed low enough to provide the necessary message reliabilities. The time taken for transmitting a L-bit message in the shared wireless medium is calculated as: $\frac{L}{Wb}$.

Notations: We use the following symbols in the rest of the chapter.

- v_0: root node in the data aggregation tree.
- V_l: set of leaf nodes in the data aggregation tree.
- T_i: task at node v_i.
- M_i: message at node v_i.
- L_i: message size at node v_i in bits.
- C_i: task size at node v_i in computation cycles.
- x_i: modulation level assigned to M_i.
- y_i: frequency level assigned to T_i.
- S_i^t: start time of the task T_i in the schedule.
- S_i^m: start time of the message M_i in the schedule.
- $prev(M_i)$: The message which is scheduled just before the message M_i in the input schedule. In figure 6.4, $prev(M_D) = M_C$.
- $next(M_i)$: The message which is scheduled just after the message M_i in the input schedule. In figure 6.4, $next(M_C) = M_D$.
- $t(e_i)$: Time taken by the entity e_i at its current performance level in the schedule.
- $t(T_i)$: Time taken by the task T_i at its current frequency level in the schedule.
- $t(M_i)$: Time taken by the message M_i at its current modulation level in the schedule.
- p_i: parent node of node v_i.
- d_i: normalized distance between the nodes v_i and p_i.
- $V_c(i)$: set of child nodes of node v_i.
- D: end-to-end deadline (latency constraint)

6.3 Problem Formulation

Given a data aggregation tree and an input schedule, the problem is to assign performance levels to each of the entities in the schedule with the objective of minimizing the total energy consumption while guaranteeing the precedence and end-to-end latency constraints. We address the above problem for the following two models. For each model, we present the corresponding analytical formulations.

- *Ideal model*: In the ideal model, the processor frequencies can be varied continuously within the specified range. Similarly, the modulation levels of each message can be scaled continuously. We use the variables x_i and y_i to denote modulation level of message M_i and frequency of task T_i, respectively.
- *Practical model*: In this model, each node in the network supports only a few discrete frequency and modulation levels. Specifically, we assume each node provides λ_t discrete frequency levels and λ_m modulation levels. Further, we denote the k^{th} frequency level and its corresponding power consumption with f_k and P_k respectively, where$k \in [1, \lambda_t]$. Similarly, the k^{th} modulation level is denoted as b_k where $k \in [1, \lambda_m]$.

6.3.1 Problem formulation: Ideal model

The above stated problem can be mathematically formulated for the ideal model as follows.

Minimize:

$$E_{total} = \sum_{v_i \in V} \left(\frac{L_i N_0}{6\text{BER}} \frac{2^{x_i} - 1}{x_i} d_i^2 + \frac{L_i(\beta_t + \beta_r)}{x_i}\right) + \sum_{v_i \in V} \beta_c C_i y_i^2 \quad (6.9)$$

Subject to:

$$x_{min} \leq x_i \leq x_{max} \quad (6.10)$$

$$y_{min} \leq y_i \leq y_{max} \quad (6.11)$$

$$S_j^m + \frac{L_j}{W x_j} - S_i^t \leq 0, \forall v_j \in V_c(i) \quad (6.12)$$

$$S_i^t + \frac{C_i}{y_i} - S_i^m \leq 0, \forall v_i \in V \quad (6.13)$$

$$S_j^m + \frac{L_j}{W x_j} - S_i^m \leq 0, v_j = prev(M_i) \quad (6.14)$$

$$S_i^t \geq 0, \forall v_i \in V_l \tag{6.15}$$

$$S_0^m + \frac{L_0}{Wx_0} \leq D \tag{6.16}$$

The above formulation accepts an input schedule and assigns the start times (variables S_i^t and S_i^m) and performance levels (variables x_i and y_i) to different tasks and messages with the objective of minimizing the total energy consumption while preserving the precedence and latency constraints guaranteed by the input schedule.

The first term of the objective function given in (6.9) totals the communication energy of all the messages while the second term totals the computation energy of all the tasks in the schedule. Further, the constraints in (6.10)-(6.11) ensure that the performance levels are maintained with in the specified range.

The remaining equations capture the ready time, deadline and precedence constraints. The constraint in (6.12) ensures that the task at each node in the network starts after it receives all the messages from its children. Similarly, the constraint in (6.13) ensures that each message is transmitted only after the local task has finished its execution. The constraint in (6.14) ensures that each message starts after its previous message finishes its transmission in the schedule. The constraints in (6.15) specify that the start times of tasks at the leaf nodes should be greater than time zero and finally, the equation (6.16) ensures that the root node completes the its message transmission before the end-to-end deadline D.

Assuming the variables x_i and y_i can be varied continuously, the objective function E_{total} can be proved to be a convex function as it is a sum of several convex functions. Moreover, the constraints in (6.11)-(6.16) are also convex in the variables x_i and y_i. Also, the variables S_i^t and S_i^m are continuous and linear in these constraints. As a result, the above problem formulation depicts a convex optimization problem and standard convex optimization tools can be applied to solve the above problem. However, the obtained solution cannot be directly applied to the practical model without losing the optimality. In the following, we address the practical problem where each node supports discrete performance levels.

6.3.2 Problem Formulation: Practical Model

In this section, we first prove that the scheduling problem for the practical model is NP-Hard. Then, we present a Mixed Integer Linear Programming (MILP) formulation to solve the problem optimally.

NP-Hard proof: Optimally solving the problem of assigning performance levels to messages and tasks of the data aggregation application for the practical model is NP-Hard. We prove this by reducing the well-known Multiple Choice Knapsack Problem (MCKP) [72] to the problem at hand. The MCKP problem is stated as follows.

Multiple Choice Knapsack Problem : Given λ classes $N_1, ...N_\lambda$ of items to pack in a knapsack of capacity c. Each item $O_{ij} \in N_i$ has a profit p_{ij} and weight w_{ij}. The problem is to choose one item from each class such that the profit sum is maximized without having the weight sum to exceed c.

The MCKP can be mathematically expressed as follows:

Maximize

$$\sum_{i=1}^{\lambda} \sum_{j \in N_i} p_{ij} z_{ij}$$

Subject to:

$$\sum_{i=1}^{\lambda} \sum_{j \in N_i} w_{ij} z_{ij} \leq c$$

$$\sum_{j \in N_i} z_{ij} = 1$$

$$z_{ij} \in \{0, 1\}$$

Theorem: The scheduling problem of assigning performance levels to different tasks and messages in the data aggregation tree to optimally minimize the total energy consumption of the input schedule while meeting the precedence and end-to-end latency constraints is NP-Hard.

Proof: For the convenience of reduction, we rewrite the maximization objective of the MCKP as the following minimization objective: $\sum_{i=1}^{\lambda} \sum_{j \in N_i} (1 - p_{ij}) z_{ij}$.

The above presented MCKP problem can be reduced to an instance of the scheduling problem at hand where the data aggregation tree has a root and a collection of leaf nodes connected directly to the root. Further, we assume the leaf nodes have no local computation to perform in this instance i.e, $C_i = 0$, $\forall v_i \in V_l$. The reduction proceeds as follows.

- *Tree construction*: Create a root node with a local task (T_0) and message (M_0). Associate the classes N_1 and N_2 to T_0 and M_0, respectively. Construct $(\lambda - 2)$ leaf nodes, all directly connected to the root node. Denote the message at leaf node, v_i as M_i. Associate the class N_{2+i} to the message M_i.
- *Object to task mapping at the root*: Considering the task T_0 and its associated class N_1, set the number of frequency levels at which T_0 can operate as the number of objects in the class N_1. Now, each object in the class N_1 corresponds to the execution of the task T_0 at a unique frequency. In other words, by selecting a frequency for the task T_0 an object from the class is correspondingly selected and viceversa. Now, set the execution time of the task T_0 at the j^{th} frequency to be w_{1j}. Similarly, set the energy consumption of T_0 when operated at j^{th} frequency level to be $(1 - p_{1j})$.
- *Object to message mapping at the root*: Considering the task M_0 and its associated class N_2, set the number of modulation levels at which M_0 can operate as the number of objects in the class N_2. Now, each object in the class N_2 corresponds to the transmission of the message M_0 at a unique modulation level. In other words, by selecting a modulation level for the task M_0 an object from the class is correspondingly selected and viceversa. Now, set the transmission time of the message M_0 at the j^{th} modulation level to be w_{2j}. Similarly, set the energy consumption of M_0 when transmitted at j^{th} modulation level to be $(1 - p_{2j})$.
- *Object to message mapping at the leaf nodes*: Now, establish a similar mapping from message M_i to the class N_{2+i}. Therefore, we have the transmission time of M_i at j^{th} modulation level equal to $w_{2+i,j}$ and corresponding energy consumption is $(1 - p_{2+i,j})$.

With this mapping the total energy consumption of all the messages and tasks in the constructed data aggregation tree is given by, $\sum_{i=1}^{\lambda} \sum_{j \in N_i} (1 - p_{i,j} z_{ij})$. Here the binary variable

z_{ij} denotes which performance level if chosen for the corresponding entity. By setting $D = c$, we have the end-to-end latency constraint as: $\sum_{i=1}^{\lambda} \sum_{j \in N_i} w_{ij} z_{ij} \leq D$.

Now the instance of the data aggregation problem constructed above has a one-to-one correspondence with the MCKP problem. This reduction takes $O(\sum_{i=1}^{\lambda} \mid N_i \mid)$ amount of time which is clearly a polynomial of the problem size. This implies that if the scheduling problem can be solved in polynomial time, then the MCKP can also be solved in polynomial time. However, it is known that the MCKP problem is NP-Hard [72]. Therefore, the scheduling problem is also NP-Hard.

6.3.2.1 MILP Formulation

We now present an MILP formulation for the data aggregation scheduling problem with tasks and messages. The formulation presented below uses the following variables: $S_i^t, S_i^m, X_{ij}, Y_{ij}$. The real variables S_i^t, S_i^m denote the start times of task T_i and message M_i respectively. The integer (binary) variables X_{ij} and Y_{ij} denote the performance level chosen for the task T_i and message M_i, respectively. Specifically, the variable X_{ij} is set to one if j^{th} modulation level is assigned to message M_i, otherwise it is set to zero. Similarly, the variable Y_{ij} is set to one if j^{th} frequency level is assigned to message T_i, otherwise it is set to zero.

The following MILP formulation optimally minimizes the total energy consumption of the input schedule while meeting the precedence and end-to-end latency constraints. The objective function in (6.17) sums up the energy consumptions for all messages and tasks in the network and is similar to (6.9). The constraints in (6.18)-(6.22) represent different precedence, ready time and deadline constraints and are similar to (6.12)-(6.16). The constraints in (6.23) and (6.24) ensure that only one performance level is assigned to each message and task in the schedule.

Minimize:

$$\sum_{v_i \in V} \sum_{k=1}^{\lambda_m} \left(\frac{L_i N_0 d_i^2}{6\text{BER}} \frac{2^{b_k}-1}{b_k} \quad + \quad \frac{L_i(\beta_t+\beta_r)}{b_k} \right) X_{ik} \quad + \quad \sum_{v_i \in V} \sum_{k=1}^{\lambda_t} \beta_c P_k \frac{C_i}{f_k} Y_{ik} \tag{6.17}$$

Subject to:

$$S_j^m + \sum_{k=1}^{\lambda_m} \frac{L_j}{Wb_k} X_{jk} - S_i^t \leq 0, \forall v_j \in V_c(i) \tag{6.18}$$

$$S_i^t + \sum_{k=1}^{\lambda_t} \frac{C_i}{f_k} Y_{ik} - S_i^m \leq 0, \forall v_i \in V \tag{6.19}$$

$$S_j^m + \sum_{k=1}^{\lambda_m} \frac{L_j}{Wb_k} X_{jk} - S_i^m \leq 0, v_j = prev(M_i) \tag{6.20}$$

$$S_i^t \geq 0, \forall i \in V_l \tag{6.21}$$

$$S_0^m + \sum_{k=1}^{\lambda_m} \frac{L_0}{Wb_k} X_{0k} \leq D \tag{6.22}$$

$$\sum_{k=1}^{\lambda_t} Y_{ik} = 1, \forall v_i \in V \tag{6.23}$$

$$\sum_{k=1}^{\lambda_m} X_{ik} = 1, \forall v_i \in V \tag{6.24}$$

$$X_{ik}, Y_{ik} \in \{0,1\} \forall i, k \tag{6.25}$$

6.4 Energy-Aware Scheduling Algorithms

Although the above presented MILP formulation solves the problem optimally, it can take exponential amount of time demanding excessively high memory and computational resources for reasonably large problem sizes. Therefore, in this section we present heuristic scheduling algorithms which run in polynomial time.

In the following, first we analyze the system-level energy-time tradeoffs by defining a novel metric called normalized energy gain. We then present two efficient algorithms which employ the energy gain metric.

6.4.1 System Level Energy-Time Tradeoffs

In the input schedule of the data aggregation tree, each entity is assigned the highest performance level. The objective of the energy-aware scheduling algorithms is to assign as low performance level as possible to each entity (message or task) without violating the precedence and end-to-end latency constraints. Assigning performance levels corresponds to allocating the available slack across different entities in the schedule.

In order to effectively reduce the total energy consumption, a good slack allocation strategy would allocate slack to the entity which results in maximum energy reduction for every additional unit of slack allocated to it. To determine the best entity for slack allocation, we define the following metric called *normalized energy gain* (or simply energy gain) for each entity as follows:

$$G(k, k-1) = \frac{E_k - E_{k-1}}{\widehat{t}_{k-1} - \widehat{t}_k} \tag{6.26}$$

where E_k and $\widehat{t}_k$ respectively denote the energy consumption and time incurred by the entity when operated at k^{th} performance level. For each entity e_j which is currently assigned the k^{th} performance level, $G(k, k-1)$ represents the energy reduction that would be obtained by reducing its performance level to $k-1$ normalized with respect to the additional time incurred for operating it at performance level $k-1$ instead of level k. The energy gain metric succinctly captures how effectively a given amount of slack is utilized by each entity and ideally slack should be allocated to the entity which offers highest normalized energy gain.

Figure 6.2 shows the normalized energy gain for the messages as a function of k (modulation level) for different values of W and d with message size $L = 1$. It also shows the DVS energy gains (horizontal lines) offered by a task of size $C = 1$ for the PXA255 processor settings. The following two important observations can be made from the figure.

1. As we decrease the performance level, the subsequent energy gains obtained are decreasing both for the messages and tasks as shown in the figure. For example, the energy gain obtained by reducing the CPU frequency from $300MHz$ to $200MHz$ is lower than that obtained by reducing the frequency from $400MHz$ to $300MHz$. Similarly, for a

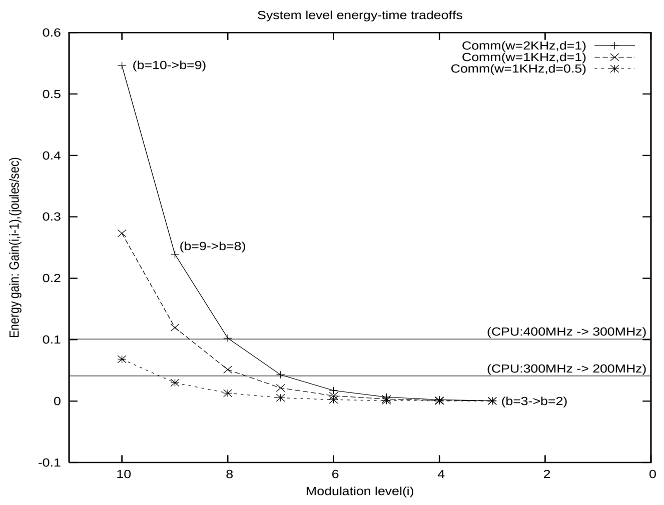


Figure 6.2 System level energy-time tradeoffs

given W and d, we have $G(k+1,k) > G(k,k-1)$. This trend suggests that we should allocate slack incrementally across messages and tasks keeping track of the decreasing energy gain values.

2. *The energy consumption of message transmission is in general higher than that of computation and it may appear on the surface that the slack should be entirely allocated to the messages. However, from the figure 6.2 we can see that the exact energy gains depend upon several parameters like W, d, the current modulation level of the message and the current frequency level of the task.* For example, in the figure considering the curve with $W = 1KHz$ and $d = 1$, we have $G(k = 8, k = 7) < G(400MHz, 300MHz)$ where as $G(k = 9, k = 8) > G(400MHz, 300MHz)$. This suggests that there are situations (specific values of W and d) where slack should be allocated to tasks rather than messages. For the cases, where the message energy gains are higher than the task energy gains, the energy gain metric helps comparing the energy reductions offered by each message and guides the slack allocation across different messages.

We have applied the above energy gain metric for a different scheduling problem for a single-hop network model in our recent work [102]. In this chapter, we apply these techniques to the data aggregation problem.

The above energy gain metric helps in choosing the best entity in the schedule for performing slack allocation. In order to ensure such slack allocation preserves the constraints satisfied by the input schedule, we first estimate the maximum safe slack for each entity prior to allocating any slack to it. Then, all slack allocations are performed without exceeding the corresponding maximum slack limit.

The maximum safe slack for each entity, e_i in the schedule is calculated based on its latest start time $lst(e_i)$ and earliest start time $est(e_i)$. The latest start time of e_i is defined as the latest time by which the entity e_i can start in the schedule without violating any constraints. Similarly, the earliest start time of e_i is defined as the earliest time by which e_i can start without violating any constraints in the schedule. For a message M_i, its latest and earliest start times are calculated as follows:

$$lst(M_i) = \min\{lst(next(M_i)), lst(T_{p_i})\} - t(M_i) \quad (6.27)$$

$$est(M_i) = \max\{(est(T_i) + t(T_i), (est(prev(M_i)) + t(prev(M_i))\} \quad (6.28)$$

Similarly, the latest and earliest start times of a task, T_i are calculated as follows:

$$lst(T_i) = lst(M_i) - t(T_i) \quad (6.29)$$

$$est(T_i) = \max_{v_j \in V_c(i)} \{est(M_j) + t(M_j)\} \quad (6.30)$$

Finally, the maximum available safe slack for each entity e_i can be calculated as follows.

$$S_{max} = lst(e_i) - est(e_i) - t(e_i) \quad (6.31)$$

In the following, we present an example to illustrate the shortcomings of the default input schedule and the working of a straight forward slack allocation scheme. In the subsequent sections, we use the same example to demonstrate the working of the gain based scheduling algorithms.

6.4.2 Illustrative Example

Consider the simple data aggregation tree shown in figure 6.3. The numbers on the edges denote the normalized distance between the corresponding nodes. We assume a 100% data correlation i.e, $\beta_f = 1$ in this example. In the given tree, we have four messages namely, M_E, M_C, M_D, M_B where each message M_i originates at node i. Similarly, there are three aggregation tasks in the system namely, T_A, T_B, T_D. To keep the example simple we ignore other entities (M_A, T_C, T_E) in the discussion. Each message is of size $L = 1024$ bits and each task is of size $C = 10^3$ computation cycles. The channel bandwidth, $W = 1MHz$.

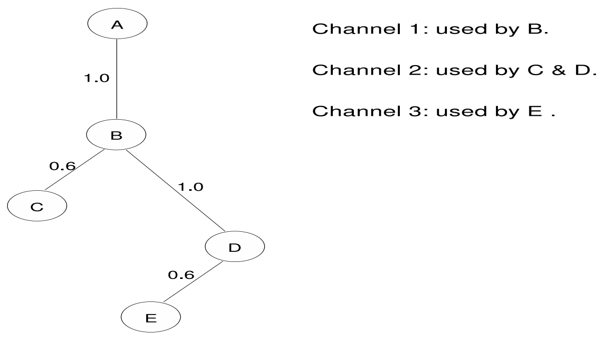


Figure 6.3 Illustrative example: Data aggregation tree

The input schedule for the data aggregation with an end-to-end deadline equal to $370\mu s$ is shown in figure 6.4. In the schedule, all the computation is shown at one place for brevity. The above input schedule operates each entity at its highest performance level while leaving a significant amount of slack unused (from time 318 to 370 in the figure).

Now, consider a straightforward *greedy slack allocation scheme* which allocates such unutilized slack to the highest energy consuming entity in the schedule. In the above example, messages M_D and M_B incur the highest energy consumption. The greedy slack allocation scheme arbitrarily chooses message M_B and allocates as much slack as possible to it. As a result the modulation level of M_B is reduced from 10 to 7 as shown in figure 6.5. Further, the left over slack cannot be utilized by any of the messages. As a result the frequency levels of all the tasks are reduced to 200 MHz. With this slack allocation, the greedy scheme reduces the energy consumption of M_B by $5745.15\mu J$ (obtained using equation 6.7) and that of each

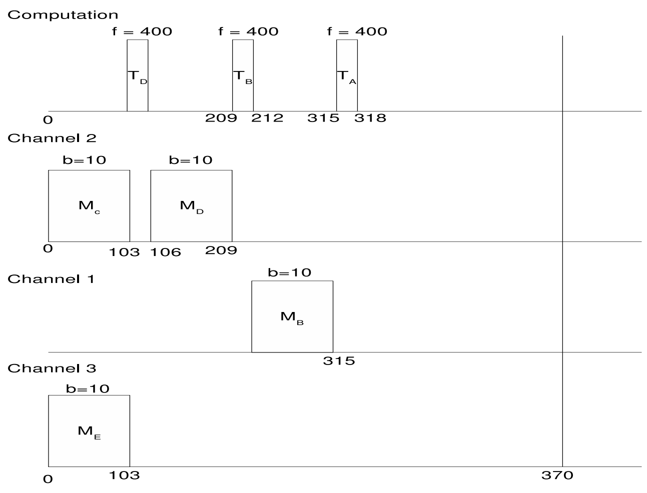


Figure 6.4 Illustrative example: Input schedule (Total energy consumption $= 18.99mJ$)

task by $2.625\mu J$ (obtained using table 6.1) which corresponds to a total energy reduction of about $5747.78\mu J$. In the following, we present gain based scheduling algorithms which apply the notion of energy gain to achieve much better energy savings.

6.4.3 Gain based Scheduling Algorithm (GSA)

We now present the gain based scheduling algorithm which assigns performance levels to each entity in the data aggregation tree while respecting the different constraints provided by the input schedule. The GSA algorithm proceeds in iterations allocating slack in an incremental fashion. In each iteration, the highest energy gain yielding entity is chosen and it is allocated just enough slack to reduce its performance mode by one level. The rest of the slack is allocated similarly to the remaining entities in the schedule. At the end of each iteration, the individual energy gain values are updated.

In order to keep track of the messages which can utilize more slack without affecting any constraints, the GSA algorithm maintains a set Q and updates it after each iteration. The algorithm terminates once the set Q becomes empty. The pseudocode for GSA is presented in Algorithm 8. It accepts an energy-unaware feasible schedule as input and outputs an energy-aware schedule where each task is assigned an operating frequency and each message is assigned

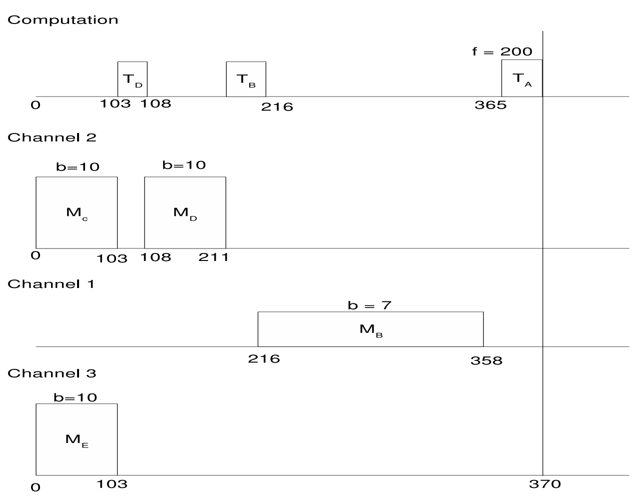


Figure 6.5 Illustrative example: Greedy slack allocation schedule (Total energy consumption = $13.25mJ$)

a modulation level. In step 2, all the messages and tasks are added in a set Q. At any point of time, the set Q contains all the messages and tasks that can accept more slack without violating deadline and precedence constraints. In step 4, the highest energy gain yielding entity is chosen from Q. In step 5 the maximum available slack for this message is determined. In step 6, the amount of slack necessary for reducing the chosen entity by one performance level is determined. If this slack is less than the available slack then its performance level is decreased by one level in step 8. Otherwise, the entity is removed from Q in step 12 as its performance level cannot be reduced further.

The worst-case computational complexity of the GSA algorithm can be derived as $O((n + m)(n\lambda_t + m\lambda_m))$ where n and m respectively denote the number of tasks and messages in the schedule.

Consider the initial schedule shown in figure 6.4. Applying the GSA algorithm, the highest energy gain yielding message M_B is picked which has an energy gain of 272.6 joules per second (obtained using equation 6.26). Consequently, the modulation level of M_B is reduced by one level. As there is more slack available, in the subsequent steps the modulation levels of M_D, M_B and M_D are reduced by one level in that order. The final GSA schedule is shown in figure 6.6. The total energy reduction obtained by the GSA schedule can be calculated as

```
    Input: Input Schedule, $H_i$
    Output: Output Schedule, $H_o$
 1  Set $H_o = H_i$;
 2  $Q$ : Set of all tasks and messages in $H_i$;
 3  while $Q \neq \emptyset$ do
 4      Pick up the highest energy gain yielding entity $e_i$ from $Q$;
 5      Determine the maximum slack, $S_{max}$ available for $e_i$
        using equation 6.31;
 6      Let $s$ be the time required to operate $e_i$ at its next lower
        performance level;
 7      if $s \leq S_{max}$ then
 8          Decrement the performance level of $e_i$ by one level;
 9          Update the schedule $H_o$;
10      end
11      else
12          Remove $e_i$ from $Q$;
13      end
14  end
```

Algorithm 8: Gain based Scheduling Algorithm

$4807.7 * 2 = 9615.4 \mu J$. In this example, GSA outperforms the greedy scheme by offering much higher energy savings.

6.4.4 Extended Gain based Scheduling Algorithm (EGSA)

Although the above presented gain based algorithm effectively reduces the energy consumption by employing the notion of energy gain, it does not exploit the parallelism offered by the tree structure of the data aggregation application. We now present the extended gain based scheduling algorithm which uses GSA as a subroutine and exploits the parallelism offered by the tree structure.

Consider the scenario shown in figure 6.7.(a), where message M_i is the highest energy gain entity in the schedule and it requires Δt additional amount of time to reduce its modulation by one level. The messages $prev(M_i)$ and M_j where $v_j \in V_c(i)$ are transmitted parallelly as their transmissions do not interfere with each other. Further, both the transmissions need to finish before M_i starts.

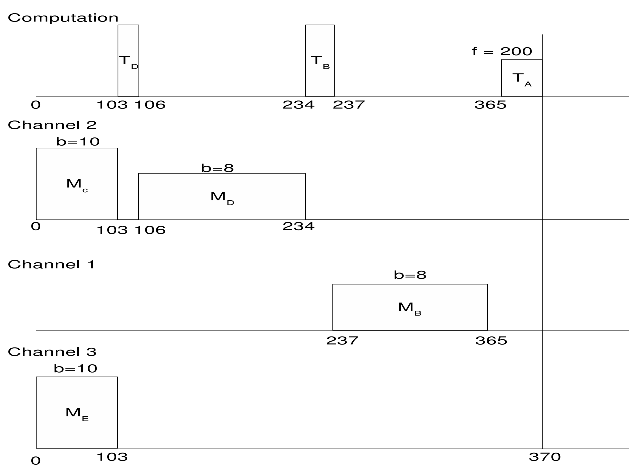


Figure 6.6 Illustrative example: GSA schedule (Total energy consumption $= 9.38mJ$)

At this point, GSA would allocate the Δt slack to message, M_i and reduce its modulation by one level resulting in a schedule shown in figure 6.7.(b). Let ΔE_1 denote the obtained energy reduction.

Now, consider an alternative where the modulation level of M_i is not reduced and the available slack Δt is utilized by both $prev(M_i)$ and M_j simultaneously as shown in figure 6.7.(c). This is possible due to the tree structure which allows both the descendants and siblings of a node to transmit parallelly. Let the energy reductions obtained by $prev(M_i)$ and M_j using no more than Δt additional slack be ΔE_2 and ΔE_3, respectively. Therefore, the total energy reduction obtained in this case is equal to $\Delta E_2 + \Delta E_3$.

Although, $\Delta E_1 > \Delta E_2$ and $\Delta E_1 > \Delta E_3$, it is possible that $\Delta E_1 < \Delta E_2 + \Delta E_3$ in which case allocating slack to $prev(M_i)$ and M_j is beneficial. This is the basic idea behind the extended gain based scheduling algorithm. It starts with a schedule produced by the GSA algorithm and for each message M_i, it evaluates the alternative where some of the M_i's slack could be simultaneously allocated to $prev(M_i)$ and descendant entities of node i (e.g., M_j in the above discussion). Based on the evaluation, it produces a schedule which yields highest energy reductions.

In this example, EGSA would verify if $(\Delta E_1 < \Delta E_2 + \Delta E_3)$. If the condition is satisfied,

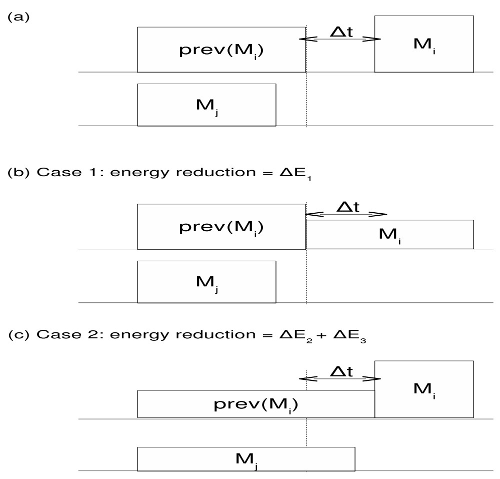

(a)
prev(M_i)
Δt
M_i
M_j
(b) Case 1: energy reduction = ΔE_1
prev(M_i)
Δt
M_i
M_j
(c) Case 2: energy reduction = ΔE_2 + ΔE_3
Δt
M_i
prev(M_i)
M_j

Figure 6.7 EGSA: Motivational example

it would follow the schedule shown in figure 6.7.(c) otherwise, it would stick with the GSA's schedule shown in figure 6.7.(b).

The pseudocode for the EGSA algorithm is presented in Algorithm 9. It uses the following terms:

- $DS(i)$: It is the descendant set of node v_i which consists of the task T_i and all the messages and tasks that belong to the descendant nodes of v_i in the aggregation tree.

- $g(\Gamma, \Delta t)$: It is the amount of energy reduction that would be obtained by applying the GSA algorithm on the entities in the Γ set to utilize the available slack, Δt. Specifically, it denotes the amount of energy reduction that would be obtained by invoking GSA and further restricting slack allocation to the entities in the Γ set. This can be accomplished by setting Q equal to Γ in step 2 of Algorithm 8.

- Extended Gain $EG(M_i, \Delta t)$: It denotes the maximum of the energy reductions that would be obtained by each of the following two options: (1) allocating slack to M_i i.e, following the GSA schedule itself (evaluated as $g(M_i, \Delta t)$), (2) allocating slack to $prev(M_i)$ and the entities in $DS(i)$ simultaneously (evaluated as $g(prev(M_i), \Delta t) + g(DS(i), \Delta t)$). Mathematically,

$$EG(M_i, \Delta t) = Max\{g(M_i, \Delta t), g(prev(M_i), \Delta t) + g(DS(i), \Delta t)\} \quad (6.32)$$

```
Input: GSA's output Schedule, $H_g$
Output: Output Schedule, $H_o$
1  Set $H_o = H_i$;
2  $Q$ : Set of all messages in $H_g$;
3  while $Q \neq \emptyset$ do
4      Pick up the message ($M_i$) operating at the least
       modulation level (say, $b_i$) from $Q$;
5      if($b_i == b_{max}$) Remove $M_i$ from $Q$ and continue;
6      Calculate $\Delta t = \frac{L_i}{Wb_i} - \frac{L_i}{W(b_i+1)}$;
7      Increment $M_i$'s modulation level to $b_i + 1$;
8      Calculate $\delta = EG(M_i, \Delta t)$ using eq. (6.32);
9      if $\delta > GSA(M_i, \Delta t)$ then
10         Update the schedule $H_o$ accordingly, as per the
           choices made in the $\delta$ calculation;
11     end
12     else
13         Decrement $M_i$'s modulation level to $b_i - 1$;
14         Remove $M_i$ from $Q$;
15     end
16 end
```

Algorithm 9: Extended Gain based Scheduling Algorithm (EGSA)

The EGSA algorithm proceeds as follows. It iterates over the messages in the set Q. In step 4, the message M_i with the least modulation level is chosen. This choice is motivated by the fact that it generates the maximum slack and incurs least energy consumption when its modulation is increased by one level. In step 5, it is verified if the modulation of M_i can be increased. If the test fails, this message is removed from the Q and the algorithm continues with the other messages in Q. In step 6, the amount of generated slack is calculated and M_i's modulation is increased by one level in the following step. Step 7 evaluates the different alternatives and calculates the best energy reduction that can be obtained using equation (6.32). If the obtained energy reductions are greater than what would be obtained by retaining the slack with M_i then the schedule is updated appropriately in step 10. Otherwise, the modulation of M_i is increased back to it original value and it is removed from the Q in steps 13 and 14.

The worst-case complexity of the algorithm can be estimated as $O(n^2\lambda_m)$ as each message is explored at most λ_m times and for each exploration the whole tree is explored in the worst case.

6.5 Performance Evaluation

We evaluated the performance of the proposed scheduling algorithms against the optimal solution obtained by solving the MILP presented in section 6.3.2.1. We used the ILOG CPLEX 10.100 software [74] to solve the MILP. We randomly generated a data aggregation tree with 100 nodes following the PTC2 tree generation algorithm presented in [103]. We assume a communication radius of $40m$. The distances between a parent and each of its child are randomly generated between $[0.05 * 40, R_f * 40]$ following a uniform distribution. Here R_f is the radius factor and is varied to study both short-range and long-range communication scenarios. In addition, we varied the following parameters: channel bandwidth (W) in hertz, data correlation factor (β_f), computational work load i.e, CPU cycles (C) per task, slack factor ($S_f = \frac{D-t_f}{D}$ where D is the deadline and t_f is the finish time of the input schedule where no performance scaling is performed. For each parameter set of interest, we performed 20 different runs each with a different random number seed and the obtained average is plotted as a single point in the graph. The communication from different child nodes to their parent is scheduled following a FCFS (first-come-first-serve) scheduling policy and the obtained schedule serves as the input to the above presented algorithms.

The performance metric in our simulation studies is the normalized total energy consumption of the schedule i.e, the total energy consumption of the schedule normalized with respect to the energy consumption of the input schedule. We compared the performance of the following algorithms in our studies: (1) Gain based computation only (comp-only) - variation of GSA where only tasks are considered, (2) Gain based communication only (comm-only) - variation of GSA where only messages are considered, (3) Greedy, (4) GSA, (5) EGSA and (6)

MILP. In the following we present two sets of results. In the first set, we assume $W = 1KHz$ (low bandwidth) and for the second we set $W = 1000KHz$ (high bandwidth).

6.5.1 Results for low bandwidth conditions

Effect of radius factor (R_f): Figure 6.8 shows the relative performance of the above schemes by varying the radius factor. The other parameters are chosen as: $s_f = 1.0, C = 10^6, \beta_f = 1.0$. With increasing R_f, the total energy consumption increases as the messages now have to be communicated over a longer range. However, the increase in the total energy consumption of the different schemes which employ DMS is much smaller than that of the input schedule where no performance scaling is performed. As a result, the normalized energy consumption values appear to be decreasing. As the comp-only scheme does not employ any message scaling its normalized energy consumption increases as expected. At low values of R_f, the computation energy dominates the communication energy for the chosen parameters. Therefore, it is very important to allocate the available slack to tasks rather than messages. Consequently, comp-only scheme performs better than the comm-only scheme in the range $R_f \in [0.05, 0.20]$ showing a maximum improvement of 14% over the comm-only scheme. As the communication range is increased, the communication energy and the benefits offered by scaling the messages increase as a result the comm-only scheme performs better than comm-only after $R_f = 0.20$. Although the greedy scheme considers both tasks and messages, due to its aggressive nature of slack allocation it performs worse than the comm-only scheme after $R_f = 0.3$. Throughout the range, both the GSA and EGSA perform better than the comp-only, comm-only and greedy schemes and offer a performance comparable to the MILP. At $R_f = 0.20$ both GSA and EGSA incur as small as 2.6% more energy than the MILP. Further, EGSA yields an average improvement of about 15% over the GSA scheme.

Effect of computational workload (C): Figure 6.9 shows the relative performance of the above schemes by varying the number of CPU cycles per task. The other parameters are chosen as: $R_f = 0.1, s_f = 0.5, \beta_f = 1.0$. For a given slack factor, with increasing C the total energy consumption increases due to increase in the workload and decrease in the available slack for

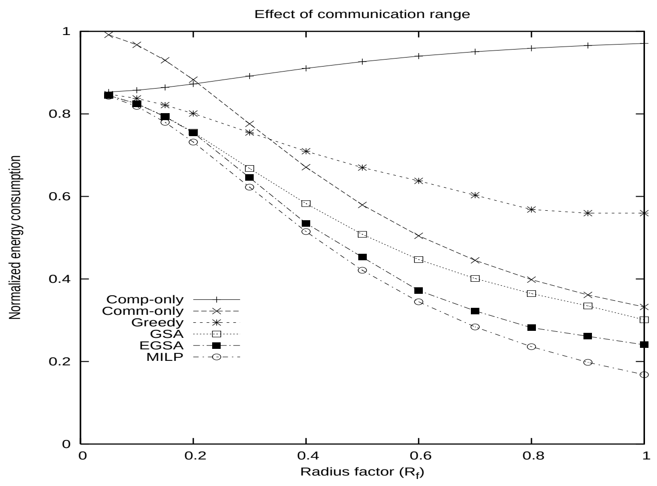


Figure 6.8 Effect of radius factor (R_f)

performance scaling. All the schemes show this trend in the figure. Also, the computation energy dominates the communication energy as result with increasing C. As a result, the comp-only scheme performs better than the comm-only scheme showing an average improvement of 8% over the comm-only scheme. Throughout the range, both the GSA and EGSA perform better than the comp-only, comm-only and greedy schemes and offer a performance comparable to the MILP. At $C = 2 * 10^5$ and $C = 6 * 10^5$ EGSA respectively incurs as small as 6.6% and 4.8% more energy than the MILP. Further, EGSA yields an improvement of about 14% over the Greedy scheme at $C = 10^5$. At low values of C the gain based scheme utilizes the ample slack to reduce the modulation level of the highest gain message in an aggressive manner. On the other hand, the EGSA scheme converts such aggressive slack allocations by employing better slack allocation strategy as a result, it offers better performance at low values of C.

6.5.2 Results for high bandwidth conditions

Effect of radius factor (R_f): Figure 6.10 shows the relative performance of the above schemes by varying the radius factor. The other parameters are chosen as: $s_f = 0.5, C = 10^6, \beta_f = 0.5$. The plot follows a similar trend depicted in figure 6.8. Throughout the range, both the GSA and EGSA perform better than the comp-only, comm-only and greedy schemes and offer a performance comparable to the MILP. At $R_f = 0.10$ and $R_f = 0.80$ EGSA respec-

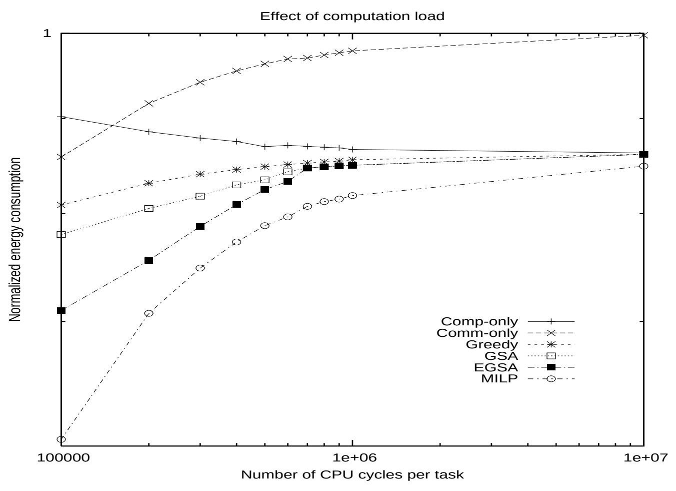


Figure 6.9 Effect of computational workload (C)

tively incurs as small as 5.8% and 4% more energy than the MILP.

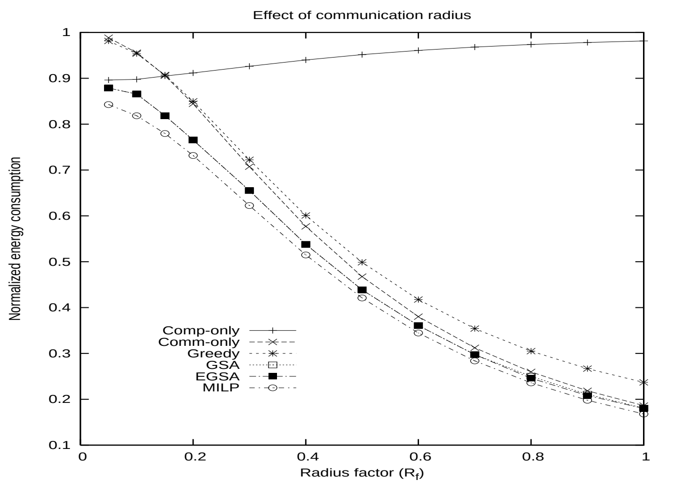


Figure 6.10 Effect of radius factor (R_f)

Effect of slack factor (S_f): Figure 6.11 shows the relative performance of the above schemes by varying the slack factor. The other parameters are chosen as: $R_f = 1.0, C = 10^6, \beta_f = 0.5$. The total energy consumption of the different schemes decrease with increasing S_f due to the increased opportunity for performance scaling. All the schemes show this trend in the figure. For high bandwidth and long range scenarios considered here, the communication energy heavily dominates the computation energy and hence comm-only scheme gives a performance close to that of GSA and EGSA. The comp-only scheme shows poor performance for the same reason. In the figure, EGSA shows as much as 80% improvement over the comp-only scheme and incurs about 12% more energy than the MILP.

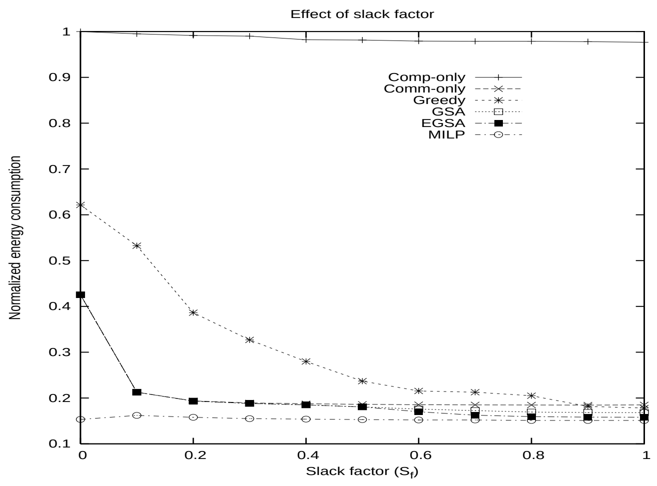


Figure 6.11 Effect of slack factor (S_f)

6.6 Discussion

In this chapter, we addressed the problem of scheduling messages and tasks in a data aggregation tree with the objective of minimizing the total energy consumption while meeting the end-to-end latency constraint. We considered a model where each node in the tree supports both DVS and DMS power management techniques. We presented analytical problem formulations both for the ideal and practical performance scaling models. We also proved that the scheduling problem is NP-Hard for the practical model and presented two heuristic scheduling algorithms which allocate slack across different messages and tasks based on the energy gain metric. We evaluated the performance of the proposed algorithms for a variety of scenarios by varying the channel bandwidth, communication range and computational work load in the network. Our results show that the energy savings obtained by the EGSA algorithm are comparable to that of the optimal solution obtained by solving the MILP. In our future work, we plan to explore the energy-time tradeoffs offered by other power management techniques like DCS and utilize these tradeoffs to address the above scheduling problem.

CHAPTER 7. Conclusions and Future Work

In this dissertation, we addressed the energy management problem in networked embedded real-time systems. We provided a comprehensive solution to this problem by addressing the energy management at computing and communication subsystems of individual nodes in the network.

Our goal was to design effective energy management strategies that would appropriately leverage the low power modes supported by different components within each node of the network to maximize overall system's energy savings while guaranteeing the necessary real-time constraints. We made the following contributions:

- *Energy management at Computing System*: Focusing on computation energy management, we addressed the energy-aware real-time task scheduling problem where the objective is to minimize the processor energy consumption while guaranteeing all the task deadlines. We employed the well known Dynamic Voltage Scaling (DVS) technique as the basic underlying energy management mechanism. The proposed scheduling algorithm exploits the control flow graph (CFG) information of each task and achieves a better idea of the exact workload in the system at run-time as early as possible. Further, this information was used to utilize the available slack to benefit the entire task-set as opposed to using it within a single task. Our simulation studies show the presented algorithm performs significantly better than the existing energy-aware task scheduling algorithms. In our future work, we plan to extend the proposed early execution concepts to periodic task system with shared resources. The early execution principle in such a system requires taking into account not only the scheduler's operation, but also the rules of the resource access control protocol.

- *Energy management at Communication System*: We addressed the energy-aware message scheduling problem where the goal is to schedule a given set of periodic messages each associated with a source and destination nodes in a single hop network. The objective of the problem is to minimize the total communication energy consumption while meeting all the message deadlines and reliability constraints. Unlike the task scheduling problem where all the tasks have identical energy-time functions, in the message scheduling problem, each message has an associated source-destination distance which governs the nature of the energy-time function of that message. As a result, this problem is fundamentally different from the task scheduling problem mentioned above. We addressed the message scheduling problem for two different models and presented analytical solutions to solve the problem optimally in both cases. Further, we also presented a polynomial time heuristic scheduling algorithm for the case where the problem becomes NP-Hard. In our future work, we plan to address the above problems for a different reliability model where the required message reliability is achieved via retransmissions instead of a single message transmission with high transmission power. We believe such a model results in lesser system-level energy consumptions and offers greater opportunities for energy management.

- *System-level Energy Management - Single-hop*: We made two major contributions here. First, we analyzed the system-level energy-time tradeoffs considering both tasks and messages in the system. This analysis can be used to solve any scheduling where slack needs to be distributed across tasks and messages in the system. Further, we addressed the specific problem of scheduling a set precedence constrained, message passing periodic tasks. Assuming a feasible schedule is available, we proposed an energy-aware scheduling algorithm which assigns CPU frequency to each task and modulation level to each message in the system. Further, we also presented dynamic scheduling algorithms which will utilize the dynamically generated slack in the system to further reduce the total energy consumption. In our future work, we plan to extend this research by providing an optimal solution through analytical approaches like integer linear programming formulation

(ILP) and compare the performance of our algorithms against the same.

- *System-level Energy Management - Multi-hop*: We addressed the problem of minimizing the total energy consumption of data aggregation with an end-to-end latency constraint while taking into account both the computational and communication workloads in the network. We then presented an analytical problem formulation for the ideal case where each node can scale its frequency and modulation continuously. We also presented a Mixed Integer Linear Programming (MILP) formulation to obtain the optimal solution for the practical case where only few discrete frequency and modulation levels are supported by each node. Further, we presented polynomial time heuristic algorithms, which employ the energy-gain metric established earlier. We evaluated the performance of the proposed algorithms for a variety of scenarios and our results show that the energy savings obtained by the proposed algorithms are comparable to that of the MILP solution. In our future work, we plan to develop online algorithms which would utilize the dynamic slack that is generated in the schedule due to temporal and spatial data correlations. We believe exploiting dynamic slack can further enhance the overall energy savings of the system.

Broader Impacts

Beside the intellectual and academic contributions of this dissertation, we expect the following impact on the research community and industry, as follows:

- By demonstrating that system-level energy management yields better energy savings, the work presented in this dissertation greatly widens the scope of the energy management in networked real-time embedded systems. As a consequence, we strongly believe that more research efforts will be focused along this direction by different research groups working in this area. To support this, in the next subsection, we present a list of practical and challenging open research problems that can be addressed based on our work here.

- Mobile device manufacturers like Apple and Sony can benefit from the cross-layer energy management algorithms presented in this thesis to implement an effective energy manager in each device e.g., cell phone or a mobile gaming device. The number of computation-intensive applications that are stacked into a single device are gradually increasing. For example, a state-of-the-art cell phone does video and audio playing in addition to performing the conventional communication tasks like making calls to people. This impending trend demands an effective energy manager which reduces overall energy consumption by adapting to the application behavior dynamically. The cross-layer research presented in this thesis directly addresses this very issue.

- Companies like sensinet and crossbow deploy sensor networks for a variety of applications ranging from environmental monitoring/learning to military surveillance. These applications require a capable system-level energy manager to ensure efficient utilization of available energy and elongate the network lifetime. Such companies can benefit by using different system-level energy management principles and algorithms presented in this thesis.

Future Work

Several exciting energy-aware research problems can be addressed in this relatively unexplored area of networked real-time embedded systems. The work presented in this dissertation opens up research in the following directions.

- The system-level energy management work presented in this thesis employs both communication energy management (DMS) and computation energy management (DVS) techniques in tandem to achieve significant energy savings. Following the system-level principles presented in this thesis, other communication energy management techniques (in place of DMS) can be explored in this framework to achieve better energy savings. Similar to the DMS technique, a variety of physical layer techniques exist which allow to gracefully tradeoff performance for communication energy savings. Such techniques include adapting the bit-error-rate and dynamically scaling code size. Each such low-level power management technique can be explored in conjunction with DVS. Each such combination needs further investigation and can be employed to obtain system-level energy savings.

- The energy management problem for a general multi-hop networked embedded real-time system has not been solved yet. Although the solutions presented in this thesis cannot be directly applied to solve the above problem, we believe that the basic principles presented are still applicable. It would be interesting to extend the presented work to address the general multi-hop problem.

- Further, the work presented in this dissertation primarily focussed on minimizing the overall energy consumption of the networked real-time embedded system. An equally important and yet to be solved problem is system-level lifetime maximization wherein, the objective is to maximize the overall lifetime of the system by appropriately leveraging both computation and communication energy management techniques while guaranteeing the required real-time deadline and channel reliability constraints. We believe that the system-level energy management principles derived in this dissertation can serve as a basic ingredient in solving this problem, however, more work needs to be done to obtain a complete solution.

- Finally, all the above listed problems as well as the system-level energy management work presented in this thesis can be re-addressed by considering: (1) resource management and access control issues along with the (2) the task preemptions. Considering these issues makes the problem more challenging while addressing a wider range of practical applications.

BIBLIOGRAPHY

[1] Coordination of Safety-Critical Mobile Real-Time Embedded Systems, *Workshop on Research Directions for Security and Networking in Critical Real-Time and Embedded Systems,* San Jose, CA, USA, 2006, apr, TCD-CS-2006-16, http://www.cs.tcd.ie/publications/tech-reports/reports.06/TCD-CS-2006-16.pdf

[2] J. P. Loyall, R. E. Schantz, D. Corman, J. L. Paunicka, and S. Fernandez, "A distributed real-time embedded application for surveillance, detection, and tracking of time critical targets," in Proc. *IEEE Real Time and Embedded Technology and Applications Symposium (RTAS),* pp. 88-97, 2005.

[3] C.D. Gill, R.K. Cytron, and D.C. Schmidt, "Multiparadigm scheduling for distributed real-time embedded computing," *Proc. the IEEE,* vol. 91, no. 1, pp. 83-97, Jan. 2003.

[4] J.A. Stankovic, "Distributed Real-time Computing: The Next Generation," *Journal of the Society of Instrument and Control Engineers of Japan,* 1992.

[5] K. Ramamritham, J.A. Stankovic, and W. Zhao, "Distributed scheduling of tasks with deadlines and resource requirements," *IEEE Trans. Computers,* vol. 38, no. 8, pp. 1110-1123, Aug. 1989.

[6] D.T. Peng and K.G. Shin, "Static allocation of periodic tasks with precedence constraints in distributed real-time systems,"In *Proc. Intl. Conf. on Distributed Computing Systems,* June 1989.

[7] K. Ramamritham, "Allocation and scheduling of precedence-related periodic tasks," *IEEE Trans. Parallel and Distributed Systems,* vol.6, no.4, pp.412-420, Apr. 1995.

[8] D.-T. Peng, K. G. Shin, and T. F. Abdelzaher, " Assignment and scheduling communicating periodic tasks in distributed real-time systems," *IEEE Trans. on Software Engineering,* vol. 23, no. 12, pp. 745-758, 1997.

[9] I. Santhoshkumar, G. Manimaran, and C. Siva Ram Murthy, "A Pre-run-time scheduling algorithm for object-based distributed real-time systems," *Journal of Systems Architecture,* vol.45, no.14, pp.1169-1188, July 1999.

[10] T. D. Burd and R. W. Brodersen, "Energy efficient CMOS microprocessor design," in Proc. *International Conf. on System Sciences*, pp. 288-297, Jan. 1995.

[11] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in Proc. *ACM Symposium on Operating System Principles*, pp.89-102, 2001.

[12] S. Lee and T. Sakurai, "Run-time voltage hopping for low-power real-time Systems," in Proc. *ACM Design Automation Conference (DAC)*, pp.806-809, 2000.

[13] A. Allavena and D. Moss, "Scheduling of frame-based embedded systems with rechargeable batteries", *Workshop on Power Management for Real-Time and Embedded Systems,* (in conjunction with RTAS 2001).

[14] C. Rusu, R. Melhem and D. Mosse, "Maximizing rewards for real-time applications with energy constraints," *IBM Journal of R&D*, vol. 46, no. 5/6, 2003.

[15] H. Aydin, R. Melhem, D.Moss, and P.M. Alvarez, "Determining optimal processor speeds for periodic real-time tasks with different power characteristics", in Proc. *Euromicro Conference on Real-Time Systems*, 2001.

[16] T. A. AlEnawy and H. Aydin, "Energy-constrained performance optimizations for real-time operating systems," in Proc. *Workshop on Compilers and Operating Systems for Low Power (COLP)*, Sept. 2003.

[17] H. Aydin, R. Melhem, D. Mosse, and P.M. Alvarez, "Power-aware scheduling for periodic real-time tasks," *IEEE Trans. Computers*, vol.53, no.5, pp.584-600, May 2004.

[18] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in Proc. *International Symposium on low power electronics and design,* pp. 197-202, 1998.

[19] D. Shin and J. Kim, "A Profile-based energy-efficient intra-task voltage scheduling tlgorithm for hard real-time applications," in Proc. *International Symposium on Low Power Electronics and Design (ISLPED)*, Aug. 2001.

[20] D. Shin, J. Kim, and S. Lee. "Intra-task voltage scheduling on DVS-enabled hard real-time systems," *IEEE Transactions on CAD*, vol.24, no.9, Sep. 2005.

[21] D. Shin, J. Kim, and S. Lee. "Intra-task voltage scheduling for low-energy hard real-time applications," *IEEE Design & Test of Computers*, vol.18, no.2, pp.20-30, 2001.

[22] G. Sudha Anil Kumar, G. Manimaran, and Z. Wang, "End-to-end energy management in networked real-time embedded systems," *IEEE Trans. on Parallel and Distributed Systems (TPDS)*, vol. 19, no. 11, pp. 1498-1510, Nov. 2008.

[23] P. Ganesan, R. Venugopalan, P. Peddabachagari, A. Dean, F. Mueller and M. Sichitiu, "Analyzing and modeling encryption overhead for sensor network nodes", *in Proc. of ACM Wireless Sensor Networks and Applications*, 2003, pp. 151-159.

[24] http://euler.slu.edu/ fritts/mediabench/mb2/index.html

[25] A. Azevedo, I. Issenin, R. Cornea, R. Gupta, N. Dutt, A. Veidenbaum, and A. Nicolau, "Profile-based dynamic voltage scheduling using program Checkpoints in the COPPER framework," in Proc. *Design Automation and Test in Europe,* 2002.

[26] D. Shin and J. Kim, "Look-ahead intra-task voltage scheduling using data flow information," in Proc. *ISOCC,* pp.148-151, Oct. 2004.

[27] J. Seo, T. Kim, and K. S. Chung, “Profile-based optimal intra-task voltage scheduling for hard real-time applications,” in Proc. *ACM Design Automation Conference (DAC)*, pp.87-92, June 2004.

[28] D. Mosse, H. Aydin, B. Childers, and R. Melhem, “Compiler-assisted dynamic power-aware scheduling for real-time applications,” in Proc. *Workshop on Compilers and Operating Systems for Low-Power*, Oct 2000.

[29] Y. Zhu and F. Mueller, ”Exploiting Synchronous and Asynchronous DVS for Feedback EDF Scheduling on an Embedded Platform”, *ACM Transactions on Embedded Computing Systems*, Vol. 7, No. 1, Dec 2007, pages 1-26.

[30] Y. Zhu and F. Mueller, ”Feedback EDF Scheduling Exploiting Dynamic Voltage Scaling,” *in Proc. of IEEE Real-Time and Embedded Technology and Applications Symposium*, 2004, pp.84-93.

[31] N. AbouGhazaleh, B. Childers, D. Mosse, R. Melhem, and M. Craven, “Energy management for real-time embedded applications with compiler support,” *ACM SIGPLAN, LCTES’03,* June 2003.

[32] H. Huang, K. G. Shin, C. Lefurgy, K. Rajamani, T. W. Keller, E. V. Hensbergen, and F. L. Rawson, “Software-Hardware Cooperative Power Management for Main Memory,” in Proc. *PACS*, 61-77, 2004.

[33] O.S. Unsal and I. Koren, “System-level power-aware design techniques in real-time systems,” *Proceedings of the IEEE,* vol. 91, no. 7, pp. 1055-1069, July 2003.

[34] A. Mahesri and V. Vardhan, “Power Consumption Breakdown on a Modern Laptop,” in Proc. *Workshop on Power Aware Computing Systems,* in conjunction with Intl. Symp. Microarchitecture, Dec. 2004.

[35] K. I. Farkas, J. Flinn, G. Back, D. Grunwald, and J. Anderson, “ Quantifying the energy consumption of a pocket computer and a Java virtual machine,” in Proc. *ACM SIGMETRICS,* pp. 252-263, June 2000.

[36] C. Schurgers, O. Aberthorne and M. B. Srivasthava, "Modulation Scaling for Energy Aware Communication Systems", *ISLPED*, 2001.

[68] V. Raghunathan, C. Schurgers, S. Park, and M. B. Srivastava, "Energy-aware wireless microsensor networks," *IEEE Signal Processing Magazine,* vol.19, no.2, pp. 40-50, Mar. 2002.

[38] C. Schurgers, V. Raghunathan, and M. B. Srivastava, "Modulation scaling for real-time energy aware packet scheduling," in Proc. *IEEE Globecom,* pp.3653-3657, Nov. 2001.

[76] V. Raghunathan, S. Ganeriwal, C. Schurgers, and M. B. Srivastava, "E2WFQ: An energy efficient fair scheduling policy for wireless systems," in Proc. *International Symposium on Low Power Electronics and Design (ISLPED),* pp. 30-35, Aug. 2002.

[40] V. Raghunathan, C.L. Pereira, M.B. Srivastava and R.K. Gupta, "Energy-aware wireless systems with adaptive power-fidelity tradeoffs",*IEEE Trans. on VLSI Systems*, Vol.13, Issue 2, Feb 2005 pp. 211-225.

[41] K. Chen, S. Shah, and K. Nahrstedt, "Cross-Layer design for data accessibility in mobile ad-hoc networks," *Wireless Personal Communications,* vol. 21, no.1, pp.49-76, 2002.

[42] Q. Wang and M. Ali Abu-Rgheff, "Cross-layer signaling for next-Generation wireless systems," in Proc. *WCNC,* 2003.

[43] F. A. Samimi, P. K. McKinley, S. M. Sajdadi, and P. Ge, "Kernel-middleware interaction to support adaptation in pervasive computing environments," in Proc. *Workshop on Middleware for Pervasive and Adhoc Computing,* 2004.

[44] W. Yuan and K. Nahrstedt, " Process group management in cross-layer adaptation," in Proc. *SPIE/ACM Multimedia Computing and Networking,* 2004.

[45] Y. Koucheryavy, D. Moltchanov, J. Harju, and G. Giambene, ' 'Cross-layer black box approach to performance evaluation of next-generation mobile networks,' in Proc. *NEW2AN,* pp.273-281, 2004.

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

[47] J. R. Lorch and A. J. Smith, "Improving Dynamic Voltage Scaling Algorithms with PACE," *in Proc. of ACM SIGMETRICS International Conference on Measurements and Modeling of Computer Systems*, June. 2001, pp. 50-61.

[48] W. Yuan and K. Nahrstedt, "Energy-Efficient Soft Real-Time Scheduling for Mobile Multimedia Systems," *in Proc. of ACM Symposium on Operating systems Principles*, 2003, pp. 149-163.

[49] R. Xu, C. Xi, R. Melhem and D. Mosse, "Pratical PACE for Embedded Systems", *in Proc. of 4th ACM International Conference on Embedded Software*, Sept. 2004, pp. 54-63.

[50] R. Xu, D. Mosse and R. Melhem,"Minimizing Expected Energy for Real-Time Embedded Systems", *in Proc. of 5th ACM SIGBED International Conference on Embedded Software*, Sept. 2005, pp. 251-254.

[78] K. Seth, A. Anantaraman, F. Mueller, E. Rotenberg, "FAST: Frequency-Aware Static Timing Analysis," *in Proc. of IEEE Real-Time Systems Symposium*, Dec. 2003, pp. 40-51.

[52] S. Mohan, F. Mueller, W. Hawkins, M. Root, D. Whalley and C. Healy, "ParaScale: Exploiting Parametric Timing Analysis for Real-Time Schedulers & Dynamic Voltage Scaling," *in Proc. of IEEE Real-Time Systems Symposium*, Dec. 2005.

[79] H. Aydin, R. Melhem, D. Mosse & P. Mejia-Alvarez, "Power-Aware Scheduling for Periodic Real-Time Tasks", *IEEE Transactions on Computers*, vol. 53, no. 5, pp. 584 - 600, 2004.

[54] G. S. Anil Kumar and G. Manimaran, “An intra-task DVS algorithm exploiting program path locality for real-time embedded systems” *in Proc. of IEEE International Conference on HiPC*, Dec. 2005, pp.225-334.

[55] N. AbouGhazaleh, D. Mosse, B. Childers, R. Melhem, and M. Craven “Collaborative Operating System and Compiler Power Management for Real-Time Applications,” *in Proc of IEEE Real-Time and Embedded Technology and Applications Symposium*, 2003, pp. 133-141.

[56] F. Gruian, ”On energy reduction in hard real-time systems containing tasks with stochastic execution times,” *in Proc. of IEEE Workshop on Power Management for Real-Time Embedded Systems*, 2001, pp.11-16.

[82] *www.intel.com/design/pca/applicationsprocessors/manuals/27878002.pdf*

[58] *www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/30417.pdf*

[59] *www.fsmlabs.com*

[60] W. Ye, J. Heidemann and D. Estrin “ An Energy-Efficient MAC Protocol for Wireless Sensor Networks,” *Proc. of IEEE INFOCOM*, 2002

[61] R. Cohen and B. Kapchits, “An Optimal Algorithm for Minimizing Energy Consumption while Limiting Maximum Delay in a Mesh Sensor Network”, *Proc. of IEEE INFOCOM*, May 2007

[62] E. Altman, K. Avrachenkov, G. Miller and B. Prabhu, “Discrete Power Control: Cooperative and Non-Cooperative Optimization”, *Proc. of IEEE INFOCOM*, May 2007.

[63] S. Banerjee and A. Misra, “Power Adaptation based Optimizations for Energy-Efficient Reliable Wireless Paths”, *Networking*, Athens, Greece, May 2004.

[64] W. Chen, M. J. Neely, and U. Mitra, “Energy Efficient Scheduling with Individual Packet Delay Constraints: Offline and Online Results,” *Proc. of IEEE INFOCOM*, May 2007

[65] C. Schurgers, “Energy-Aware Communication Systems,” *Ph.D. Thesis* in Electrical Engineering, University of California, Los Angeles, 2002.

[66] B. Prabhakar, E. Uysal-Biyikoglu and A. El Gamal “Energy-efficient Transmission over a Wireless Link via Lazy Packet Scheduling,” *Proc. of IEEE INFOCOM*, 2001.

[67] W. Chen and U. Mitra, "Energy Efficient Scheduling with Individual Packet Delay Constraints," *Proc. of IEEE INFOCOM*, 2006.

[68] C. Schurgers, O. Aberthorne and M. B. Srivasthava, "Modulation Scaling for Energy Aware Communication Systems", *ISLPED*, 2001.

[69] A. El Gamal, C. Nair, B. Prabhakar, E. Uysal-Biyikoglu and S. Zahedi "Energy-efficient Scheduling of Packet Transmissions over Wireless Networks," *Proc. of IEEE INFOCOM*, 2002.

[70] Y. Yang, B. Krishnamachari and V.K. Prasanna, "Energy Minimization for Real-Time Data Gathering in Wireless Sensor Networks", *IEEE Trans. on Wireless Communications*, Vol. 5, No. 11, Nov. 2006.

[71] C. Siva Ram Murthy and G. Manimaran, "Resource Management in Real-time Systems and Networks", *MIT Press, USA*, April 2001.

[72] K. Dudzinski and S. Walukiewicz, " Exact methods for the knapsack problem and its generalizations,", *European Journal of Operations Reseach*, vol. 28, 1987, pp. 3-21.

[73] S. Boyd and L. Vandenberghe, " Convex Optimization," *Cambridge University Press*, 2004.

[74] www.cplex.com

[75] V. Raghunathan, S. Ganeriwal, C. Schurgers, and M. B. Srivastava, "E2WFQ: An energy efficient fair scheduling policy for wireless systems," *in Proc. of ISLPED*, pp. 30-35, Aug. 2002.

[76] C. Schurgers, V. Raghunathan, and M. B. Srivastava, "Modulation scaling for real-time energy aware packet scheduling," *in Proc. IEEE Globecom*, pp.3653-3657, Nov. 2001.

[77] V. Raghunathan, C.L. Pereira, M.B. Srivastava and R.K. Gupta, "Energy-aware wireless systems with adaptive power-fidelity tradeoffs," *IEEE Trans. on VLSI Systems*, Vol.13, no. 2, 2005 pp. 211-225.

[78] K. Seth, A. Anantaraman, F. Mueller, E. Rotenberg, "FAST: Frequency-Aware Static Timing Analysis," *in Proc. of IEEE RTSS*, pp.40-51, Dec. 2003.

[79] H. Aydin, R. Melhem, D. Moss & P. Mejia-Alvarez, "Power-Aware Scheduling for Periodic Real-Time Tasks", *IEEE Trans. on Computers*, vol. 53, no. 5, pp. 584 - 600, 2004.

[80] H. Aydin, V. Devadas and D. Zhu, "System-level Energy Management for Periodic Real-Time Tasks", *In Proc. of IEEE RTSS*, Dec. 2006.

[81] Bren Mochocki, Dinesh Rajan, Xiaobo Sharon Hu, Christian Poellabauer, Kathleen Otten, and Thidapat Chantem, "Network-Aware Dynamic Voltage and Frequency Scaling", *In Proc. of IEEE RTAS*, April 2007.

[82] *www.intel.com/design/pca/applicationsprocessors/manuals/27878002.pdf*

[83] J. S. Pathasuntharam, A. Das and P. Mohapatra, "A Flow Control Framework for Improving Throughput and Energy Efficiency in CSMA/CA based Wireless Multihop Networks", *WOWMOM'06*, pp.143-149, 2006.

[84] C. Zhu and M. S. Corson, "QoS routing for mobile ad hoc networks", *IEEE Infocom*, June 2001

[85] J.M. Miller, C. Sengul and I. Gupta, "Exploring the Energy-Latency Trade-Off for Broadcasts in Energy-Saving Sensor Networks", *ICDCS*, 2005.

[86] A.G. Ruzzelli, L. Evers, S. Dulman, L.F.W. van Hoesel and P.J.M. Havinga, "On the design of an energy-efficient low-latency integrated protocol for distributed mobile sensor networks", *Intl. Workshop on Wireless Ad-Hoc Networks*, 2004, pp.35-44

[87] L. Y. Zhang, G. Ye and J. Hou, "Energy-efficient real-time scheduling in IEEE 802.11 wireless LANs", *ICDCS*, 2003, pp. 658-677

[88] Y. Yang, B. Krishnamachari and V.K. Prasanna, "Energy-latency tradeoffs for data gathering in wireless sensor networks", *IEEE Infocom*, 2004.

[89] R. Mangharam, S. Pollin, B. Bougard, R. Rajkumar, F. Catthoor, L. V. Perre and I. Moemann, “Optimal Fixed and Scalable Energy Management for Wireless Networks”, *in Proc. of IEEE Infocom*, 2005.

[90] S. Choi and K.G. Shin, “A Unified Wireless LAN Architecture for Real-Time and Non-Real-Time Communication Services”, *IEEE/ACM Transactions on Networking*, vol. 8, no. 1, pp.44-59, 2000.

[91] H. Li, P. Shenoy and K. Ramamritham, “Scheduling messages with deadlines in multi-hop real-time sensor networks”, *in Proc. of IEEE RTAS*, pp. 415-425, May 2005.

[92] Q. Liu, S. Zhou, and G.B.Giannakis, “Cross-Layer Scheduling With Prescribed QoS Guarantees in Adaptive Wireless Networks”, *IEEE Journal on Selected Areas in Communications*vol. 23, No.5, May 2005.

[93] S. Pollin, R. Goyens, W. Cleeren and B. Bougard, “Cross-Layer Energy-Throughput Evaluation of Multi-hop/path Communication and Link Adaptation for IEEE 802.11a”, *in Proc. of SIPS*, 2005.

[94] J. H. Chang and L. Tassiulas, “Energy Conserving Routing in Wireless Ad-Hoc Networks”, *in proc. of IEEE Infocomm*, pp. 22-31, 2000.

[95] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, “Introduction to Algorithms,” *MIT press*, Cambridge, MA, U.S.A, 2001.

[96] K. Ramamritham, “Allocation and scheduling of precedence-related periodic tasks,” *IEEE Trans. Parallel and Distributed Systems*, vol.6, no.4, pp.412-420, Apr. 1995.

[97] J. Proakis, “Digital Communications,” *McGraw-Hill*, 4th Ed., 2001.

[98] B. przydatek, D. Song, and A. Perrig, “ Secure information aggregation in sensor networks,” *in Proc. of ACM Intl. Conf. on Embedded Networked Sensor Systems*, Nov. 2003.

[99] Y. Le Borgne, G. Bontempi, "Round robin cycle for predictions in wireless sensor networks," *Proc. of 2nd Intl. Conf. on Intelligent sensors, Sensor networks and Information Processing*, 2005.

[100] A. Desphande, C. Guestrin, S. Madden, J. Hellerstein, W. Hong, "Model driven data acquisition in sensor networks," *Proc. of the 30th VLDB conference*, 2004.

[101] L. Tong, Q. Zhao, and G. Mergen, " Multipacket reception in random access wireless networks: From signal processing to optimal MAC," *IEEE Comm. Mag.*, vol. 39, no. 11, pp. 108-112, Nov. 2001.

[102] G. Sudha Anil Kumar, G. Manimaran, and Z. Wang, "Energy-aware Scheduling of Real-Time Tasks in Wireless Networked Embedded Systems," *in Proc. of IEEE Real-Time Systems Symposium*, pp. 15-24, Tucson, AZ, Dec. 2007.

[103] S. Luke and L. Panait, "A Survey and Comparison of Tree Generation Algorithms," *In Proc. of Genetic and Evolutionary Computation Conference*, 2001.

[104] G. Sudha Anil Kumar, G. Manimaran, and Z.Wang, "Energy-aware scheduling with probabilistic deadline constraints in wireless networks", *Elsevier Journal on Ad Hoc Networks*, vol. 7, no. 7, pp. 1400-1413, Sep. 2009.

[105] G. Sudha Anil Kumar, G. Manimaran, and Z. Wang, Energy-aware scheduling with deadline and reliability constraints in wireless networks, *in Proc. of IEEE Broadnets - Wireless Networks Symposium*, pp. 96-105, Raleigh, NC, Sept. 2007.