This Dissertation

entitled

Real-Time Scheduling in Cyber-Physical System

typeset with NDdiss2$_\varepsilon$ v3.0 (2005/07/27) on August 17, 2014 for

Shengyan Hong

This LaTeX 2$_\varepsilon$ classfile conforms to the University of Notre Dame style guidelines established in Spring 2004. However it is still possible to generate a non-conformant document if the instructions in the class file documentation are not followed!

> Be sure to refer to the published Graduate School guidelines at `http://graduateschool.nd.edu` as well. Those guidelines override everything mentioned about formatting in the documentation for this NDdiss2$_\varepsilon$ class file.

It is YOUR responsibility to ensure that the Chapter titles and Table caption titles are put in CAPS LETTERS. This classfile does *NOT* do that!

This page can be disabled by specifying the "`noinfo`*" option to the class invocation.* (i.e.,`\documentclass[...,noinfo]{nddiss2e}`)

This page is *NOT* part of the dissertation/thesis, but MUST be turned in to the proofreader(s) or the reviwer(s)!

NDdiss2$_\varepsilon$ documentation can be found at these locations:

`http://www.gsu.nd.edu`
`http://graduateschool.nd.edu`

Real-Time Scheduling in Cyber-Physical System

A Dissertation

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy
in
Computer Science and Engineering

by

Shengyan Hong, B.S., M.S.

Xiaobo Sharon Hu, Director

Graduate Program in Computer Science and Engineering
Notre Dame, Indiana
August 2014

CONTENTS

FIGURES

TABLES

ACKNOWLEDGMENTS

I would like to appreciate the careful guidance of my advisor, Professor Xiaobo Sharon Hu, during my study in University of Notre Dame. Professor Hu brought me to the area of real-time scheduling, trained me to be an assertive, independent and solid researcher, and was always glad to point out my little merits. She expects me to achieve the PHD degree and enjoy a happy life in future.

Professor Lemmon have given me lots of valuable advice on the application of real-time scheduling in the control systems. He is a very passionate, strict and sincere scientist. Professor Christian Poellabauer taught me the application of real-time scheduling in the wireless sensor networks. Professor Aaron D. Striegel gave me important suggestions in my PHD candidacy oral exam.

I show my gratitudes to Dr. Thidapat Chantem and Dr. Jun Yi. Dr. Thidapat Chantem improved my research skills greatly in the the implementation of ideas and the writing of papers. Dr. Yi provided valuable advice to me on how to apply real-time scheduling in the wireless sensor networks.

I am indebted to Professor Song Han, Professor Liqiang Zhang, Professor Shangping Ren and Miao Song. Professor Han has guided me to complete my last project. He is very effective in improving the quality and pushing the progress of this project. Professor Zhang taught me a lot on the knowledge of wireless sensor networks. I collaborated with Professor Ren and Miao Song to improve one of my previous works in the area of distributed real-time systems.

This work is supported in part by National Science Foundation (NSF) under grant numbers CNS-0720457, CNS-0931195, CNS-0702761, CPS-0931195, CSR-1319718 and CSR-1319904.

Real-Time Scheduling in Cyber-Physical System

Abstract

by

Shengyan Hong

A Cyber-Physical System (CPS) is a system where physical components and computational components are tightly integrated. Tasks in a CPS generally need to be accomplished correctly in terms of not only functionality but also punctuality. Real-time scheduling provides the methodology of determining the task execution order on a shared resource in order to make as many tasks in a CPS as possible to meet their deadlines. We addressed four different challenges in this dissertation, i.e., minimizing delay variations of real-time control tasks in a CPS, schedulability of jobs in a distributed real-time system (DRTS), tradeoff between energy savings and real-time stream deadline meetings in a wireless sensor network, and minimizing the impact of network dynamics on a wireless networked control system (WNCS).

For many CPSs, control performance is strongly dependent on delay variations of the control tasks. Such variations can come from a number of sources including task preemptions, variations in task workloads and perturbations in the physical environment. We designed a general adaptive framework that incorporates a powerful heuristic aiming to minimize delay variations.

In a DRTS, jobs are often executed on a number of processors and must complete by their end-to-end deadlines. Job deadline requirements may be violated if

resource competition among different jobs on a given processor is not considered. We designed a distributed, locally optimal algorithm to assign local deadlines to the jobs on each processor to meet as many jobs' end-to-end deadline requirements as possible in a distributed soft real-time system.

Most of the wireless sensors are powered by batteries with a limited amount of energy, hence require the transmission to be energy efficient. Lower transmission rates can greatly reduce transmission energy. However, if the lowest transmission rate is selected, many messages can miss their deadlines, which degrades the Quality of Service (QoS) for CPS applications. We have designed an on-line transmission rate selection approach to maximize the number of packets to meet their deadlines with a small increase in the energy dissipation.

A key design challenge in a WNCS is to design efficient data link layer scheduling algorithms to achieve deterministic end-to-end real-time communication while the WNCS is disturbed by various physical events. In this work, we adopted a rhythmic task in adaptive to external disturbances and designed an effective approach to adjust existing schedule for all the nodes in the WNCS when the disturbances happen.

CHAPTER 1

INTRODUCTION

1.1 Overview

A CPS [124] is a system where physical components and computational components are tightly integrated. The physical components are typically monitored by sensors, and the sensor signals are then transmitted to the computational components by the communication infrastructure. The computational components make decisions for future actions, and these are transmitted to the actuators of the physical components. The computational components coordinate and control the physical operations to satisfy the overall requirements. Most of computational components in CPS applications are control systems which are good at managing and regulating the behaviors of physical components. CPS is prevalent in different areas, such as transportation [119], health care [61], power grid [50],[122], building and environmental control [89], and factory automation [130]. Figure 1.1 shows a CPS application example which integrates physical components and computational components by employing the wireless network. In the example, the states of physical plants are monitored and sampled by sensors, and the sensor signals are transmitted to the controllers by a series of relay nodes in the multi-hop wireless network. After the controllers receive the sensor signals, they will generate the actuator signals that contain decisions for future actions. The actuator signals are

transmitted to the actuators of plants by a series of relay nodes in the multi-hop wireless network to improve the performance of plants.

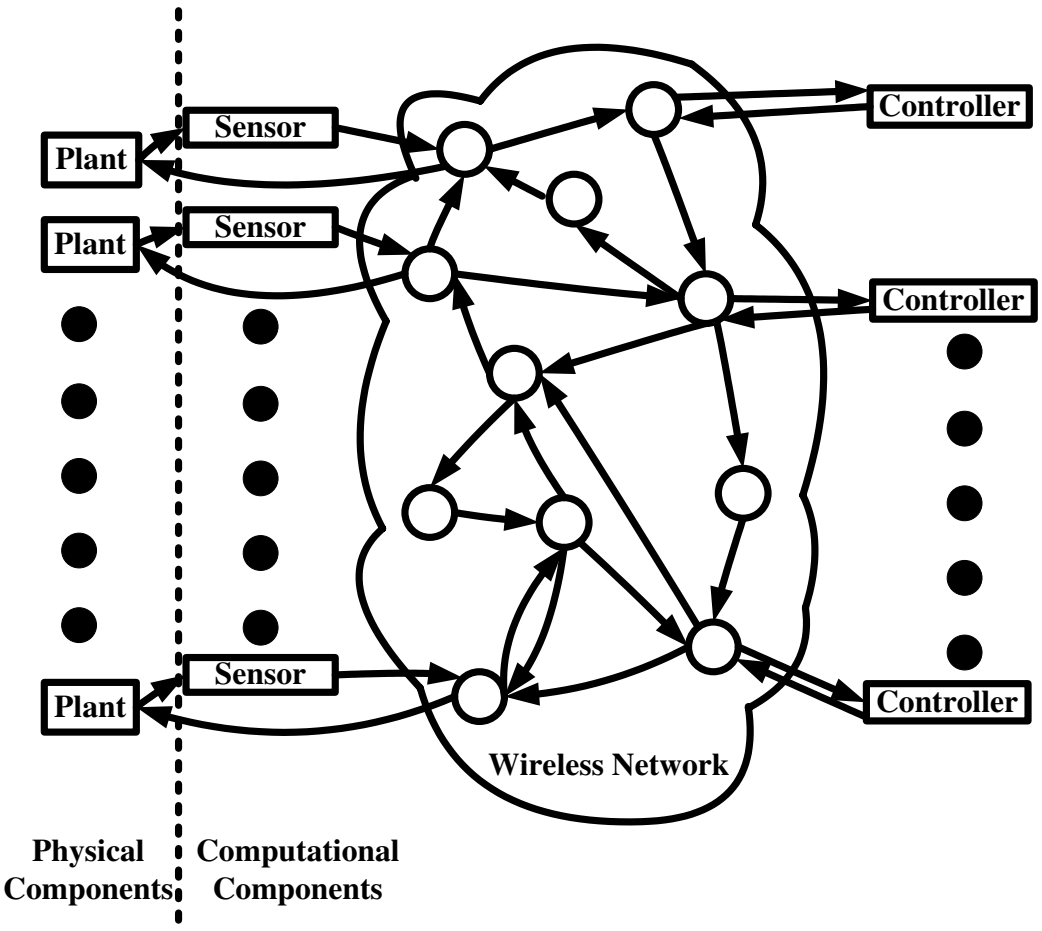


Figure 1.1. A CPS application example.

Most CPS must deal with real-time constraints. That is, a computational task needs to be accomplished correctly in terms of not only functionality but also

time. A delayed reaction can lead to unsatisfied customers (such as in a video game scenario) or total catastrophy (such as in a vehicle anti-lock braking scenario). The real-time requirements of CPS are generally described by deadlines associated with tasks. A task is typically executed repeatedly. Each instance of a task is a job in the real-time scheduling area. Since a computational component (i.e., a processor) in CPS almost always handle multiple real-time tasks, the execution order of such tasks plays a big role in meeting the deadlines. CPS is required to have predictable and reliable behaviors in terms of meeting the time requirements of all tasks instead of completing a single task fast. The predicability and reliability of CPS's can be satisfied by employing certain real-time scheduling methods in the CPS design. Real-time scheduling [29] [101] provides the methodology of determining the task execution order on a shared resource in order to make as many tasks as possible to meet their deadlines. Real-time scheduling is widely employed in CPS applications such as flight control systems [70], wireless networked control systems [10] and battery management systems [80].

Specifically, real-time scheduling [16] assigns priorities to tasks on a shared resource such as a processor. The shared resource then schedules the tasks according to their priorities through a Real-Time Operating System (RTOS). There are two different categories of real-time scheduling algorithms, i.e., static-priority scheduling and dynamic-priority scheduling. In static-priority scheduling, a task's priority never changes once it is assigned. Rate-Monotonic (RM) scheduling [29, 101] and Deadline-Monotonic (DM) scheduling [29, 101] are two well-known scheduling algorithms that belong to the static-priority scheduling category. In contrast, dynamic-priority scheduling allows the task priorities to change over time. Earliest Deadline First (EDF) scheduling [29] [101] is a widely used dynamic-priority

scheduling algorithm.

DRTSs are widely employed in CPS applications such as vehicle control and teleconferencing application (e.g., [48, 110, 160]). A DRTS contains a set of tasks periodically or aperiodically releasing jobs which typically have end-to-end deadlines. Many works, e.g. [62, 70, 71, 73, 118, 158] have studied the schedulability of task set in the DRTSs, while the others have proposed many heuristics on the static-priority or dynamic-priority assignment problem for DRTSs, e.g., [6, 27, 62, 65, 132].

Since the physical world is not entirely predictable, CPS's present more challenges to real-time scheduling. The performance of CPS applications is greatly influenced by the environmental perturbations such as the failures of critical civil infrastructures and malicious attacks. In response to external events, the works [28, 81, 92] employ approaches of adjusting the sampling rates and relative deadlines of tasks on-line in control systems. However, high sampling rates of control tasks can increase the workload of the system, delay the completion of control tasks and result in high delay variations of tasks at sampling and actuation instants. Such results can severely cause the unschedulability of the task set and the degradation of control plant performance. Many works have been done to select optimal sampling rates and deadlines of control tasks to guarantee the robustness of CPS under the varying environment, e.g. [8, 9, 12, 13, 23, 26, 33]. Since periodic sampling causes a large usage of processing resource, event-triggered controllers have emerged in CPS's in recent years. Event-triggered controller only executes control tasks when the controller needs the execution based on the sampling information, which greatly saves the usage of processing resource. In event-triggered control systems, the real-time scheduling is required to determine the minimum

or average sampling periods of control tasks to guarantee the schedulability of sporadic task set, e.g., [91, 142, 153].

Most of CPS's are composed of a great number of computational components, sensors and actuators, which are interconnected through large scale wired or wireless networks. Wireless network enables computational components to communicate with remote sensors and actuators in a low cost and flexible manner. To guarantee the overall CPS application performance, sampling and actuation data are required to reach their destinations by their end-to-end deadlines in a multihop wireless network, while the interference in wireless network results in a high unpredictability of data transmission delay. Many works have studied the scheduling of data transmission in the wireless network. The works in [5, 38, 72, 85, 125] propose approaches to solve the flow rate assignment problem in order to improve the QoS's of CPS's while still guaranteeing the end-to-end deadline meeting of all the streams. A MAC layer protocol and two dynamic-priority assignment approaches are proposed in [151] and [90, 159], respectively, to provide the timeliness support in a resource efficient manner. There are other research problems in the multihop wireless network, such as modeling the effect of transmit power and interference in the wireless transmission [147], energy saving data transmission [37, 149, 154] and power control [143].

Although wireless network provides the service of information communication in many business applications, it is not suitable for the control applications in CPS's. Control applications usually have a very stringent requirement on the real-time responses of control tasks and the adaption to the physical perturbations, which guarantees not only the freshness but also the integrity of physical information. Such a special requirement may not be fully guaranteed by the gen-

eral protocol of wireless networks such as 802.11 [55], which are quite unreliable and nondeterministic in the timeliness support. In contrast, WirelessHART standard [36] is very suitable for the control applications since it has been designed to provide hard timing guarantees and be robust enough against the environmental noise. WirelessHART network has the following features, which makes it appropriate for the control applications of CPS's.

1. WirelessHART network is able to adapt to the changes in plant infrastructure automatically.

2. WirelessHART network spreads the communication among all the channels in response to the interference from the environment.

3. WirelessHART takes robust security measures to protect the security of network and information in transmission.

The research works on the implementation of WirelessHART standard, e.g., [79, 116], the communication resource assignment, e.g., [51, 60, 126, 127, 156, 157], and the schedulability analysis of the stream set, e.g., [128], have been proposed in recent years. In both standards, a centralized network architecture is adopted for network resource allocation and data management. However, WirelessHART is based on static Time Division Multiple Access (TDMA)-based data link layer, which cannot handle the situation that the sampling rates and relative deadlines of tasks are adjusted promptly.

During my Ph.D. study, I am working to present solutions to some of the above challenges. Till now, I have presented approaches in different areas, e.g., minimizing the delay variations of control tasks on a uniprocessor, guaranteeing the schedulability of DRTSs, minimizing the energy expenditure of the real-time

data transmission in the wireless network and minimizing the impact of wireless network dynamics on existing tasks in the CPS. I will introduce the contributions briefly in Section 1.2 and present the technical details in Chapters 2, 3, 4, 5 and 6.

1.2 Contributions

In this section, I summarize the contributions that I have made till now. During the past four year, I have mainly worked on three projects, i.e, regulating delay variations of real-time control tasks, local-deadline assignment for DRTSs and minimizing energy consumption in wireless sensor networks.

1.2.1 Regulating Delay Variations of Real-Time Control Tasks

Control theory is fundamental to a large set of CPS. Control tasks are used to determine the actuator behavior according to the sensing signals. For such CPS applications, besides satisfying the deadlines of the control tasks, the delay variations of the control tasks can also have a significant impact on the system performance. Such variations can come from a number of sources including task preemptions (the operation of suspending the running task with a low priority is called preemption.), variations in task workloads and perturbations in the physical environment. Such variations can cause degraded control system performance, such as sluggish response and erroneous behavior.

When considering the various sources that cause delay variation in a physical system and the significant influence of delay variation on the stability and performance of control systems, it is imperative that a delay variation reduction process be integrated in the control loop so as to regulate delay variation whenever

there are strong internal and external perturbations. To accomplish this, I have designed an on-line adaptive approach which directly minimizes delay variations for control tasks in response to strong internal and external perturbations. Our approach formulates the delay variation minimization problem as an optimization problem. An efficient algorithm is designed based on the generalized elastic scheduling heuristic [34]. The efficiency of the algorithm readily supports an adaptive framework which can adjust deadlines of control tasks on-line in response to dynamic changes in workloads.

1.2.2 Local-Deadline Assignment for Distributed Real-Time Systems

DRTSs are widely employed in CPS applications such as vehicle control and teleconferencing (e.g., [48, 110, 160]). Such systems typically have stringent real-time requirements and may experience large variations in terms of their operating environments. Jobs in a DRTS are often executed on a number of processors and must complete by their end-to-end deadlines. Job deadline requirements may be violated and the performance of CPS may be unpredictable and unreliable if resource competition among different jobs on a given processor is not considered. The scale of these systems often prohibits a centralized resource management approach. Designing low-overhead, distributed scheduling solutions is critical to reliable deployment of such systems.

I have designed a distributed, locally optimal algorithm to assign local deadlines to the jobs on each processor without any restrictions on the mappings of the applications to the processors. The algorithm leads to improved schedulability results since it considers disparate workloads among the processors due to competing jobs having different paths. Given its distributed nature, the algorithm is

adaptive to dynamic changes of the applications and avoids the overhead of global time synchronization. In order to make the proposed algorithm more practical, two derivatives of the algorithm are proposed and compared. Simulation results based on randomly generated workloads indicate that the approach outperforms existing work both in terms of the number of feasible task sets and the number of feasible jobs. In addition, our simulation results illustrate the excellent performance of the approach in real-world case studies.

1.2.3 An Adaptive Transmission Rate Control Approach to Minimize Energy Consumption

Wireless sensor network is widely used in CPS applications, such as health care and environment monitoring (e.g. [103, 140]). Most of the wireless sensors are powered by batteries with a limited amount of energy, hence require the transmission to be energy efficient. Lower transmission rates can greatly reduce transmission energy. However, if the lowest transmission rate is selected, many messages can miss their deadlines, which degrades the Quality of Service (QoS) for CPS applications. There is a tradeoff between saving energy and meeting real-time stream deadlines in wireless sensor networks. Therefore, it is important to design an efficient approach for adjusting transmission rates in order to not only achieve energy saving but also maximize the QoS for CPS applications.

I have designed an on-line transmission rate selection approach based on an optimal dynamic voltage frequency scaling algorithm, Lp-EDF [148]. Our approach exploits the periodicity property of the real-time streams to predict the future jobs' timing information and finds an optimal transmission rate schedule. The approach is able to make more messages meet their deadlines. Preliminary

results show that the approach achieves a higher success ratio with a lower timing cost compared with existing works, although the energy dissipation caused by the approach sees a small increase.

1.2.4 Data Link Layer Scheduling in Dynamic Wireless Networked Control Systems with Minimum On-line Schedule Update

WNCSs have received significant attention over the past several decades [58, 67, 111, 113, 117] because WNCSs are widely used in many areas such as telerobotics [44], aircraft control [150], civil infrastructure monitoring [77], medication service [63] and power management [56]. In a WNCS, sensors, controllers, actuators and other relay nodes are geographically distributed and connected over wireless network media. A task in a WNCS generally delivers measurements from a sensor to the gateway and sends control signals from the gateway to an actuator within an end-to-end deadline. In the real-world, the performance of WNCSs is usually degraded by various physical events, such as the failures of critical civil infrastructures and malicious attacks.

In this work, I consider a WNCS adopting a centralized network architecture, which utilizes a static schedule when there is no physical disturbances in the WNCS. The WNCS contains a set of periodic tasks and a rhythmic task, where the period and relative deadline of rhythmic task are reduced suddenly and then return to their nominal values gradually. Since rhythmic task is critical for the WNCS to respond to physical disturbances, all the packets of rhythmic task must meet their deadlines. In contrast, packets of periodic tasks are allowed to be dropped to give up some bandwidth to rhythmic task. After rhythmic task returns to its nominal state, the WNCS is required to reuse the static schedule immediately

after a specific time slot (called switch point) in order to reduce the overhead of packet transmissions and prepare for any future external event. Therefore, there exists a transient time duration when the WNCS cannot use the static schedule. During such a transient duration, there may be a high bandwidth competition in the WNCS, which may result in the end-to-end deadline misses of periodic packets. Although the WNCS allows periodic packets to miss their end-to-end deadlines, frequent deadline misses can degrade the Quality of Service (QoS) of the system. Therefore, it is critical to determine a transient time duration and design an on-line data link scheduling algorithm to make periodic packets meet deadlines as many as possible.

This work designs an on-line data link layer scheduling problem determining a transient time duration and a dynamic schedule in order to minimize the number of dropped periodic packets. In the problem, I propose various practical constraints of making the problem usable for real-world applications. To solve the problem, I propose an on-line approach to determine a transient time duration and construct a dynamic schedule for this time duration with bounded time and schedule update overheads. The effectiveness and efficiency of the proposed approach are validated thorough extensive experimental results.

CHAPTER 2

Reducing Delay Jitter through Adaptive Deadline Adjustments

For many control systems, control performance is strongly dependent on delay variations of the control tasks. Such variations can come from a number of sources including task preemptions, variations in task workloads and perturbations in the physical environment. Existing work has considered improving control task delay variations due to task preemption only.

This chapter presents a general adaptive framework that incorporates a powerful heuristic aiming to further reduce delay variations. Results indicate that the heuristic significantly improves existing approaches in terms of the number of feasible task sets (between 13% to 50% on average) and the delay variation value (between 113% to 250% on average).

2.1 Introduction

For many CPSs, intelligent coordination between control design and its corresponding computer implementation can lead to improved control performance and/or reduced resource demands [7, 47, 102]. A prime example that benefits from such coordination is regulating delay variations (jitter) in control tasks. For many control systems, control performance strongly depends on delay variations in control tasks. Such variations can come from numerous sources including task

preemptions, variations in task workloads and perturbations in the physical environment, and can cause degraded control system performance, such as sluggish response and erroneous behavior. An integrated approach to regulate delay variation has the potential to significantly improve a physical system performance.

There are a number of published papers related to reducing delay variations. A somewhat indirect way of reducing delay variations is to reduce task deadlines, which has been investigated by many researchers, e.g., [13, 14, 21, 26, 64]. A common theme of all these methods is to focus on reducing deadlines of either tasks or subtasks. Because deadlines are only allowed to be reduced, these methods cannot effectively explore the design space where deadlines of certain tasks/subtasks may be increased (within some upper bounds) to reduce the overall delay variations.

Another set of methods are based on a task decomposition based approach where each task is partitioned into three subtasks, i.e., Initial, Mandatory, and Final Subtasks (referred as the IMF model), and the delay variation of the final subtask (corresponding to control update) is minimized. The task-decomposition based methods [11, 12] suffer less, but still obvious performance degradation (compared with direct deadline reduction methods) when deadlines are only allowed to be decreased greedily. The decomposition task model is acceptable for control tasks where only a small amount of data needs to be passed to control update subtasks, otherwise context switching cost could be prohibitive. In addition, these methods require repeated worst-case response time computation under the Earliest Deadline First (EDF) scheduling policy, which can be quite time consuming.

Some recent works have focused on studying the influence of task delay variations on control system performance, and how to reduce such influence by scheduling the tasks intelligently in order to enhance system performance. A Matlab-

based toolbox for real-time control performance analysis, taking the timing effects into account, is presented in [32, 95]. A computational model has been proposed in [31] to provide small jitter and short input-output latencies of control tasks to facilitate co-design of flexible real time control systems. Theory of jitter margin is proposed in [33], and applied in [18, 19, 33] to guarantee the stability and performance of controllers in the target system. Some straightforward jitter control methods to improve control system performance, e.g., task splitting, advancing deadlines and enforcing non-preemption, are evaluated in [26]. [23] proposes a delay-aware period assignment algorithm under fixed priority scheduling to reduce the control performance cost. Some of these works [23, 31, 33] adjust control task periods and change the workload of the control system, which may over or under utilize the control system resources, while some of them [18, 19, 26] are not suitable for on-line use due to their exceedingly long computation time of the schedulability analysis.

When considering the various sources that cause delay variation in a physical system and the significant influence of delay variation on the stability and performance of control systems, it is imperative that a delay variation reduction process be integrated in the control loop so as to regulate delay variation whenever there are strong internal and external perturbations. To accomplish this, we need a delay variation reduction approach that is effective, efficient and adaptive. In this chapter, we propose an on-line adaptive approach which directly minimizes delay variations for both decomposable and non-decomposable control tasks simultaneously. The approach leverages the IMF based task model for both types of tasks and formulates the delay variation minimization problem as an optimization problem. An efficient algorithm is designed based on the generalized elastic scheduling

heuristic [34]. The efficiency of the algorithm readily supports an adaptive framework which can adjust deadlines of control tasks on-line in response to dynamic changes in workloads.

The rest of the chapter is organized as follow. Section 2.2 reviews system models and Section 2.3 provides motivations to our work. Section 2.4 presents our heuristic to solve the delay variation reduction problem. Experimental results are presented and discussed in Section 2.5.

2.2 System Model

We consider a computer system which handles a set Γ of N real-time control tasks, $\{\tau_1, \tau_2, \cdots, \tau_N\}$, each with the attributes: (C_i, D_i, P_i), where C_i is the worst case execution time of τ_i, D_i is τ_i's deadline, P_i is its period, and $C_i \leq D_i \leq P_i$. Without loss of generality, we adopt the IMF task modeling approach introduced in [12]. Specifically, we let τ_i be composed of three subtasks, the initial part τ_{ii} for sampling input data, the mandatory part τ_{im} for executing the control algorithm, and the final part τ_{if} to deliver the control action. Thus, a task set Γ_{IMF} consists of $3N$ subtasks $(\tau_{1i}, \tau_{1m}, \tau_{1f}, ..., \tau_{Ni}, \tau_{Nm}, \tau_{Nf})$, each with the parameters

$$\tau_{ii} = (C_{ii}, D_{ii}, P_i, O_{ii}) \quad \tau_{im} = (C_{im}, D_{im}, P_i, O_{im}) \quad \tau_{if} = (C_{if}, D_{if}, P_i, O_{if})$$

where $O_{i\star}$ is the offset of the corresponding subtask. Note that in order for the IMF model to faithfully represent the original task set, each τ_{ii} must be executed before τ_{im}, which must in turn be executed before τ_{if}. For a non-decomposable task, say τ_i, we simply have $C_{ii} = C_{im} = D_{ii} = D_{im} = 0$, and $C_{if} = C_i$. Some tasks may also be partially decomposable, i.e., we may have non-zero C_{ii} and D_{ii} but $C_{im} = D_{im} = 0$.

To achieve desirable control performance, control actions should be delivered at regular time intervals periodically. However, preemptions, variations in task workloads, and perturbations in the physical environment make each instance of the control actions experience different delays. Similar to [12], we define the delay variation as the difference between the worst and best case response times of the same final subtask relative to its period, i.e.,

$$DV_i = \frac{WCRT_{if} - BCRT_{if}}{P_i}, \tag{2.1}$$

where $WCRT_{if}$, $BCRT_{if}$ are the worst case and best case response times respectively. The definition of delay variation gives information on the delay variance that a task will suffer in the control action delivery within a period. Our problem is to minimize the delay variations of all the final subtasks.

We use Earliest Deadline First (EDF) scheduling algorithm. A necessary and sufficient condition for a synchronous task set to be schedulable under EDF is given below.

Theorem 1. *A set of synchronous periodic tasks with relative deadlines less than or equal to periods can be scheduled by EDF if and only if* $\forall L \in \{d_K | d_K = K \cdot P_i + D_i \leq \min(L_{ip}, H, B_p)\}$ *the following constraint is satisfied,*

$$L \geq \sum_{i=1}^{N} (\lfloor \frac{L - D_i}{P_i} \rfloor + 1) \cdot C_i \tag{2.2}$$

where $L_{ip} = \frac{\sum_{i=1}^{N}(P_i - D_i)U_i}{1-U}, U_i = \frac{C_i}{P_i}, U = \sum_{i=1}^{N} \frac{C_i}{P_i}$, $K \in \mathbb{N}$ *(the set of natural numbers including 0),* H *is the hyperperiod, and* B_p *is the busy period [17, 29].*

For an asynchronous task set, the condition in Theorem 1 can be used as a sufficient condition [17].

2.3 Motivation

We use a simple robotic example, similar to the one in [12], to illustrate the deficiencies of existing approaches for delay variation reduction. The example contains four control tasks, i.e., the speed, strength, position and sense tasks. The tasks and the original delay variations under EDF are shown in Table 2.1. We consider two representative methods for delay variation reduction, described as the followings.

TABLE 2.1

A Motivational Example Containing Four Tasks with the Second and Fourth Tasks being Non-Decomposable.

Task Name	Computation Exec. Time	Deadline	Period	Original Delay Variations (%)
Speed	5000	27000	27000	18.52
Strength	8000	30000	320000	1.56
Position	10000	45000	50000	32
Sense	13000	60000	70000	40

In [11, 12], a task decomposition based method, denoted as TDB, is proposed to reduce delay variations of final subtasks. The algorithm replaces the deadlines

of final subtasks by their respective worst case response times in the main loop greedily and efficiently. However, the method neglects the subtask dependencies in the IMF task model and tends to generate infeasible solutions in high utilization task sets.

Another greedy algorithm, presented in [14] and denoted as DSB, indirectly reduces task delay variations by the deadline scaling based technique. The algorithm repeatedly reduces the deadlines of all the tasks by the same scaling factor, until the task set becomes unschedulable. The blind deadline reduction may increase the delay variations.

TABLE 2.2

Delay Variation of A Motivational Example by Adaptively Applying Existing Methods

Task Name	New Delay Variations (%) DSB / TDB / DVR	Delay Variations before Reassignment (%) DSB / TDB / DVR	Delay Variations after Reassignment (%) DSB / TDB / DVR
Speed	44.54 / Fail / 0.44	44.54 / 33.33 / 0.44	41.48 / Fail / 0.44
Strength	3.36 / Fail / 5.94	33.59 / 28.13 / 59.4	31.25 / Fail / 15.59
Position	34.75 / Fail / 1	34.75 / 44 / 1	34 / Fail / 1
Sense	32.86 / Fail / 9.27	32.86 / 48.57 / 9.27	32.86 / Fail / 20

Suppose that decomposing the strength and sense tasks in the robotic example would cause non-negligible context switch overhead and we opt to only partition the speed and position tasks according to the IMF model. Assume that the IMF decomposition is made considering that the initial and final subtasks consume 10% of the execution time of the corresponding control task. By applying DSB and TDB, new delay variation values are obtained and shown as the first two values in column 2 of Table 2.2. TDB actually fails to find a feasible solution and employs the original deadline assignment, while DSB gives worse delay variations. With the DSB method, three tasks suffer more than 30% delay variation.

Now, assume that at some time interval, the execution rate of the strength task increases by 10 times. If the same deadline assignments are used for the tasks/subtasks, the delay variation of the strength task increases to 33.59% and 28.13% for DSB and TDB, respectively (see the first two values in column 3 of Table 2.2). Since TDB cannot find a feasible solution before the workload change, it still employs the original deadline assignment after the change. Suppose we apply the DSB and TDB methods on-line in response to this change, the new delay variation values are shown as the first two values in column 4 of Table 2.2. It turns out that the TDB still fails to find a feasible solution while DSB does not provide much improvement over the delay variations before the reassignment.

With our proposed approach (DVR), smaller delay variations can be obtained for all the cases considered above. In particular, for each respective scenario, we have applied our approach and the delay variation values are shown as the third number in columns 2-4 of Table 2.2. Though for some tasks, delay variations see a small increase, most of the tasks which suffer from large delay variations due to the other methods are now having much smaller delay variations.

2.4 Approach

2.4.1 Problem Formulation

From the previous section, one can see that delay variations could be improved significantly if more appropriate deadline assignments can be identified. In this section, we describe our adaptive delay variation reduction (DVR) approach. DVR is built on three basic elements. First, the general IMF model as given in the previous section is used for both decomposable and non-decomposable tasks. Second, the delay variation reduction problem is formulated as an optimization problem. Third, an efficient heuristic is developed to solve the optimization problem. The heuristic is then incorporated into an adaptive framework.

We adopt the generalized IMF task model described in Section 2.2 to represent the task set under consideration. The general IMF model allows both decomposable and non-decomposable tasks to be treated equivalently. Given an IMF task set, there may exist numerous sets of feasible deadlines (D_{ii}, D_{im}, D_{if}) which allow the original task set to be schedulable. However, different sets of deadlines could lead to different delay variations of the original tasks. To find the particular subtask deadline assignment that results in the minimum delay variation, we formulate the deadline selection problem as a constrained optimization problem. Though existing work such as [34, 83] has considered the deadline selection problem as an optimization problem, there are two major differences between our present formulation and theirs. First, our formulation directly minimizes delay variations. Second and more importantly, our formulation leverages special properties of the IMF task model and thus allows much more effective delay variation reduction.

The delay variation minimization problem is to minimize the total delay vari-

ation bounds of (2.1) subject to the schedulability constraints as given in (2.2) while considering the IMF task model. The decision variables in the problem are subtask deadlines D_{ii}, D_{im} and D_{if} (while L as defined in Theorem 1 is dependent on D_{i*}'s). Specifically, we have

$$\text{min: } \sum_{i=1}^{N} w_i(\frac{D_{if} - C_{if}}{P_i})^2 \tag{2.3}$$

$$\text{s.t. } \sum_{i=1}^{N}[(\lfloor\frac{L - D_{ii}}{P_i}\rfloor + 1)\cdot C_{ii} + (\lfloor\frac{L - D_{im}}{P_i}\rfloor + 1)\cdot C_{im} + (\lfloor\frac{L - D_{if}}{P_i}\rfloor + 1)\cdot C_{if}] \leq L,$$

$$\forall L \in \{d_K | d_K = K \cdot P_i + D_i \leq min(L_{ip}, H, B_p)\}, \tag{2.4}$$

$$D_{ii} = D_{im}, \tag{2.5}$$

$$C_{if} \leq D_{if} \leq \min(D_{im}, D_i - D_{im}), \tag{2.6}$$

$$C_{im} \leq D_{im} \leq D_i - C_{if}, \tag{2.7}$$

where $0 \leq w_i \leq 1$ is a constant, L_{ip}, H, B_p and K are as defined in Theorem 1. If task τ_i is not decomposable, (2.6) and (2.7) are replaced by

$$C_{if} \leq D_{if} \leq D_i, \tag{2.8}$$

$$D_{im} = 0. \tag{2.9}$$

To see why the above formulation can lead to valid deadline assignments that minimize delay variations, first note that deadline D_{if} is the upper bound of the WCRT of the final subtask of τ_i, and C_{if} is the lower bound of the BCRT of the final subtask of τ_i. Hence, the objective function in (2.3) is simply the weighted sum of the squares of worst-case delay variations as defined in (2.1). The use of

w_i allows one to capture the relative importance of control tasks in the objective function. The choice of the objective function is based on two observations. First, the quadratic form effectively reduces the variation of jitter distribution. Second, the formulation leads to an efficient heuristic to solve the quadratic programming problem. We do not directly optimize control performance as such formulation can be quite expensive computationally[32]. (Our experiments will show that the objective function is effective in improving control performance.)

To guarantee schedulability under the IMF model, we have introduced a set of constraints in our formulation. Constraint (2.4) helps ensure the schedulability of the task set according to Theorem 1. However, this constraint alone is not sufficient since the precedence requirement must be obeyed when executing the initial, mandatory and final subtasks of any decomposable task. (Note that ensuring the subtask dependencies during task execution is straightforward. The difficulty lies in capturing this in the schedulability test without either introducing more variables (i.e., O_{ii}, O_{im}, O_{if}) or being overly pessimistic.) To capture the fact that τ_{ii} is always executed before τ_{im}, we can set $D_{ii} \leq D_{im}$ (and hence τ_{ii} has a higher priority than τ_{im} as long as $O_{ii} = O_{im} = 0$). For simplicity, we let $D_{ii} = D_{im}$, assuming that a tiebreak goes to τ_{ii}, which is constraint (2.5). Since τ_{if} must start after τ_{im} is completed, we let $O_{if} = D_{im}$. Furthermore, to guarantee that task τ_i finishes by its deadline D_i, we must have $O_{if} + D_{if} \leq D_i$. We thus have $D_{if} \leq D_i - D_{im}$, which leads to one part of (2.6). The other part of (2.6), i.e., $D_{if} \leq D_{im}$, reflects the observation that smaller deadlines should be assigned to the final subtask compared to that of the mandatory subtask so as to help reduce delay variation of the final subtask (as this would be the delay variation of interests). (2.7) constrains the space of D_{im} and is obtained simply

by combining $D_{im} \leq D_i - D_{if}$ and $D_{if} \geq C_{if}$. Constraints (2.8) and (2.9) replace (2.6) and (2.7) for tasks that are not decomposable. Since they are simpler than (2.6) and (2.7), our following discussions focus more on constraints (2.5)-(2.7).

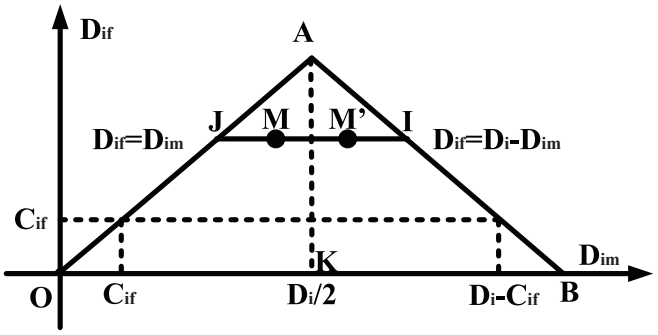


Figure 2.1. Feasible deadline region for mandatory/final subtasks.

Based on constraints (2.5)-(2.7), Figure 2.1 depicts the feasible region of (D_{im}, D_{if}), which is bounded by $\triangle\ ABO$ and corresponds to the search region for the optimal solution to (2.3)-(2.7). To make our search more efficient, we would like to reduce the search region as much as possible without sacrificing the optimization solution quality. An observation discussed in [15] can be exploited to reduce the search region. We summarize this observation in the following theorem by using our notation.

Theorem 2. *Given a set Γ_{IMF} of N tasks. If the necessary and sufficient condition for schedulability in Theorem 1 is satisfied for a synchronous task set Γ_{IMF} with $(D_{ii} = D_{im}, D_{im}, D_{if})$ for $i = 1, ..., N$, then the same condition is satisfied*

for a synchronous task set Γ'_{IMF} *with* $(D'_{ii}, D'_{im}, D_{if})$*, where* $D'_{ii} = D'_{im} \geq D_{im}$ *for* $i = 1, ..., N$*. [15]*

Applying Theorem 2 to the search region depicted in Figure 2.1, one can readily see that point $M^{'}$ on the segment JI leads to a schedulable solution if point M leads to a schedulable solution. Since D_{im} corresponding to $M^{'}$ is larger than that of M, $M^{'}$ is a more desirable solution than M as it leads to a smaller D_{if}. Based on this observation, we can reduce the search region by 1/2 by replacing constraints (2.6)-(2.7) in the optimization problem by the following:

$$C_{if} \leq D_{if} \leq D_i - D_{im}, \tag{2.10}$$

$$\frac{D_i}{2} \leq D_{im} \leq D_i - C_{if}. \tag{2.11}$$

If task τ_i is not decomposable, constraint (2.11) is replaced by

$$D_{im} = 0. \tag{2.12}$$

Hence jitter minimization can be achieved by solving the optimization problem defined by (2.3-2.5), (2.10-2.12).

Solving the optimization problem specified in (2.3) together with (2.4), (2.5), (2.10), (2.11) and (2.12) is not trivial as it involves dealing with a discontinuous function (the floor function). Heuristic techniques such as the one presented in [34] may be leveraged to solve the problem, but it would take many iterations to reach convergence. In addition, this heuristic either fails to find a solution or finds a very pessimistic solution for task sets with high utilization. We have developed a better heuristic to avoid such problems, which we refer to as DVR. DVR in essence

is a branch-and-bound type algorithm. It achieves its efficiency and effectiveness by exploiting a number of observations based on the general time demand analysis to direct the search process to focused regions.

2.4.2 DVR Heuristic

DVR solves the optimization problem as follows. (The high-level process of DVR is similar to that used in [34], but there are significant differences between DVR and that in [34] in the way that the actual search is conducted.) For an initial solution $(D_{ii} = D_{im}, D_{if})$, the value of L is computed, and an updated set of D'_{if} is obtained by solving the optimization subproblem defined in (2.3)-(2.5) and (2.10). The new set of D'_{if} is adjusted to make the solution $(D_{ii} = D_{im}, D'_{if})$ become schedulable, i.e., satisfy the necessary and sufficient condition in Theorem 1; and then D_{im} is updated to D'_{im} by considering not only the constraint on D_{im} but also D_{im}'s effect on future D_{if} selection. These new values, $D'_{ii} = D'_{im}$ and D'_{if}, are then used as the initial solution for the next iteration. This process is repeated in an attempt to find the best set (D_{im}, D_{if}) that minimizes the objective function (2.3).

Algorithm 1 summarizes the main procedure of DVR. In the algorithm, the current and best solutions found are represented by $currD$ and $bestD$, respectively. DVR starts with several straightforward initialization procedures (Line 1-Line 4). The main loop of DVR spans from Line 5 to Line 36, where DVR searches the best solution $bestD$ to minimize the objective function (2.3). In each iteration of the main loop, a schedulability check is performed to test whether the current task set Γ_{IMF} satisfies (2.4) (Line 9). If the current solution satisfies (2.4), the corresponding objective function value is evaluated by (2.3) and the new

solution is either recorded as the best solution found or discarded (Line 11-Line 18). Furthermore, either $currD_{if}$ or $currD_{im}$ is updated according to the value of *state* (Line 19-Line 23). If constraint (2.4) is not satisfied, subtasks' deadlines (D_{if} or D_{im}) are adjusted according to the value of *state* as shown in Lines 27 to 31.

As outlined above, in each iteration of the main loop, DVR updates either the final subtasks' deadlines or the mandatory subtasks' deadlines according to the value of *state*. By allowing only one set of deadlines to vary in each iteration, we not only simplify the optimization problem, but also manage search space reduction effectively. We use parameter "*state*" in each iteration to indicate whether the iteration is to update D_{if} ($state = 0$) or D_{im} ($state = 1$). Below, we present the details on the deadline update procedures.

DVR updates the final subtasks' deadlines D_{if} by solving the optimization problem described by (2.3-2.5) and (2.10) for a fixed L and D_{im} values. The obvious difficulty here is how to deal with the discontinuous function (2.4). Instead of simply adopting the strategy introduced in [34], i.e., removing the floor operator from (2.4), we propose to use a less pessimistic way to tackle the difficulty. Specifically, we replace constraint (2.4) by the following constraint

$$\sum_{i=1}^{N}[(\frac{L-D_{ii}}{P_i}+1)\cdot C_{ii}+(\frac{L-D_{im}}{P_i}+1)\cdot C_{im}+(\frac{L-D_{if}}{P_i}+1)\cdot C_{if}] = L, L = L_{ip}, \tag{2.13}$$

where L_{ip} is as defined in Theorem 1. Constraint (2.13) replaces (2.4) to form a new optimization subproblem defined by (2.3), (2.5), (2.10), (2.13). Compared with L^* employed in [34], L_{ip} in (2.13) leads to less pessimistic solutions. Solving the optimization subproblem defined by (2.3), (2.5), (2.10), (2.13) can be done efficiently by leveraging the Karush-Kuhn-Tucker Theorem [114]. Specifically,

function *Optimize_Solution* in Line 22 finds the solution according to Theorem 3 for given $L = L_{ip}$ and D_{im} values.

Theorem 3. *Given the constrained optimization problem as specified in (2.3), (2.5), (2.10), (2.13), for fixed values of $L = L_{ip}$ and mandatory subtask deadlines $D_{im}, \forall i$, let*

$$\tilde{D} = \sum_{i=1}^{N} L \cdot U_i - \sum_{i=1}^{N} D_{im} \cdot (U_{ii} + U_{im}) + \sum_{i=1}^{N} C_i - L - \sum_{D_{if} \neq D_{ifmax}} D_{ifmin} U_{if}$$

$$- \sum_{D_{if} = D_{ifmax}} D_{ifmax} U_{if}, \tag{2.14}$$

$$\tilde{S} = \sum_{D_{if} \neq D_{ifmax}} \frac{U_{if}^2}{w_i} P_i^2. \tag{2.15}$$

A solution, D_{if}^*, is optimal, if and only if

$$D_{if}^* = \frac{\tilde{D} U_{if}}{\tilde{S} w_i} P_i^2 + D_{ifmin}, \tag{2.16}$$

where $U_i = \frac{C_i}{P_i}$, $U_{ii} = \frac{C_{ii}}{P_i}$, $U_{im} = \frac{C_{im}}{P_i}$, $U_{if} = \frac{C_{if}}{P_i}$, $D_{ifmin} = C_{if}$ and $D_{ifmax} = D_i - D_{im}$.

Proof: *We prove the theorem by utilizing the KKT necessary conditions for the solution to the given problem, which can be written in terms of the Lagrangian function for the problem as*

$$J_a(D, \mu) = \sum_{i=1}^{N} \frac{w_i}{P_i^2} (D_{if} - D_{ifmin})^2 + \mu_0 (L \sum_{i=1}^{N} U_i - \sum_{i=1}^{N} D_{im} (U_{ii} + U_{im})$$

$$- \sum_{i=1}^{N} D_{if} U_{if} - L + \sum_{i=1}^{N} C_i) + \sum_{i=1}^{N} \lambda_{if} (D_{if} - D_{ifmax}) + \sum_{i=1}^{N} \mu_{if} (D_{ifmin} - D_{if}), \tag{2.17}$$

where μ_0, μ_{if}, *and* λ_{if} *are Lagrange multipliers,* $\mu_0 \geq 0$, $\mu_{if} \geq 0$, *and* $\lambda_{if} \geq 0$, *for* $i = 1, ..., N$. *The necessary conditions for the existence of a relative minimum at* D^*_{if} *are, for all* $i = 1, ..., N$,

$$0 = \frac{\partial J_a(D, \mu)}{\partial D_{if}} = -\frac{2w_i}{P_i^2}(D_{ifmin} - D^*_{if}) - \mu_0 U_{if} + \lambda_{if} - \mu_{if} \tag{2.18}$$

$$\mu_0(L\sum_{i=1}^{N} U_i - L - \sum_{i=1}^{N} D_{im}(U_{ii} + U_{im}) - \sum_{i=1}^{N} D^*_{if}U_{if} + \sum_{i=1}^{N} C_i) = 0 \tag{2.19}$$

$$\lambda_{if}(D_{if} - D_{ifmax}) = 0 \tag{2.20}$$

$$\mu_{if}(D_{ifmin} - D_{if}) = 0 \tag{2.21}$$

Consider the KKT conditions given in (2.18), (2.19), (2.20), (2.21). Suppose that $D^*_{kf} = D_{kfmin}$, $\mu_{kf} \geq 0$, *and* $\lambda_{kf} = 0$. *Then, from (2.18),* $\mu_{kf} = -\mu_0 U_{kf}$. *Since* $\mu_{kf} \geq 0$, $\mu_0 \geq 0$ *and* $U_{kf} > 0$, *we have*

$$\mu_{kf} = \mu_0 = 0. \tag{2.22}$$

Similarly, if $D^*_{kf} = D_{kfmax}$, *then* $\lambda_{kf} \geq 0$, $\mu_{kf} = 0$ *and*

$$\mu_0 = \frac{\frac{2w_k(D_{kfmax} - D_{kfmin})}{P_k^2} + \lambda_{kf}}{U_{kf}}. \tag{2.23}$$

Assume that constraints in (2.10) are inactive (i.e., $D_{kfmin} < D^*_{kf} < D_{kfmax}$, $\mu_{kf} = \lambda_{kf} = 0$*). Then (2.18) becomes*

$$\mu_0 = \frac{2(D_{kf} - D_{kfmin})U_{kf}}{\frac{U_{kf}^2 P_k^2}{w_k}}. \tag{2.24}$$

Summing (2.22) and (2.24) up for all i, and assuming that constraint (2.13) is

active for any optimal solution to the given optimization problem, we have

$$\mu_0 = 2\frac{\tilde{D}}{\tilde{S}}. \tag{2.25}$$

By combining (2.25) with (2.18), we get

$$D^*_{if} = \frac{\tilde{D}U_{if}}{\tilde{S}w_i}P_i^2 + D_{ifmin}. \tag{2.26}$$

□

Applying Theorem 3 for given L_{ip} and D_{im} values results in a new set of D_{if} values. However, tasks with this set of deadlines may or may not be schedulable since constraint (2.13) itself is not equivalent to (2.4). To check whether the set of deadlines can indeed satisfy the feasibility condition in Theorem 1, we evaluate (2.4) for every scheduling point $L \leq min(L_{ip}, H, B_p)$, where L_{ip} corresponds to newly found D_{if}'s, to determine if the D_{if}'s can be satisfied by the tasks. If (2.4) is satisfied by all $L \leq min(L_{ip}, H, B_p)$, the solution will be used for deriving new deadlines in the next iteration. Otherwise, adjustments to the found deadlines need to be made. DVR adjusts, i.e., extends, the final subtasks' deadline through function *Final_Deadline_Adjust* (Line 28). The value of D_{if} is adjusted to a new value such that the number of τ_{if}'s jobs to be completed within the time interval $(0, min(L_{ip}, H, B_p)]$ is decreased by 1. The D_{if}'s are adjusted one at a time in the decreasing order of the task execution times, and stops as soon as (2.4) is satisfied by all $L \leq min(L_{ip}, H, B_p)$. This method greedily reduces the workload of the task set within the time interval $(0, min(L_{ip}, H, B_p)]$ and tends to quickly find a set of schedulable deadlines.

To update mandatory subtasks' deadline values for constructing a new opti-

mization subproblem, DVR increases previous mandatory subtasks' deadlines D_{im} to $D'_{im} = D_i - D_{if}$. This is implemented in function *Construct_New_Subproblem* (Line 20). By setting $D'_{im} = D_i - D_{if}$, (2.10) is satisfied. More importantly, DVR skips all D''_{im} values that satisfy $D_{im} < D''_{im} < D_i - D_{if}$, because $D'_{im} = D_i - D_{if}$ leads to a smaller D_{if} than D''_{im} according to Theorem 2. Additionally, DVR does not increase D_{im} beyond $D_i - D_{if}$, because for $D''_{im} > D_i - D_{if}$, DVR would miss the optimal solution contained in the new subproblem based on $D'_{im} = D_i - D_{if}$. If the constructed new subproblem contains no schedulable solution, DVR will incrementally extend the mandatory subtasks' deadlines, which is implemented in *Mandatory_Deadline_Adjust* and similar to adjusting the final subtasks' deadlines discussed above.

To best utilize the new subproblem formulation in the search process, care must be taken in the subtasks' deadline initialization. DVR starts the search process from the minimum value of D_{im} and the maximum value of D_{if} (implemented in function *Construct_Initial_Solution_and_Subproblem* (Line 2)). Specifically, the D_{im} of a decomposable task is set to $\frac{D_i}{2}$, which satisfies (2.11), while D_{im} of a non-decomposable task is set to 0 according to (2.12). The initial deadline of the final subtask of any task τ_i is set to $D_i - D_{im}$, according to (2.10). With each update of D_{im}, the upper bound of D_{if} decreases as the number of iterations increases, implying a smaller D_{if} by Theorem 3 (and leads to smaller jitter). To accelerate the search process and make DVR flexible, we also allow user-defined deadlines to be used as an initial solution.

In the main loop, DVR needs stopping criteria to end its search process. We choose variables $BestObjF$ and *duplicate* to set up the stopping criteria for DVR. $BestObjF$ is the objective function value of the best solution found so far, and

duplicate records the number of the found feasible solutions whose objective function values satisfy $|ObjF - BestObjF| < \delta$, where δ is a user-defined parameter. If $|ObjF - BestObjF| < \delta$, *duplicate* is incremented by 1, as shown in Lines 12 to 14. When the number of "duplicated" solutions is equal to a user-defined parameter β, the program exits from the main loop (Line 7). To handle the case where final subtasks' deadlines do not converge to some fixed values (or when it may take too long for the solution to converge), the algorithm uses another user-defined parameter, $maxIter$, to limit the maximum number of iterations.

The time complexity of DVR is dominated by the busy period computation in Line 3 and the main `for` loop starting at Line 5. The time complexity of the busy period computation algorithm proposed in [141] is $O(\frac{L_{busy}}{C_{min}})$, where L_{busy} is the first busy period and C_{min} is the minimum task computation time. Inside the `for` loop, the most timing consuming operations appear in the $Feasibility\_Test$ in Line 9 with the time complexity $O(N \cdot max\{P_i - D_{i\star}\})$. Thus, the time complexity of DVR is $O(max(\frac{L_{busy}}{C_{min}}, N \cdot max\{P_i - D_{i\star}\} \cdot maxIter))$.

2.4.3 Adaptive Delay Variation Reduction

As we have seen from the motivational example, dynamic workload changes could cause large delay variations if the original task/subtask deadlines were used. Dynamic workload changes may be caused by task-period adjustment in response to an event in the changing environment as shown in [25, 34]. Note that such task periods may assume any value within a range and it can be impractical to precompute all possible delay variations. Furthermore, task workloads may also fluctuate widely due to non-deterministic task computation times [25, 120]. Thus, it is desirable to deploy an on-line adaptive framework to adjust task/subtask deadlines

when workloads change significantly. The key to such an adaptive framework is an efficient method of solving the optimization problem posed earlier. Our heuristic, DVR, satisfies such a requirement. Hence, we propose an adaptive framework built on DVR.

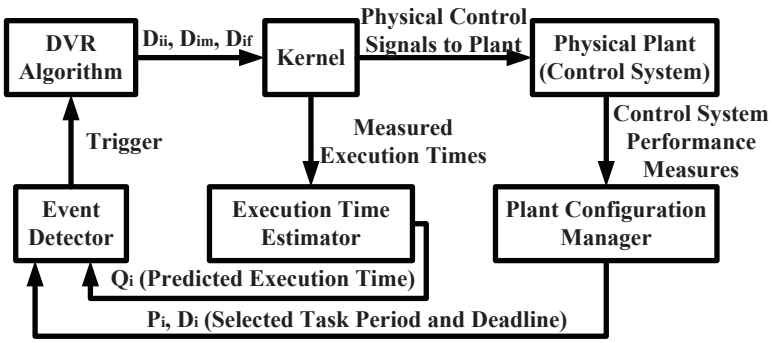


Figure 2.2. Adaptive framework for delay variation reduction.

Our proposed framework is similar to the one in [25] and is shown in Figure 2.2. This framework triggers the execution of the DVR algorithm in response to changes in task parameters. The DVR algorithm selects the subtask deadlines, D_{ii}, D_{im}, and D_{if} based on the task set's current parameters. These deadlines are handed over to the *Kernel* which then executes the control tasks and issues the control signals to *Physical Plant*. The performance of the physically controlled plant is monitored by a *plant configuration Manager* which may request a readjustment of the control task's deadlines and periods based on the use of *anytime control* formalism [52]. The performance of *Kernel* is monitored by an *Execution*

Time Estimator. This estimator uses kernel-based measurements of the mean and worst-case execution times to determine a predicted execution time Q_i. The predicted execution time, Q_i, the selected period, P_i, and task deadline D_i, are then handed over to an *Event Detector*. This detector compares these parameters to task parameters that were previously used to compute the subtask deadlines. If there is a large difference, then a *trigger* is issued to cause the DVR algorithm to recompute the subtask deadlines. Determining the "optimal" triggering condition is being investigated. The triggering condition can be a constant empirically chosen threshold. It may also be possible to use a time-varying state-dependent threshold [69] to compute thresholds that optimize overall control system performance subject to a constraint on the average rate at which the intermediate deadlines are readjusted.

2.5 Evaluation

In this section, we first evaluate the performance and efficiency of our heuristic DVR based on randomly generated task sets and compare DVR with the iterative method TDB in [12] and the greedy method DSB in [14]. Then, we illustrate the use of our heuristic in solving real-world problems. Last, we demonstrate the effectiveness of our adaptive framework through the simulation of actual control systems.

2.5.1 Simulation Setup

We generated 1000 random task sets consisting of 5 tasks each were randomly generated for 9 different utilization levels ($U_{level} = 0.1, ..., 0.9$) with a total of 9000 task sets. The utilization level is defined to be $U_{level_i} = \sum_{j=1}^{5} \frac{C_j}{P_j}, i = 0.1, ..., 0.9$.

Each task is initially schedulable with (C_j, D_j, P_j), using the necessary and sufficient condition (2.2) in Theorem 1. In our experiment, we set the maximum hyperperiod, minimum period, and maximum period to $500,000$, $10,000$, and $40,000$, respectively. The precision was specified to be 100, whereas the maximum number of tries was set to $10,000$. The precision denotes the minimum increment in any task period. For example, if the precision is 100, a task period could be 5200, but not 5010. In a nutshell, the following steps were taken to generate a task set. First of all, a set of periods were randomly generated based on the minimum period, maximum period, hyperperiod bound, and precision. Task periods were generated in such a way that the hyperperiod was no larger than the maximum hyperperiod. Each task was randomly assigned an execution time such that the total utilization was equal to that specified by the user. No task will have a utilization that is greater than half of the specified total utilization. Then, each task was assigned an initial deadline $D_i^{initial}$ that ensured $\sum_{i=1}^{N} \frac{C_i}{D_i^{initial}} > 1$. As a final step, the random task set generator tested the schedulability of a task set using the necessary and sufficient condition in Theorem 1. If the task set was unschedulable, task deadlines $D_i^{initial}$ were randomly increased such that the new deadline was greater than the previous deadline but $\sum_{i=1}^{N} \frac{C_i}{D_i^{initial}}$ was still greater than 1. This final step was repeated until either a feasible task set had been found or the maximum number of tries had been reached. To allow the decomposable task model adopted by both DVR and TDB to be investigated, we randomly selected 3 tasks out of each task set to be decomposable, and assumed that the initial and final subtasks consumed 10% of the execution time of the corresponding control task. Furthermore, because the constraints of the subtask dependencies in the decomposable task model made it harder to construct initially feasible task

sets, we increased each task's deadline to $\frac{D_j+P_j}{2}$.

Our heuristic was implemented in C++, running on a Sun Ultra 20 (x86-64) workstation with Red Hat Enterprise Linux 4. To demonstrate the performance of DVR, we complete the following comparisons in this section. First, we compare the number of problems that DVR is able to solve with what can be solved by TDB and DSB. Second, to assess the solution quality of DVR, we compare the solutions obtained by DVR with the results by TDB and DSB. Third, to show the efficiency of our heuristic, we compare the DVR's execution time for solving a batch of problems with those of TDB and DSB.

2.5.2 Performance of DVR

In the first experiment, we compare the percentage of solutions found by our heuristic, as opposed to those by TDB and DSB. Figure 2.3 compares the number of solutions found by DVR, TDB and DSB, respectively. If a task set cannot be

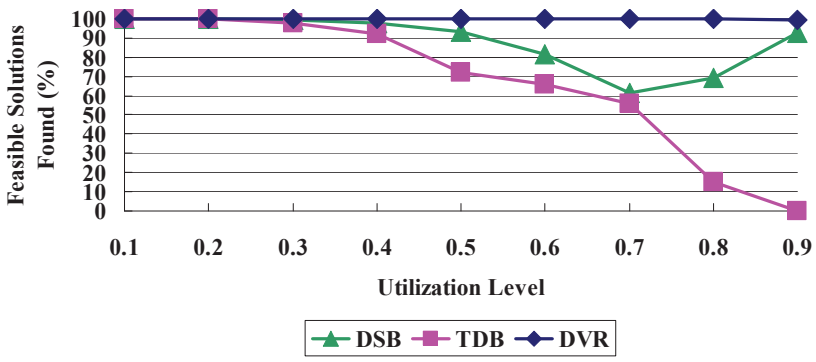


Figure 2.3. Comparison of DVR, TDB and DSB in terms of percentage of feasible solutions found.

solved by a method, its solution is said to be not found by this method. The x-axis shows the different utilization levels, whereas the y-axis shows the percentage of solutions found. It is clear from the plot that DVR is able to find solutions for all task sets with utilization levels 0.1 to 0.8, while for utilization level 0.9 DVR finds 995 solutions out of 1000 task sets. In contrast, TDB and DSB suffer from various degrees of degradation. With increasing utilization levels, more and more solutions found by TDB cannot satisfy the subtask dependency constraint, i.e., the found solutions are infeasible, because TDB blindly reduces deadlines of final subtasks, neglecting the deadlines of mandatory subtasks. DSB works better than TDB, but it cannot find solutions of many task sets for utilization levels 0.6 to 0.8. Compared with TDB and DSB, DVR performs excellently in obtaining a schedulable solution while guaranteeing subtask dependencies.

The second experiment examines the quality of the solutions found by our heuristic with respect to the original delay variations and that of the solutions found by TDB and DSB. Figure 2.4 illustrates the solution quality of DVR as well as TDB and DSB. As before, the x-axis shows the different utilization levels. The y-axis shows the average delay variations of the found solutions at each utilization level, i.e., $\frac{\sum_{j=1}^{5} \frac{DV_j}{5}}{1000}$, where DV_j is the delay variation of task τ_j in a task set. To guarantee the fairness of the comparison, the average delay variation of a task set that cannot be solved by a specific method is set to the original average delay variation. The first, and most obvious, observation is that the average delay variations resulted from applying DVR are much smaller than the values by DSB and the original average delay variations. In addition, with increasing utilization levels, such a delay variation difference becomes greater. Actually, DSB gives worse delay variations than the original delay variations at each utilization level,

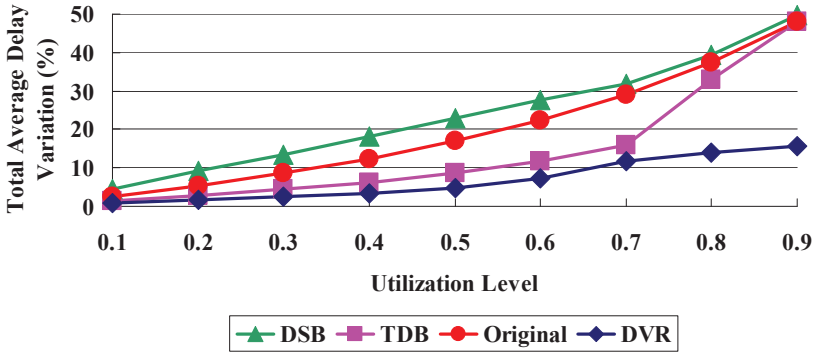


Figure 2.4. Comparison of DVR, TDB and DSB in terms of solution quality.

because its blind deadline reduction increases the delay variations. Second, for utilization levels less than or equal to 0.7, TDB performs a little worse than DVR, while for utilization levels greater than 0.7, the performance of TDB degrades drastically. The reason for this is that the numbers of found solutions by TDB for utilization levels greater than 0.7 decrease greatly (see Figure 2.3), and the average delay variations for these task sets without a solution by TDB are the original delay variations. The results of the first and second experiments show that DVR performs best in applications with various utilizations among the three methods.

To examine whether DVR is suitable for on-line dynamic deadline adjustments, we study the execution times of DVR and compare them with TDB and DSB in the third experiment. Figure 2.5 compares the execution times of DVR, TDB and DSB. The x-axis shows the different utilization levels, whereas the y-axis shows the average execution time which it takes by an algorithm. As shown in Figure 2.5, TDB spends 24.5 milliseconds on average in searching a solution at utilization level 0.9, and only finds 1 solution out of 1000 task sets finally. DVR and DSB

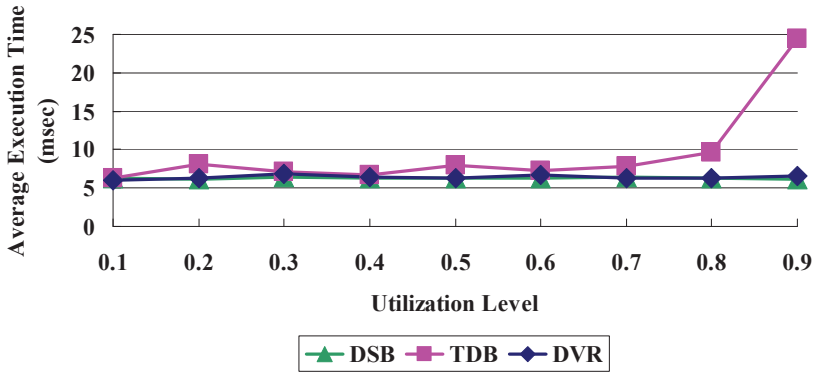


Figure 2.5. Comparison of DVR, TDB and DSB in terms of average execution time.

only take 6.58 and 6.21 milliseconds on average to search a solution at utilization level 0.9, respectively, which is almost 4 times faster than TDB. Furthermore, at all utilization levels, TDB always spends longer execution time than DVR and DSB, and DVR has the execution time comparable with that of the greedy method DSB.

To further investigate the convergence characteristics of DVR, the average and maximum numbers of iterations to search for a solution by DVR at various utilization levels are summarized in Table 2.3. As shown in Table 2.3, DVR converges very fast at utilization levels 0.1 to 0.6 with the average number of iterations less than 10 and the maximum number of iterations 19. For higher utilization levels 0.7 to 0.9, the maximum number of iterations increase obviously. However, most of the task sets at high utilization levels can be solved within 50 iterations. For example, 894 task sets at utilization level 0.9 can be solved within 50 iterations, 568 out of which only need 25 iterations or less. The relatively high number of iterations at higher utilization levels is caused by the increased amount

of preemption, which requires the adjustment of subtasks' deadlines repeatedly in DVR. However, the execution time of the deadline adjustment is very short, which effectively restrains the average execution time at high utilization levels from increasing drastically.

TABLE 2.3

Number of Iterations for Solving a Task Set by DVR.

Utilization Level	Average Number of Iterations	Maximum Number of Iterations
0.1	9.43	19
0.2	9.29	19
0.3	9.21	19
0.4	9.33	19
0.5	9.59	19
0.6	9.5	19
0.7	11.43	38
0.8	16.31	58
0.9	25.98	106

The experimental results on the execution time and convergence rate of DVR demonstrate that DVR can solve problems with various utilization levels efficiently

and have an acceptable convergence rate even for task sets with a high utilization level. The power of DVR lies in its effective problem formulation as well as its efficient search strategy.

2.5.3 Experimental Data for Real-World Workload

Using a large number of randomly generated task sets, we have shown that our heuristic can reduce delay variations greatly in almost all the task sets at various utilization levels. However, it is important to quantify the performance of DVR under real-world workloads. In this section, we compare DVR with TDB and DSB using three real-world applications. The applications we consider are a videophone application, a computerized numerical control (CNC) application, and an avionics application. We will describe the benchmarks one by one, and compare the results obtained by the three methods.

1) *Videophone Application:* A benchmark for a typical videophone application is presented in [137]. The task set is composed of four real-time tasks: video encoding, video decoding, speech encoding, and speech decoding. Delay variations in such an application would degrade the user perceived quality of voice or images. The worst-case execution time and period for each task is given. Task deadlines are randomly generated to be at least 95% of the period by using a uniform distribution. Since the videophone application was proposed in 2001, it is reasonable to reduce each task execution time by 10%. Thus, the utilization of the task set becomes 0.89. Assume that the context switch overhead of partitioning the encoding tasks is non-negligible, so only the video decoding and speech decoding tasks are decomposed based on the IMF model, where the initial and final subtasks consume 10% of the execution time of the corresponding control

task.

DVR, TDB and DSB are applied to the application to reduce delay variations. DVR takes 7 iterations to reduce average delay variations from 58.05% to 5.96%, while both TDB and DSB fail to find a feasible solution.

2) *Computerized Numerical Control Application* A benchmark is presented in [82], consisting of eight tasks with measured execution times and derived periods and deadlines of some CNC controller tasks. Consider the situation where the system is overloaded (e.g., the primary computer is down and the backup computer is less powerful). The increased task execution times can cause much higher delay variations and degrade the controller performance significantly. Thus, it is necessary to employ an efficient method to reduce delay variations. To model such a scenario, we multiply each task execution time by a factor of 1.6 and increase the utilization of the task set to 0.78. We assume that all the tasks in the application are decomposed based on the IMF model where the initial and final subtasks consume 10% of the execution time of the corresponding control task.

DVR can reduce the average delay variation from 69.9% to 2.17%, requiring 23 iterations in total, while TDB fails to find a feasible solution and DSB reduces the average delay variation only to 67.8%.

3) *Generic Avionics Application* The authors in [104] present a benchmark containing one aperiodic task and 17 periodic tasks with given period and execution time for each task, for a Generic Avionics Platform (GAP). We randomly assign the task deadlines to be between 50% and 60% of the periods using a uniform distribution. The aperiodic task arrives with a specific deadline and a minimum inter-arrival time. In addition, since the GAP benchmark dates back to 1991, it is reasonable to reduce each task execution time by 20%. The utilization of the

task set becomes 0.72. In this application, we still assume that all the tasks are decomposed based on the IMF model where the initial and final subtasks consume 10% of the execution time of the corresponding control task.

DVR reduces the average delay variation from 23.42% to 1.63% within 22 iterations. TDB is unable to find a feasible solution, and DSB actually increases the average delay variation to 30.64% after 324 iterations.

According to the experimental results of the three real-world benchmarks, we show that DVR is capable of reducing delay variations greatly within a limited number of iterations. In contrast, TDB always fails to find a feasible solution for such high utilization level applications, while DSB cannot guarantee to find a satisfactory solution even after quite a few iterations.

2.5.4 Performance of Adaptive-DVR

We use a simple system composed of one hard real-time task and three control tasks, similar to the systems employed in [18, 19, 26], to illustrate the efficiencies of DVR for delay variation reduction and control performance improvement. The hard real-time task is a non-control task without jitter and delay requirement except for schedulability, while each control task τ_i actuates a continuous-time system of two stable poles P_i. In order to vary the sensitivity towards delay variations, we employ the method presented in [23] to generate random plants P_i, which are given by

$$\begin{aligned} P_1(s) &= \frac{1000}{(s+0.5)(s-0.2)}, \\ P_2(s) &= \frac{1000}{(s+0.5)(s-0.3)}, \\ P_3(s) &= \frac{1000}{(s+0.2)(s-0.7)}. \end{aligned} \tag{2.27}$$

In [23, 31], the control performance cost J_i of each system is defined to be

$$J_i = \lim_{T \longrightarrow \infty} \int_0^T (y_i(t)^2 + u_i(t)^2) dt, \tag{2.28}$$

where $y_i(t)$ and $u_i(t)$ are the system output and input controlled by task τ_i, respectively. Thus, the total control performance cost of the continuous-time system is

$$J = \sum_{i=1}^{3} J_i. \tag{2.29}$$

In addition, one hard real-time task and three control tasks are generated by the method presented in [23], as shown in columns 2 to 4 of Table 2.4. All the tasks have transient and location dependent deadlines, similar to the cases in [135, 136].

We compare the delay variation values of the control tasks by applying DVR with those obtained by TDB and DSB. Furthermore, given the delay variation and sampling periods of a control system, we compute control performance cost by using Jitterbug proposed in [32, 95].

The delay variation value and control performance cost of each control task in the original task set and the corresponding values obtained by DSB, TDB and DVR are shown in column 2 of Table 2.5. For each task, two rows of data are shown in column 2. The top row corresponds to delay variation while the bottom one corresponds to control performance cost. The data clearly show that DVR reduces the average delay variation from 11.59% to 5.94% and improves the total performance cost from 330.8 to 271. However, DSB neither reduces the delay variations nor improves the total control performance. TDB fails to find a feasible solution for the task set.

Now, assume that at some time interval, the execution times of all the tasks

TABLE 2.4

A Control System Containing One Hard Real-Time Task and Three Control Tasks.

Task Name	Computation Exec. Time	Deadline	Period	New Computation Exec. Time	New Deadline
Hard Real-time Task	570	3810	3810	1140	2290
Control Task τ_1	1570	10000	10000	3140	10000
Control Task τ_2	855	1500	6860	1710	5710
Control Task τ_3	429	1500	8570	857	7570

increase by 2 times, due to the primary processor being down and the backup processor being much slower. Meanwhile, the deadlines of the hard real-time task and the control tasks τ_2 and τ_3 are also changed to satisfy the user's requirement. The new execution times and deadlines of the tasks are shown in columns 5 and 6 of Table 2.4. If the previous deadline assignments generated by DSB and DVR are reused for the tasks/subtasks, the deadlines of the hard real-time task and the control tasks will be missed. If no delay variation reduction method is applied to the current case, the delay variation and control performance cost of each control task are shown as the first numbers in column 3 of Table 2.5 (for each task,

top row for delay variation and bottom for control performance). The average delay variation and total control performance cost increase to 57.86% and 2453.79, respectively, which may not be acceptable to the system.

Suppose DSB, TDB and DVR methods are applied on-line in response to the workload and deadline change, the new delay variation values and control performance costs are shown as the second to the fourth numbers in column 3 of Table 2.5. With our proposed approach (DVR), smaller average delay variation 4.74% and total control performance cost 266 can be obtained in the current situation, which overcomes the negative effects on the system caused by the workload and deadline change. However, TDB fails to find a feasible solution while DSB gives average delay variation 57.77% and total control performance cost 1617.3, much worse than the results obtained by DVR.

Algorithm 1 DVR(Γ_{IMF}, Γ, *maxIter*)

1: $ePID = Sort\_in\_Task\_Exe\_Time(\Gamma)$
2: $Construct\_Initial\_Solution\_and\_Subproblem(\Gamma_{IMF}, \Gamma, currD)$
3: $Compute\_Busy\_Period\_and\_Hyper\_Period(\Gamma, L\_busy, Hyper\_Period)$
4: $state = 0$, $duplicate = 0$, $BestObjF = +\infty$
5: **for** $h = 0, h < maxIter, h = h + 1$ **do**
6: **if** $duplicate == \beta$ **then**
7: break
8: **end if**
9: $feasibility = Feasibility\_Test(\Gamma_{IMF}, \Gamma, currD, L\_busy, Hyper\_Period, L, time\_demand, h)$ //test whether $currD$ satisfies constraint (2.4)
10: **if** $feasibility == 1$ **then** //feasible
11: $ObjF = Obj\_Compute(\Gamma, \Gamma_{IMF}, currD)$
12: **if** $|ObjF - BestObjF| < \delta$ **then**
13: $duplicate = duplicate + 1$
14: **end if**
15: **if** $ObjF < BestObjF$ **then**
16: $Record\_Current\_Solution(\Gamma, \Gamma_{IMF}, bestD, currD, BestObjF, ObjF)$
17: $duplicate = 0$
18: **end if**
19: **if** $state == 1$ **then**
20: $Construct\_New\_Subproblem(\Gamma, \Gamma_{IMF}, currD)$ //update $currD_{im} = D_i - currD_{if}$ to formulate a new subproblem
21: **else**
22: $Optimize\_Solution(\Gamma, \Gamma_{IMF}, currD)$ //apply Theorem 3 to compute a new set of $currD_{if}$
23: **end if**
24: $state = !state$
25: **else if** $feasibility == 0$ **then** //not feasible
26: $overload = time\_demand - L$
27: **if** $state == 1$ **then**
28: $adjust\_result = Final\_Deadline\_Adjust(\Gamma, \Gamma_{IMF}, currD, overload, L, ePID)$ //adjust $currD_{if}$ to make the found solution schedulable
29: **else**
30: $adjust\_result = Mandatory\_Deadline\_Adjust(\Gamma, \Gamma_{IMF}, currD, overload, L, ePID)$ //adjust $currD_{im}$ to make solution feasible
31: **end if**
32: **if** $adjust\_result == 0$ **then** //the deadlines cannot be adjusted
33: break
34: **end if**
35: **end if**
36: **end for**
37: **return** *bestD*

TABLE 2.5

Control Task Delay Variations and Their Control Performance by Adaptively Applying DVR.

Task Name	Delay Variations (%) Control Performance Cost Original / DSB / TDB / DVR	New Delay Variations (%) Control Performance Cost Original / DSB / TDB / DVR
Hard	33.7 / 33.7 / Fail / 37.82	7.27 / 23.12 / Fail / 29.52
Real-time Task	NA / NA / NA / NA	NA / NA / NA / NA
Control	18.54 / 18.54 / Fail / 12.84	59.87 / 56.84 / Fail / 8.08
Task τ_1	153.3 / 153.3 / Fail / 112.5	2089.49 / 1246.6 / Fail / 105.2
Control	6.25 / 6.25 / Fail / 1.16	45.58 / 48.52 / Fail / 4.17
Task τ_2	82.2 / 82.2 / Fail / 71.9	123.8 / 130.2 / Fail / 74.2
Control	9.98 / 9.98 / Fail / 3.83	68.14 / 67.94 / Fail / 1.98
Task τ_3	95.3 / 95.3 / Fail / 86.6	240.5 / 240.5 / Fail / 86.6
Average Delay Variation	11.59 / 11.59 / Fail / 5.94	57.86 / 57.77 / Fail / 4.74
Total Performance Cost	330.8 / 330.8 / Fail / 271	2453.79 / 1617.3 / Fail / 266

CHAPTER 3

Local-Deadline Assignment for Distributed Real-Time Systems

In a DRTS, jobs are often executed on a number of processors and must complete by their end-to-end deadlines. Job deadline requirements may be violated if resource competition among different jobs on a given processor is not considered. We introduce a distributed, locally optimal algorithm to assign local deadlines to the jobs on each processor without any restrictions on the mappings of the applications to the processors in the distributed soft real-time system. Improved schedulability results are achieved by the algorithm since disparate workloads among the processors due to competing jobs having different paths are considered. Given its distributed nature, the proposed algorithm is adaptive to dynamic changes of the applications and avoids the overhead of global clock synchronization.

3.1 Introduction

Distributed soft real-time systems are widely used in cyber-physical applications such as the multimedia [68, 84], telecommunication [106], and automatic control and monitoring systems [94]. Since such systems often experience large variations in terms of their operating environments, a number of task deadlines may be missed without severely degrading performance. The scale of these distributed soft real-time systems often prohibits a centralized resource management

approach. Designing low-overhead, distributed scheduling solutions is critical to a reliable operation of such systems.

A DRTS contains a set of tasks periodically or aperiodically releasing jobs which typically have end-to-end deadlines. Each job is composed of a set of sub-jobs that are executed on different processors. Since different tasks may require execution on different sets of processors, there may be high resource competition among sub-jobs on a given processor, which could severely increase job response times, potentially resulting in end-to-end deadline misses. Although the distributed soft real-time system allows some jobs to miss their end-to-end deadlines, frequent deadline misses can degrade the Quality of Service (QoS) of the system. Therefore, it is important to properly assign local sub-job priorities in order to meet as many job deadlines as possible.

A number of recent papers investigated the sub-job priority assignment problem for DRTSs. Most of the local-deadline assignment approaches [27, 76, 158] divide the end-to-end deadline of a job into segments to be used as local deadlines by the processors that execute the sub-jobs. The division may depend on the number of processors on which the sub-job is executed [158] or the execution time distribution of the job among the processors [27, 76]. The local deadlines then dictate sub-job priorities according to the earliest-deadline-first (EDF) scheduling policy [29, 101]. While efficient, such approaches [27, 76, 158] do not consider resource competition of different sub-jobs on a processor, which may lead to local deadline misses and eventually end-to-end deadline violations.

To ensure the schedulability of the tasks on each processor, some work combines the local-deadline assignment problem with feasibility analysis so that the resulting deadline assignment is guaranteed to be schedulable. The approaches

proposed in [65, 132] assign local deadlines to the sub-jobs on-line by considering the schedulability of sub-jobs on each processor in a distributed manner. The approach [65] is based on a strong assumption that each processor knows the local release times and upper bounds on the local deadlines of all the future sub-jobs, which may be impractical for real-world applications. In [132], the absolute local deadline of each sub-job is derived on-line based on the sub-job completion time on the preceding processors and the given relative local deadline of each subtask. However, the work can not handle the situation where the relative local deadlines of subtasks are not given off-line. In contrast, the works in [87, 109, 131] assign intermediate deadlines to subtasks and consider resource contention among subtasks off-line. The schedulability condition used in work [87] (from [99]) utilizes the ratio of subtask execution time over subtask local deadline in the schedulability analysis. According to [29], this condition can be very pessimistic in testing the schedulability of subtask set when the subtask period is not equal to the subtask local deadline or the subtask is not periodic. The work in [109] employs the feasibility condition from [17] to assign local deadlines to subtasks on each processor in an off-line, iterative manner. The drawback of the approach is that it is time consuming and cannot adapt to dynamic changes in applications. In addition, the analysis assumes that all the periodic subtasks are synchronized, which is pessimistic in testing the schedulability of subtask set. The authors in [131] proposed a local-deadline assignment scheme to minimize processor resource requirements for a single task, yet many DRTSs need to execute multiple tasks.

To address the shortcomings of existing work, we present an on-line distributed approach which combines local-deadline assignment with feasibility analysis to meet as many applications' end-to-end deadline requirements as possible in a

distributed soft real-time system. Since the proposed approach is targeted towards soft real-time systems, it supports possibly infeasible applications. By extending our previous work [65], our local-deadline assignment algorithm supports soft real-time applications which can be modeled as a directed acyclic graph (DAG) and partitioned onto processors by whichever means. Our general application model covers a wide range of CPSs, e.g., multimedia system, data processing back-end systems, signal processing systems, control systems and wireless network systems.

In order to efficiently solve the local-deadline assignment problem, we formulate the local-deadline assignment problem for a given processor as a mixed integer linear programming (MILP) problem. We further introduce a locally optimal algorithm that can solve the MILP based local-deadline assignment problem in $O(N^4)$ time, where N is the number of sub-jobs executed by the processor. We should point out that the locally optimal solution may not be a globally optimal solution for the DRTS. Given the algorithm's distributed nature, the proposed algorithm avoids the overhead of global clock synchronization. In addition, the observations made in the proofs reveal several interesting properties (such as when a busy time interval occurs) for some special sub-job subsets used in our algorithm and can be applied to similar feasibility studies.

3.2 System Model

We consider a DRTS where a set of real-time tasks arrive either periodically or aperiodically and require execution on an arbitrary sequence of processors. Each task T_n is composed of a set of subtasks $T_{n,k}$ and has a relative end-to-end deadline $\mathcal{D}_n$. Since our focus is on an on-line distributed local-deadline assignment method, we only consider individual task and sub-task instances, i.e., jobs and sub-jobs,

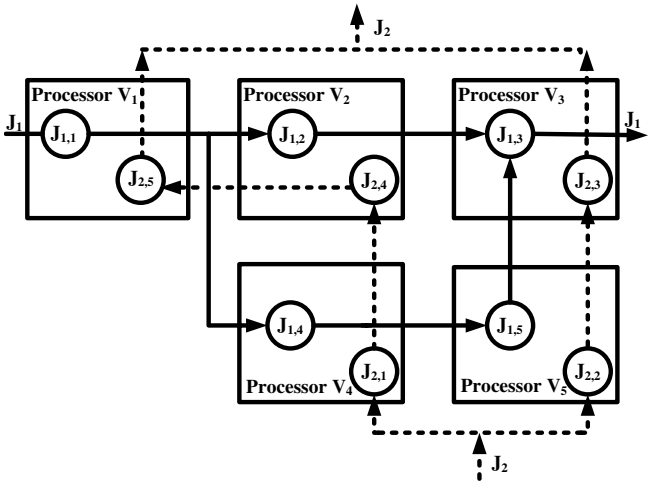


Figure 3.1. An example system containing two jobs, each with 5 sub-jobs being executed on 5 processors. In the example, $J_{1,1} \preceq J_{1,2} \preceq J_{1,3}$, $J_{1,1} \preceq J_{1,4} \preceq J_{1,5} \preceq J_{1,3}$, $J_{2,1} \preceq J_{2,4} \preceq J_{2,5}$, $J_{2,2} \preceq J_{2,3}$, $J_{1,1}$, $J_{2,1}$, $J_{2,2}$ are input sub-jobs, and $J_{1,3}$, $J_{2,5}$, $J_{2,3}$ are output sub-jobs.

respectively, without any assumption on task periodicity. Job J_i is composed of M_i sub-jobs $J_{i,k}$, $k = 1, ..., M_i$, where i and k are the index numbers of job J_i and subtask $T_{n,k}$, respectively[1]. Figure 3.1 shows a DRTS containing 2 jobs, J_1 and J_2, and each has 5 sub-jobs.

The precedence relationship among the sub-jobs of J_i is given by a directed acyclic graph (DAG). If sub-job $J_{i,k'}$ cannot begin its execution until sub-job $J_{i,k}$ has completed its execution, $J_{i,k}$ is a *predecessor* of $J_{i,k'}$, and $J_{i,k'}$ is the *successor* of $J_{i,k}$ (denoted as $J_{i,k} \prec J_{i,k'}$). $J_{i,k'}$ is an *immediate successor* of $J_{i,k}$ and $J_{i,k}$ is an *immediate predecessor* of $J_{i,k'}$ (denoted as $J_{i,k} \preccurlyeq J_{i,k'}$) if $J_{i,k}$ is the predecessor of $J_{i,k'}$ and no job $J_{i,h}$ satisfies $J_{i,k} \prec J_{i,h} \prec J_{i,k'}$. After all the immediate predecessors of a sub-job $J_{i,k}$ have finished their execution, sub-job $J_{i,k}$ is released and can start

[1]We omit a task's index number n when referring to a sub-job $J_{i,k}$ because we only consider jobs and sub-jobs.

TABLE 3.1

Summary of Key Notations Used

Symbol	Definition
V_x	A processor in the system
T_n, $T_{n,k}$	The nth task, and its kth subtask
$\Psi(V_x)$	Set of subtasks to be executed on processor V_x
$\Omega(V_x)$	Set of sub-jobs to be scheduled
J_i, $J_{i,k}$	The ith job, and sub-job $J_{i,k}$
$\mathcal{D}_n$	Relative end-to-end deadline of task T_n
D_i,	Absolute end-to-end deadline of job J_i
R_i	Release time of job J_i
$d_{i,k}$	Local deadline of sub-job $J_{i,k}$
$r_{i,k}$	Release time of sub-job $J_{i,k}$
$UB_{i,k}$	Local deadline upper bound of sub-job $J_{i,k}$
$s_{i,k}$	Time slack of sub-job $J_{i,k}$
$P_{i,k,k'}$, $\mathcal{P}_{i,k}$	A sub-job path starting from $J_{i,k}$ and ending with output sub-job $J_{i,k'}$, and the set of all the $J_{i,k}$'s paths
$C_{i,k}$	The worst-case execution time of sub-job $J_{i,k}$
$C_{i,k}^{cri}$	Critical execution time of sub-job $J_{i,k}$ (See (3.1))

executing. A sub-job without any predecessor is called an *input sub-job* and a sub-job without any successor is called an *output sub-job*. A sub-job path $P_{i,k,k'}$ is a chain of successive sub-jobs starting with sub-job $J_{i,k}$ and ending with an output sub-job $J_{i,k'}$. A sub-job may belong to multiple paths. We let $P_{i,k}$ be the set of paths $P_{i,k,k'}$'s starting from $J_{i,k}$. See Figure 3.1 for examples of these definitions.

Job J_i is released at time R_i, and must be completed by its absolute end-to-end deadline, D_i, which is equal to $R_i + \mathcal{D}_n$. All the input sub-jobs of J_i are released at time R_i, and all the output sub-jobs of J_i must be completed by

time D_i. The worst-case execution time of $J_{i,k}$ is $C_{i,k}$, and $J_{i,k}$ is associated with an absolute release time $r_{i,k}$ and absolute local deadline $d_{i,k}$, both of which are to be determined during the local-deadline assignment process. (We adopt the convention of using upper letters to indicate known values and lower letters for variables.)

We consider a multiprocessor system, where each processor V_x has a set $\Omega(V_x)$ of sub-jobs. We use $J_{i,k} \in \Omega(V_x)$ to indicate that sub-job $J_{i,k}$, an instance of subtask $T_{n,k}$, is executed on processor V_x. Subtask $T_{n,k}$ belongs to set $\Psi(V_x)$ of subtasks that reside on V_x, i.e., $T_{n,k} \in \Psi(V_x)$. Note that we do not assume any execution order among the processors in the distributed system. That is, processor V_x may appear before processor V_y in a sub-job's path while the order of the two processors may be reversed in another sub-job's path. In Figure 3.1, we have 5 processors, where $J_{1,1}, J_{2,5} \in \Omega(V_1)$, $J_{1,2}, J_{2,4} \in \Omega(V_2)$, $J_{1,3}, J_{2,3} \in \Omega(V_3)$, $J_{1,4}, J_{2,1} \in \Omega(V_4)$, and $J_{1,5}, J_{2,2} \in \Omega(V_5)$.

One way to meet the jobs' end-to-end deadlines is to assign local deadlines such that all the sub-jobs on every processor are schedulable and that the local deadlines of all the output sub-jobs are less than or equal to the respective end-to-end deadlines. In order to ensure that end-to-end deadlines are not violated, it is important for predecessor sub-jobs not to overuse their shares of slacks and to leave enough time for successor sub-jobs. We define the critical execution time $C_{i,k}^{cri}$ as the longest execution time among all the paths in $\mathcal{P}_{i,k}$, i.e.,

$$C_{i,k}^{cri} = \max_{P_{i,k,k'} \in \mathcal{P}_{i,k}} \sum_{\substack{\forall J_{i,h} \in P_{i,k,k'} \\ J_{i,k} \prec J_{i,h}}} C_{i,h}. \tag{3.1}$$

Using the definition of the critical execution time $C_{i,k}^{cri}$, we define the time slack of

$J_{i,k}$ as the difference between D_i relative to $d_{i,k}$ and $C_{i,k}^{cri}$, i.e.,

$$s_{i,k} = D_i - d_{i,k} - C_{i,k}^{cri}. \tag{3.2}$$

The time slack provides information on the longest delay that a job can endure after the execution of sub-job $J_{i,k}$ for all of the respective output sub-jobs to meet their end-to-end deadlines. By using the end-to-end deadline and the critical execution time of the sub-job $J_{i,k}$, we define the upper bound on the local deadline of $J_{i,k}$ as

$$UB_{i,k} = D_i - C_{i,k}^{cri}, \tag{3.3}$$

which gives the maximum allowable value for the local deadline of $J_{i,k}$. According to (3.2) and (3.3), we see that maximizing the time slack of each sub-job on any processor provides the best opportunity for each sub-job to meet its local deadline, and for each job to satisfy its end-to-end deadline requirement. Table 3.1 presents the notations and definitions of the parameters and variables used throughout the paper.

We assume that EDF [29] is used on each processor since it is optimal in terms of meeting job deadlines for a uniprocessor[2]. A necessary and sufficient condition for schedulability under EDF on a uniprocessor is restated below with the notation introduced earlier.

Theorem 4. [39, 40] *Sub-job set $\Omega(V_x)$ can be scheduled by EDF if and only if*

[2]This does not imply that EDF is optimal for distributed systems.

$\forall J_{i,k}, J_{j,h} \in \Omega(V_x)$, $r_{i,k} \leq d_{j,h}$,

$$d_{j,h} - r_{i,k} \geq \sum_{\substack{\forall J_{p,q} \in \Omega(V_x), \\ r_{p,q} \geq r_{i,k}, \\ d_{p,q} \leq d_{j,h}}} C_{p,q}. \tag{3.4}$$

3.3 Motivation

We use a simple DRTS to illustrate the drawbacks of existing approaches in terms of satisfying the real-time requirements. The example application contains 2 jobs, J_1 and J_2, and both jobs are composed of a chain of four sub-jobs, which are sequentially executed on four processors, V_1, V_2, V_3 and V_4. The sub-jobs' execution times and the jobs' end-to-end deadlines are shown in columns 2 to 5 in Table 3.2.

TABLE 3.2

A Motivating Example Containing Two Jobs that Traverse Four Processors

Processor Name	Job J_1		Job J_2	
	Execution Time	End-to-End Deadline	Execution Time	End-to-End Deadline
Processor V_1	100	N/A	70	N/A
Processor V_2	200	N/A	430	N/A
Processor V_3	100	N/A	100	N/A
Processor V_4	600	1100	100	930

We consider two representative priority assignment methods: JA [70, 73] and BBW [27]. JA is a job-level fixed-priority based approach, employed by Jayachandran and Abdelzaher in [70, 73]. BBW, proposed by Buttazzo, Bini and Wu, is an end-to-end deadline partitioning based method. In [27], the local-deadline assignment by BBW is used as an input to partitioning hard real-time tasks onto multiprocessors. However, BBW can also be utilized to assign local deadlines to sub-jobs in a distributed soft real-time system when tasks have been partitioned onto different processors since it efficiently decomposes the job's end-to-end deadline in proportion to the sub-job's execution times on different processors.

In the motivating example, the local deadlines assigned by BBW (and the resultant sub-job response times) at each processor are indicated by the first value in columns 2 and 3 (and the first value in columns 4 and 5) of Table 3.3. The resultant sub-job response times at each processor obtained by JA are shown in the second value in columns 8 and 9. For example, under BBW, the local absolute deadline of sub-job $J_{1,1}$ on processor V_1 is 111 time units and the response time is 170 time units. BBW causes job J_1 to complete its execution on processor V_4 at time 1170 and miss its end-to-end deadline by 70 time units. The reason for J_1's end-to-end deadline miss is that BBW ignores the resource competition on individual processors and does not make the best use of the given resources, which results in an idle time interval $[470, 700]$ on processor V_3. JA performs much worse than BBW in reducing the response time of job J_1, and causes sub-job $J_{1,1}$ to complete its execution on processor V_4 at time 1400. This is because job J_1 is assigned a lower priority by JA and is preempted by J_2 on both processors V_1 and V_2. As a result, J_1 fails to meet its end-to-end deadline when its sub-job $J_{1,4}$ has a large execution time of 600 on processor V_4.

If there exists an alternative local-deadline assignment method that can consider both the workloads on a job's execution path and resource competition among different sub-jobs on a shared processor, adopting such a method may result in meeting the deadline requirements for both jobs J_1 and J_2. We will present one such method, OLDA (Omniscient Local-Deadline Assignment), in the subsequent sections. The new local deadlines obtained by OLDA are shown as the second values in columns 2 and 3 and the resultant response times are as given by the third values in columns 4 and 5 in Table 3.3. It is clear that this local-deadline assignment allows both jobs to meet their end-to-end deadlines.

TABLE 3.3

Local-Deadline Assignment and Response Time of A Motivating Example

Processor Name	Local-Deadline Assignment BBW / OLDA		Response Time BBW / JA / OLDA	
	Job J_1	Job J_2	Job J_1	Job J_2
Processor V_1	111 / 100	90 / 170	170 / 170 / 100	70 / 70 / 170
Processor V_2	331 / 300	663 / 730	370 / 700 / 300	700 / 500 / 730
Processor V_3	441 / 400	797 / 830	470 / 800 / 400	800 / 600 / 830
Processor V_4	1100 / 1100	930 / 930	1170 / 1400 / 1100	900 / 700 / 930

If there exists an alternative local-deadline assignment method that can con-

sider both the workloads on a job's execution path and resource competition among different sub-jobs on a shared processor, adopting such a method may result in meeting the deadline requirements for both jobs J_1 and J_2. We will present one such method, OLDA (Omniscient Local-Deadline Algorithm), in the subsequent sections. The new local deadlines obtained by OLDA are shown as the second values in columns 2 and 3 and the resultant response times are as given by the third values in columns 4 and 5 in Table 3.3. It is clear that this local-deadline assignment allows both jobs to meet their end-to-end deadlines.

3.4 Approach

In this section, we provide a high-level overview of our approach and present the detailed MILP formulation for finding a locally optimal local-deadline assignment. Since our objective is to assign local deadlines to a set of sub-jobs on-line, we will only use the concepts of jobs and sub-jobs from now on.

3.4.1 Overview

As shown in the last section, the probability that jobs meet their end-to-end deadlines can be greatly increased if appropriate local deadlines are assigned to the sub-jobs on different processors. Although it is possible to accomplish local sub-job deadline assignment in a global manner using mathematical programming or dynamic programming, such approaches incur high computation overhead and are not suitable for on-line use.

We adopt a distributed, on-line approach to determine local sub-job deadlines on each processor. At Algorithm 2, every time a new sub-job arrives at processor V_x, new deadlines are assigned for both the newly arrived sub-job and current

Algorithm 2 Distributed On-Line Approach in Processor V_x

1: Upon completing sub-job $J_{j,h}$ in V_x:
2: Send a message to V_y's that are to execute $J_{j,h'}$'s which satisfy $J_{j,h} \preccurlyeq J_{j,h'}$ and $J_{j,h'} \in \Omega(V_y)$
3: $\Omega(V_x) = \Omega(V_x) - \{J_{j,h}\}$
4: Execute $J_{j',h'}$ which satisfies $d_{j',h'} = \min_{J_{i,k} \in \Omega(V_x)}\{d_{i,k}\}$
5: Upon receiving a message on the completion of sub-job $J_{j,h}$ from V_y:
6: Suspend the currently executing sub-job
7: Release $J_{j,h'}$'s which satisfy $J_{j,h} \preccurlyeq J_{j,h'}$ and $J_{j,h'} \in \Omega(V_y)$, and calculate $UB_{j,h'}$'s of $J_{j,h'}$'s
8: Re-assign $d_{i,k}$'s to $J_{i,k}$'s in V_x
9: Update the dropped job record in V_x
10: Send an acknowledgement message to V_y
11: Execute $J_{j',h'}$ which satisfies $d_{j',h'} = \min_{J_{i,k} \in \Omega(V_x)}\{d_{i,k}\}$
12: Upon receiving an acknowledgement message from V_y:
13: Update the dropped job record in V_x
14: Upon dropping a subset of sub-jobs in V_x:
15: Update the dropped job record in V_x

active sub-jobs which are already in V_x and may have been partially executed (Section 4.1). Upon the completion of a sub-job at V_x, V_x sends a message to those downstream processors which are to execute the immediate successors of the completed sub-job. The downstream processors utilize the information contained in the message to release new sub-jobs. Consider the example shown in Figure 3.1. Suppose at time t, $J_{2,4}$ arrives at V_2 which is executing $J_{1,2}$. Processor V_2 suspends the execution of $J_{1,2}$ and assigns new local deadlines to $J_{2,4}$ and $J_{1,2}$. If at the same time, $J_{2,3}$ arrives at V_3, V_3 simultaneously assigns the new local deadlines to $J_{2,3}$ and $J_{1,3}$. If a feasible deadline assignment is not found, a sub-job dropping policy is followed to remove a job from further processing. The drop information is propagated to the subsequent processors using some communication mechanism (Section 4.3.1).

The key to making the above distributed approach effective lies in the design

of an appropriate local-deadline assignment algorithm to be run on each processor such that some specific QoS metric for the DRTS is achieved, e.g., the number of jobs dropped is minimized. In our framework, each processor determines the local-deadline assignment to maximize the minimum time slack of sub-jobs on the corresponding processor. (Readers can see the explanation of this objective in our previous work [65]). To achieve this goal, we formulate an MILP problem to capture the local-deadline assignment on each processor (Section 3.5). Then, we devise an exact off-line algorithm that can solve the MILP problem in polynomial time (more details in Section 3.6). The MILP problem and the off-line algorithm provide a theoretical foundation for the practical on-line local-deadline assignment algorithms (Chapter 4). It is important to note that our overall framework is a heuristic since the objective used by each processor to determine the local-deadline assignment does not guarantee to always lead to a globally optimal solution. (The local-deadline assignment problem for DRTS is an NP-hard problem [20].) Below, we present the MILP formulation for local-deadline assignment as it forms the basis for our off-line algorithm.

3.5 Mathematical Programming Formulation

In order to ensure that a local-deadline assignment leads to a feasible schedule on a processor, we must consider resource competition among different sub-jobs on that processor. Furthermore, a sub-job should leave as much time slack (see the definition in (3.2)) as possible for successor sub-jobs to help satisfy the end-to-end deadline requirements. Hence, our goal is to determine the local deadline $d_{i,k}$ for sub-job $J_{i,k}$ such that the end-to-end deadline of J_i is met, the sub-job set $\Omega(V_x)$ on processor V_x is schedulable (see Theorem 4), and (3.2) is maximized. Assume

that the release times and upper bounds on the local deadlines of all the sub-jobs are known, we capture the problem as a constrained optimization problem given below:

$$\text{max:} \quad \left\{D_i - d_{i,k} - C_{i,k}^{cri}\right\} \qquad \forall J_{i,k} \in \Omega(V_x) \tag{3.5}$$

$$\text{s.t.} \quad r_{i,k} + C_{i,k} \leq d_{i,k} \leq D_i - C_{i,k}^{cri}, \qquad \forall J_{i,k} \in \Omega(V_x) \tag{3.6}$$

$$d_{j,h} - r_{i,k} \geq \sum_{\substack{\forall J_{p,q} \in \Omega(V_x), \\ r_{i,k} \geq r_{p,q}, \\ d_{p,q} \leq d_{j,h}}} C_{p,q}, \qquad \forall J_{j,h}, J_{i,k} \in \Omega(V_x). \tag{3.7}$$

It is difficult to maximize the time slacks of all sub-jobs on a processor, since the sub-jobs compete with each other for the shared computation resource severely. Therefore, we resort to the objective of maximizing the minimum time slack among all the sub-jobs executed on V_x as follows,

$$\text{max:} \min_{J_{i,k} \in \Omega(V_x)} \quad \left\{D_i - d_{i,k} - C_{i,k}^{cri}\right\}. \tag{3.8}$$

Constraints (3.6)–(3.7) are used to guarantee schedulability on each processor. Specifically, constraint (3.6) bounds the local deadlines of sub-jobs executed on V_x by the earliest completion time of the sub-job (left side of (3.6)) and the latest allowable completion time of the sub-job (right side of (3.6)). Constraint (3.7) is simply a restatement of (3.4).

An astute reader would notice that the above constraint optimization problem formulation cannot be straightforwardly solved by a mathematical programming solver. This is because the actual terms to be included in the summation on the

right hand side of constraint (3.7) depend on local deadlines which are themselves decision variables. To overcome this challenge, we introduce an observation in Lemma 1 below which can be used to convert constraint (3.7) to a form readily solvable by a mathematical programming solver. The essence of the observation is that the EDF schedulability of a sub-job set can be checked by examining certain behavior of all the sub-job subsets in the set. This observation also plays a key role in developing the efficient algorithm to be presented later.

Lemma 1. *Given sub-job set $\Omega(V_x)$ to be executed on processor V_x according to the EDF policy, let $\omega(V_x)$ represent any subset of $\Omega(V_x)$. $\Omega(V_x)$ is schedulable if and only if*

$$\max_{J_{i,k}\in\omega(V_x)}\{d_{i,k}\} - \min_{J_{i,k}\in\omega(V_x)}\{r_{i,k}\} \geq \sum_{\forall J_{i,k}\in\omega(V_x)} C_{i,k},$$
$$\forall\omega(V_x) \subseteq \Omega(V_x). \tag{3.9}$$

Proof: *We first prove the "if" part. Assume that $\forall\omega(V_x) \subseteq \Omega(V_x)$, constraint (3.9) holds, i.e.,*

$$\max_{J_{i,k}\in\omega(V_x)}\{d_{i,k}\} - \min_{J_{i,k}\in\omega(V_x)}\{r_{i,k}\} \geq \sum_{\forall J_{i,k}\in\omega(V_x)} C_{i,k}, \forall\omega(V_x) \subseteq \Omega(V_x). \tag{3.10}$$

Then for each pair of $r_{j,h}$ and $d_{j',h'}$ with $J_{j,h}, J_{j',h'} \in \Omega(V_x)$, there must exist a sub-job subset $\omega(V_x)$ in $\Omega(V_x)$ such that

$$\omega(V_x) = \{J_{i,k} | r_{i,k} \geq r_{j,h}, d_{i,k} \leq d_{j',h'}, \forall J_{i,k} \in \Omega(V_x)\}.$$

Thus, $\forall J_{j,h}, J_{j',h'} \in \Omega(V_x)$*, we have*

$$\max_{J_{i,k}\in\omega(V_x)}\{d_{i,k}\}-\min_{J_{i,k}\in\omega(V_x)}\{r_{i,k}\} = d_{j',h'}-r_{j,h} \geq \sum_{\forall J_{i,k}\in\omega(V_x)} C_{i,k} = \sum_{\substack{\forall J_{i,k}\in\Omega(V_x),\\ r_{i,k}\geq r_{j,h},\\ d_{i,k}\leq d_{j',h'}}} C_{i,k}.$$

By Theorem 4, sub-job set $\Omega(V_x)$ *is schedulable.*

Next, we prove the "only if" part. Assume that the sub-job set $\Omega(V_x)$ *is schedulable on* V_x*. According to Theorem 4,* $\forall J_{j,h}, J_{j',h'} \in \Omega(V_x)$*,* $r_{j,h} \leq d_{j',h'}$*, and*

$$d_{j',h'} - r_{j,h} \geq \sum_{\substack{\forall J_{i,k}\in\Omega(V_x),\\ r_{i,k}\geq r_{j,h},\\ d_{i,k}\leq d_{j',h'}}} C_{i,k}. \tag{3.11}$$

Given any sub-job subset $\omega(V_x)$*, let* $\max_{J_{i,k}\in\omega(V_x)}\{d_{i,k}\} = d_{j',h'}$ *and* $\min_{J_{i,k}\in\omega(V_x)}\{r_{i,k}\} = r_{j,h}$*. We have*

$$\max_{J_{i,k}\in\omega(V_x)}\{d_{i,k}\} - \min_{J_{i,k}\in\omega(V_x)}\{r_{i,k}\} \geq \sum_{\substack{\forall J_{i,k}\in\Omega(V_x),\\ r_{i,k}\geq r_{j,h},\\ d_{i,k}\leq d_{j',h'}}} C_{i,k} \geq \sum_{\forall J_{i,k}\in\omega(V_x)} C_{i,k}. \tag{3.12}$$

Therefore, constraint (3.9) holds and the lemma is proved. □

Based on Lemma 1, we can substitute constraint (3.7) in our optimization problem with constraint (3.9). We claimed that the formulated problem (3.8) together with (3.6) and (3.9) is an MILP problem. Constraint (3.6) is already in an MILP form. We now show how to transform the max and min functions in (3.8) and (3.9) into expressions in an MILP form. We introduce a variable $s^{min}_{\Omega(V_x)}$ to replace $\min_{J_{i,k}\in\Omega(V_x)}\{D_i - d_{i,k} - C^{cri}_{i,k}\}$ and transform (3.8) to the following objective plus

a new constraint,

$$\max: \quad s^{min}_{\Omega(V_x)}, \tag{3.13}$$

and

$$s^{min}_{\Omega(V_x)} \leq D_i - d_{i,k} - C^{cri}_{i,k}, \qquad \forall J_{i,k} \in \Omega(V_x). \tag{3.14}$$

Since the objective (3.13) is to maximize $s^{min}_{\Omega(V_x)}$ and (3.14) ensures that $s^{min}_{\Omega(V_x)}$ is smaller than or equal to the slack $s_{i,k}$ of any sub-job $J_{i,k}$ on processor V_x, $s^{min}_{\Omega(V_x)}$ is equal to the minimum time slack among all the sub-jobs executed on V_x.

To remove the max function in (3.9), we use a variable $d_{max,\omega(V_x)}$ to represent $\max_{J_{i,k}\in\omega(V_x)}\{d_{i,k}\}$ and then replace constraint (3.9) with the following constraints,

$$d_{max,\omega(V_x)} - \min_{J_{i,k}\in\omega(V_x)}\{r_{i,k}\} \geq \sum_{\forall J_{i,k}\in\omega(V_x)} C_{i,k}, \quad \forall\omega(V_x) \subseteq \Omega(V_x), \tag{3.15}$$

$$d_{max,\omega(V_x)} \geq d_{i,k}, \quad \forall J_{i,k} \in \omega(V_x), \quad \forall\omega(V_x) \subseteq \Omega(V_x), \tag{3.16}$$

$$d_{max,\omega(V_x)} \leq \sum_{\forall J_{i,k}\in\omega(V_x)} d_{i,k} \cdot A_{i,k,\omega(V_x)}, \quad \forall\omega(V_x) \subseteq \Omega(V_x), \tag{3.17}$$

and

$$\sum_{\forall J_{i,k}\in\omega(V_x)} A_{i,k,\omega(V_x)} = 1, \quad A_{i,k,\omega(V_x)} \in \{0,1\}, \quad \forall\omega(V_x) \subseteq \Omega(V_x), \tag{3.18}$$

where $A_{i,k,\omega(V_x)}$ are binary variables. Constraints (3.17) and (3.18) indicate that $d_{max,\omega(V_x)}$ is no greater than the deadline of one sub-job in $\omega(V_x)$. Without loss of generality, we suppose this sub-job is $J_{i',k'}$ whose deadline is $d_{i',k'}$. Meanwhile, since $d_{max,\omega(V_x)} \geq d_{i,k}$ for any sub-job $J_{i,k}$ in $\omega(V_x)$, we have $d_{max,\omega(V_x)} = d_{i',k'} = \max_{J_{i,k}\in\omega(V_x)}\{d_{i,k}\}$.

Although we remove the max function in (3.9), constraint (3.17) is non-linear.

To linearize (3.17), we use variable $\alpha_{i,k,\omega(V_x)}$ which is defined as follows,

$$\alpha_{i,k,\omega(V_x)} = \begin{cases} d_{i,k} & : \quad A_{i,k,\omega(V_x)} = 1 \\ 0 & : \quad A_{i,k,\omega(V_x)} = 0 \end{cases}, \quad \forall J_{i,k} \in \omega(V_x), \quad \forall \omega(V_x) \subseteq \Omega(V_x). \tag{3.19}$$

By replacing $d_{i,k} \cdot A_{i,k,\omega(V_x)}$ with $\alpha_{i,k,\omega(V_x)}$, the following constraint can be used instead of (3.17).

$$d_{max,\omega(V_x)} \leq \sum_{\forall J_{i,k} \in \omega(V_x)} \alpha_{i,k,\omega(V_x)}, \quad \forall \omega(V_x) \subseteq \Omega(V_x). \tag{3.20}$$

In addition to ensure that $\alpha_{i,k,\omega(V_x)}$ is as defined in (3.19), the following two constraints are needed,

$$\alpha_{i,k,\omega(V_x)} \leq A_{i,k,\omega(V_x)} \cdot M, \quad \forall J_{i,k} \in \omega(V_x), \quad \forall \omega(V_x) \subseteq \Omega(V_x), \tag{3.21}$$

and

$$\alpha_{i,k,\omega(V_x)} \leq d_{i,k,\omega(V_x)} + (1 - A_{i,k,\omega(V_x)}) \cdot M, \quad \forall J_{i,k} \in \omega(V_x), \quad \forall \omega(V_x) \subseteq \Omega(V_x), \tag{3.22}$$

where M is some large constant greater than zero. In summary, the problem in (3.8) together with (3.6) and (3.9) can be expressed as (3.13), (3.6), (3.14), (3.15), (3.16), (3.18), (3.20), (3.21) and (3.22).

If the release times of sub-jobs are known when computing the local deadlines, which is true in the proposed on-line approach, the resulting problem specified by (3.8), together with (3.6) and (3.9), can be solved by a mathematical programming solver. Though a solver can be used for solving the local-deadline assignment problem, it can be too time consuming for on-line use (see Section 4.4). In the

next section, we introduce a polynomial time algorithm to solve the problem exactly.

3.6 Omniscient Local-Deadline Assignment

In this section, we present the Omniscient Local-Deadline Assignment (OLDA), the canonical version of our local-deadline assignment algorithm, which solves the optimization problem given in (3.8), (3.6) and (3.9) in $O(N^4)$ (where N is the number of sub-jobs), assuming that the release times of all the existing and future sub-jobs are known a priori. Although OLDA is an off-line algoirthm, it forms the basis of the desired on-line algorithms (see Chapter 4). There are multiple challenges in designing OLDA. The most obvious difficulty is how to avoid checking the combinatorial number of subsets of $\Omega(V_x)$ in constraint (3.9). Another challenge is how to maximize the objective function in (3.8) while ensuring sub-job schedulability and meeting all jobs' end-to-end deadlines.

Below, we discuss how our algorithm overcomes these challenges and describe the algorithm in detail along with the theoretical foundations behind it. Unless explicitly noted, the deadline of a sub-job in this section always means the local deadline of the sub-job on the processor under consideration.

3.6.1 Base Subset and Base Sub-job

One key idea in OLDA is to construct a unique subset from a given sub-job set $\Omega(V_x)$. Using this sub-job subset, OLDA can determine the local deadline of at least one sub-job in $\Omega(V_x)$. This local deadline is guaranteed to belong to an optimal solution for the problem given in (3.8), (3.6) and (3.9). We refer to this unique sub-job subset of $\Omega(V_x)$ as the *base subset* of $\Omega(V_x)$ and define it as follows.

We first describe sub-job subsets that are candidates for the base subset of $\Omega(V_x)$. Subset $\omega^c(V_x)$ is a candidate for the base subset of $\Omega(V_x)$ if

$$d_{c,k^c} = \min_{J_{i,k} \in \omega^c(V_x)} \{r_{i,k}\} + \sum_{\forall J_{i,k} \in \omega^c(V_x)} C_{i,k} \geq \min_{J_{i,k} \in \omega(V_x)} \{r_{i,k}\} + \sum_{\forall J_{i,k} \in \omega(V_x)} C_{i,k}, \forall \omega(V_x) \subseteq \Omega(V_x),$$

where d_{c,k^c} is the earliest completion time of $\omega^c(V_x)$. We now formally define the base subset of $\Omega(V_x)$. Let $\{\omega^c(V_x) | \forall \omega^c(V_x) \subseteq \Omega(V_x)\}$ contain all the candidates of the base subset.

Definition 1. *$\omega^*(V_x)$ is a base subset of $\Omega(V_x)$, if it satisfies*

$$\min_{J_{i,k} \in \omega^*(V_x)} \{r_{i,k}\} > \min_{J_{i,k} \in \omega(V_x)} \{r_{i,k}\}, \omega^*(V_x) \in \{\omega^c(V_x) | \forall \omega^c(V_x)$$

$$\subseteq \Omega(V_x)\}, \forall \omega(V_x) \in \{\omega^c(V_x) | \forall \omega^c(V_x) \subseteq \Omega(V_x)\}.$$

The definition of the base subset of $\Omega(V_x)$ simply states that the completion time of the sub-jobs in the base subset is no less than that in any other sub-job subset in $\Omega(V_x)$ (such a property of the base subset will be proved in Lemma 5). If the completion times of all the sub-jobs in multiple subsets are the same, the base subset is the subset which has the latest released sub-job in $\Omega(V_x)$.

For a given base subset, determining which sub-job to assign a deadline to and what value the deadline should have constitutes the other key idea in OLDA. Recall that our optimization goal is to maximize the sub-job time slacks. Hence, we select this sub-job based on the local deadline upper bounds of all the sub-jobs

in the base subset. Let $J_{c,k^c} \in \omega^*(V_x)$ be a candidate for the base sub-job if

$$UB_{c,k^c} \geq UB_{i,k} \quad \forall J_{i,k} \in \omega^*(V_x).$$

Let $\{J_{c,k^c} | \forall J_{c,k^c} \in \Omega(V_x)\}$ contain all the candidates for the base sub-job in $\omega^*(V_x)$. We refer to the selected sub-job as the base sub-job and define it as follows.

Definition 2. *$J_{*,k^*} \in \{J_{c,k^c} | \forall J_{c,k^c} \in \Omega(V_x)\}$ is a base sub-job for sub-job set $\Omega(V_x)$, if it satisfies*

$$(* > i) \quad or \ (* = i \ \ and \ k^* > k) \quad \forall J_{i,k} \in \{J_{c,k^c} | \forall J_{c,k^c} \in \Omega(V_x)\}.$$

The base sub-job has the largest local deadline upper bound among all the sub-jobs in the base subset. Ties are broken in favour of the sub-job with the largest job identifier and then in favour of the sub-job with the largest subtask identifier.

We use a simple example to illustrate how to find base subset and base sub-job. Consider a sub-job set $\Omega(V_x)$ with its timing parameters as shown in the top part of Table 3.4. It is easy to verify that subset $\{J_{2,1}, J_{3,1}, J_{4,1}\}$ is the base subset $\omega^*(V_x)$ (see the bottom part of Table 3.4), where d^* is 9. Among the three sub-jobs in $\omega^*(V_x)$, sub-job $J_{2,1}$ is the base sub-job according to Definition 2 since it has the largest local deadline upper bound of 42. OLDA uses the base subset and base sub-job to accomplish local-deadline assignment. The details of OLDA is given in the next subsection.

TABLE 3.4

A Sub-job Set Example

Sub-job	$r_{i,k}$	$C_{i,k}$	$UB_{i,k}$
$J_{1,1}$	0	2	35
$J_{2,1}$	4	2	42
$J_{3,1}$	5	2	39
$J_{4,1}$	6	1	35

$\omega(V_x)$	$\min_{J_{i,k}\in\omega(V_x)}\{r_{i,k}\} + \sum_{\forall J_{i,k}\in\omega(V_x)} C_{i,k}$
$\{J_{1,1}, J_{2,1}, J_{3,1}, J_{4,1}\}$	7
$\{J_{2,1}, J_{3,1}, J_{4,1}\}$ $(\omega^*(V_x))$	9
$\{J_{3,1}, J_{4,1}\}$	8
$\{J_{4,1}\}$	7

3.6.2 OLDA Algorithm Design

Given a sub-job set $\Omega(V_x)$, OLDA first constructs the base subset for the sub-job set. It then finds the base sub-job and assigns a local deadline to that base sub-job. The base sub-job is then removed from the sub-job set and the process is repeated until all the sub-jobs have been assigned deadlines.

Algorithm 3 summarizes the main steps in OLDA. (Recall that this algorithm is used by each processor in a distributed manner, so the pseudocode is given for processor V_x.) The inputs to OLDA are the sub-job set $\Omega(V_x)$ and the variable $Max\_Allowed\_Drop\_Num$. $\Omega(V_x)$ contains all the active and future sub-jobs $J_{i,k}$'s. Without loss of generality, a sub-job is always associated with its local release time, execution time, and local deadline upper bound, and the local deadline upper bound of sub-job is computed before the call of OLDA. Thus, we do not use the local release time, execution time and local deadline upper bound as

Algorithm 3 OLDA($\Omega(V_x)$, $Max\_Allowed\_Drop\_Num$)

1: $\mathbf{d} = \emptyset$
2: $\Omega(V_x) = Sort\_Sub\_Jobs(\Omega(V_x))$
3: **while** $(\Omega(V_x) \neq \emptyset)$ **do**
4: $\omega(V_x) = \Omega(V_x)$
5: $\omega^*(V_x) = \Omega(V_x)$
6: $Max\_Deadline = 0$
7: $Temp\_Deadline = 0$
8: **while** $\omega(V_x) \neq \emptyset$ **do**
9: $Temp\_Deadline = \min_{J_{i,k} \in \omega(V_x)}\{r_{i,k}\} + \sum_{J_{i,k} \in \omega(V_x)}\{C_{i,k}\}$
10: **if** $Temp\_Deadline \geq Max\_Deadline$ **then**
11: $Max\_Deadline = Temp\_Deadline$
12: $\omega^*(V_x) = \omega(V_x)$
13: **end if**
14: $\omega(V_x) = Remove\_Earliest\_Released\_Sub\_Job(\omega(V_x))$
15: **end while**
16: $J_{*,k^*} = Find\_Base\_Sub\_Job(\omega^*(V_x))$ //Find base sub-job J_{*,k^*} according to Definition 2
17: **if** $(UB_{*,k^*} \geq Max\_Deadline)$ **then**
18: $d_{*,k^*} = Max\_Deadline$
19: $\mathbf{d} = \mathbf{d} \bigcup \{d_{*,k^*}\}$
20: $\Omega(V_x) = \Omega(V_x) - J_{*,k^*}$
21: **else**
22: $\mathbf{J^{drop}} = Drop\_Sub\_Jobs(\omega^*(V_x), Max\_Allowed\_Drop\_Num)$ //Remove a subset of sub-jobs from $\omega^*(V_x)$ according to some sub-job dropping policy, and return the subset containing the dropped sub-jobs
23: $\mathbf{d} = \emptyset$
24: break
25: **end if**
26: **end while**
27: return $\mathbf{d}$ //$\mathbf{d} = \{d_{i,k}\}$

the input variables in OLDA. The variable $Max\_Allowed\_Drop\_Num$ is used in Function $Drop\_Sub\_Jobs()$, which will be discussed in Section 4.2. OLDA starts by initializing the set of sub-job deadlines (Line 1) and sorting the given sub-jobs in a non-decreasing order of their release times (Line 2), which breaks ties in favour of the sub-job with the largest job identifier and then in favour of the sub-job with the largest subtask identifier. Then, the algorithm enters the main loop spanning from Line 3 to Line 26. The first part in the main loop (Lines 4–15) constructs the

base subset $\omega^*(V_x)$ for the given sub-job set and computes the desired deadline value ($Max\_Deadline$) according to Definition 1. ($Max\_Deadline$ is in fact the completion time of all the sub-jobs in the base subset, as will be shown in the next subsection.) The second part of the main loop (Line 16) applies Definition 2 to find the base sub-job in the base subset.

If the desired deadline value is smaller than or equal to UB_{*,k^*} of the base sub-job J_{*,k^*} (Line 17), the third part of the main loop (Lines 17–26) assigns the desired deadline value to the base sub-job as its local deadline (denoted by d_{*,k^*}) (Line 18), adds d_{*,k^*} to the set of sub-job deadlines (Line 19), and removes J_{*,k^*} from $\Omega(V_x)$ (Line 20). This process is repeated in the main loop until each sub-job in $\Omega(V_x)$ obtains a local deadline. In the case where the desired deadline value is larger than UB_{*,k^*} (Line 21), at least one sub-job will miss its deadline and a subset of sub-jobs $\mathbf{J^{drop}}$ are removed from the subset $\omega^*(V_x)$ based on some sub-job dropping policy (Line 22).(The discussion on the sub-job dropping policy is provided in Section 4.3.3.) Then, the set of sub-job deadlines is set to be empty (Line 23) and OLDA exits (Line 24). OLDA either returns the set of sub-job deadlines to be used by the processor in performing EDF scheduling or an empty set to processor V_x. In the latter case, V_x calls OLDA repeatedly until a feasible solution is found or all the sub-jobs in $\Omega(V_x)$ have been dropped. The time complexity of OLDA is $O(|\Omega(V_x)|^3)$, which is proved in Theorem 5, and a processor takes $O(|\Omega(V_x)|^4)$ time to solve a set $\Omega(V_x)$ using OLDA.

We use the example in Table 3.4 to illustrate the steps taken by OLDA to assign local deadlines given sub-job set $\{J_{1,1}, J_{2,1}, J_{3,1}, J_{4,1}\}$. In the first iteration of the main loop, OLDA finds the base subset $\{J_{2,1}, J_{3,1}, J_{4,1}\}$ and selects the base sub-job $J_{2,1}$. OLDA assigns the completion time of all the sub-jobs in the base

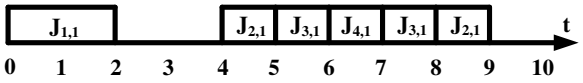


Figure 3.2. The example of executing sub-jobs with local deadlines assigned by OLDA

subset, 9, to $J_{2,1}$ as its local deadline. In the next iteration, OLDA works on sub-job set $\{J_{1,1}, J_{3,1}, J_{4,1}\}$ and the process is repeated until all the sub-jobs have been assigned local deadlines. In the example, the local deadlines for sub-jobs $J_{1,1}$, $J_{2,1}$, $J_{3,1}$ and $J_{4,1}$ are 2, 9, 8 and 7, respectively. The base subset and base sub-job in each iteration are shown in Table 3.5.

A possible schedule for sub-jobs $J_{1,1}$, $J_{2,1}$, $J_{3,1}$ and $J_{4,1}$ is shown in Figure 3.2.

It is worth noting that we assume that a processor knows the release times and local deadline upper bounds of all the future sub-jobs in OLDA (This assumption will be relaxed in Chapter 4). Thus, OLDA only requires information known upon a sub-job's release (such as the maximum allowed response time of a sub-job at the completion time of the sub-job's intermediate predecessor), which

TABLE 3.5

Base Subset and Base Sub-job in Each Iteration

Iter. Number	Sub-job Set	Base Subset	Base Sub-job
1	$\{J_{1,1}, J_{2,1}, J_{3,1}, J_{4,1}\}$	$\{J_{2,1}, J_{3,1}, J_{4,1}\}$	$J_{2,1}$
2	$\{J_{1,1}, J_{3,1}, J_{4,1}\}$	$\{J_{3,1}, J_{4,1}\}$	$J_{3,1}$
3	$\{J_{1,1}, J_{4,1}\}$	$\{J_{4,1}\}$	$J_{4,1}$
4	$\{J_{1,1}\}$	$\{J_{1,1}\}$	$J_{1,1}$

can be relayed between processors with the support of a specific distributed communication mechanism (Section 4.3). Therefore, OLDA does not require global clock synchronization.

3.6.3 Optimality of OLDA Algorithm

We claim that OLDA solves the optimization problem given by (3.8), (3.6) and (3.9). That is, if there exists a solution to the problem, OLDA always finds it. Furthermore, if there is no feasible solution to the problem, OLDA always identifies such a case, i.e., drop a job following some sub-job dropping policy. To support our claim, we first show that the local-deadline assignment made by OLDA (when no sub-jobs are dropped) satisfies the constraints in (3.6) and (3.9). This is given in Lemmas 2 and 3, respectively.

Lemma 2. *Given sub-job set $\Omega(V_x)$, let $d^*_{i,k}$ be the local deadline assigned by OLDA to $J_{i,k} \in \Omega(V_x)$. Then*

$$r_{i,k} + C_{i,k} \leq d^*_{i,k} \leq D_i - C^{cri}_{i,k} \quad \forall J_{i,k} \in \Omega(V_x). \tag{3.23}$$

Proof: *First, we prove that $r_{i,k} + C_{i,k} \leq d^*_{i,k}$ for any $J_{i,k}$ in any solution found by OLDA. Without loss of generality (WLOG), we prove the lemma for the first local deadline determined by OLDA (denoted as d_{*,k^*}). Let $d^{\omega}_{i,k} = \min_{J_{i,k} \in \omega(V_x)}\{r_{i,k}\} + \sum_{\forall J_{i,k} \in \omega(V_x)} C_{i,k}$ for any $\omega(V_x) \subseteq \Omega(V_x)$. According to Lines 4–15 of Algorithm 3, OLDA selects the sub-job subset $\omega^*(V_x)$, which has the maximum $d^{\omega}_{i,k}$ among all the subsets of $\Omega(V_x)$, and assigns this value to sub-job J_{*,k^*} as its local deadline.*

That is,

$$d_{*,k^*} = d_{i,k}^{\omega^*} \geq d_{i,k}^{\omega}, \qquad \forall \omega(V_x) \subseteq \Omega(V_x). \tag{3.24}$$

Suppose that $d_{*,k^*} < r_{*,k^*} + C_{*,k^*}$. *In such a case, we can find a sub-job subset* $\omega'(V_x)$ *containing only one sub-job* J_{*,k^*} *such that* $d_{i,k}^{\omega'} = r_{*,k^*} + C_{*,k^*} > d_{i,k}^{\omega^*}$. *This contradicts the condition in (3.24).*

Next, we prove that $d_{i,k}^* \leq D_i - C_{i,k}^{cri}$ *for any* $J_{i,k}$ *in the solution found by OLDA. Suppose there is a sub-job* $J_{p,q}$ *where* $d_{p,q} > D_p - C_{p,q}^{cri}$. *In such a case, OLDA exits without returning a solution, which contradicts the assumption that OLDA returns a solution. Therefore, the solution returned by OLDA satisfies (3.23).* □

Lemma 3. *Given sub-job set* $\Omega(V_x)$, *let* $d_{i,k}^*$ *be the local deadline assigned by OLDA to* $J_{i,k} \in \Omega(V_x)$. *We have*

$$\max_{J_{i,k} \in \omega(V_x)} \{d_{i,k}^*\} - \min_{J_{i,k} \in \omega(V_x)} \{r_{i,k}\} \geq \sum_{\forall J_{i,k} \in \omega(V_x)} C_{i,k}, \forall \omega(V_x) \subseteq \Omega(V_x). \tag{3.25}$$

Proof: *Suppose sub-job* $J_{j,h}$ *has the maximum local deadline* $d_{j,h}$ *among all the sub-jobs in sub-job subset* $\omega^{'}(V_x)$, *which satisfies*

$$d_{j,h} = \max_{J_{i,k} \in \omega^{'}(V_x)} \{d_{i,k}^*\} \tag{3.26}$$

Meanwhile, suppose $J_{j,h}$ *is the base sub-job in the base subset* $\omega^*(V_x)$ *determined by OLDA, which satisfies*

$$d_{j,h} = \min_{J_{i,k} \in \omega^*(V_x)} \{r_{i,k}\} + \sum_{\forall J_{i,k} \in \omega^*(V_x)} C_{i,k} \geq \min_{J_{i,k} \in \omega^{'}(V_x)} \{r_{i,k}\} + \sum_{\forall J_{i,k} \in \omega^{'}(V_x)} C_{i,k}. \tag{3.27}$$

By combining (3.26) and (3.27), we obtain (3.25). □

To show that OLDA always identifies the case where there is no feasible solution to the optimization problem, we observe that OLDA always finds a local-deadline assignment without dropping any job if there exists a feasible solution that satisfies constraints (3.6) and (3.9). This is stated in the following lemma.

Lemma 4. *Given sub-job set $\Omega(V_x)$, if there exists $d_{i,k}$ for every $J_{i,k} \in \Omega(V_x)$ that satisfies (3.6) and (3.9), OLDA always finds a feasible local-deadline assignment for every $J_{i,k} \in \Omega(V_x)$.*

Proof: *We prove Lemma 4 by contradiction. Suppose there is one solution* **d** *satisfying (3.6) and (3.9) but OLDA fails to find a solution. This means that the* `if` *condition on Line 17 in Algorithm 3 is false, and consequently, for sub-job J_{*,k^*}, we have*

$$D_* - C^{cri}_{*,k^*} < d_{*,k^*}. \tag{3.28}$$

In addition,

$$d_{*,k^*} = \min_{J_{i,k} \in \omega^*(V_x)} \{r_{i,k}\} + \sum_{\forall J_{i,k} \in \omega^*(V_x)} C_{i,k}. \tag{3.29}$$

Given that $\Omega(V_x)$ is schedulable, according to Lemma 1, we have

$$\min_{J_{i,k} \in \omega^*(V_x)} \{r_{i,k}\} + \sum_{\forall J_{i,k} \in \omega^*(V_x)} C_{i,k} \leq \max_{J_{i,k} \in \omega^*(V_x)} \{d_{i,k}\}. \tag{3.30}$$

Combining (3.28), (3.29) and (3.30), we have

$$D_* - C^{cri}_{*,k^*} < \max_{J_{i,k} \in \omega^*(V_x)} \{d_{i,k}\}. \tag{3.31}$$

Assume that $J_{p,q}$ has the maximum local deadline among all the sub-jobs in $\omega^*(V_x)$

in the feasible solution $\mathbf{d}$. *Then,*

$$D_* - C^{cri}_{*,k^*} < \max_{J_{i,k} \in \omega^*(V_x)} \{d_{i,k}\} = d_{p,q} \leq D_p - C^{cri}_{p,q}. \tag{3.32}$$

In Algorithm 3, J_{*,k^*} *is selected because it has the maximum upper bound on the local deadline among all the sub-jobs in* $\omega^*(V_x)$ *(Line 16). However,* $D_* - C^{cri}_{*,k^*} < D_p - C^{cri}_{p,q}$, *which is a contradiction.* □

Proving that the local-deadline assignment made by OLDA indeed maximizes the objective function in (3.8) requires analyzing the relationship among the sub-jobs' time slacks. Since OLDA assigns sub-job local deadlines by identifying the base sub-job in each base subset, a special property that the base subset possesses greatly simplifies the analysis process. Lemma 5 below summarizes this property.

Lemma 5. *Let* $\omega^*(V_x)$ *be a base subset of sub-job set* $\Omega(V_x)$ *and* $r^* = \min_{J_{i,k} \in \omega^*(V_x)}\{r_{i,k}\}$. *Under the work-conserving EDF policy, processor* V_x *is never idle once it starts to execute the sub-jobs in* $\omega^*(V_x)$ *at* r^* *and before it completes all the sub-jobs in* $\omega^*(V_x)$. *In addition, the busy interval during which the sub-jobs in* $\omega^*(V_x)$ *are executed is* $[r^*, r^* + \sum_{\forall J_{i,k} \in \omega^*(V_x)} C_{i,k}]$. *Furthermore, there is at least one sub-job unfinished at any time instant within* $[r^*, r^* + \sum_{\forall J_{i,k} \in \omega^*(V_x)} C_{i,k})$.

Proof: *We prove the lemma by contradiction. Since* $\omega^*(V_x)$ *is the base subset of* $\Omega(V_x)$, *according to Definition 1, we have* $\forall \omega(V_x) \subseteq \Omega(V_x)$,

$$r^* + \sum_{\forall J_{i,k} \in \omega^*(V_x)} C_{i,k} \geq \min_{J_{i,k} \in \omega(V_x)} \{r_{i,k}\} + \sum_{\forall J_{i,k} \in \omega(V_x)} C_{i,k}. \tag{3.33}$$

Under EDF scheduling policy, the earliest start time of the first sub-job in $\omega^*(V_x)$ *is simply* $r^* = \min_{J_{i,k} \in \omega^*(V_x)}\{r_{i,k}\}$, *and the earliest completion time of the last sub-job in* $\omega^*(V_x)$ *(denoted by* f^**) satisfies* $f^* = r^* + \sum_{\forall J_{i,k} \in \omega^*(V_x)} C_{i,k}$.

Suppose there are multiple idle time intervals inside $[r^*, f^*]$ *when the sub-jobs in* $\omega^*(V_x)$ *are executed. Let* $T_{idle,sum}$ *be the total duration of the idle times. Thus, the completion time of the last sub-job in* $\omega^*(V_x)$ *can be expressed as*

$$T_{complete,\omega^*(V_x)} = f^* + T_{idle,sum}. \tag{3.34}$$

Let $[t_{start}, t_{end}]$ *be the latest idle time interval among all the idle time intervals. Under the work-conserving EDF policy, an idle interval means that no sub-job is ready to be executed during the interval. In other words,* t_{end} *coincides with the release time* $r_{p,q}$ *of some sub-job* $J_{p,q}$ *in* $\omega^*(V_x)$*. The completion time of* $\omega^*(V_x)$ *can also be expressed as,*

$$T_{complete,\omega^*(V_x)} = r_{p,q} + \sum_{\substack{\forall J_{i,k} \in \omega^*(V_x) \\ r_{i,k} \geq r_{p,q}}} C_{i,k}. \tag{3.35}$$

This implies that,

$$r_{p,q} + \sum_{\substack{\forall J_{i,k} \in \omega^*(V_x) \\ r_{i,k} \geq r_{p,q}}} C_{i,k} > r^* + \sum_{\forall J_{i,k} \in \omega^*(V_x)} C_{i,k}.$$

This contradicts (3.33) stated earlier.

Since there is no idle time when executing the sub-jobs in $\omega^*(V_x)$*, and since the start time of the first sub-job and the total execution time of the sub-jobs in* $\omega^*(V_x)$ *are* r^* *and* $\sum_{\forall J_{i,k} \in \omega^*(V_x)} C_{i,k}$*, respectively, the completion time of the last sub-job in* $\omega^*(V_x)$ *is then equal to* $r^* + \sum_{\forall J_{i,k} \in \omega^*(V_x)} C_{i,k}$*. It follows that the busy interval is* $[r^*, r^* + \sum_{\forall J_{i,k} \in \omega^*(V_x)} C_{i,k}]$*.*

Suppose there exists a time instant that all the sub-jobs released earlier than

this time instant has been finished. Since there is no idle time when executing the sub-jobs in $\omega^*(V_x)$, such an instant can only occur at the completion time of a sub-job. Without loss of generality, we assume that when sub-job $J_{p,q}$ is finished at t, there is no unfinished sub-job released earlier than t. This implies that any sub-job which is released earlier than t has been finished already before t. In addition, at least one sub-job in $\omega^*(V_x)$ is released at time t. Let sub-job subset $\omega^+(V_x)$ contain all the sub-jobs released later than or equal to t in $\omega^*(V_x)$. Then we have,

$$r^* + \sum_{\forall J_{i,k} \in \omega^*(V_x)} C_{i,k} = t + \sum_{\forall J_{i,k} \in \omega^+(V_x)} C_{i,k}. \tag{3.36}$$

According to Definition 1, OLDA should select $\omega^+(V_x)$ instead of $\omega^*(V_x)$ to be the base subset. This contradicts that $\omega^*(V_x)$ is the base subset.

□

Based on Lemma 5, it can be proved that the local-deadline assignment made by OLDA maximizes the objective function (3.8), which is stated in Theorem 5.

Theorem 5. *Given sub-job set $\Omega(V_x)$, let $d^*_{i,k}$ be the local deadline assigned to each $J_{i,k} \in \Omega(V_x)$ by OLDA. Then $d^*_{i,k}$ maximizes the minimum time slack, $\{D_i - d_{i,k} - C^{cri}_{i,k}\}$, among all the sub-jobs executed on V_x, i.e.,*

$$max: \min_{J_{i,k} \in \Omega(V_x)} \left\{ D_i - d_{i,k} - C^{cri}_{i,k} \right\}. \tag{3.37}$$

Proof: *Suppose there exists an optimal set of local deadlines $\mathbf{d}^+$ that is different from the solution $\mathbf{d}^*$ returned by OLDA. Then, there exists at least one sub-job $J_{i,k}$ whose $d^+_{i,k}$ is different from $d^*_{i,k}$. We need to show that such differences in local-deadline assignments do not affect the value of the objective function in (3.37).*

Let the sub-jobs in $\Omega(V_x)$ be arranged in the order by which a sub-job obtains its local absolute deadline in OLDA. Without loss of generality, suppose that sub-job $J_{p,q}$ is the first sub-job in $\Omega(V_x)$ and has different deadlines $d^+_{p,q}$ and $d^*_{p,q}$ in solutions $\mathbf{d}^+$ and $\mathbf{d}^*$, respectively. According to Lemma 5, $d^*_{p,q}$ is the completion time of the base sub-job in base subset $\omega^*(V_x)$ and is equal to $r^* + \sum_{\forall J_{i,k} \in \omega^*(V_x)} C_{i,k}$. Hence, $d^*_{p,q}$ is the longest absolute local deadline of the sub-jobs in $\omega^*(V_x)$ for $\mathbf{d}^*$. Assume that in $\mathbf{d}^+$, the sub-job $J_{p',q'}$ has the longest absolute local deadline among the sub-jobs in $\omega^*(V_x)$. We consider the following scenarios: $J_{p,q} = J_{p',q'}$ and $J_{p,q} \neq J_{p',q'}$.

Case 1 *($J_{p,q} = J_{p',q'}$) : Recall that $d^*_{p,q} = r^* + \sum_{\forall J_{i,k} \in \omega^*(V_x)} C_{i,k}$. According to the definition of $d^+_{p',q'}$, we have*

$$d^+_{p',q'} = \max_{J_{i,k} \in \omega^*(V_x)} \{d^+_{i,k}\} \geq r^* + \sum_{\forall J_{i,k} \in \omega^*(V_x)} C_{i,k} = d^*_{p,q}. \tag{3.38}$$

Hence, we have $s^*_{p,q} \geq s^+_{p',q'}$, and the objective function of the OLDA's solution is larger than or equal to that of $\mathbf{J}^+$.

Case 2 *($J_{p,q} \neq J_{p',q'}$) : In this case, we first prove that $s^+_{p,q} \geq s^+_{p',q'}$ and $s^+_{p,q}$ does not have an impact on the value of the objective function $\min_{J_{i,k} \in \omega^+(V_x)}\{s^+_{i,k}\}$. We then prove that $s^*_{p,q} \geq s^+_{p',q'}$ and $s^*_{p',q'} \geq s^+_{p',q'}$, which leads to the observation that the value of the objective function obtained by OLDA is larger than or equal to that of the assumed optimal solution.*

Since $d^+_{p',q'} \geq d^+_{p,q}$ (see the definition of $d^+_{p',q'}$ above), and $D_p - C^{cri}_{p,q} \geq D_{p'} - C^{cri}_{p',q'}$ according to Line 16 of Algorithm 3, we have $s^+_{p,q} \geq s^+_{p',q'} \geq \min_{J_{i,k} \in \omega^*(V_x)}\{s^+_{i,k}\}$. Therefore, $s^+_{p,q}$ has no impact on the value of the objective function. In addition, since $d^+_{p',q'} \geq d^*_{p,q}$, and $D_p - C^{cri}_{p,q} \geq D_{p'} - C^{cri}_{p',q'}$, we have $s^*_{p,q} \geq s^+_{p',q'}$. Similarly, since $d^*_{p',q'} \leq d^*_{p,q}$ and $d^*_{p,q} \leq d^+_{p',q'}$, we have $s^*_{p',q'} \geq s^+_{p',q'}$. As a result, the value

of the objective function obtained by OLDA is larger than or equal to that of the optimal solution in either case, hence the solution found by OLDA is optimal. □

Based on Theorem 5, we conclude that the solution found by OLDA maximizes the objective function (3.37). Note that the found solution may not be globally optimal for DRTS.

Based on Lemmas 2, 3, 4 and Theorem 5, we have the following theorem.

Theorem 6. *In $O(|\Omega(V_x)|^3)$ time, OLDA returns a set of local deadlines if and only if there exists a solution to the optimization problem specified in (3.8), (3.6) and (3.9). Furthermore, the returned set of local deadlines is a solution that maximizes the objective function (3.8).*

Proof: *By Lemmas 2, 3 and 4, OLDA returns a set of local deadlines if and only if there exists a solution to the optimization problem specified in (3.8), (3.6) and (3.9). Furthermore, by Theorem 5, the returned set of local deadlines is a solution to the optimization problem.*

The time complexity of OLDA is dominated by the main `while` *loop starting at Line 3 of Algorithm 3. Inside the* `while` *loop, the most time consuming operations appear inside the inner* `while` *loop from Lines 8–15. Every time OLDA computes the local absolute deadline of a sub-job, it considers $|\Omega(V_x)|$ number of subsets. Furthermore, the number of sub-jobs in $\Omega(V_x)$ is always reduced by 1 in each iteration. Hence, OLDA considers $\frac{|\Omega(V_x)|\cdot|\Omega(V_x)+1|}{2}$ number of subsets of $\Omega(V_x)$ in total, instead of a combinatorial number of them. In addition, it takes $|\omega(V_x)|$ iterations to compute $\min_{J_{i,k}\in\omega(V_x)}\{r_{i,k}\} + \sum_{\forall J_{i,k}\in\omega(V_x)} C_{i,k}$ for each subset $\omega(V_x)$. Therefore, the time complexity of OLDA is $O(|\Omega(V_x)|^3)$.* □

The importance of Theorem 6 is that the deadline assignment problem can be solved exactly by OLDA in polynomial time even though the original MILP for-

mulation contains $(|\Omega(V_x)| + 2^{|\Omega(V_x)|})$ constraints. Note that processor V_x needs $O(|\Omega(V_x)|^4)$ time to solve $\Omega(V_x)$ using OLDA since V_x may call OLDA for at most $|\Omega(V_x)|$ number of times due to dropping sub-jobs.

CHAPTER 4

More Practical Versions of OLDA

Although the local-deadline assignment problem can be solved efficiently and effectively by the canonical version of our algorithm, it is based on the strong assumption that each processor knows the local release times and the local deadline upper bounds of all the future sub-jobs as in [65]. Fundamentally, the canonical version of our algorithm is an off-line algorithm. To relax this assumption and make our algorithm practical for real-world applications, we propose two derivatives of our algorithm. In the first derivative, each processor only considers the currently active local sub-jobs, which are released (ready to be executed) but have not been finished. In the second derivative, the processor employs a prediction mechanism to estimate the timing information of future sub-jobs in order to further exploit the capability of the ideal algorithm to improve system performance. We prove that the first derivative can find the same solution to that generated by the canonical version of our algorithm if both solve the same set of sub-jobs. Additionally, we discuss other practical considerations such as communication among processors and investigate the time overhead on the performance of the system. Since our algorithm needs to be run upon each release of a sub-job, the time overhead of the proposed algorithm grow relatively quickly as the number of sub-jobs on each processor increases. Thus, our algorithm is suitable for DRTSs

with the number of active sub-jobs in the order of tens. Such DRTSs often appear in avionics and automotive applications.

We have conducted simulation-based studies of our approach and compared it to two existing representative methods, JA [70, 73] and BBW [27]. To evaluate our proposed approach, we used two different types of workloads, the stream-type (ST) workloads and the general-type (GT) workloads to emulate different kinds of application scenarios. The ST workloads are intended to simulate pipelined DRTSs, often found in multimedia systems, performance-sensitive server farms, data processing back-end systems, and shipboard computing clusters [35, 66, 70, 73, 110]. The GT workloads emulate more complex DRTSs, such as signal processing systems, control systems and wireless network systems [98, 129, 159–161].

Our studies reveal that, for ST workloads whose distribution on processors is somewhat balanced, JA and BBW drop 179% and 165% more jobs on average than our approach. For ST workloads whose distribution on processors varies noticeably, the averages are 61% and 313%. For GT workloads, 160% and 51% more jobs, on average, are dropped by JA and BBW than those by our proposed approach. Furthermore, for balanced ST workloads, our approach can find, on average, 71% and 22% more feasible solutions than JA and BBW, respectively. For imbalanced ST workloads, our approach leads to, on average, 60% and 48% more feasible solutions than JA and BBW, respectively. For GT workloads, our approach can solve, on average, 13% and 12% more of task sets than JA and BBW, respectively. These results show that our approach indeed leads to improved quality of service for DRTSs.

4.1 Active Local-Deadline Assignment

In this section, we present a local-deadline assignment algorithm in which each processor considers only the active sub-jobs. We refer to this new algorithm as ALDA. In ALDA, sub-job set $\Omega^a(V_x)$ contains only the active sub-jobs on the processor when ALDA is invoked. Whenever a sub-job is completed, it is removed from $\Omega^a(V_x)$. Every time a new sub-job arrives at the processor, the processor stops its current execution and calls ALDA to determine the deadlines of all the active sub-jobs. The remaining execution times of the active sub-jobs, which is maintained by the processor, are used by ALDA instead of the original execution times. Note that ALDA only returns the solution of $\Omega^a(V_x)$. For a given sub-job set $\Omega(V_x)$ containing sub-jobs with different release times, a sub-job can be assigned local deadlines by ALDA multiple times from its release to its completion. This is because multiple sub-jobs may be released during such a time interval. Hence, the solution of $\Omega(V_x)$ by ALDA is a set of local deadlines, $\{d_{i,k}\}$, where $d_{i,k}$ is the last local deadline assigned to each sub-job $J_{i,k} \in \Omega(V_x)$ by ALDA.

ALDA actually is very similar to OLDA in that both algorithms need to find the base subset and the base sub-job and then assign the local deadline to the base sub-job. However, ALDA only considers all the active sub-jobs on the processor, which possesses a special property. The property can greatly reduce the time complexity of OLDA and is summarized in Lemma 6.

Lemma 6. *Given sub-job set $\Omega^a(V_x)$, if all the sub-jobs are ready for execution, the base subset $\omega^*(V_x)$ is just $\Omega^a(V_x)$.*

Proof: *We prove the lemma by contradiction. Suppose $\Omega^a(V_x)$ is not a base*

subset, and there is one base subset $\omega^*(V_x) \subset \Omega^a(V_x)$*. Then we have*

$$\min_{J_{i,k} \in \omega^*(V_x)} \{r_{i,k}\} + \sum_{\forall J_{i,k} \in \omega^*(V_x)} C_{i,k} > \min_{J_{i,k} \in \Omega^a(V_x)} \{r_{i,k}\} + \sum_{\forall J_{i,k} \in \Omega^a(V_x)} C_{i,k}. \quad (4.1)$$

Since all the sub-jobs are ready at the current time, we have $\min_{J_{i,k} \in \omega^*(V_x)}\{r_{i,k}\} = \min_{J_{i,k} \in \Omega^a(V_x)}\{r_{i,k}\}$*. Thus, we have,* $\sum_{\forall J_{i,k} \in \omega^*(V_x)} C_{i,k} > \sum_{\forall J_{i,k} \in \Omega^a(V_x)} C_{i,k}$*, which contradicts our initial assumption that* $\omega^*(V_x)$ *is a subset of* $\Omega^a(V_x)$*.* □

Based on Lemma 6, it costs ALDA a lower time overhead to identify the base subset than that of OLDA.

The steps of ALDA are summarized in Algorithm 4. We briefly discuss the key steps and omit the ones that are similar to OLDA. The inputs to ALDA are the newly released sub-job $J_{j,h}$ and the sub-job set $\Omega^a(V_x)$ that contains all the active sub-jobs that are already in V_x before the current invocation of ALDA. The sub-jobs in $\Omega^a(V_x)$ are sorted in the non-decreasing order of the upper bound on the local deadline of each sub-job in $\Omega^a(V_x)$. Ties are broken in favour of the sub-job with the largest job identifier and then in favour of the sub-job with the largest subtask identifier. Since the sub-job set $\Omega^a(V_x)$ is the base subset according to Lemma 6, the sorting of sub-jobs in $\Omega^a(V_x)$ makes the tail sub-job in $\Omega^a(V_x)$ the base sub-job according to Definition 2. In addition, ALDA directly calculates the desired local deadline value, $Max\_Deadline$, for $\Omega^a(V_x)$ according to Lemma 5 (Lines 4–7).

The algorithm then enters the main loop spanning from Line 8 to Line 27. ALDA finds the base sub-job J_{*,k^*}, which is the last sub-job of the sub-job set in $\Omega^a(V_x)$ according to Lemma 6 (Line 9). If the desired local deadline value is smaller than or equal to UB_{*,k^*}, ALDA updates $Max\_Deadline$ for the next

Algorithm 4 ALDA($\Omega^a(V_x)$, $J_{j,h}$)

1: $\Omega^a(V_x) = Insert\_by\_Non\_Dec\_Local\_Deadline\_UB(\Omega^a(V_x), J_{j,h})$ //Insert $J_{j,h}$ into the sub-job set in the non-decreasing order of the upper bound on the local deadline of each sub-job in $\Omega^a(V_x)$
2: $\mathbf{d} = \emptyset$
3: $\Omega^{'}(V_x) = \emptyset$
4: $Max\_Deadline = 0$
5: **for** $(J_{i,k} \in \Omega^a(V_x))$ **do**
6: $\quad Max\_Deadline = Max\_Deadline + C_{i,k}$
7: **end for**
8: **while** $(\Omega^a(V_x) \neq \emptyset)$ **do**
9: $\quad J_{*,k^*} = Tail(\Omega^a(V_x))$ //Select the base sub-job which is the last sub-job of the sub-job set in $\Omega^a(V_x)$
10: $\quad$ **if** $(UB_{*,k^*} \geq Max\_Deadline)$ **then**
11: $\quad\quad d_{*,k^*} = Max\_Deadline$
12: $\quad\quad \mathbf{d} = \mathbf{d} \bigcup \{d_{*,k^*}\}$
13: $\quad\quad \Omega^a(V_x) = \Omega^a(V_x) - J_{*,k^*}$
14: $\quad\quad \Omega^{'}(V_x) = \Omega^{'}(V_x) \bigcup \{J_{*,k^*}\}$
15: $\quad\quad Max\_Deadline = Max\_Deadline - C_{*,k^*}$
16: $\quad$ **else**
17: $\quad\quad \mathbf{J^{drop}} = Drop\_Sub\_Jobs(\Omega^a(V_x))$ // Remove a subset of sub-jobs from $\Omega^a(V_x)$ according to some sub-job dropping policy, and return the subset containing the dropped sub-jobs
18: $\quad\quad Temp\_C = 0$
19: $\quad\quad$ **for** $(J_{i,k} \in \mathbf{J^{drop}})$ **do**
20: $\quad\quad\quad Temp\_C = Temp\_C + C_{i,k}$
21: $\quad\quad$ **end for**
22: $\quad\quad Max\_Deadline = Max\_Deadline - Temp\_C$
23: $\quad\quad$ **for** $(J_{i,k} \in \Omega^{'}(V_x))$ **do**
24: $\quad\quad\quad d_{i,k} = d_{i,k} - Temp\_C$
25: $\quad\quad$ **end for**
26: $\quad$ **end if**
27: **end while**
28: return $\mathbf{d}$ //$\mathbf{d} = \{d_{i,k}\}$

iteration (Line 15). If the desired local deadline value is larger than UB_{*,k^*} of the base sub-job J_{*,k^*}, the total execution time of the removed sub-jobs, $Temp\_C$, is calculated (Lines 18–21). Since the sub-jobs in $\mathbf{J^{drop}}$ are removed from the base subset $\Omega^a(V_x)$, $Temp\_C$ is reduced from $Max\_Deadline$ (Line 22). According to the local-deadline assignment in ALDA (Lines 4–15), a sub-job which is assigned

its local deadline earlier will have a longer local deadline than a sub-job being assigned its local deadline later. This implies that a sub-job that has been moved to $\Omega'(V_x)$ will be completed after the sub-jobs currently still in $\Omega^a(V_x)$. Thus, after removing the sub-jobs in $\mathbf{J^{drop}}$ from $\Omega^a(V_x)$, each sub-job in $\Omega'(V_x)$ can be completed earlier by $Temp\_C$ and each previously assigned local deadline is reduced by $Temp\_C$ (Lines 23–25). The above process is repeated until each sub-job in $\Omega^a(V_x)$ either receives a deadline or is dropped. ALDA eventually returns the set of sub-job deadlines to be used by the processor in performing EDF based scheduling.

Since ALDA simplifies OLDA by only considering the active sub-jobs on a local processor, all the lemmas and theorems in Section 3.6.3 still hold for ALDA except for the time complexity of ALDA. The time complexity of ALDA is dominated by the main `while` loop starting at Line 8. (Refer to Algorithm 4.) Every time a subset of sub-jobs are to be removed from $\Omega^a(V_x)$ (Line 17), OLDA needs to traverse $|\Omega^a(V_x)|$ number of sub-jobs in Function $Drop\_Sub\_Jobs()$. Hence, the time complexity of ALDA when handling $|\Omega^a(V_x)|$ number of sub-jobs on processor V_x is $O(|\Omega^a(V_x)|^2)$, where $|\Omega^a(V_x)|$ is the number of active sub-jobs on processor V_x. Processor V_x calls ALDA every time a new sub-job from $\Omega(V_x)$ is released at V_x. Since $|\Omega^a(V_x)| \leq |\Omega(V_x)|$, processor V_x takes $O(|\Omega(V_x)|^3)$ time to solve a set $\Omega(V_x)$ using ALDA. Compared with OLDA, ALDA is much more efficient in solving the local-deadline assignment problem.

We show next that ALDA is equivalent to OLDA. First, we have the following lemma to show that ALDA can solve the sub-job set if and only if the sub-job set is schedulable.

Lemma 7. *Let $\Omega(V_x)$ contain all the sub-jobs to be scheduled by OLDA. If and*

only if there exists a schedulable solution for $\Omega(V_x)$, ALDA can find a feasible solution for $\Omega(V_x)$.

Proof: *We first prove the "only if" part. After processor V_x assigns local deadlines to released sub-jobs by running ALDA, the processor keeps executing the released sub-jobs until the release of a new sub-job. A sub-job can be assigned local deadlines for multiple times from its release to its completion since multiple sub-jobs may be released during the time interval. However, the local deadline $d_{i,k}$ assigned to $J_{i,k}$ for the last time is the finish time of $J_{i,k}$. Suppose at time t, processor assigns $d_{p,q}$ to $J_{p,q}$ by using ALDA and let $J_{p,q}$ be completed without any more local-deadline assignment. According to Lines 4-15 of Algorithm 4, we have*

$$d_{p,q} \geq C'_{p,q} + t, \tag{4.2}$$

where $C'_{p,q}$ is the remaining execution times of $J_{p,q}$. The other execution time $C_{p,q} - C'_{p,q}$ of sub-job $J_{p,q}$ has been finished within time interval $[r_{p,q}, t]$. Thus, we have

$$t - r_{p,q} \geq C_{p,q} - C'_{p,q}. \tag{4.3}$$

By combining (4.2) and (4.3), we have $r_{p,q} + C_{p,q} \leq d_{p,q}$. In addition, the local deadline $d_{p,q}$ assigned to sub-job $J_{p,q}$ at time t satisfies $d_{p,q} \leq UB_{p,q}$. Otherwise, ALDA will drop some sub-jobs in Line 17 of Algorithm 4. Hence, constraint (3.6) is satisfied.

Given a set of local deadlines determined by ALDA, we assume that subset $\omega(V_x)$ cannot satisfy condition (3.9) and $d_{p,q}$ has the maximum local deadline

among all the sub-jobs in $\omega(V_x)$*, i.e.,*

$$d_{p,q} = \max_{J_{i,k} \in \omega(V_x)} \{d_{i,k}\} < \min_{J_{i,k} \in \omega(V_x)} \{r_{i,k}\} + \sum_{\forall J_{i,k} \in \omega(V_x)} C_{i,k}. \tag{4.4}$$

Since $d_{i,k}$ *determined by ALDA for the last time is the completion time of sub-job* $J_{i,k}$*,* $J_{p,q}$ *should be finished after all the other sub-jobs in* $\omega(V_x)$ *have been completed, i.e.,*

$$\min_{J_{i,k} \in \omega(V_x)} \{r_{i,k}\} + \sum_{\forall J_{i,k} \in \omega(V_x)} C_{i,k} \leq d_{p,q}. \tag{4.5}$$

By combining (4.4) and (4.5), we have $d_{p,q} < d_{p,q}$*, which is a contradiction. Therefore, if ALDA finds a solution, this solution must be a schedulable solution.*

We next prove the "if" part by contradiction. Suppose ALDA cannot find a feasible solution. Since ALDA employs EDF scheduling algorithm and uses $UB_{i,k}$ *of each sub-job* $J_{i,k}$ *in* $\omega(V_x)$ *as* $J_{i,k}$*'s deadline, we can find a subset* $\omega(V_x)$ *which satisfies*

$$\max_{J_{i,k} \in \omega(V_x)} \{UB_{i,k}\} - \min_{J_{i,k} \in \omega(V_x)} \{r_{i,k}\} < \sum_{\forall J_{i,k} \in \omega(V_x)} C_{i,k}, \tag{4.6}$$

according to Lemma 1. Meanwhile, there exists a feasible solution for $\Omega(V_x)$*. According to Lemma 1, we have*

$$\max_{J_{i,k} \in \omega(V_x)} \{d_{i,k}\} - \min_{J_{i,k} \in \omega(V_x)} \{r_{i,k}\} \geq \sum_{\forall J_{i,k} \in \omega(V_x)} C_{i,k}. \tag{4.7}$$

Combining constraints (4.6) and (4.7), we have $\max_{J_{i,k} \in \omega(V_x)}\{UB_{i,k}\} < \max_{J_{i,k} \in \omega(V_x)}\{d_{i,k}\}$*, which is a contradiction.* □

Second, we have the following lemma to show that the equivalence of the solutions found by ALDA and OLDA.

Lemma 8. *The solutions found by ALDA and OLDA are the same if $\Omega(V_x)$ is schedulable.*

Proof: *We prove the lemma by contradiction. Suppose the solution* $\mathbf{d}$ *by ALDA is different from the solution* $\mathbf{d}^*$ *by OLDA. Then, there exists at least one sub-job $J_{i,k}$ whose $d_{i,k}$ is different from $d^*_{i,k}$. Let the sub-jobs in $\Omega(V_x)$ be arranged in the order by which a sub-job obtains its local absolute deadline in OLDA. Without loss of generality, suppose that sub-job $J_{p,q}$ is the first sub-job in $\Omega(V_x)$ which has different deadlines $d_{p,q}$ and $d^*_{p,q}$ in solutions* $\mathbf{d}$ *and* $\mathbf{d}^*$*, respectively. Hence, $d^*_{p,q}$ is the longest absolute local deadline of the sub-jobs in $\omega^*(V_x)$ for* $\mathbf{d}^*$*. We consider two cases: $d_{p,q} = \max_{J_{i,k} \in \omega^*(V_x)}\{d_{i,k}\}$ and $d_{p,q} \neq \max_{J_{i,k} \in \omega^*(V_x)}\{d_{i,k}\}$.*

Case 1 *$d_{p,q}$ is equal to $d^*_{p,q}$ according to Lemma 5.*

Case 2 *Since $J_{p,q}$ is the base sub-job in $\omega^*(v_x)$, ALDA always put $J_{p,q}$ to the tail of the sub-job list in $\Omega^a(V_x)$ when there are only the sub-jobs in $\omega^*(v_x)$ on V_x. Hence, any sub-job in $\omega^*(V_x)$ released earlier than $J_{p,q}$ should have been finished when $J_{p,q}$ is finished. However, this contradicts the property that there is always at least one sub-job unfinished within $[r^*, r^* + \sum_{\forall J_{i,k} \in \omega^*(V_x)} C_{i,k})$, which is proved in Lemma 5.* □

Since ALDA and OLDA are equivalent for schedulable sub-job sets and we have proved the optimality of OLDA, ALDA is also an optimal algorithm to solve the proposed problem.

Based on Theorem 6, Lemma 7 and Lemma 8, we have the following theorem.

Theorem 7. *In $O(|\Omega(V_x)|^2)$ time, ALDA returns a set of local deadlines if and only if there exists a solution to the optimization problem specified in (3.8), (3.6) and (3.9). Furthermore, the returned set of local deadlines is a solution that*

maximizes the objective function (3.8).

Proof: *By Lemma 7, ALDA returns a set of local deadlines if and only if there exists a solution to the optimization problem specified in (3.8), (3.6) and (3.9). Furthermore, by Lemma 8, the returned solution by ALDA is the same to that by OLDA if* $\Omega(V_x)$ *is schedulable. Since the solution found by OLDA maximizes the objective function (3.8) according to Theorem 6, the returned set of local deadlines maximizes the objective function (3.8).*

When ALDA does not drop any sub-job, the most time consuming operations of ALDA appear in the calculation of desired local deadline in Lines 4-7 of Algorithm 4, which considers $|\Omega^a(V_x)|$ *number of sub-jobs. In addition, every time a sub-job from* $\Omega(V_x)$ *is released, ALDA will be called to assign local deadlines to active sub-jobs in* $\Omega^a(V_x)$*. Since the number of active sub-jobs on* V_x *is smaller than or equal to* $|\Omega(V_x)|$*, ALDA needs* $O(|\Omega(V_x)|^2)$ *time to find a solution for a schedulabe sub-job set* $\Omega(V_x)$*.* □

Theorem 7 shows that ALDA is able to solve the local-deadline assignment problem exactly in polynomial time, which demonstrates the same performance to that of OLDA.

4.2 WLDA

Although ALDA is extremely efficient in assigning local deadlines to the active sub-jobs, it does not consider any future sub-job when judging the schedulability of a sub-job subset. That is, the sub-job set which is deemed schedulable by ALDA at the current time may not actually be schedulable after additional sub-jobs arrive at the processor. If a processor considers the workload of some future sub-jobs when assigning local deadlines, an infeasible sub-job subset can be detected ahead of

time, and a subset of sub-jobs can be dropped as early as possible without wasting valuable resources. To achieve this, we propose another derivative of OLDA, WLDA, which leverages workload prediction during local-deadline assignment to improve resource utilization and avoid dropping jobs. In WLDA, a processor can estimate the release times and local deadline upper bounds of future sub-jobs and then apply OLDA directly to all the active sub-jobs and predicted future sub-jobs on the processor.

Similar to OLDA and ALDA, WLDA is invoked to perform local-deadline assignment every time a new sub-job arrives at the processor. However, WLDA considers both the active sub-jobs and the predicted future sub-jobs whose release times and upper bounds on the local deadlines are within some given window. In WLDA, the set of active sub-jobs and future sub-jobs predicted by V_x is denoted by $\Omega^w(V_x)$. We set the time window as $W = [t, t + \alpha \cdot (\max_{J_{i,k} \in \Omega^w(V_x)}\{D_i - C_{i,k}^{cri}\} - t)], \alpha > 0$, where t is the current time and α is a scaling factor whose value can be selected by the user. When α is 0, WLDA is equivalent to ALDA, i.e., it only considers the active sub-jobs. When α is 1, WLDA considers all the future sub-jobs whose upper bounds on their local deadlines are smaller than or equal to the largest local deadline upper bound among all the sub-jobs in $\Omega^w(V_x)$. With increasing α value, WLDA considers more future sub-jobs, which leads to more precise schedulability prediction at the cost of higher time overhead. The actual deadline assignment part of WLDA is done by the original OLDA. The main task of WLDA is to predict the release times as well as local deadline upper bounds of future sub-jobs. A straightforward way of estimating these values is to simply use the corresponding task periods if tasks are periodic. However, such estimation can have large errors due to the variations in sub-job completion times on different

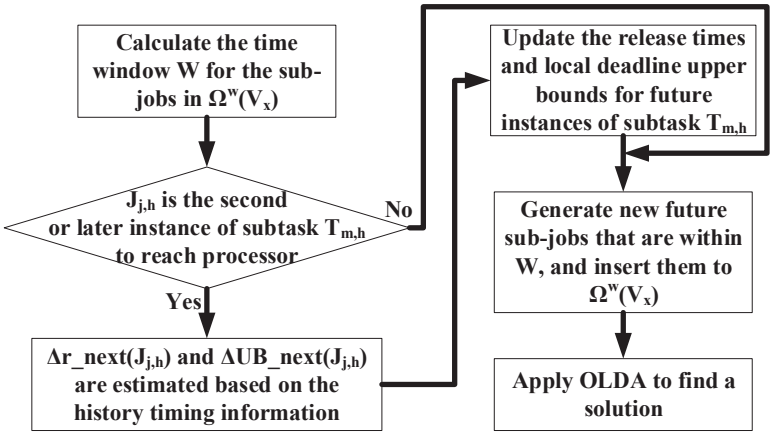


Figure 4.1. WLDA flow to determine future release times and upper bounds on the local deadlines of future sub-jobs and assign local deadlines to the newly released sub-job as well as the active and future sub-jobs in $\Omega^w(V_x)$.

processors. Furthermore, when tasks are not periodic, WLDA needs to employ other estimation methods.

WLDA utilizes recent history data to predict future timing data, as is shown in 4.1. Specifically, it estimates the release interval between subsequent future sub-jobs and the difference between the local deadline upper bounds of subsequent future sub-jobs. Once these values are available, release times and upper bounds on the local deadlines for future sub-jobs can be readily predicted.

Prediction of sub-job release times and local deadline upper bounds is performed for sub-jobs of the same subtask. We use subtask $T_{m,h}$ as an example for our following discussion. Suppose sub-jobs $J_{j,h}$ and $J_{j',h}$, are instances of subtask $T_{m,h}$ on processor V_x. Let $Ind(j)$ be a function which returns $J_{j,h}$'s instance index of subtask $T_{m,h}$. Then we have the average release interval between the sub-jobs, denoted as $\overline{\Delta r(J_{j,h}, J_{j',h})}$, to be the difference between the release times of $J_{j,h}$

and $J_{j',h}$ relative to the instance index difference

$$\overline{\Delta r(J_{j,h}, J_{j',h})} = \frac{r(J_{j',h}) - r(J_{j,h})}{Ind(j') - Ind(j)}, \qquad Ind(j') > Ind(j). \tag{4.8}$$

If $Ind(j') > Ind(j)$, we have $r_{j,h} < r_{j',h}$. This observation is summarized in Lemma 9.

Lemma 9. *For sub-jobs $J_{j',h}$ and $J_{j,h}$, if the instance index $Ind(j')$ is larger than $Ind(j)$, the release time $r_{j,h}$ is smaller than the release time $r_{j',h}$.*

Proof: *We prove this lemma by induction. Consider jobs J_j and $J_{j'}$ that satisfy $ind(j') > ind(j)$. We first prove the base case. Suppose sub-jobs $J_{j,h}$ and $J_{j',h}$ are input sub-jobs of J_j and $J_{j'}$, respectively. Since $ind(j') > ind(j)$, we have $r_{j,h} < r_{j',h}$ by the definition of the absolute release time of a sub-job. The base case holds.*

Consider sub-jobs $J_{j,h'}$ and $J_{j',h'}$, which are not input sub-jobs and have immediate successors $J_{j,h''}$ and $J_{j',h''}$, respectively. Assume that we have $r_{j,h'} < r_{j',h'}$, we then need to prove $r_{j,h''} < r_{j',h''}$. According to the dependencies of the subtasks, we have $r_{j,h''} = \max_{J_{j,h'} \preccurlyeq J_{j,h''}}\{d_{j,h'}\}$ and $r_{j',h''} = \max_{J_{j',h'} \preccurlyeq J_{j',h''}}\{d_{j',h'}\}$. In order to prove $r_{j,h''} < r_{j',h''}$, we need to prove $d_{j,h'} < d_{j',h'}$ for sub-jobs $J_{j,h'}$ and $J_{j',h'}$.

We prove $d_{j,h'} < d_{j',h'}$ by contradiction. According to the first paragraph in the proof of Lemma 5, $d_{j,h'}$ and $d_{j',h'}$ are equal to the completion times of $J_{j,h'}$ and $J_{j',h'}$, respectively. Since $J_{j,h'}$ and $J_{j',h'}$ cannot be completed on V_x at the same time, $d_{j,h'} \neq d_{j',h'}$. Suppose we have $d_{j,h'} > d_{j',h'}$. Without loss of generality, $J_{j,h'}$ and $J_{j',h'}$ are the base sub-jobs of base subsets $\omega'(V_x)$ and $\omega''(V_x)$, respectively.

Therefore, $d_{j,h'}$ and $d_{j',h'}$ satisfy

$$d_{j,h'} = \min_{J_{i,k} \in \omega'(V_x)} \{r_{i,k}\} + \sum_{\forall J_{i,k} \in \omega'(V_x)} C_{i,k} > \min_{J_{i,k} \in \omega''(V_x)} \{r_{i,k}\} + \sum_{\forall J_{i,k} \in \omega''(V_x)} C_{i,k} = d_{j',h'}. \tag{4.9}$$

Hence, according to Lines 4–15 of Algorithm 3, OLDA should select $\omega'(V_x)$ as the base subset before selecting $\omega''(V_x)$. Thus, $J_{j',h'}$ is still in $\Omega^w(V_x)$ when OLDA selects $\omega'(V_x)$ as the base subset. Since $r_{j,h'} < r_{j',h'}$, according to Definition 2, $J_{j',h'}$ is in $\omega'(V_x)$ and maximizes $\min_{J_{i,k} \in \omega'(V_x)}\{r_{i,k}\} + \sum_{\forall J_{i,k} \in \omega'(V_x)} C_{i,k}$. In addition, since $r_{j,h'} < r_{j',h'}$ and $C^{cri}_{j,h'} = C^{cri}_{j',h'}$, we have $D_j < D_{j'}$ and $UB_{j,h'} < UB_{j',h'}$. According to Definition 2, OLDA should not select $J_{j,h'}$ as the base sub-job for the base subset $\omega'(V_x)$. This contradicts the assumption that $J_{j,h'}$ is the base sub-job of $\omega'(V_x)$. Thus, we proved $d_{j,h'} < d_{j',h'}$.

Since $d_{j,h'} < d_{j',h'}$, $r_{j,h''} = \max_{J_{j,k'} \preccurlyeq J_{j,k''}}\{d_{j,k'}\}$ and $r_{j',h''} = \max_{J_{j',k'} \preccurlyeq J_{j',k''}}\{d_{j',k'}\}$ hold, we have $r_{j,h''} < r_{j',h''}$ and the lemma holds. □

If $Ind(j') = Ind(j) + 1$, the average release interval $r(J_{j,h}, J_{j',h})$ is called the next release interval $\Delta r\_next(J_{j,h})$ of $J_{j,h}$, i.e.,

$$\Delta r\_next(J_{j,h}) = \overline{\Delta r(J_{j,h}, J_{j',h})}, \qquad \forall Ind(j') = Ind(j) + 1. \tag{4.10}$$

We further define $Ins(j,h)$ to be the number of $T_{m,h}$'s instances that have arrived at processor V_x before or at time t (arrival time of $J_{j,h}$). Assume that sub-job $J_{j^-,h}$ is the latest arriving sub-job of subtask $T_{m,h}$ before time t, i.e. $Ins(j^-,h) = Ins(j,h) - 1$. By using the timing information of $J_{j^-,h}$ and $J_{j,h}$, processor V_x

estimates $\Delta r\_next(J_{j,h})$ of $J_{j,h}$ as,

$$\Delta\tilde{r}\_next(J_{j,h}) = \begin{cases} \frac{\Delta\tilde{r}\_next(J_{j^-,h}) + \overline{\Delta r(J_{j,h}, J_{j^-,h})}}{2} & : Ins(j,h) \geq 3 \\ \overline{\Delta r(J_{j,h}, J_{j^-,h})} & : Ins(j,h) = 2 \end{cases}, \tag{4.11}$$

where $\Delta\tilde{r}\_next(J_{j,h})$ and $\Delta\tilde{r}\_next(J_{j^-,h})$ are the estimated next release intervals of $J_{j,h}$ and $J_{j^-,h}$, respectively, and $\overline{\Delta r(J_{j,h}, J_{j^-,h})}$ is the average release interval between $J_{j,h}$ and $J_{j^-,h}$. The processor starts to estimate $\Delta\tilde{r}\_next(J_{j,h})$ after two instances of subtask $T_{m,h}$ have reached processor V_x, i.e., $Ins(j,h) \geq 2$. When $J_{j,h}$ is the second instance to reach the processor, $\Delta\tilde{r}\_next(J_{j,h})$ is estimated to be the release interval $\overline{\Delta r(J_{j,h}, J_{j^-,h})}$ between sub-jobs $J_{j,h}$ and $J_{j^-,h}$. Otherwise, $\Delta\tilde{r}\_next(J_{j,h})$ is estimated to be the average of $\Delta\tilde{r}\_next(J_{j^-,h})$ and $\overline{\Delta r(J_{j,h}, J_{j^-,h})}$.

Similarly, we define the average difference in local deadline upper bounds $\overline{\Delta UB(J_{j,h}, J_{j',h})}$ to be the difference between local deadline upper bounds of sub-jobs $J_{j',h}$ and $J_{j,h}$ relative to their instance index difference, i.e.,

$$\overline{\Delta UB(J_{j,h}, J_{j',h})} = \frac{UB_{j',h} - UB_{j,h}}{Ind(j') - Ind(j)}, \qquad Ind(j') > Ind(j). \tag{4.12}$$

If $Ind(j') = Ind(j) + 1$, $\overline{\Delta UB(J_{j,h}, J_{j',h})}$ is the next local deadline upper bound difference $\Delta UB\_next(J_{j,h})$, i.e.,

$$\Delta UB\_next(J_{j,h}) = \overline{\Delta UB(J_{j,h}, J_{j',h})}, \forall Ind(j') = Ind(j) + 1. \tag{4.13}$$

Processor V_x estimates $\Delta UB\_next(J_{j,h})$ of $J_{j,h}$ as,

$$\Delta\tilde{UB}\_next(J_{j,h}) = \begin{cases} \frac{\Delta\tilde{UB}\_next(J_{j^-,h}) + \overline{\Delta UB(J_{j,h}, J_{j^-,h})}}{2} & : Ins(j,h) \geq 3 \\ \overline{\Delta UB(J_{j,h}, J_{j^-,h})} & : Ins(j,h) = 2 \end{cases}, \tag{4.14}$$

where $\Delta\tilde{UB}\_next(J_{j,h})$ and $\Delta\tilde{UB}\_next(J_{j^-,h})$ are the estimated next local deadline upper bound differences of $J_{j,h}$ and $J_{j^-,h}$, respectively, and $\overline{\Delta UB(J_{j,h}, J_{j^-,h})}$ is the average difference between the local deadline upper bounds of $J_{j,h}$ and $J_{j^-,h}$.

The details of WLDA are summarized in Algorithm 5. The input variables of WLDA are user-defined parameters, α and $Max\_Allowed\_Drop\_Num$, the sub-job sets $\Omega^w(V_x)$ and $\Omega^-(V_x)$, the newly released sub-job $J_{j,h}$, and the sub-task set $\Psi(V_x)$. $\Omega^w(V_x)$ contains all the active sub-jobs that have been released but not finished before $r_{j,h}$ and all the future sub-jobs that have been considered in the previous invocation of WLDA. $\Omega^-(V_x)$ contains the latest arriving sub-job of each sub-task on V_x prior to the arrival of $J_{j,h}$. When $J_{j,h}$ arrives at processor V_x, WLDA first determines if $J_{j,h}$ should be dropped in Function $Determine\_Drop\_Sub\_Job()$ (Line 1) based on the drop counter value corresponding to $J_{j,h}$. The drop counter of the future sub-job corresponding to $J_{j,h}$ indicates how many times WLDA has intended to drop $J_{j,h}$ before the release of $J_{j,h}$ in the previous calls of WLDA. That is, every time when WLDA calls OLDA in Line 32 of Algorithm 5 and decides to drop future sub-job $J_{j,h}$ in Function $Drop\_Sub\_Jobs()$ of OLDA, the corresponding drop counter of $J_{j,h}$ is incremented. If this drop counter value is accumulated to be higher than $Max\_Allowed\_Drop\_Num$, $J_{j,h}$ is dropped directly as soon as it is released in Function $Delete\_Sub\_Job()$ (Line 3) of Algorithm 5. Otherwise, WLDA computes $UB_{j,h}$, updates $\Omega^w(V_x)$, and determines the time window, W, as given in Lines 5–8. It then removes the obsolete future sub-

jobs whose predecessors have been predicted to be dropped on other processors (Line 9). If $Ins(j,h) \geq 2$, WLDA updates the release times and local deadline upper bounds for future instances of $T_{m,h}$ that are already in set $\Omega^w(V_x)$ in function $Update\_Release\_Times\_Local\_Deadline\_UB()$ (Lines 10–12), which is introduced in more details in Algorithm 6. Then, WLDA add $J_{j,h}$ to $\Omega^-(V_x)$ since it is the latest arriving sub-job of $T_{m,h}$.

After updating the release times and local deadline upper bounds for the future instances of $T_{m,h}$ already in $\Omega^w(V_x)$, WLDA generates new future sub-jobs $J_{i',k}$'s whose $r_{i',k}$'s and $UB_{i',k}$'s are predicted to be within W (Lines 14–28). By considering these newly predicted sub-jobs in the schedulability analysis of $\Omega^w(V_x)$, WLDA can more accurately predict the load of V_x and give a more precise schedulability prediction of $\Omega^w(V_x)$. To predict release times and local deadline upper bounds for new future instances $J_{i',k}$'s of $T_{n,k}$, WLDA need to utilize the timing information of the latest arriving sub-job $J_{i^-,k}$ and the latest predicted sub-job $J_{i,k}$ by the current time t. Specifically, WLDA first obtains the latest predicted future sub-job $J_{i,k}$ and latest arriving sub-job $J_{i^-,k}$ of each $T_{n,k}$ in order to achieve $r_{i,k}$, $UB_{i,k}$, $\Delta\tilde{r}\_next(J_{i^-,k})$ and $\Delta\tilde{UB}\_next(J_{i^-,k})$ (Lines 15–16). If WLDA has never constructed a future sub-job of a specific subtask $T_{n,k}$, WLDA replaces $J_{i,k}$ with $J_{i^-,k}$, whose $r_{i^-,k}$ and $UB_{i^-,k}$ substitute for $r_{i,k}$ and $UB_{i,k}$ in predicting new future sub-jobs (Lines 17–18). Next, WLDA iteratively adds $\Delta\tilde{r}\_next(J_{i^-,k})$ and $\Delta\tilde{UB}\_next(J_{i^-,k})$ to $r_{i,k}$ and $UB_{i,k}$ to predict release times and local deadline upper bounds of future sub-jobs $J_{i,k}$'s, respectively (Lines 20–22). As long as the newly predicted release time and local deadline upper bound are within W, WLDA constructs a new sub-job $J_{i',k}$ and adds it to $\Omega^w(V_x)$ (Lines 23–27). Finally, OLDA is invoked for the new sub-job set $\Omega^w(V_x)$ and returns the local-deadline assignment

Algorithm 5 WLDA(α, $Max\_Allowed\_Drop\_Num$, $\Omega^w(V_x)$, $\Omega^-(V_x)$, $J_{j,h}$, $\Psi(V_x)$)

1: $Flag = Determine\_Drop\_Sub\_Job(\Omega^w(V_x), J_{j,h}, Max\_Allowed\_Drop\_Num)$
2: **if** $(Flag = 1)$ **then**
3: $\quad Delete\_Sub\_Job(J_{j,h})$
4: **else**
5: $\quad UB_{j,h} = Comp\_Local\_Deadline\_UB(J_{j,h})$
6: $\quad \Omega^w(V_x) = \Omega^w(V_x) \bigcup \{J_{j,h}\}$
7: **end if**
8: $W = Calc\_Schedule\_Window(\Omega^w(V_x), \alpha)$
9: $\Omega^w(V_x) = Remove\_Obsolete\_Future\_Sub\_Jobs(\Omega^w(V_x))$
10: **if** $(Ins(j,h) \geq 2)$ **then**
11: $\quad Update\_Release\_Times\_Local\_Deadline\_UB(\Omega^w(V_x), \Omega^-(V_x), T_{m,h}, J_{j,h})$
12: **end if**
13: $\Omega^-(V_x) = \Omega^-(V_x) \bigcup \{J_{j,h}\}$
14: **for** $(T_{n,k} \in \Psi(V_x))$ **do**
15: $\quad J_{i,k} = Find\_Latest\_Future\_Sub\_Job(\Omega^w(V_x), T_{n,k})$
16: $\quad J_{i^-,k} = Find\_Latest\_Sub\_Job(\Omega^-(V_x), T_{n,k})$
17: $\quad$ **if** $(J_{i,k} = NULL)$ **then**
18: $\quad\quad J_{i,k} = J_{i^-,k}$
19: $\quad$ **end if**
20: $\quad$ **while** $(Ins(i^-, k) \geq 2)$ **do**
21: $\quad\quad Next\_RT = r_{i,k} + \Delta\tilde{r}\_next(J_{i^-,k})$
22: $\quad\quad Next\_UB = UB_{i,k} + \Delta\tilde{UB}\_next(J_{i^-,k})$
23: $\quad\quad$ **if** $([Next\_RT, Next\_UB] \nsubseteq W)$ **then**
24: $\quad\quad\quad break$
25: $\quad\quad$ **end if**
26: $\quad\quad J_{i',k} = Construct\_a\_Future\_Sub\_Job(Next\_RT,\ Next\_UB, C_{i,k})$
27: $\quad\quad \Omega^w(V_x) = \Omega^w(V_x) \bigcup \{J_{i',k}\}$
28: $\quad$ **end while**
29: **end for**
30: $\mathbf{d} = \emptyset$
31: **while** $(\mathbf{d} = \emptyset)$ **do**
32: $\quad \mathbf{d} = OLDA(\Omega^w(V_x), Max\_Allowed\_Drop\_Num)$
33: **end while**
34: return **d**

for all the existing and future sub-jobs (Lines 30–34).

Algorithm 6 presents function $Update\_Release\_Times\_Local\_Deadline\_UB()$, which updates the release times and local deadline upper bounds for future in-

Algorithm 6 Update_Release_Times_Local_Deadline_UB($\Omega^w(V_x)$, $\Omega^-(V_x)$, $T_{m,h}$, $J_{j,h}$)

1: $J_{j^-,h} = Find\_Latest\_Sub\_Job(\Omega^-(V_x), T_{m,h})$
2: $\Delta\tilde{r}\_next(J_{j,h}) = Estimate\_Next\_Release\_Interval(J_{j^-,h}, J_{j,h})$
3: $\Delta\tilde{UB}\_next(J_{j,h}) = Estimate\_Next\_UB\_Difference(J_{j^-,h}, J_{j,h})$
4: $Next\_RT = r_{j^-,h} + \Delta\tilde{r}\_next(J_{j,h})$
5: $Next\_UB = UB_{j^-,h} + \Delta\tilde{UB}\_next(J_{j,h})$
6: **for** ($J_{i,k} \in \Omega^w(V_x)$) **do**
7: **if** ($J_{i,k}$ is a future instance of $T_{m,h}$) **then**
8: $r_{i,k} = Next\_RT$
9: $UB_{i,k} = Next\_UB$
10: $Next\_RT = Next\_RT + \Delta\tilde{r}\_next\ (J_{j,h})$
11: $Next\_UB = Next\_UB + \Delta\tilde{UB}\_next\ (J_{j,h})$
12: **if** ($[r_{i,k}, UB_{i,k}] \not\subseteq W$) **then**
13: $\Omega^w(V_x) = \Omega^w(V_x) - J_{i,k}$
14: **end if**
15: **end if**
16: **end for**
17: $\Omega^-(V_x) = \Omega^-(V_x) - \{J_{j^-,h}\}$

stances of $T_{m,h}$ that are already in set $\Omega^w(V_x)$. Note that $J_{j,h}$ is an instance of $T_{m,h}$. WLDA obtains the latest arriving sub-job $J_{j^-,h}$ of $T_{m,h}$ prior to the arrival of $J_{j,h}$ from $\Omega^-(V_x)$ (Line 1). Based on the timing information of $J_{j^-,h}$ and $J_{j,h}$, $\Delta\tilde{r}\_next(J_{j,h})$ and $\Delta\tilde{UB}\_next(J_{j,h})$ of $J_{j,h}$ are computed according to (4.11) and (4.14) (Lines 2–3). Based on $\Delta\tilde{r}\_next(J_{j,h})$ and $\Delta\tilde{UB}\_next(J_{j,h})$, $r_{i,k}$ and $UB_{i,k}$ of a future sub-job $J_{i,k}$ are updated (Lines 4–11) if $J_{i,k}$ is an instance of $T_{m,h}$. If the newly obtained $r_{i,k}$ and $UB_{i,k}$ of $J_{i,k}$ are out of W, $J_{i,k}$ is removed from $\Omega^w(V_x)$ (Lines 12–13). Finally, $J_{j^-,h}$ is removed from $\Omega^-(V_x)$ because $J_{j^-,h}$ is not the latest arriving sub-job of $T_{m,h}$ due to the arrival of $J_{j,h}$.

Since the prediction by WLDA may not always be accurate and may lead to an overestimation of the resource required by future sub-jobs, we need to be judicious when dropping sub-jobs (see Lines 21–25 in Algorithm 3). To avoid dropping a sub-job prematurely, in function *Drop_Sub_Jobs*(), we introduce a user-defined

threshold parameter $Max\_Allowed\_Drop\_Num$ to help balance the prediction inaccuracy. Specifically, if the counter value is higher than $Max\_Allowed\_Drop\_Num$, the sub-job is then actually dropped.

The most time consuming part of Algorithm 5 is running OLDA at Line 32, which is $O(|\Omega^w(V_x)|^3)$. In addition, WLDA calls OLDA for at most $|\Omega^w(V_x)|$ times if some sub-jobs are dropped. Therefore, the time complexity of WLDA is $O(|\Omega^w(V_x)|^4)$, where $|\Omega^w(V_x)|$ is the number of active and predicted future sub-jobs on processor V_x. Since processor V_x calls WLDA every time a new sub-job from $\Omega(V_x)$ is released at V_x, processor V_x takes $O(|\Omega(V_x)| \cdot |\Omega^w(V_x)|^4)$ time to solve a set $|\Omega(V_x)|$ using WLDA.

Since WLDA considers some future sub-jobs, it may be able to detect an unschedulable sub-job subset earlier and hence drop sub-jobs earlier than ALDA. Doing so avoids wasteful execution of sub-jobs whose successors would be dropped later, and results in a possible decrease in the total number of dropped sub-jobs. However, if ALDA cannot find a feasible local-deadline assignment for a task set, neither can WLDA. This observation is summarized in Lemma 10.

Lemma 10. *If ALDA drops any sub-jobs due to an infeasible deadline assignment, WLDA must also drop some sub-jobs.*

Proof: *We prove the lemma by contradiction. Suppose there is a task set that cannot be solved by ALDA, but it can be solved by WLDA. Assume that processor V_x has not dropped any job using ALDA until it finds an unschedulable sub-job set $\Omega^a(V_x)$ at time t. Processor V_x would encounter the sub-job set $\Omega'(V_x)$ that contains all the same active sub-jobs in $\Omega^a(V_x)$ at time t using WLDA, because it processes sub-jobs in the same order as using ALDA. Since processor V_x considers not only the existing sub-jobs, but also the future sub-jobs when analyzing the schedulability*

of $\Omega'(V_x)$ at time t using WLDA, V_x would consider $\Omega'(V_x)$ unschedulable, and drop at least one job at time t. This contradicts our initial assumption, and the lemma holds. □

4.3 Practical Consideration

In this section, we discuss two important issues applicable to all versions of OLDA. Specifically, we present a communication mechanism to support timely release of a sub-job whose predecessors, possibly on different processors, have finished execution. In addition, we discuss the influence of time overhead by OLDA on the performance of the DRTSs.

4.3.1 Communication Mechanism

OLDA and its derivatives all rely on the following generic communication scheme. A processor completing a sub-job sends a message to downstream processors which are to execute the immediate successors of the completed sub-job. The message contains the identifier of the completed sub-job and the identifier of the subtask that the completed sub-job belongs to, which downstream processors utilize to release new sub-jobs. In addition, the maximum allowed response time of job comprising the completed sub-job is included in the message, based on which downstream processors calculate the upper bound on the local deadline of the newly released sub-job and compute the necessary local deadlines. We define the maximum allowed response time $resp_i(t)$ of job J_i at time t to be the difference between the relative end-to-end deadline $\mathcal{D}_m$ of task T_m that J_i belongs to and the total delay that J_i has experienced up to time t. Moreover, the message includes the dropped job identifiers which have been recorded in the local processor but

never been told to downstream processors. The downstream processors utilize such information to drop sub-jobs for which other sub-jobs composing the same job have been dropped in other processors. Notice that whenever a sub-job is dropped, the job that this dropped sub-job belongs to cannot meet its end-to-end deadline and all the sub-jobs belonging to the job needs to be dropped. To support our proposed algorithms, we can employ a low-cost communication mechanism implemented in a bus-based network similar to those discussed in [30, 57, 59, 161].

To reduce network traffic, we do not require global clock synchronization when implementing our algorithms. The main challenge lies in how to calculate the local deadline upper bound of a newly released sub-job without requiring global clock synchronization. For any newly released sub-job, its execution time, release time and local deadline upper bound is required in OLDA. The execution time and release time of a sub-job are known locally. In contrast, the local deadline upper bound of a sub-job is determined by the end-to-end deadline of the corresponding job comprising the sub-job according to (3.3), which may be different on different processors in an asynchronous DRTS. Therefore, downstream processors cannot directly use the end-to-end deadline value of a job delivered from the local processor. We employ a distributed method to calculate the local deadline upper bound of a newly released sub-job, which leverages the relative end-to-end deadline of a *task*. Below, we illustrate how to accomplish this without requiring global clock synchronization.

Assume that sub-job $J_{i,k}$ belonging to subtask $T_{m,k}$ is assigned to processor V_y. To calculate $UB_{i,k}$ of $J_{i,k}$, processor V_y needs to obtain the end-to-end deadline D_i according to (3.3), where D_i can be calculated by using the following equation,

$$D_i = r_{i,k} + resp_i(r_{i,k}). \tag{4.15}$$

When all the immediate predecessors of $J_{i,k}$ have finished execution, $J_{i,k}$ is immediately released. Therefore, $resp_i(r_{i,k})$ of J_i at time $r_{i,k}$ is calculated by

$$resp_i(r_{i,k}) = \min_{\forall J_{i,h} \preccurlyeq J_{i,k}} \{resp_i(d_{i,h})\}. \tag{4.16}$$

Without loss of generality, suppose an immediate predecessor $J_{i,h}$ of $J_{i,k}$ is finished on processor V_x at time $d_{i,h}$. Then, processor V_x can obtain $resp_i(d_{i,h})$ by using equation

$$resp_i(d_{i,h}) = resp_i(r_{i,h}) - (d_{i,h} - r_{i,h}). \tag{4.17}$$

If $J_{i,k}$ is an input sub-job, the maximum allowed response time $resp_i(r_{i,k})$ of J_i is equal to the relative end-to-end deadline $\mathcal{D}_m$ of task T_m. Therefore, to calculate $UB_{i,k}$ of $J_{i,k}$, a message triggered by the completion of $J_{i,h}$ at $d_{i,h}$ is sent from V_x to V_y to inform V_y about the completion of $J_{i,h}$. Then, processor V_y reads $r_{i,k}$ directly and uses (3.3) to calculate the local deadline upper bound $UB_{i,k}$ of $J_{i,k}$ without global clock synchronization.

Furthermore, every time a processor drops a sub-job or overhears that a sub-job has been dropped by other processors, the local processor adds the drop information to a list of jobs whose sub-jobs have been dropped. When a message is sent from V_x to V_y to inform V_y about the completion of $J_{i,h}$, the message will include the dropped jobs' identifiers which have been recorded in V_x's list but never been told to V_y by V_x. Similarly, when V_y receives a message that a sub-job finished execution at processor V_x, V_y will send V_x an acknowledgement message. The acknowledgement message includes the dropped jobs' identifiers that have been recorded in V_y's list and have never been told to V_x by V_y. The list of dropped jobs kept by each processor is then updated and the corresponding sub-jobs are

dropped when they arrive at the local processor.

In all, the message transmitted upon the completion of a sub-job contains a small amount of information, which can be supported by the bus-based network platforms, e.g., Controller Area Network (CAN). There exist some communication delays due to message transmissions among processors. When the on-line derivatives of OLDA, ALDA and WLDA, are implemented in a DRTS, the communication delays between the processors can increase the maximum allowed response time of jobs on the downstream processors and reduce the local deadline upper bounds of sub-jobs on the upstream processors. If the communication delays along the downstream paths of sub-jobs can be estimated, these delays can be readily incorporated into the maximum allowed response time of jobs and the local deadline upper bounds of sub-jobs during runtime. Hence, though ALDA and WLDA cannot precisely handle communication delays along the downstream paths, they can indirectly account for such delays.

4.3.2 Influence of Time Overhead by OLDA

The time overhead associated with OLDA may cause some sub-jobs to miss their local deadlines. There are two factors that determine the effects of time overhead of OLDA on the performance of the distributed system. The first factor is the density level of a task T_n, i.e., $\frac{C_n}{\mathcal{D}_n}$. A job of a task with a high density level has a higher probability of missing its end-to-end deadline when it is delayed due to the execution of OLDA. The second factor is the ratio of the time overhead over the relative end-to-end deadline of task T_n, i.e., $\frac{Overhead}{\mathcal{D}_n}$. If the relative end-to-end deadline of a job is not large enough to accommodate the time overhead of OLDA, the job will violate its end-to-end deadline. The time overhead is determined by

the time complexity of OLDA and the frequency of the call to OLDA. Since the time complexity of OLDA's derivatives is at least quadratic in the number of sub-jobs and OLDA needs to be run each time a sub-job enters a local processor, our algorithm is suitable for DRTSs where dozens of active sub-jobs are to be executed, e.g., avionics and automotive control applications. We discuss quantitatively the effect of the time overhead of OLDA in Section 4.5.4.

4.3.3 Sub-Job Dropping Policies

There exist a large number of job dropping policies for soft and firm real-time systems. Examples include, but are not limited to, on-demand job dropping policy [22], deadline based job dropping policy [75], Markov Chain job dropping policy [100], and uniform job dropping policy [115]. All these policies focus on improving the performance of firm real-time systems, and not on reducing the job-drop rates, on uniprocessor systems. For DRTSs, the work in [107, 151, 158] proposed different job or packet dropping policies. Specifically, the authors in [107] assume that an active instance of task T_m will be discarded when a new instance of T_m is released. This policy may drop a job unnecessarily if that job can still catch up with its end-to-end deadline. Furthermore, it may not drop a job that can no longer meet its end-to-end deadline, thus wasting resources. The work in [151] assumes that a packet will be dropped after 7 retransmissions of that packet. Again, such a policy does not drop a packet early enough to save computation resources. In [158], the authors proposed a packet dropping policy that considers packet latency and the utilization of a given node. Such a policy neglects the impact of dropping a packet on the utilizations of downstream processors in the system.

In OLDA, a processor must determine a subset of sub-jobs to drop when it cannot find a feasible local-deadline assignment. Note that when a processor drops any sub-job of a job, the job can no longer meet its end-to-end deadline and all the sub-jobs of this job need to be dropped. When OLDA decides to drop a subset of future sub-jobs, these future sub-jobs are not removed from $\Omega(V_x)$ immediately. Instead, each time a future sub-job is considered by OLDA to be dropped, a corresponding drop counter of this future sub-job is incremented in $Drop\_Sub\_Jobs()$. In contrast, if OLDA decides to drop a subset of active sub-jobs in function $Drop\_Sub\_Jobs()$ of Algorithm 2, these sub-jobs are immediately removed from $\Omega(V_x)$. In order to maximize the number of jobs meeting their end-to-end deadlines, it is desirable that as few jobs are abandoned by the system as possible. We consider two sub-job dropping policies, as explained next.

The first policy (denoted as MLET, for Maximum Local Execution Time) abandons a job with the largest execution time on the processor first. This step is repeated until a local-deadline assignment can be found for the remaining sub-jobs. Therefore, MLET ensures that the least number of jobs are abandoned locally. The MLET policy focuses on the execution times of the existing sub-jobs without considering the execution times of their successors. Therefore, by using MLET, it is possible for a processor to keep jobs that have smaller local execution times but larger remaining execution times than the dropped job. This, in turns, may lead to higher future workload on subsequent processors and result in high resource competition in the near future.

In contrast, the second policy (denoted as MRET, for Maximum Remaining Execution Time) addresses the deficiencies of MLET by giving up the job with the maximum remaining execution time first. We define the remaining ex-

ecution time of a job to be the job's remaining local execution time plus the sum of path execution times among all the paths of all the job's local active instances. For example, the remaining execution time of job J_i on processor V_x is $\sum_{\forall k, J_{i,k} \in \Omega(V_x)} (C_{i,k} + \sum_{\forall p_{i,k,k'} \in \mathcal{P}_{i,k}} \sum_{\substack{\forall J_{i,h} \in P_{i,k,k'} \\ J_{i,k} \prec J_{i,h}}} C_{i,h})$. By using MRET, OLDA can maximally reduce future workload on subsequent processors and alleviate potentially high resource competition.

4.4 Evaluation

In this section, we analyze the performance and efficiency of our proposed algorithms using generated task sets. We start by selecting sub-job dropping policy for ALDA and WLDA, and determining optimal input parameters for WLDA. Next, we evaluate ALDA with a specific sub-job dropping policy for ALDA, and then compare ALDA against WLDA. Note that we do not evaluate OLDA since it is not practical in real settings, as explained in Section 4.1. To determine how our proposed algorithms fare against existing techniques, we select one derivative with a better performance out of ALDA and WLDA and compare this derivative against JA and BBW for different types of workloads. Notice that ALDA performs better than WLDA for the ST workloads while WLDA performs better than ALDA for the GT workloads. Finally, we show the performance of ALDA under real-world workloads. Below, we describe the simulation setup and then discuss simulation results.

4.4.1 Simulation Setup

The distributed system consists of 8 processors. We use two different types of workloads, the stream-type (ST) workloads and the general-type (GT) workloads

to emulate different kinds of application scenarios. For the ST workloads, each task is composed of a chain of subtasks. Such tasks can be found in many signal processing and multimedia applications. In contrast, the GT workloads are more general where (i) a sub-job may have multiple successors and predecessors, and (ii) there is no fixed execution order in the system.

The ST workloads consist of randomly generated task sets in order to evaluate two different processor loading scenarios. For the first set of workloads, the execution time of a job is randomly distributed along its execution path. As a result, processor loads tend to be balanced. As a stress test, the second set of workload represents a somewhat imbalanced workload distribution among the processors. The workloads were generated in such a way that the first few subtasks as well as the last few subtasks are more heavily loaded. This set of workloads was designed to test the usefulness in considering severe resource competition among different jobs on a given processor in meeting end-to-end deadlines. (Note that imbalanced workload scenarios may occur in real life if an originally balanced design experiences processor failures and the original workload must be redistributed.)

Both sets of the ST workloads contain 100 randomly generated task sets of 50 tasks each for 10 different system utilization levels $(400\%, 425\%, \ldots, 625\%)$, for a total of 1,000 task sets. Each task is composed of a chain of 4 to 6 subtasks. Each subtask is assigned to a processor such that no two subtasks of the same task run on a common processor. Task periods were randomly generated within the range of from 100,000 to 1000,000 microseconds and the end-to-end deadlines were set to their corresponding periods. We used the UUnifast algorithm [22] to generate the total execution time of each task since UUnifast provides better control on how to assign execution times to subtasks than a random assignment.

After the call to the UUnifast algorithm, the set of processors used by task T_i was randomly selected based on the actual number of subtasks M_i for each task T_i and the execution time of each subtask was determined. Each task set was generated with the guarantee that the total utilization at each processor is no larger than 1.

Similar to the ST workloads, the GT workloads also contain 1000 task sets, but each set only has between 25 and 100 subtasks. The GT workloads were generated using TGFF [49]. Task periods were generated using uniform distribution and can take any value between $[10,000, 150,000]$ microseconds. The end-to-end deadline of each job was set to be equal to the release time of the job plus the period of the corresponding task. The execution time of a subtask was randomly generated and was within $[1, 10,000]$ microseconds. After the task set was generated, the execution time of each sub-job was uniformly scaled down so that the total utilization of the task set is equal to the desired utilization.

To ensure a fair comparison of the different algorithms under consideration, we made some modifications to JA and BBW. The original versions of JA and BBW require global clock synchronization. We removed this requirement by implementing JA and BBW on-line on each processor. We implemented our proposed algorithms (ALDA and WLDA) as well as two sub-job dropping policies. The first policy (denoted as MLET, for Maximum Local Execution Time) abandons a job with the largest execution time on the processor first. The second policy (denoted as MRET, for Maximum Remaining Execution Time) drops the job with the largest remaining execution time on the processor first. All algorithms were implemented in C++. Experimental data were collected on a computer cluster, which is composed of 8 quad-core 2.3 GHz AMD Opteron processors with Red Hat Linux 4.1.2-50. Each task set was simulated for the time interval $[0, 100 \cdot max\_period]$,

where *max_period* is the maximum period among the periods of all the tasks in the task set.

We measure the performance of each algorithm with three metrics. The first metric is the *job drop rate*, i.e., the ratio between the number of jobs dropped and the number of jobs released in the system. This metric measures the algorithm's dynamic behavior in a soft real-time system. The second metric is the *number of schedulable task sets*. This metric indicates each algorithm's ability in finding feasible solutions (i.e., static behavior). The third metric is the *running time* of each algorithm (averaged on each processor) to solve a task set. This metric shows the time overhead of each algorithm.

4.5 Comparison of Sub-Job Dropping Policies

The job-drop rates due to the use of MRET and MLET in ALDA are summarized in table 4.1. The differences of job-drop rates due to the use of MRET and MLET are very small because both sub-job dropping policies are very effective in reducing the job drop number for the workloads. Specifically, experimental results show that MLET drops 1.02%, 0.93% and 3.28% (up to 1.23%, 1.76%, and 7.98%) more jobs on average than MRET for the balanced ST workloads, the imbalanced ST workloads and the GT workloads, respectively. This improvement of MRET over MLET can be attributed to that MRET considers the execution times of both the active sub-jobs on the processor and the successors of these sub-jobs, which helps ease future resource competition. In addition, MRET solves more task sets than MLET by one for the GT workloads while both policies solve the same numbers of task sets for the balanced and imbalanced ST workloads. Thus, we will use MRET in ALDA in the rest of the paper. The selection of the sub-job

TABLE 4.1

Job-Drop Rates Generated by ALDA Employing Sub-Job Dropping Policies, MRET and MLET, for the Different Workloads

Utilization Level	GT (%) MRET / MLET	Balanced ST (%) MRET / MLET	Imbalanced ST (%) MRET / MLET
400%	0 / 0	0 / 0	0 / 0
425%	0 / 0	0 / 0	0 / 0
450%	0.693 / 0.693	0 / 0	0.000478 / 0.000478
475%	0.0628 / 0.064	0 / 0	0.000398 / 0.000398
500%	0.217 / 0.217	0.000796 / 0.000796	0.00335 / 0.00335
525%	1.06 / 1.08	0.00765 / 0.00765	0.0146 / 0.0143
550%	1.49 / 1.52	0.0130 / 0.0131	0.0263 / 0.0266
575%	2.13 / 2.26	0.0579 / 0.0584	0.0362 / 0.0369
600%	3.43 / 3.69	0.0591 / 0.0598	0.0691 / 0.0698
625%	8.23 / 8.32	0.0556 / 0.0562	0.105 / 0.106

dropping policies for different workloads by ALDA and WLDA is summarized in table 4.2.

4.5.1 Selection of Optimal Input Parameters for WLDA

Since the performance of WLDA depends on the parameters $Max\_Allowed\_Drop\_Num$ and α, we start by calibrating these parameters to fully exploit the potential of WLDA. We adopt two metrics when selecting the values. The main metric is the number of dropped jobs caused by WLDA. If multiple variable settings for WLDA result in the same number of dropped jobs, we employ the number of solved sets

TABLE 4.2

Selection of Sub-job Dropping Policies for Different Types of Workloads by ALDA and WLDA.

Workload Type	GT	Balanced ST	Imbalanced ST
ALDA	MRET	MRET	MRET
WLDA	MLET	MLET	MLET

by WLDA as the secondary metric. That is, we will use the variable setting which results in more sets being solved. Note that for ALDA, we do not need to set the values for $Max\_Allowed\_Drop\_Num$ since ALDA does not consider future sub-jobs. As a starting point, we tested WLDA using different values $(1, 2, 3, \ldots, 10)$, for $Max\_Allowed\_Drop\_Num$ while fixing the scaling factor α to 1. The average drop rates for the GT workloads, the balanced ST workloads and the imbalanced ST workloads are shown in Figures 4.2, 4.3 and 4.4, respectively. The data reveal that setting $Max\_Allowed\_Drop\_Num$ to 1 and employing MLET as the sub-job dropping policy result in the fewest number of dropped jobs for the GT workloads. However, the number of dropped jobs using WLDA is never fewer than that using ALDA for the balanced and imbalanced ST workloads, regardless of the values of $Max\_Allowed\_Drop\_Num$ and the selection of the sub-job dropping policies. Thus, we set $Max\_Allowed\_Drop\_Num$ to 1 and 10 for the balanced and imbalanced ST workloads, respectively, and use MLET as the sub-job dropping policy for the ST workloads to obtain the best performance. The numbers of the schedulable task sets by WLDA for the three sets of the workloads are shown in table 4.3.

TABLE 4.3

Number of Schedulable Task Sets by WLDA with Different $Max\_Allowed\_Drop\_Num$ Values

$Max\_Allowed\_Drop\_Num$		1	2	3	4	5	6	7	8	9	10
GT	MLET	817	817	817	815	815	815	818	816	817	817
	MRET	818	817	815	817	818	813	816	816	815	816
Balanced ST	MLET	848	845	846	845	845	844	845	845	844	845
	MRET	847	846	847	844	844	846	846	845	843	848
Imbalanced ST	MLET	715	715	714	717	715	718	716	715	714	715
	MRET	713	715	714	714	715	717	716	717	719	716

Next, we tested different values $(0.2, 0.4, 0.6, \ldots, 1.2)$ for the scaling factor α while fixing $Max\_Allowed\_Drop\_Num$ to the values selected above for all three sets of the workloads. The average drop rates by WLDA with different α values are shown in Figure 4.5. The left y-axis shows the average drop rates at different α values for the balanced and imbalanced ST workloads, while the right y-axis shows the average drop rates at different α values for the GT workloads. Based on the results, we find that setting α to 0.6 minimizes the number of dropped jobs for the GT workloads. However, regardless of the value of α, the number of dropped jobs using WLDA is never fewer than that using ALDA for the balanced and imbalanced ST workloads. Thus, for the balanced and imbalanced ST workloads, we set α to 0.2 to obtain the best performance. The numbers of the solved task sets by WLDA with different α values are shown in table 4.4.

TABLE 4.4

Number of Schedulable Task Sets by WLDA with Different α Values Using MLET

	0	0.2	0.4	0.6	0.8	1.0	1.2
GT	843	841	839	834	829	817	797
Balanced ST	861	859	858	856	855	848	834
Imbalanced ST	745	739	735	735	724	715	694

4.5.2 Comparing OLDA Derivatives

We now discuss the comparison results for our proposed algorithms, ALDA and WLDA for the ST and GT workloads. Since the performance of WLDA depends on some input parameters (e.g., $Max\_Allowed\_Drop\_Num$ and α), we set these parameters to optimal values in order to fully exploit the potential of WLDA.

The job-drop rates for the balanced ST workloads, the imbalanced ST workloads and GT workloads obtained by ALDA and WLDA are shown in Figures 4.6, 4.7 and 4.8, respectively. The numbers of solved task sets by ALDA and WLDA are summarized in table 4.5. ALDA solves 2, 6, and 10 more task sets than WLDA for the balanced ST workloads, the imbalanced ST workloads and GT workloads, respectively. WLDA drops more jobs than ALDA in 72, 40 and 75 sets of the GT workloads, the balanced ST workloads and the imbalanced ST workloads, respectively.

The running time of the task sets at different utilization levels for the balanced ST workloads, the imbalanced ST workloads and GT workloads obtained by ALDA and WLDA are shown in Figures 4.9, 4.10 and 4.11, respectively. The

TABLE 4.5

Numbers of Solved Tasks Generated by ALDA and WLDA for the Different Workloads

Utilization Level		400%	425%	450%	475%	500%
GT	ALDA	100	100	98	98	98
	WLDA	100	99	98	97	97
Balanced ST	ALDA	100	100	100	99	95
	WLDA	100	100	100	99	95
Imbalanced ST	ALDA	100	100	96	96	90
	WLDA	100	100	96	95	90
Utilization Level		525%	550%	575%	600%	625%
GT	ALDA	91	86	75	61	37
	WLDA	91	83	75	58	36
Balanced ST	ALDA	92	88	73	67	47
	WLDA	92	88	72	67	46
Imbalanced ST	ALDA	81	72	51	35	24
	WLDA	81	71	50	34	22

average number of cycles required to run WLDA once is 7797, 12719 and 4508 for the balanced ST workloads, imbalanced ST workloads and GT workloads, respectively. These numbers are dependent on the number of active sub-jobs plus that of the predicted sub-jobs, which are 3, 4 and 8 on average for the balanced ST workloads, imbalanced ST workloads, and GT workloads, respectively.

Table 4.6 summarizes the total job drop rates, total solved task set numbers and total running times of solving all the 1000 task sets by ALDA and WLDA for different types of workloads. It is found that WLDA drops 5.21% fewer jobs (up to

TABLE 4.6

Comparison of ALDA and WLDA in terms of the Three Metrics for Different Types of Workloads.

Metrics		Job Drop Rate (%)	Solved Set Number	Running Time (μs)
GT	ALDA	1.70	844	4549292
	WLDA	1.62	834	86812928
Balanced ST	ALDA	0.0195	861	6604004
	WLDA	0.0203	859	68431633
Imbalanced ST	ALDA	0.0257	745	6924910
	WLDA	0.0279	739	70330281

9.09%) on average than ALDA for the GT workloads. In contrast, WLDA drops 4.49% and 8.80% more jobs (up to 7.54% and 120%) on average than ALDA for the balanced and imbalanced ST workloads, respectively. Our results show that ALDA can solve 2, 6 and 10, more task sets (out of 1000 task sets) than WLDA for the balanced ST workloads, imbalanced ST workloads and GT workloads, respectively. WLDA requires 11, 11, and 19 times more cycles on average than ALDA for the balanced ST workloads, imbalanced ST workloads and GT workloads, respectively.

Ideally, WLDA can use its prediction mechanism to reduce the number of dropped jobs. However, WLDA may drop a schedulable job by mistake due to a mis-prediction of future sub-jobs. Moreover, the time overhead caused by the prediction mechanism can greatly degrade the performance of WLDA. This time overhead is caused by several maintenance operations (such as the update of the timing information of the future sub-jobs already considered in the previous assignment, the addition and removal of some new and obsolete future sub-jobs,

respectively, etc.) in the prediction mechanism. The time overhead caused by the prediction mechanism in WLDA makes some jobs not only miss their assigned local deadlines but also violate their local deadline upper bounds. In summary, our results indicate that ALDA performs better than WLDA for the ST workloads while WLDA outperforms ALDA for the GT workloads. The higher time overhead of WLDA makes it unsuitable for ST workloads, as a sub-job $J_{i,k}$ can delay the execution of all its successors $J_{i,k'}$'s since a job is composed of a chain of sub-jobs. In contrast, for the GT workloads, the time overhead incurred due to scheduling will most likely not delay the execution of all the other active sub-jobs $J_{i,k'}$'s in J_i since some active sub-jobs in J_i are not successors of $J_{i,k}$. Therefore, we focus on ALDA for the ST workloads and WLDA for the GT workloads in the discussion below.

4.5.3 Performance of OLDA against Other Algorithms

We compare the performance of OLDA with JA and BBW, the two representative priority assignment methods. We use ALDA and WLDA to test the performance of OLDA in the ST and GT workloads, respectively. In the first experiment, we compare the average job drop rates of infeasible task sets when using different algorithms for balanced ST workloads, imbalanced ST workloads and GT workloads. A job is dropped either because no local-deadline assignment can be found for the sub-job set on a processor using OLDA or the job's end-to-end deadline is missed using BBW and JA. The job drop rates for the three algorithms for the balanced ST workloads, imbalanced ST workloads, and GT workloads are shown in Figures 4.12, 4.5.3, and 4.14, respectively.

It is clear that OLDA drops much fewer jobs than the other two methods.

Specifically, for balanced ST workloads, BBW and JA drop 179% and 165% more jobs on average than OLDA, respectively. For imbalanced ST workloads, the averages are 61% and 313%, respectively. For GT workloads, 160% and 51% more jobs are dropped by BBW and JA than those by OLDA on average, respectively.

In the second experiment, we compare the percentage of feasible task sets (over the 100 task sets at each utilization level) found by our algorithm, with those found by JA and BBW for the three sets of the workloads. The results are summarized in Figures 4.15, 4.16 and 4.17, respectively. The data shows that OLDA finds far more feasible sets than the other two methods. Specifically, for balanced ST workloads, using OLDA leads to 71% and 22% on average (and up to 2,250% and 124%) more feasible task sets than using BBW and JA, respectively. For imbalanced ST workloads, using OLDA results in 60% and 48% on average (and up to 338% and 2300%) more feasible task sets than BBW and JA, respectively. For GT workloads, the number of solutions found by OLDA is 13% and 12% on average (and up to 100% and 200%) more than that found by BBW and JA. Observe that OLDA performs much better than existing techniques at high utilization levels where there are more jobs in the system. We would also like to point out that sometimes OLDA may not be able to find a feasible solution even though such solutions indeed exist, since OLDA finds local sub-job deadlines for each processor independently instead of using a global approach. For balanced ST workloads, OLDA can find on average 98.81% and 99.86% of those found by BBW and JA, respectively. For imbalanced ST workloads, OLDA can find on average 95.27% and 99.40% of the feasible task sets found by BBW and JA, respectively. For GT workloads, OLDA can find on average 99.86% and 99.60% of the feasible task sets found by BBW and JA, respectively. These results demonstrate that

OLDA not only finds more feasible task sets than BBW and JA, but also solves most of the problems that BBW and JA can solve.

To see how well OLDA fares compared to an MILP solver, we randomly selected 3 workloads containing 4, 20 and 26 tasks, respectively, and compared the solutions obtained by OLDA and lp_solve [3], an MILP solver. For the workload with 4 tasks, both OLDA and lp_solve find the same solution. For the workload with 20 tasks, OLDA and lp_solve find two different feasible solutions, however, the objective function values by the two solutions are the same. For the workload with 26 tasks, OLDA is able to solve the problem containing 10 sub-jobs within 6 ns while lp_solve fails to find a solution after running for 48 hours. The comparisons support our earlier claim that OLDA always finds an optimal solution to the problem stated in (3.8), (3.6) and (3.9) whenever a feasible solution exists. Furthermore, the execution time of OLDA is more suitable for on-line use than that of lp_solve.

4.5.4 Time Overhead of OLDA

In our evaluations, we consider the time overhead due to OLDA when simulating a task set. That is, every time OLDA is called by a local processor, our simulator records the running time of OLDA and postpones the execution of all the active local sub-jobs for that time duration to simulate the influence of the time overhead due to OLDA. To examine whether OLDA is suitable for on-line local-deadline assignments, we show the total running time overheads of OLDA, BBW and JA for the balanced ST workloads, imbalanced ST workloads and GT workloads in Figures 4.18, 4.19 and 4.20, respectively. We still use ALDA and WLDA to test the performance of OLDA in the ST and GT workloads, respectively. Based on the results, we compare the number of cycles required by OLDA

against those of JA and BBW. For the balanced ST workloads, OLDA requires on average 1.76 and 2.34 times more cycles per task set (with 50 tasks) than BBW and JA, respectively. For the imbalanced ST workloads, OLDA needs about 1.81 and 2.43 times more cycles per task set than BBW and JA, respectively. For the GT workloads, OLDA requires on average 30 and 41 times more cycles per task set than BBW and JA. Although OLDA has a longer running time than both BBW and JA, the average numbers of cycles required to run OLDA for once are about 292, 295 and 4507 cycles for the balanced ST workloads, imbalanced ST workloads and GT workloads, respectively, while the average number of the sub-jobs handled by an activation of OLDA is 3, 3 and 8 for the balanced ST workloads, imbalanced ST workloads and GT workloads, respectively. Such runtime overhead is tolerable in DRTSs executing computationally demanding real-time jobs, e.g. in avionics and automotive control applications [70, 129], where dozens of active sub-jobs are to be executed.

TABLE 4.7

Specification of A Flight Control System

Task	Execution Time (ms)								Period	End2End
Name	AH	NV	FC	BS	FG	AP	SV	PF	(ms)	Deadline (ms)
FCP	0	0	15	29	10	15	0	10	500	450
PAA	10	0	0	16	15	20	10	0	100	100
NIP	0	10	0	14	20	0	0	0	250	200

Using a large number of randomly generated task sets, we have shown that A-OLDA outperforms existing methods. However, it is important to quantify the performance of A-OLDA under real-world workloads. We use a simplified flight control system [70], and a fault-tolerant distributed system based on the examples in [48, 145], to illustrate how A-OLDA adapts to changing requirements on-line to guarantee the end-to-end deadlines of jobs in actual DRTSs.

The flight control as shown in Figure 4.21 system contains 3 periodic chain subtasks and 8 heterogeneous processors. Flight control processing (FCP) task reads input data from Flight Control Processor (FC), Flight Guidance System (FG) and Auto-Pilot (AP) process the input data sequentially, and Primary Flight Display (PF) displays the results. Pitch adjustment actuation (PAA) subtask receives periodic sensor readings from Attitude and Heading Reference System (AH). The information is processed by FG and AP sequentially, and AH sends control signals to Elevator Servo (SV). Navigation information processing (NIP) subtask periodically receives sensor readings from Navigation Radio (NV) and processes them using FG. All input commands and sensor readings reach FG through a Bus (BS). The subtask execution times on the processors (in ms) are given in columns 2 to 9 and the task end-to-end deadlines, also in ms, in column 11 of table 4.7.

We simulated the application for the time interval $[0, 9000ms]$ ($9000ms$ is the least common multiplier of all the task periods before and after the emergency situation) when applying ALDA, JA and BBW. The results show that no job misses its end-to-end deadline, as shown in column 2 of table 4.8. Now, assume

TABLE 4.8

Case Study of A Flight Control System

Task Name	Dropped Job Num. Before Emergency A-OLDA / JA / BBW	End2End Deadline (=Period)	Dropped Job Num. After Emergency ALDA / JA / BBW
FCP	0 / 0 / 0	120	0 / 0 / 0
PAA	0 / 0 / 0	72	0 / 30 / 5
NIP	0 / 0 / 0	75	0 / 0 / 5

that at some time interval, the airplane encounters some emergency, such as air turbulence or a mechanical malfunction. For safety, the periods and end-to-end deadlines of the three tasks, in ms, are decreased, as shown in column 3 of table 4.7. Suppose we apply JA and BBW in response to the workload and deadline changes. By using JA, 30 jobs are dropped from the PAA task. BBW performs a little better than JA. By using BBW, 5 jobs are dropped from the NIP and PAA tasks each (see the second and third numbers in column 4 of table 4.8). In contrast, applying ALDA leads to all the jobs meeting their end-to-end deadlines. For both situations, we run each algorithm for 10 times and summarize the average time overhead of each algorithm on the 8 processors in Row 2 of Table 4.9.

In addition to applying ALDA in a real-world ST example (the flight control system), we also tested ALDA for a real-world GT example, the fault-tolerant distributed system studied in [48] and [145]. The system contains 6 tasks, composed of 43 subtasks and 36 messages, deployed onto a hardware platform with 8

TABLE 4.9

Average Time Overhead for Both Case Studies

	Time Overhead for Nominal Situation (μs) ALDA / JA / BBW	Time Overhead for New Situation (μs) ALDA / JA / BBW
FCS	100 / 0 / 100	99.9 / 100 / 0
FTDS	99.9 / 0 / 0	200 / 99.9 / 100

processors and a single bus. The subtasks and their dependencies for all the tasks are shown in Figure 4.22. The task identifiers, task periods, subtask identifiers, subtask execution times and processor assignment are given in table 4.10, while the message transmission between different subtasks is given in table 4.11. Here, tasks 0, 1, 2, 3 are presented in columns 1 to 6, while tasks 4 and 5 are presented in columns 7 to 12. The periods and execution times are in ms. The sizes (in bytes) of the messages sent between subtasks are given in columns 6 and 12. For example, "500→1 1500→2" of subtask 0 indicates that two messages of 500 bytes and 1500 bytes are sent to subtasks 1 and 2 from subtask 0, respectively. We assume that each message is fragmented to several packets and the fragmentation threshold is 1000 bytes, which is similar to the bus-based network system model employed in [57]. We assume that the transmission time slot of each packet is 1 ms, which can be realized on a high-speed bus. In addition, we assume that the relative end-to-end deadline of each task is equal to its period.

We simulated the application for the time interval $[0, 4200ms]$ ($4200ms$ is the least common multiplier of all the task periods), and found that all the jobs can meet their end-to-end deadlines by applying ALDA, JA and BBW. Now assume

at some time, processor 4 is down and the backup processor is used. Because the backup processor is less powerful, the execution times of the sub-tasks on processor 4 is increased by 1.5 times. Suppose we still apply JA and BBW to adapt to the workload change, 3 and 1 jobs out of the 112 released jobs are dropped from task 4 and task 1, respectively. In contrast, employing ALDA results in all end-to-end deadlines being met. For both case studies, we also run each algorithm for 10 times and summarize the average time overhead of each algorithm on the 8 processors in Row 3 of Table 4.9.

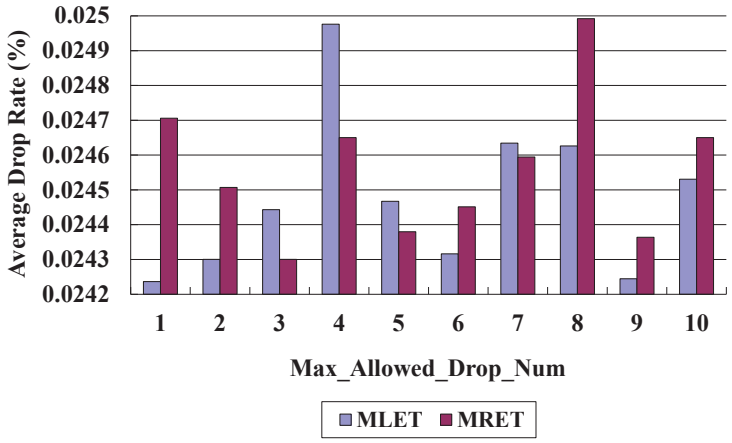


Figure 4.2. Comparison of different $Max\_Allowed\_Drop\_Num$ values in terms of average drop rate by WLDA for balanced ST workloads.

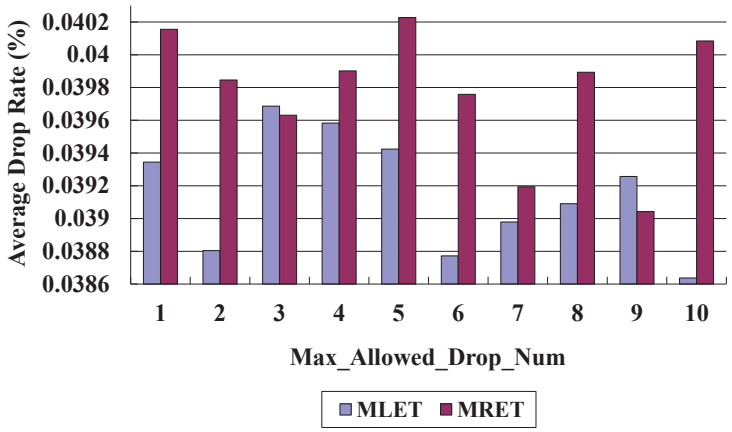


Figure 4.3. Comparison of different $Max\_Allowed\_Drop\_Num$ values in terms of average drop rate by WLDA for imbalanced ST workloads.

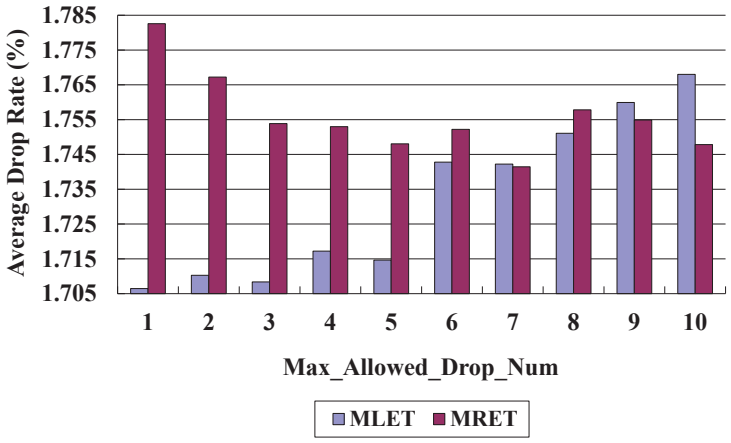


Figure 4.4. Comparison of different $Max\_Allowed\_Drop\_Num$ values in terms of average drop rate by WLDA for GT workloads.

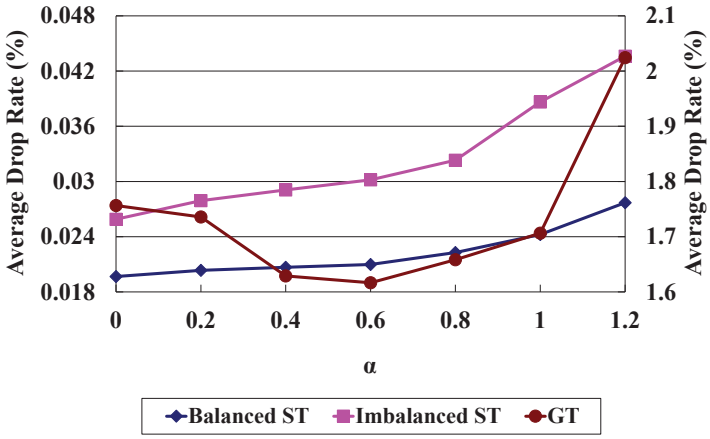


Figure 4.5. Comparison of different α values in terms of average drop rate by WLDA for ST and GT workloads.

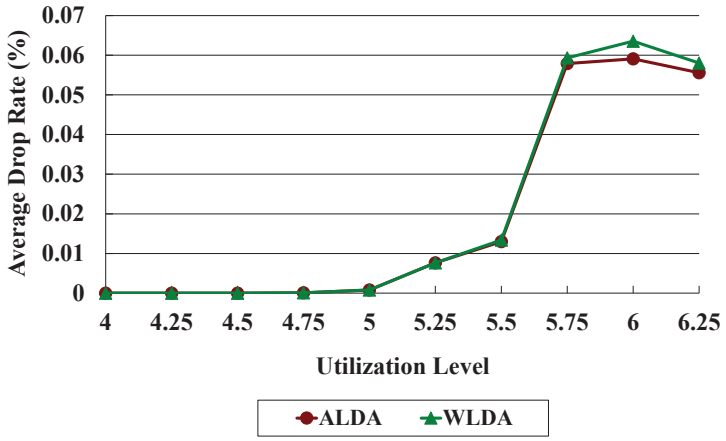


Figure 4.6. Average drop rate for balanced workloads (ST workloads).

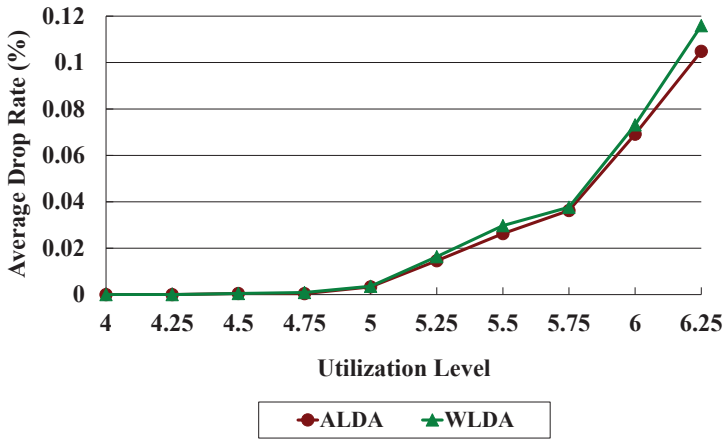


Figure 4.7. Average drop rate for imbalanced workloads (ST workloads).

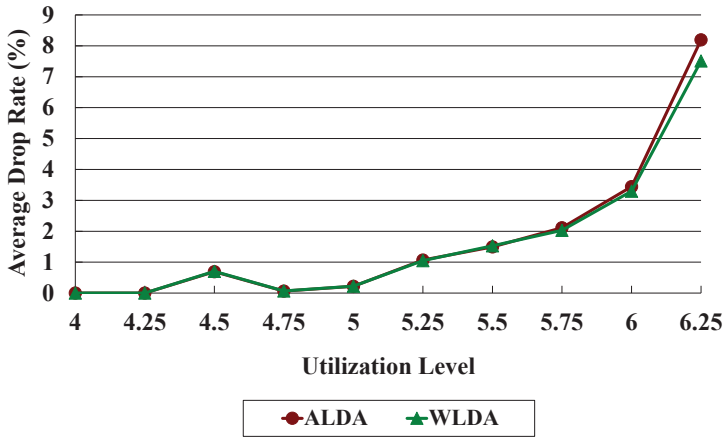


Figure 4.8. Average drop rate for GT workloads.

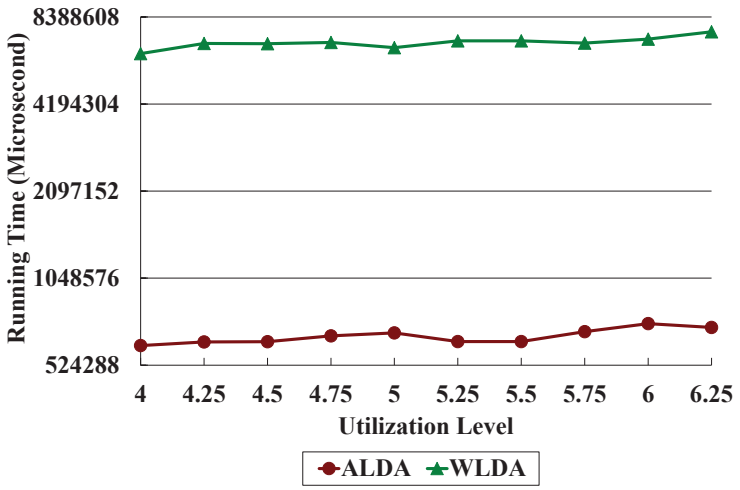


Figure 4.9. Total running time for balanced workloads (ST workloads)

.

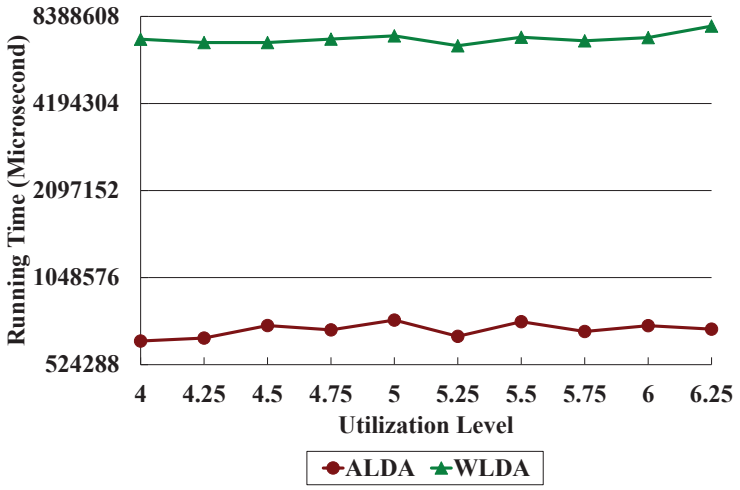


Figure 4.10. Total running time for imbalanced workloads (ST workloads).

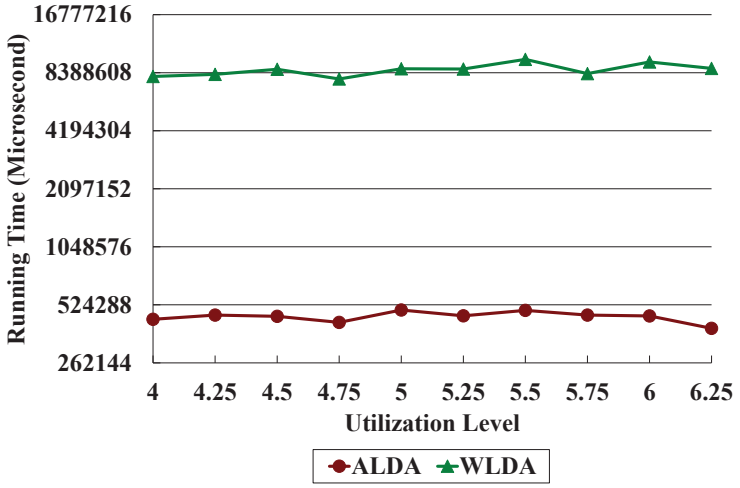


Figure 4.11. Total running time for GT workloads.

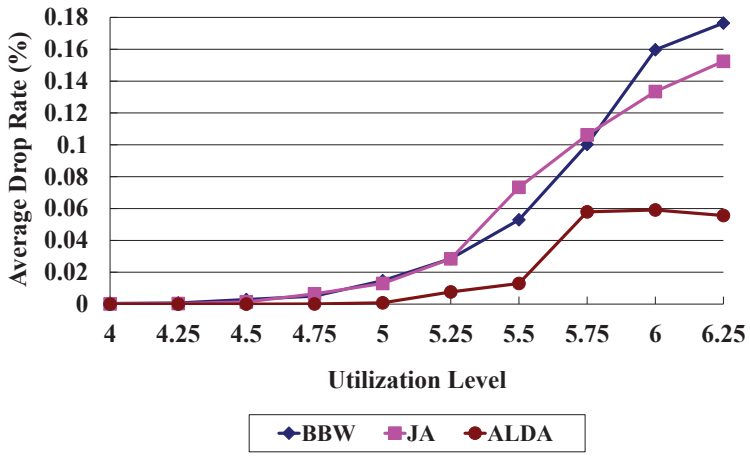


Figure 4.12. Average drop rate for balanced workloads (ST workloads).

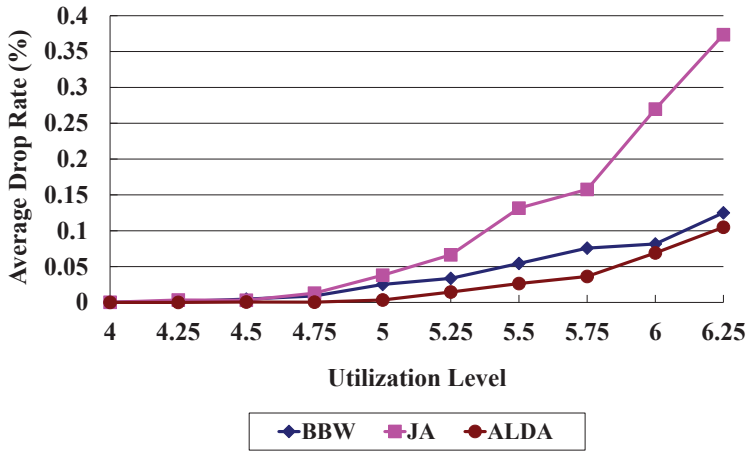


Figure 4.13. Average drop rate for imbalanced workloads (ST workloads).

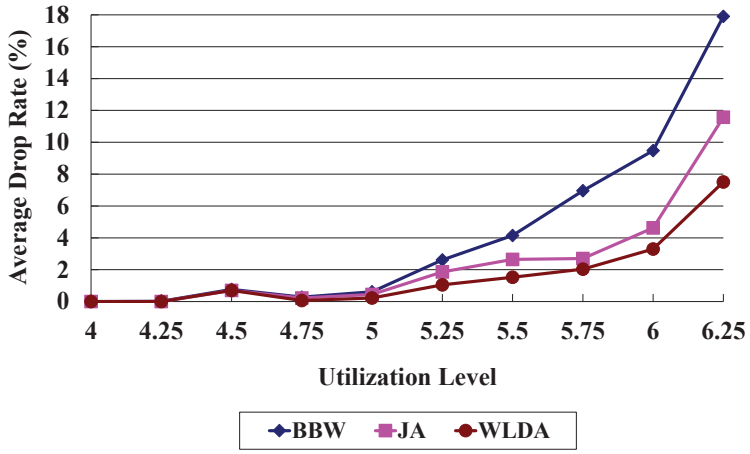


Figure 4.14. Average drop rate for GT workloads.

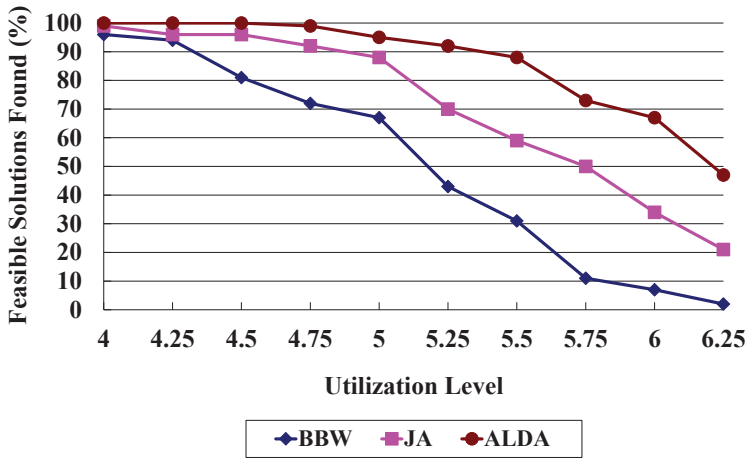


Figure 4.15. Percentage of feasible task sets found for balanced workloads (ST workloads).

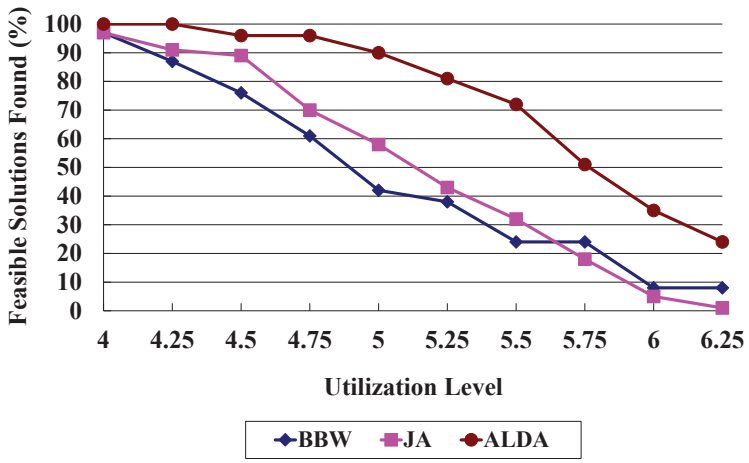


Figure 4.16. Percentage of feasible task sets found for imbalanced workloads (ST workloads).

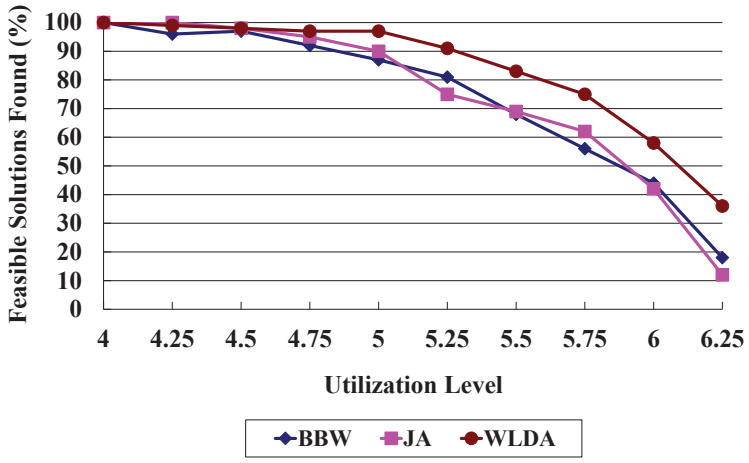


Figure 4.17. Percentage of feasible task sets found for GT workloads.

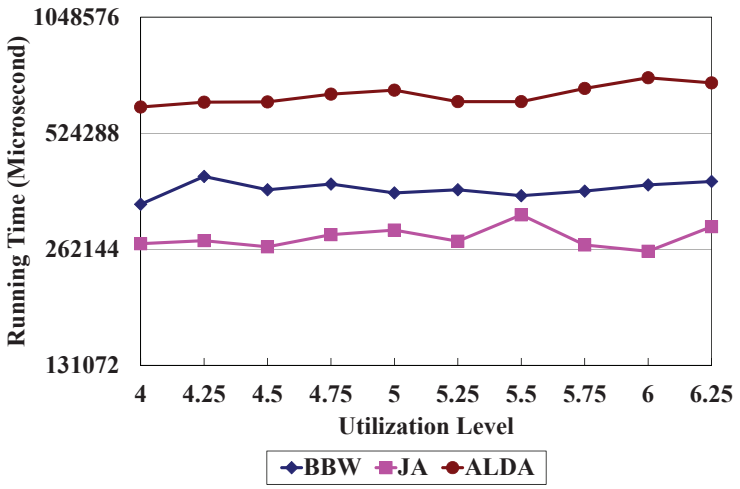


Figure 4.18. Total running time for balanced workloads (ST workloads).

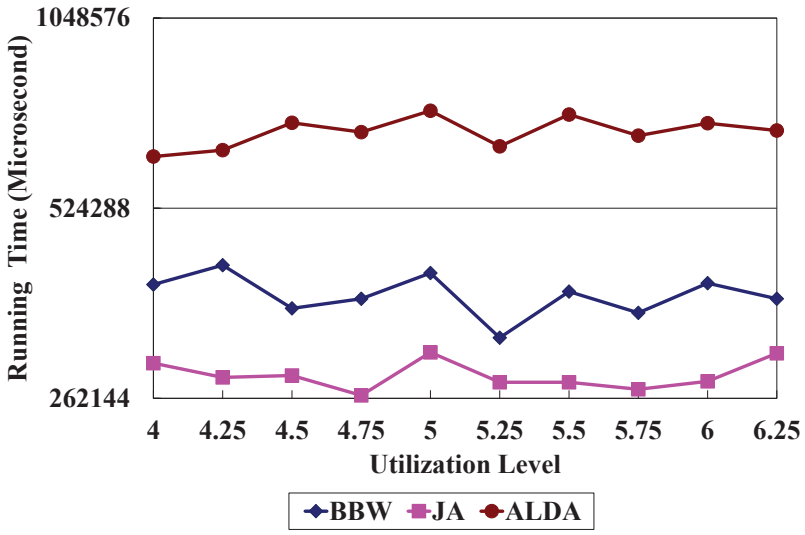


Figure 4.19. Total running time for imbalanced workloads (ST workloads).

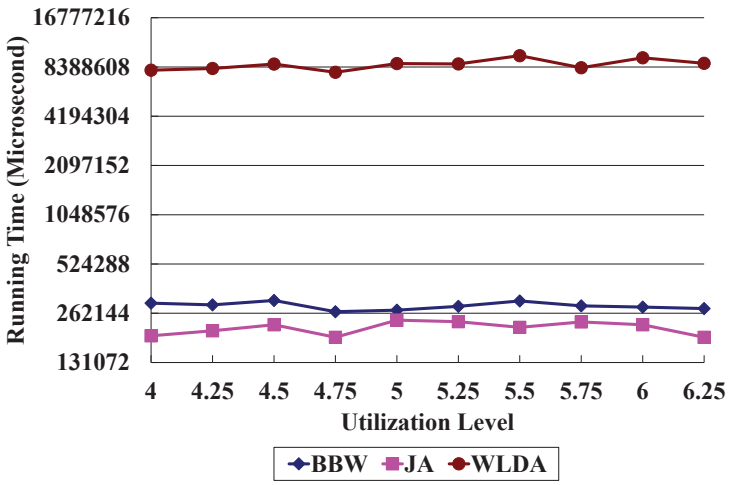
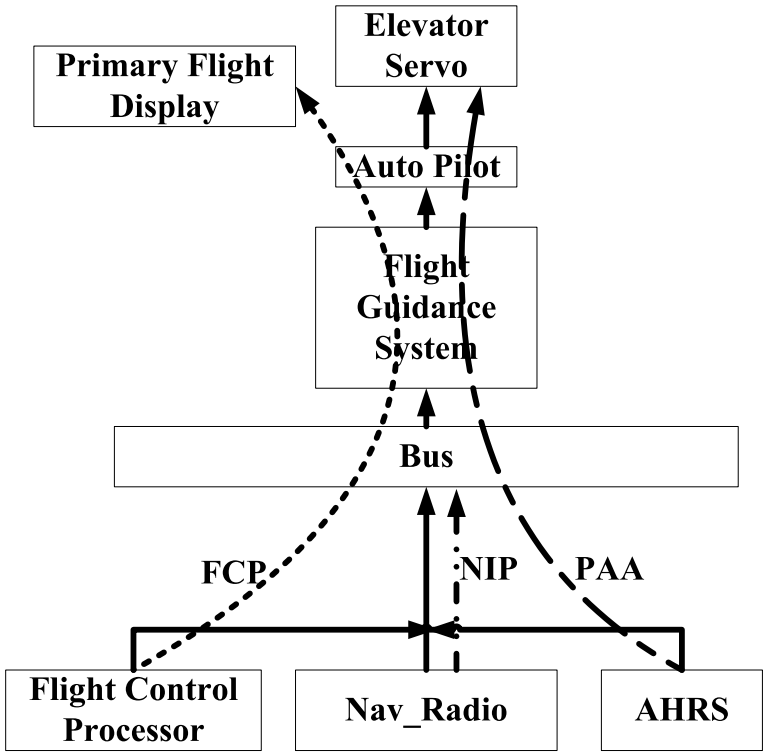


Figure 4.20. Total running time for GT workloads.



FCP: Flight Control Processing

NIP: Navigation Information Processing

PAA: Pitch Adjustment Actuation

Figure 4.21. Flight control system.

TABLE 4.10

Specification of A Fault-tolerant Distributed System

Task	Period	Subtask	Processor	Exec. Time	Task	Period	Subtask	Processor	Exec. Time
0	600	0	0	40	4	140	20	5	10
		1	0	40			21	3	20
		2	6	20			22	0	10
		3	1	20			23	7	10
		4	2	20			24	1	10
		5	4	40			25	7	10
		6	2	60			26	6	20
1	350	7	1	20			27	6	10
		8	1	20			28	1	10
		9	0	80			29	6	10
		10	4	140			30	7	10
		11	1	40			31	6	20
2	140	12	2	20			32	7	20
		13	5	20	5	200	33	3	30
		14	5	20			34	0	20
		15	3	20			35	5	20
		16	3	20			36	7	20
		17	2	20			37	5	20
3	350	18	1	10			38	2	30
		19	1	10			39	1	20
							40	4	20
							41	6	20
							42	7	20

TABLE 4.11

Data Dependencies of Subtasks in A Fault-tolerant Distributed System

Subtask	Messages Size→Succ. Subtask	Subtask	Messages Size→Succ. Subtask	Subtask	Messages Size→Succ. Subtask
0	500→1, 1500→2	14	500→15	29	
1	600→3, 700→4, 300→5	15		30	500→31
		16	500→17	31	700→32
2	200→3	17		32	
3		18	500→19	33	500→35
4	600→6	19		34	500→35
5	800→6	20	400→21	35	600→36, 600→37
6		21		36	
7	400→8	22	400→23	37	
8		23	400→24	38	500→40
9	900→11	24	200→25	39	500→40
10	2500→11	25	200→26	40	600→41, 600→42
11		26	200→27, 200→28	41	
12	1500→13, 1500→14	27	500→29	42	
13	500→15	28	300→29		

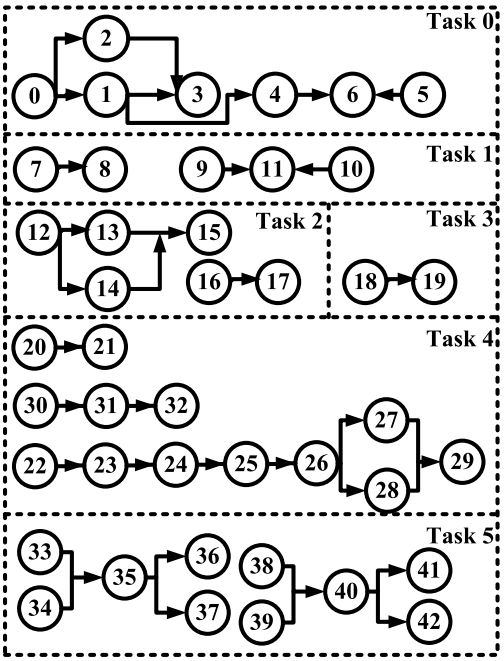


Figure 4.22. Subtasks and their dependencies for all the tasks of the fault-tolerant distributed system.

CHAPTER 5

An Adaptive Transmission Rate Control Approach to Minimize Energy Consumption

Wireless sensor network is widely used in CPS applications, such as the health care and environment monitoring (e.g. [103, 140]). Most of the wireless sensors are powered by batteries and store a limited amount of energy, hence require the transmission to be energy efficient. Lower transmission rates can greatly reduce transmission energy. However, if the lowest transmission rate is selected, many messages could miss their deadlines, which degrades the quality of service (QoS) for real-time applications. Therefore, it is important to design an efficient approach for adjusting transmission rates in order to not only achieve energy saving, but also maximize the QoS.

There are some recent publications on the transmission rate adjustment to minimize energy dissipation while still satisfying the timing requirement of real-time streams under earliest deadline first (EDF) scheduling. Some papers, [144, 155] and [53], propose optimal approaches, by assuming a given amount of data needed to be transmitted within an absolute deadline, to minimize the energy consumption and maximize the data throughput, respectively. The works of [54, 121] assume that all packets to be transmitted have a common absolute deadline. The situation considered in these papers, cannot be directly applied to handle cases where different packets have different absolute deadlines. There are works select-

ing rates for packets based on that each packet has its own absolute deadline. Some of these, e.g., [4, 37, 78], assume that the energy function for all the packets are the same, while other more general approaches, e.g., [112, 162] are proposed based on the fact that the energy functions are influenced by the fading channel state, transmission distance, and so on. All of these works assume that a packet arriving earlier always has an earlier deadline. Furthermore, some of the above approaches [4, 37, 54, 121] assume that the arrival times of future packets are known apriori precisely, and others [4, 53, 144, 155], assume that a packet transmission can be preempted at arbitrary timing during transmission, neither of which are realistic in wireless sensor networks. Moreover, the algorithms proposed by [53, 112] is extremely time consuming. In addition to the energy-aware transmission, some works have already proposed heuristics to guarantee the

In addition to the study of the rate control in wireless sensor networks, there is also much research work on dynamic voltage frequency scaling (DVFS), which is similar to the rate control in sensor nodes. Some papers, e.g., [86, 93, 148], propose CPU speed selection approaches for a set of preemptive jobs. The preemptive execution of CPU jobs does not map well to packet transmission in wireless sensor networks. The work in [74] proposes a CPU speed slow down method for periodic tasks that have maximum blocking times. However, the schedulability condition [17, 29, 101] employed in this work [74] is not only pessimistic, but also time consuming. A busy period decomposition method is proposed in [108] for a set of non-preemptive jobs based on the assumption that a job arriving earlier always has an earlier deadline. All of the proposed approaches know the release times of jobs exactly, which is unrealistic in wireless sensor networks.

In this work, we propose an on-line transmission rate selection approach based

on an optimal dynamic voltage frequency scaling algorithm Lp-EDF [148]. Our approach exploits the periodicity property of the real-time streams to predict the future jobs' timing information and find an optimal transmission rate schedule. We are designing our approach to make more messages meet their deadlines. Preliminary results show that our approach achieves a higher success ratio with a lower timing cost compared with existing works, although the energy dissipation caused by our approach sees a small increase.

5.1 System Model

We consider a system composed of a set of streams $\{S_i\} = \{S_1, S_2, ..., S_N\}$. Stream S_i periodically generates a message of C_i bytes with a period T_i. The message generated at time $R_{ij} = O_i + (j-1) \cdot T_i - J_{ij}$ is denoted as M_{ij}, where O_i is the release offset of the stream S_i and J_{ij} is the jitter of the message arrivals. We assume that the jitter satisfies a uniform distribution $J_{ij} \sim U(0, T_i)$. Each message M_{ij} has a relative deadline D_i, and its absolute deadline $AD_{ij} = O_i + (j-1) \cdot T_i + D_i - J_{ij}$. According to [55], the fragmentation threshold $Threshold$ is the maximum non-preemption length of each message. Each message M_{ij} is fragmented to $X_i = \lceil \frac{C_i}{Threshold} \rceil$ packets, and the first $X_i - 1$ packets and the last packet have the length $Threshold$ and $C_i - (X_i - 1) \cdot Threshold$, respectively. Similar to 802.11a [55], before a node transmits a packet, it needs $Overhead\_Time$ time to transmit the preamble and the "Signal" part of the PLCP header. Then, the "Service" part of the PLCP header, with its size $PLCP\_Length$, will be transmitted with the payload. Based on these definitions, we define the intensity

$g(I)$ of a time interval $I = [t, t']$ to be,

$$g(I) = \frac{\sum_{i=1}^{N} Y_i \cdot (C_i + X_i \cdot PLCP\_Length)}{t' - t - \sum_{i=1}^{N} Y_i \cdot X_i \cdot Overhead\_Time}, \tag{5.1}$$

where Y_i is the number of messages with $[R_{ij}, AD_{ij}] \subseteq I$.

We consider a single wireless sensor node which has transmission rate r, which can take on any value in $[min\_rate, max\_rate]$, where min_rate and max_rate are the minimum and maximum allowed rates for the node, respectively. The wireless node handles the given set of streams $\{S_i\}$. The transmission power P is a convex function of the transmission rate according to [121]. In this work, we assume that the transmission power function is,

$$P(r(t)) = max\_rate \cdot Noise \cdot L^2(2^{\frac{2 \cdot r(t)}{max\_rate}} - 1), \tag{5.2}$$

where L is the transmission distance and $Noise$ is the noise power according to [162]. The packets of the streams are stored in a buffer, whose size is $MaxSize$.

We refer to $[R_{ij}, AD_{ij}]$ as the active interval of the message M_{ij}. A schedule $S(t_0, t_1)$ is a pair of $(r(t), Message(t))$ functions defined over the given time interval $[t_0, t_1]$, where $Message(t)$ defines the message being transmitted at time t with rate r(t) (or idle if $r(t) = 0$). The total energy consumed during a given time interval $[t_0, t_1]$ is

$$E(S) = \int_{t_0}^{t_1} P(r(t))dt. \tag{5.3}$$

The goal of our scheduling problem is to find a feasible schedule that minimizes the transmission energy, while the following constraint is satisfied for any message

whose interval is within the time interval $[t_0, t_1]$,

$$\int_{R_{ij}}^{AD_{ij}} r(t)\delta(Message(t), M_{ij})dt = C_i, \qquad \forall M_{ij}, [R_{ij}, D_{ij}] \subseteq [t_0, t_1], \tag{5.4}$$

where $\delta(Message(t), M_{ij}) = 1$ if $Message(t) = M_{ij}$ and 0 otherwise. To make all the messages reach their destinations within their deadlines, the constraint (5.4) should be satisfied for any message. However, in wireless sensor networks, the jitters of the message arrivals, the non-preemption property of a packet, and the dynamic interference in the transmission environment will cause some of the messages miss their deadlines inevitably. We define the message success ratio SR within a given time interval to be

$$SR = \frac{Num\_Message\_Sucess}{Num\_Message}, \tag{5.5}$$

where $Num\_Message$ is the number of transmitted messages within the time interval, while $Num\_Message\_Sucess$ is the number of messages successfully delivered within the deadlines. Message success ratio SR represents the percentage of the messages that satisfy constraint (5.4) within a given time interval if we assume that the impact of interference has been adequately handled by the rate assignment.

Our problem is to find rates for the messages to be transmitted within a given time interval, in order to not only minimize the transmission energy (5.3), but also to make as many messages as possible to satisfy the schedulability constraint (5.4). To accomplish this, we need an adaptive rate control approach that is able to adjust the rates efficiently in response to the jitters of the message arrivals, and effectively reduce the influence of the packet non-preemption property on the

success ratio. Specifically, when a new message arrives at the buffer at time t_0, we have

$$\min_{r(t)} \int_{t_0}^{t_1} P(r(t))dt \tag{5.6}$$

$$\text{s.t.} \int_{R_{ij}}^{AD_{ij}} r(t)\delta(Message(t), M_{ij})dt = C_i, \qquad \forall M_{ij}, [R_{ij}, D_{ij}] \subseteq [t_0, t_1], \tag{5.7}$$

where t_0 is the current time, i.e., the release time of the new message, and t_1 is set to be $\max_{M_{ij}|R_{ij} \leq t_0}\{AD_{ij}\}$.

We use EDF scheduling algorithm since it is optimal in scheduling a set of periodic streams on a single sensor node. An optimal minimum energy scheduler Lp-EDF is proposed in [148] under preemptive EDF scheduling. Though Lp-EDF was originally for scheduling CPU tasks, it can be modified to schedule messages in wireless sensor networks, where messages are fragmented to non-preemptive packets and message arrival times are not known precisely.

5.2 Our Approach

Our problem is to find message transmission rates within a given time interval such that the transmission energy is minimized and the transmission success ratio is as high as possible. We solve the problem by an on-line approach so as to better respond to dynamic variations including arrival time jitters, failed delivery, etc. An outline of our approach is as follows. Every time a new message arrives at the buffer, the sensor node will compute the rates for the messages in the buffer by solving an optimization problem as given in (5.6) and (5.7). If a packet is being transmitted upon the arrival of a new message, the node will finish the transmission of this packet before computing a new schedule. To make the on-line approach work well, we need to address issues including prediction of future load

and solving the resulting optimization problem. With the predicted packets and the packets already in the buffer, solving the optimization problem defined in (5.6) and (5.7) can employ the Lp-EDF algorithm introduced in [148].

Since future messages may compete for the time resource with the messages already in the buffer and greatly influence the transmission success ratio and the energy consumption, it is necessary to predict the future messages that release within some time window. We define the time interval $[t_0, \max_{M_{ij}|R_{ij}\leq t_0}\{AD_{ij}\}]$ as the scheduling window W at time t_0. When computing the transmission rates at time t_0, the node considers not only the messages M_{ij} already in the buffer, but also the future messages M_{km} whose release time R_{km} is larger than t_0 but smaller than $\max_{M_{ij}|R_{ij}\leq t_0}\{AD_{ij}\}$.

We consider two ways to predict the future messages. The first one is called **proportional prediction** (Lp-EDF-p). If the absolute deadline AD_{km} of a predicted message M_{km} is within the window W, we include message as is. However, if $AD_{km} > \max_{M_{ij}|R_{ij}\leq t_0}\{AD_{ij}\}$, we modify the message size and deadline to C'_k and AD'_{km}, where $C'_k = C_k \cdot \frac{\max_{M_{ij}|R_{ij}\leq t_0}\{AD_{ij}\}-R_{km}}{AD_{km}-R_{km}}$ and $AD'_{km} = \max_{M_{ij}|R_{ij}\leq t_0}\{AD_{ij}\}$, respectively. The packet number X'_k under this schedule computation is equal to $\lceil \frac{C'_k}{Threshold} \rceil$. The other way is to treat the future message M_{km} with AD_{km} beyond the window W by the same way as treating the messages whose absolute deadlines are within the window W, which is called **complete prediction** (Lp-EDF-c). Any future message M_{km} with its release time within the window W has the message size C_k and absolute deadline AD_{km}.

The improved Lp-EDF algorithm identifies a critical interval $I^* = [t', t'']$ whose intensity $g(I^*)$ is maximum within the time interval $[t_0, t_1]$. We incorporate the timing overhead of the packet transmission, such as the preamble and the PLCP

header, into the intensity computation as shown in (5.1). Then, the rates r of the messages are set to be $g(I^*)$ if their release times and absolute deadlines are within the critical interval I^*, and these messages are deleted from the message set. This process is repeated until all the messages obtain their rates.

5.3 Evaluation

We evaluate the performance and efficiency of our proposed on-line approach on randomly generated stream sets and compare the modified Lp-EDF approach, both Lp-EDF-p and Lp-EDF-c, with the original Lp-EDF and ZM algorithm in [4]. Our modified Lp-EDF algorithm was implemented in C++, running on an AMD Phenom(tm) II X4 940 workstation with Red Hat Enterprise Linux 4. 1000 stream sets consisting of 5 streams each were randomly generated for 9 different bandwidth levels ($Bandwidth_{level} = 0.1Mbps, ..., 0.9Mbps$) with a total of 9000 stream sets. The bandwidth level is defined to be $Bandwidth_{level_i} = \sum_{j=1}^{5} \frac{C_j}{T_j}, i = 0.1, ..., 0.9$. We employ IEEE 802.11a [55] as the MAC protocol, which has the minimum and maximum rates as $6Mbps$ and $54Mbps$, respectively. To fully show different performance of the stream sets at different bandwidth levels in 802.11a, we multiply the message length of each stream by 54 times, and change the bandwidth levels to be $5.4Mbps, ..., 48.6Mbps$. In addition, the fragmentation threshold $Threshold$, the overhead time $Overhead\_Time$ and the length of PLCP header $PLCP\_length$ are set to be 2346 bytes, 40 μs, and 2 bytes, respectively. We assume that the stream S_i with the highest density $\frac{C_i}{D_i}$ in a stream set releases its messages with jitter J_{ij}, which satisfies a uniform distribution $J_{ij} \sim U(0, T_i)$. For the details of the stream set generation, readers can refer to [34].

Although Lp-EDF is optimal under preemptive EDF, its computation time is

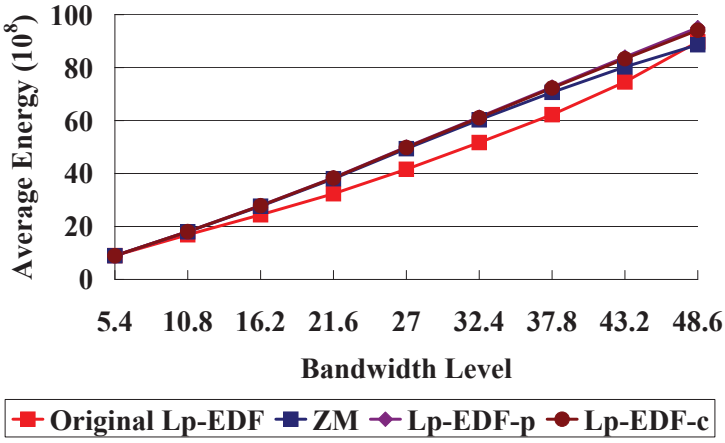


Figure 5.1. Comparison of original Lp-EDF, ZM, Lp-EDF-p and Lp-EDF-c in terms of average energy consumption.

extremely long because of its time complexity $O(NUM^3)$ for NUM messages. In addition, the performance of Lp-EDF is negatively influenced by the jitters of the messages and the packet non-preemption property.

Another energy minimization scheduling algorithm, denoted as ZM, presented in [4], dynamically computes the lowest rate for the messages to minimize the energy consumption. There are two disadvantages of ZM. First, ZM is not optimal when a message arriving later has an earlier deadline. Second, ZM suffers less but still obviously from the jitters of message arrivals and packet non-preemption property.

In the first experiment, we compare energy consumption resulted from applying the original Lp-EDF, ZM and our approach, as shown in Figure 5.1. The x-axis represents the bandwidth level, while the y-axis represents the average energy consumption per stream set. It is illustrated that the original Lp-EDF performs a little better than our approach and ZM in energy saving, while our approach

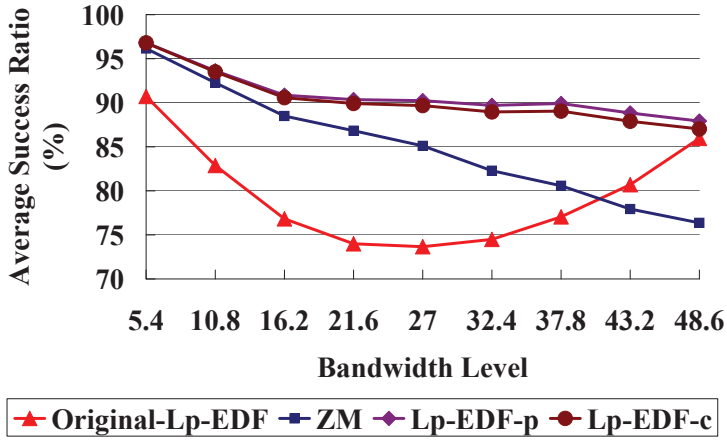


Figure 5.2. Comparison of original Lp-EDF, ZM, Lp-EDF-p and Lp-EDF-c in terms of average success ratio.

achieves the energy saving comparable with that of ZM at all bandwidth levels.

The second experiment shows the average success ratio obtained by the original Lp-EDF, ZM, and our approach in Figure 5.2. The x-axis shows the bandwidth level, whereas the y-axis represents the average success ratio, which is the percentage of the successfully transmitted messages among the 1000 stream sets at each bandwidth level. First, the average success ratios resulted from the original Lp-EDF are much lower than the results by our approach at bandwidth levels $5.4Mbps$ to $43.2Mbps$, because the original Lp-EDF statically computes all the rates for the whole time interval under preemptive EDF and neglects the message jitters. Second, for bandwidth levels greater than $16.2Mbps$, the performance of ZM degrades drastically, because ZM cannot handle the case that a message arriving later has an earlier deadline, which appears frequently when bandwidth levels become higher.

To further compare the performance of different approaches, the minimum suc-

cess ratio among the 1000 stream sets at each bandwidth level is shown in Figure 5.3. The x-axis represents the bandwidth level, while the y-axis represents the minimum success ratio. For the bandwidth levels less than or equal to $16.2Mbps$, the minimum success ratios by ZM are a little lower than those obtained by our approach, while for bandwidth levels greater than $16.2Mbps$, the minimum success ratios resulted by ZM are much lower than those obtained by our approach. In contrast, for bandwidth levels $21.6Mbps$ to $48.6Mbps$, the minimum success ratios by the original Lp-EDF are a little higher than those by our approach.

We study the average computation time of a stream set by our approach and compare them with the original Lp-EDF and ZM in the third experiment, as shown in Figure 5.4. The x-axis represents the bandwidth level, while the y-axis represents the average rate computation time. As shown in Figure 5.4, our method runs 200 to 4000 times faster than the original Lp-EDF, while it is 10 to 36 times slower than the ZM algorithm.

Based on the preliminary results, our approach can achieve much higher success ratios on average than the original Lp-EDF and ZM, though the energy dissipation by applying our approach sees a small increase compared with the original Lp-EDF. Furthermore, the computational cost of our approach is significantly smaller than that of the original Lp-EDF. ZM takes less time than our approach to compute the rates, but its success ratios are not satisfactory at high bandwidth levels.

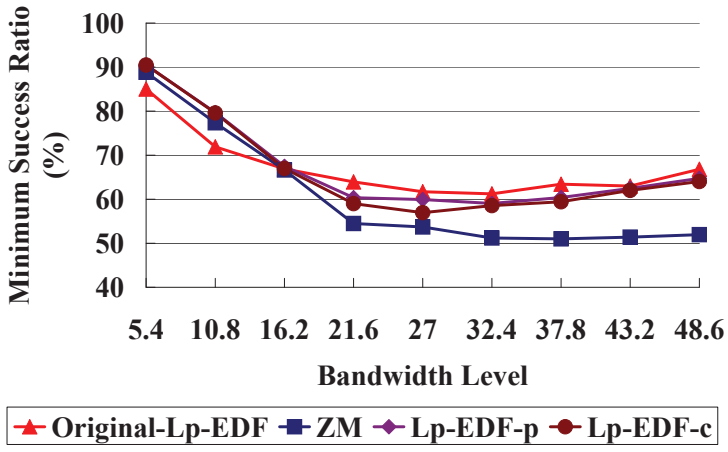


Figure 5.3. Comparison of original Lp-EDF, ZM, Lp-EDF-p and Lp-EDF-c in terms of minimum success ratio.

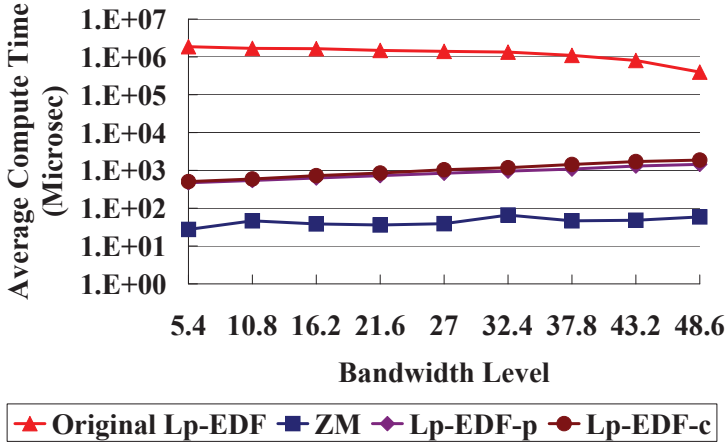


Figure 5.4. Comparison of original Lp-EDF, ZM, Lp-EDF-p and Lp-EDF-c in terms of computational cost.

CHAPTER 6

Data Link Layer Scheduling in Dynamic Wireless Networked Control Systems with Minimum On-line Schedule Update

WNCSs have received tremendous research interests recently due to their great advantages in easier deployment, enhanced mobility, and reduced maintenance cost. A key design challenge in such systems is to design efficient data link layer scheduling algorithms to achieve deterministic end-to-end real-time communication. Previous research works have a common assumption that the WNCS is static and the network communication schedule, once constructed and distributed, will stay unchanged. This assumption, however does not hold in many real-world system setup. In this work, we propose the data link layer scheduling problem in dynamic WNCSs. We employ a rhythmic task in adaptive to external disturbances and introduce an effective approach to adjust existing schedule for all the nodes in the WNCS when the disturbances happen. The approach determines the time duration of dynamic schedule and generates the schedule for that time duration to minimize the impact of network dynamics on existing network flows with bounded overheads. The effectiveness and efficiency of the proposed algorithm are validated through sufficient experimental results. Simulation results based on randomly generated task sets indicate that the proposed approach outperforms existing work both in terms of the number of feasible task sets (between 122%

and 128% on average) and the number of feasible periodic packets (between 263% and 321% on average).

6.1 Introduction

WNCSs have received significant attention over the past several decades [58, 67, 111, 113, 117] because WNCSs are widely used in many areas such as telerobotics [44], aircraft control [150], civil infrastructure monitoring [77], medication service [63] and power management [56]. In a WNCS, sensors, controllers, actuators and other relay nodes are geographically distributed and connected over wireless network media. A task in a WNCS generally delivers measurements from a sensor to the gateway and sends control signals from the gateway to an actuator within an end-to-end deadline. In the real-world, the performance of WNCSs is usually degraded by various physical events, such as the failures of critical civil infrastructures and malicious attacks. In response to external events, the works [28, 81, 92] employ approaches of adjusting the sampling rates of tasks on-line in control systems. For example, the authors in [81] proposed a rhythmic task model, which continuously changes the period and relative deadline of one specific task in mechanical subsystems. However, such approaches cannot be applied to WNCSs straightforwardly because they do not consider the stringent time requirement of end-to-end packet delivery in the wireless network media.

The wireless network media is usually unreliable for packet delivery due to the intermittent connection, shifting signal strength, power outage and etc. Missing or delaying the sensing and control packets in a WNCS may severely degrade the performance of system or even damage the system. To address these problems, two prevalent international standards, ISA 100 [2] and WirelessHART [36] are de-

veloped to guarantee the timely delivery of packets in WNCSs. In both standards, a centralized network architecture is adopted for network resource allocation and data management. Specifically, a gateway, sensors, actuators and relays nodes in the WNCS are connected through wireless network technology and a network manager residing in the gateway allocates network resource. Both standards are based on Time Division Multiple Access (TDMA)-based data link layer, which cannot handle the situation that the sampling rates and deadlines of tasks are adjusted promptly.

To achieve deterministic end-to-end real-time communication in WNCSs, extensive research works have been devoted on the design of data link layer scheduling algorithms. The algorithms based on TDMA are presented in works [51, 60, 126, 127, 138, 139, 146, 156]. The work [146] designed and implemented a real-time high-speed data link layer protocol called RT-WiFi. The authors in [51, 60, 139] utilizes Rate Monotonic (RM) algorithm [29] to generate data link layer communication schedules. In [126], the authors proposed an optimal branch and bound scheduling algorithm and a practical heuristic to assign time slots to transmissions of packets. The work in [127] designed an optimal algorithm and an efficient heuristic for priority assignment of the periodic flows, based on which data link layer schedules can be generated under fixed priority. The works in [138] and [156] studied the joint data layer link scheduling and channel assignment problem for convergecast in different network topologies. All the aforementioned works assume that the network is static after an initial configuration phase, which are not suitable for handling on-line workload changes in the WNCS.

There are also some centralized data link layer approaches [41–43, 46, 133, 134] that can adapt to the network dynamics. The work [46] and [134] proposed

wireless MAC protocols in adaptive to network dynamic dynamics such as a packet loss, a node failure and route changes. The work [133] proposed an approach to dynamically select and switch MAC protocols in response to changes in ambient conditions and application requirements. The protocols proposed in papers [41–43] can dynamically update the transmission schedule in response to workload changes. The proposed protocols in [46, 134] are unable to respond to on-line workload change while the paper in [133] assumes that only a fixed number of MACs are stored in the system. The algorithms proposed in the papers [41–43] are only suitable for the data aggregation in a tree-like network, which does not support the system containing multiple tasks ending at different actuators.

In this work, we consider a WNCS adopting a centralized network architecture, which utilizes a static schedule when there is no physical disturbances in the WNCS. The WNCS contains a set of periodic tasks and a rhythmic task, where the period and relative deadline of rhythmic task are reduced suddenly and then return to their nominal values gradually. Since rhythmic task is critical for the WNCS to respond to physical disturbances, all the packets of rhythmic task must meet their deadlines. In contrast, packets of periodic tasks are allowed to be dropped to give up some bandwidth to rhythmic task. After rhythmic task returns to its nominal state, the WNCS is required to reuse the static schedule immediately after a specific time slot (called switch point) in order to reduce the overhead of packet transmissions and prepare for any future external event. Therefore, there exists a transient time duration when the WNCS cannot use the static schedule. During such a transient duration, there may be a high bandwidth competition in the WNCS, which may result in the end-to-end deadline misses of periodic packets. Although the WNCS allows periodic packets to miss their end-to-end

deadlines, frequent deadline misses can degrade the Quality of Service (QoS) of the system. Therefore, it is critical to determine a transient time duration and design an on-line data link scheduling algorithm to make periodic packets meet deadlines as many as possible.

This work designs an on-line data link layer scheduling problem determining a transient time duration and a dynamic schedule in order to minimize the number of dropped periodic packets. In the problem, we propose various practical constraints of making the problem usable for real-world applications, e.g., the upper bound on switch point, the time overhead limit for solution search process, and the schedule update overhead limit for the dynamic schedule. To solve the problem, we propose an on-line approach to determine a transient time duration and construct a dynamic schedule for this time duration with bounded time and schedule update overheads. The approach is composed of an efficient framework and an effective heuristic to be called by the framework. The framework determines possible time durations by judiciously choosing switch point candidates, adopts an algorithm to generate a dynamic schedule for each possible time duration, and reduces the schedule update overhead of constructed schedule. Our proposed heuristic is a modified dynamic programming algorithm with a pseudo polynomial time complexity, which generates a dynamic schedule minimizing the number of dropped periodic packets. The effectiveness and efficiency of the proposed approach are validated through thorough extensive experimental results.

6.2 System Model

This work adopts the system architecture of typical WNCSs which consist of one gateway, a set of sensors, a set of actuators, and a set of intermediate nodes.

A sensor or an actuator can also serve as a relay node. A sensor samples physical measurements and sends them to the gateway through multiple relay nodes. The gateway extracts the sampled data, executes some given control algorithm, encapsulates the actuation data to packets and sends them to specific actuators through multiple relay nodes. All these devices collectively form a node set denoted as V= $\{V_0, V_1, V_2, \cdots, V_g\}$, where V_g represents the gateway. A direct link $(V_j, V_{j'})$ exists if and only if V_j can send data to $V_{j'}$ reliably. This work assumes that there is a single communication channel shared by all the data link layer communication in the network and our future work will remove this assumption. For the reader's convenience, we summarize notations of system model in Table 6.1 [1].

A set of tasks $\mathcal{T} = \{\tau_0, \tau_1, \tau_2, \cdots \tau_n, \tau_{n+1}\}$ are running on the node set $\mathcal{V}$ in the WNCS. Task τ_{n+1} is a *broadcast task* which delivers the schedule update from the gateway to all the nodes in the WNCS. (Such a concept has been used in many other link layer scheduling frameworks, e.g., [1, 2, 24, 90, 152].) We will introduce broadcast task in details in Section 6.3. Task τ_0 is a *rhythmic task* while any task τ_i $(1 \leq i \leq n)$ is a *periodic task*. A rhythmic or periodic task generally starts from a sensor, delivers the sensor measurements to the gateway and finally sends the control signals to an actuator. Each task τ_i is associated with a period P_i, a relative deadline D_i and a hop number H_i. Hop number H_i is the number of hops along task τ_i's routing path.

A rhythmic task can be in one of two states. When τ_0 is in the *nominal state*, it has a constant period, *i.e.*, P_0, and a constant relative deadline, D_0. When τ_0 is in the *rhythmic state* (*i.e.*, needs to respond to disturbances), its period and relative deadline are reduced abruptly and then return to their nominal values

[1] Unless specified, task identifier i satisfies $0 \leq i \leq n+1$ in this table.

TABLE 6.1

Summary of Notations Used for System Model

Parameter	Definition
V_g, V_i, $0 \leq i < g$	Gateway, sensors, actuators and other nodes
$\chi_{i,k}$	Rhythmic, periodic and broadcast packet $\chi_{i,k}$
τ_0, τ_{n+1}	Rhythmic task and broadcast task
$r_{i,k}$, $d_{i,k}$, $0 \leq i \leq n$	Release time and deadline of $\chi_{i,k}$
τ_i, $1 \leq i \leq n$	Periodic task
$\bar{r}_{0,k}$, $\bar{d}_{0,k}$	Nominal release time and nominal deadline of $\chi_{0,k}$
H_i	Hop number of τ_i
$\chi_{i,k}(h)$	Rhythmic, periodic and broadcast transmission
P_i, D_i	Period and relative deadline of τ_i
$r_{i,k}(h)$, $d_{i,k}(h)$ $f_{i,k}(h)$, $1 \leq i \leq n$	Release time, deadline and finish time of rhythmic or periodic transmission $\chi_{i,k}(h)$
$\vec{P}_0$, $\vec{D}_0$	Vectors of periods and relative deadlines
$t_{n\to r}$, $t_{r\to n}$	Slot when τ_0 leaves the nominal state and the rhythmic state, respectively

by following some monotonically non-decreasing functions. (The actual functions depend on the environmental disturbance that the system suffers.) We use vectors $\vec{P}_0 = [P_{0,x}, x = 1, \ldots, \mathcal{R}]^T$ and $\vec{D}_0 = [D_{0,x}, x = 1, \ldots, \mathcal{R}]^T$ to represent the periods and relative deadlines of τ_0 when it is in the rhythmic state. (To be more precise, $P_{0,x}$ is the time duration between two consecutive releases but we still call it period to simplify our notation.) The periods $P_{0,x}$'s and deadlines $D_{0,x}$'s satisfy $P_{0,x} \leq P_{0,x+1} \leq P_0$ and $D_{0,x} \leq D_{0,x+1} \leq D_0$, respectively. Since the rhythmic task enters the rhythmic state, the period and relative deadline of τ_0 follow the elements of $\vec{P}_0$ and $\vec{D}_0$ in sequence, respectively. Task τ_0 returns to the nominal

state when the $(R+1)$-th rhythmic packet is released.

TABLE 6.2

An Example WNCS with 4 Tasks Running on 8 Nodes

Task	Routing Path	Period	Relative Deadline	Period Vector	Deadline Vector
τ_0	$V_0 \rightarrow V_7 \rightarrow V_4$	9	6	$[3,6]^T$	$[2,5]^T$
τ_1	$V_1 \rightarrow V_7 \rightarrow V_3 \rightarrow V_5$	9	7	N/A	N/A
τ_2	$V_2 \rightarrow V_7 \rightarrow V_6$	9	9	N/A	N/A
τ_3	$V_7 \rightarrow *^2$	9	N/A	N/A	9

In the WNCS, a rhythmic or periodic task τ_i $(0 \leq i \leq n)$ follows a designated single routing path, consisting of $H_i + 1$ number of specific nodes that τ_i needs to traverse. Rhythmic or periodic task τ_i transmits the sensor measurements from a sensor to the gateway and then transmits the control signals from the gateway to an actuator. (Here we assume that such transmissions are always successful and leave packet loss recovery mechanisms like FEC and retransmission for future work.) Since this work concerns data link layer scheduling, we ignore the computation overhead in the gateway and only focus on the communication aspect of rhythmic and periodic tasks. Broadcast task τ_{n+1} has a fixed tree-like routing path, starting from V_g and ending at each sensor or actuator in the WNCS. The

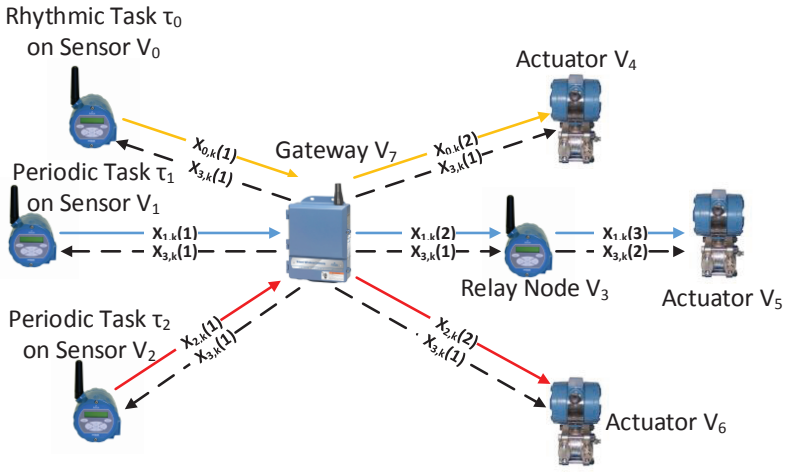


Figure 6.1. Topology of An Example WNCS with 4 Tasks Running on 8 Nodes.

routing path is constructed by using Breadth-First-Search (BFS) Algorithm [45]. Figure 6.1 gives an example WNCS which contains 4 tasks, *i.e.*, τ_0, τ_1, τ_2 and τ_3 running on 8 nodes, *i.e.*, V_i, $i = 0, \ldots, 7$, where V_0, V_1 and V_2 are sensors, V_4, V_5, V_6 are actuators, V_3 is a relay node and V_7 is the gateway. Task τ_0 is a rhythmic task, tasks τ_1 and τ_2 are periodic tasks, and task τ_3 is a broadcast task. The routing paths of all the tasks are shown in the second column of Table 6.2 .

An instance of a task is referred to as a packet, *i.e.*, packet $\chi_{i,k}$ corresponds to the k-th instance of τ_i. An instance of a periodic task is called a *periodic packet*, an instance of the rhythmic task is called a *rhythmic packet* and an instance of broadcast task is called a *broadcast packet*. Periodic packet $\chi_{i,k}$ is associated with a release time $r_{i,k}$ (equal to $(k-1) \cdot P_i$) and an absolute deadline $d_{i,k}$ (equal to $(k-1) \cdot P_i + D_i$). Rhythmic task τ_0 is associated with actual release time $r_{0,k}$

[2] τ_3 is a broadcast packet, which has 2 hops. The 1-st hop is from V_7 to V_0, V_1, V_2, V_3, V_4, V_6, and the 2-nd hop is from V_3 to V_5.

and actual deadline $d_{0,k}$. In this work, we assume that $d_{i,k} \leq r_{i,k+1}$ is satisfied no matter whether $\chi_{i,k}$ is a rhythmic or periodic packet.

We assume that τ_0 is in the nominal state initially. Let τ_0 switches from the nominal state to the rhythmic state at $r_{0,m+1}$ (denoted as $t_{n\to r}$), and returns to the nominal state from the rhythmic state at $r_{0,m+\mathcal{R}+1}$ (denoted as $t_{r\to n}$). Then, τ_0 stays in the rhythmic state within interval $[t_{n\to r}+1, t_{r\to n}]$, where $t_{n\to r}$ and $t_{r\to n}$ satisfy

$$t_{r\to n} = t_{n\to r} + \sum_{x=1}^{\mathcal{R}} P_{0,x}. \tag{6.1}$$

Assume that the first rhythmic packet $\chi_{0,1}$ is released at time 0. The actual release time $r_{0,k+1}$ is determined based on the state of τ_0 when $r_{0,k}$ is released, *i.e.*,

$$r_{0,k+1} = \begin{cases} r_{0,k} + P_0 & \text{if } k \leq m \text{ or } k > m+R \\ r_{0,k} + P_{0,k-m} & \text{if } m < k \leq m+R \end{cases}. \tag{6.2}$$

Similarly, $d_{0,k}$ of $\chi_{0,k}$ is equal to

$$d_{0,k+1} = \begin{cases} r_{0,k+1} + D_0 & \text{if } k \leq m-1 \text{ or } k \geq m+R \\ r_{0,k+1} + D_{0,k+1-m} & \text{if } m \leq k < m+R \end{cases}. \tag{6.3}$$

To differentiate from the actual release time $r_{0,k}$ and actual deadline $d_{0,k}$, we define nominal release time $\bar{r}_{0,k}$ and nominal deadline $\bar{d}_{0,k}$ to be the release time and deadline of $\chi_{0,k}$ when τ_0 has never entered the rhythmic state. To simplify the paper, we use release time and deadline to refer to actual release time and actual deadline in the rest of the paper unless we need to differ actual and nominal release times (deadlines). Figure 6.2 shows the release times and deadlines of rhythmic packets when rhythmic task τ_0 is in the different states. The top part of

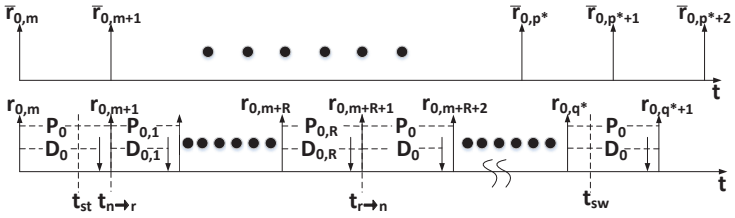


Figure 6.2. Release times and deadlines of rhythmic packets when rhythmic task τ_0 is in the different states. Rhythmic task τ_0 enters the rhythmic state from the nominal state at time slot $t_{n\to r}$ and returns to the nominal state from the rhythmic state at time slot $t_{r\to n}$.

the figure shows the nominal release times of rhythmic packets while the bottom part of the figure shows the actual release times and actual deadlines of rhythmic packets.

Following WirelessHART and ISA100.11a standards, we apply a TDMA protocol for data link layer communication. At any given time, a node in the WNCS always follows a schedule to transmit or receive a packet under the TDMA protocol. Specifically, at the h-th hop along the routing path of rhythmic or periodic task τ_i, the sender sends packet $\chi_{n+1,k}$ to the receiver and receives an acknowledgement packet from the receiver. In contrast, at the h-th hop of broadcast task τ_{n+1}, the sender sends packet to one or multiple receivers by following the tree-like routing path. Such a process is referred to as a transmission denoted by $\chi_{i,k}(h)$, $h = 1, ..., H_i$, and a transmission must be completed within one time slot. Similar to the definition of packet, a transmission can be either rhythmic, periodic and broadcast. Due to the sequential dependence among the transmissions of a packet, we say $\chi_{i,k}(h)$ is a (or an immediate) successor of $\chi_{i,k}(h')$ and $\chi_{i,k}(h')$ is a (or an immediate) predecessor of $\chi_{i,k}(h)$ if $h \geq h' + 1$ (or $h = h' + 1$). As an

example, in Figure 6.1, each task τ_i has one packet $\chi_{i,k}$ which is composed of a series of transmissions $\chi_{i,k}(h)$'s. Transmission $\chi_{i,k}(h)$ is associated with a release time $r_{i,k}(h)$, an absolute deadline $d_{i,k}(h)$ and a finish time $f_{i,k}(h)$. Finish time $f_{i,k}(h)$ represents the time slot when $\chi_{i,k}$ completes the $(h+1)$-th hop. In this work, $r_{n+1,k}(h)$, $d_{n+1,k}(h)$ and $f_{n+1,k}(h)$ are designed off-line. However, $r_{i,k}(h)$, $d_{i,k}(h)$ and $f_{i,k}(h)$ of rhythmic or periodic packet need to be determined on-line.

6.3 Problem statement

Given the fact that most control systems must deal with unexpected disturbances, the rhythmic task model provides a general approach for control systems to respond to such disturbances. However, as pointed out earlier, existing work in data link layer scheduling for WNCS mainly focuses on static scheduling, which would unfortunately either require excessive amount of network resource to satisfy timing requirements or result in poor control performance in the WNCS with rhythmic tasks. Dynamic data link layer scheduling could be much more responsive, but its success critically depends on the efficiency of deriving and delivering such schedules, which is the focus of our work. Below, we describe in more detail the specific problem we aim to solve.

Initially, the system starts by following $\bar{S}$. A static schedule $\bar{S}$ is designed off-line and is only used when each rhythmic packet is released at its nominal release time. In the static schedule, all the rhythmic and periodic packets can meet their nominal deadlines. A static schedule is defined as $\bar{S} = \{(t, i, h)\}$, where t is the time slot identifier, i is a task identifier and h is a hop index. If $i = -1$, no task uses the time slot and we say the slot is idle. Given any time slot t, we have $\bar{S}[t] = (i, h)$. If the time slot is idle, we have $\bar{S}[t] = (-1, -1)$. The length $\mathcal{L}$

of static schedule $\bar{S}$ is the least common multiple of the task periods (assuming τ_0 is in the nominal state) and the schedule is followed in a cyclic fashion, *i.e.*, $\bar{S}[t] = \bar{S}[t + \mathcal{L}]$. A node only stores the information on when it should transmit or receive a packet in the static schedule $\bar{S}$, and hence the WNCS desires to use static schedule $\bar{S}$ to reduce the overhead of transmissions. (The design of static schedule is not in the scope of this work and can be constructed using the approaches presented in [60] and [126].)

When $\chi_{0,m}$ reaches the gateway V_g at time slot $f_{0,m}(h)$ and V_g determines the state switch of τ_0 at $t_{n\to r}$, V_g generates a dynamic schedule S. Before the time slot update information reaches all the nodes in the WNCS, the WNCS still follows $\bar{S}$ to guarantee the coordination among all the nodes in the WNCS. To reduce the communication overhead, the WNCS expects to employ $\bar{S}$ again at a certain time slot after $t_{r\to n}$. Therefore, V_g is responsible for generating a dynamic schedule S which all the nodes will follow during a specific time interval $[t_{st}, t_{sw}]$, where t_{st} and t_{sw} are the start point and switch point of S, respectively. We choose $f_{n+1,m}(H_{n+1}) + 1$ as the start point t_{st} of the dynamic schedule when τ_0 enters the rhythmic state immediately after $r_{0,m+1}$. This is because $f_{n+1,m}(H_{n+1})$ is the earliest time when all the nodes in the WNCS know the dynamic schedule. To guarantee the WNCS to use S immediately after $t_{n\to r}$, $\bar{S}$ is designed to satisfy $f_{n+1,m}(H_{n+1}) \leq t_{n\to r}$. Dynamic schedule $S[t_{st}, t_{sw}] = \{(t, i, h)\}$ stores information on the assignment of each time slot within time interval $[t_{st}, t_{sw}]$. If t is assigned to packet $\chi_{i,k}(h)$, $S[t] = (i, h)$ is satisfied. If t is set to idle in S, $S[t] = (-1, -1)$ is satisfied.

When V_g generates a dynamic schedule S, V_g piggybacks the information of S to the nearest-in-time broadcast packet $\chi_{n+1,k}$. Without loss of generality

(WLOG), the instance index k of such a nearest-in-time broadcast packet $\chi_{n+1,k}$ is set to the instance index m of rhythmic packet $\chi_{0,m}$ which just arrives at V_g. Then, $\chi_{n+1,m}$ is transmitted to all the nodes in the WNCS along the broadcast routing tree by following $\bar{S}$. Since it is unpredictable when τ_0 enters the rhythmic state, $\bar{S}$ pre-allocates slots to τ_{n+1} periodically by setting $P_{n+1} = P_0$. In this work, we require $D_{n+1} = P_0$ to be satisfied in $\bar{S}$ to make $\chi_{n+1,m}$ be broadcast to all the nodes in the WNCS by $t_{n\to r}$ such that the WNCS can use S when τ_0 enters the rhythmic state. Meanwhile, transmission $\chi_{n+1,m}(1)$ is assigned a time slot $f_{n+1,m}(1)$ as late as possible in $\bar{S}$ to guarantee V_g to have enough time to construct S.

Since transmitting S is done through piggybacking S to a broadcast packet, only the information about those slots whose assignments (*i.e.*, task identifier and hop index) have been changed from $\bar{S}$ is required to be broadcast to all the nodes in the WNCS. A slot t whose assignment is different between $S[t]$ and $\bar{S}[t]$ is called an updated slot. Let Δ^u and $S[t_{st}, t_{sw}].\delta$ represent the maximum allowed and actual number of updated slots in $S[t_{st}, t_{sw}]$, respectively. To further constrain $S[t_{st}, t_{sw}].\delta$, t is not regarded as an updated time slot when t is idle, *i.e.*, $S[t] = (-1, -1)$. Hence, a time slot is an updated slot if and only if it satisfies

$$S[t] = (i, h) \neq (-1, -1) \text{ and } \bar{S}[t] \neq (i, h). \tag{6.4}$$

Since the gateway determines a dynamic schedule S within time interval $[t_{st}, t_{sw}]$, the gateway only needs to schedule a set of packets, $\Omega(t_{sw})$. Packet set $\Omega(t_{sw})$ contains any packet if and only if this packet has at least one transmission required to be completed within $[t_{st}, t_{sw}]$. If and only if a transmission is to be finished in $[t_{st}, t_{sw}]$, this transmission is contained in transmission set $\Psi(t_{sw})$. If $\Omega(t_{sw})$

is unschedulable, periodic packets are dropped to make $\Omega(t_{sw})$ schedulable. In addition, when $S[t_{st}, t_{sw}]$ updates more than Δ^u slots, $S[t_{st}, t_{sw}].\delta$ can be reduced by dropping periodic packets. This is because an idle slot is not regarded as an updated slot in $S[t_{st}, t_{sw}]$. However, the performance of WNCS is degraded by dropping packets according to the works [88, 96, 97, 105, 123]. Therefore, the primary goal of dynamic data link layer scheduling is to make all the rhythmic packets and as many periodic packets as possible satisfy the end-to-end (E2E) deadlines. Thus, the objective of dynamic scheduling is

$$\min \quad S[t_{st}, t_{sw}].\rho. \tag{6.5}$$

Each rhythmic packet $\chi_{0,k}$ in $\Omega(t_{sw})$ meets its deadline $d_{0,k}$ if and only if its last transmission $\chi_{0,k}(H_0)$ is completed by $d_{0,k}$, *i.e.*,

Constraint 1. $f_{0,k}(H_0) \leq d_{0,k}$.

To guarantee the performance of the WNCS, switch point is required to be smaller than or equal to a switch point upper bound (denoted as t^u_{sw}), which is the latest time slot when the WNCS is desired to respond to new physical disturbances. Thus, we have constraint

Constraint 2. $t_{sw} \leq t^u_{sw}$.

To make the information of updated slots in $S[t_{st}, t_{sw}]$ be piggybacked to packet $\chi_{n+1,m}$, $S[t_{st}, t_{sw}]$ must be generated within the time interval $[f_{0,m}(h)+1, f_{n+1,m}(1)-1]$, *i.e.*,

Constraint 3. *the computation time of S must be smaller than or equal to the maximum allowed running time $T_{max} = f_{n+1,m}(1) - 1 - f_{0,m}(h)$.*

Since a broadcast transmission can deliver the information for at most Δ^u updated slots within one time slot, we require

Constraint 4. $S[t_{st}, t_{sw}].\delta \leq \Delta^u$.

In addition, in the case $\bar{S}[t] = (i,h) \neq (-1,-1)$ and $S[t] = (-1,-1)$, $\chi_{i,k}(h)$ should have either not been released at t or been completed earlier than t. Otherwise, the WNCS will transmit $\chi_{i,k}(h)$ at slot t by following $\bar{S}$ because slot t is not updated when $S[t] = (-1,-1)$, which is called an implicit schedule conflict. Thus, we have

Constraint 5. *if* $\bar{S}[t] = (i,h) \neq (-1,-1)$, $r_{i,k}(h) \leq t$ *and* $f_{i,k}(h) \geq t$, *then* t *cannot be set idle,* i.e., $S[t] \neq (-1,-1)$.

Based on the above discussion, our aim is to solve the following problem:

Problem 1. *Given task set* $\mathcal{T}$, *static schedule* $\bar{S}$, $t_{n\to r}$ *and* t_{st}, *determine* t_{sw} *and* $S[t_{st}, t_{sw}]$ *such that objective function (6.5) is achieved subject to Constraints 1, 2, 3, 4 and 5 are satisfied.*

We summarize the notations used for Problem 1 in Table 6.3. In the next section, we will use a motivational example to show the necessity of solving our problem.

6.4 Motivation

We use a simple example to show the deficiency of using the static schedule in a WNCS when physical disturbance is present. Consider the example shown in Figure 6.1. There is one rhythmic task τ_0, two periodic tasks τ_1 and τ_2 and one broadcast task τ_3. The periodic tasks and rhythmic task are synchronous and packets $\chi_{0,1}$, $\chi_{1,1}$ and $\chi_{2,1}$ are released at time slot 0. The routing paths, periods

TABLE 6.3

Summary of Notations Used for Problem 1

t_{st}, t_{sw}, t^c_{sw}, t^u_{sw}	Start point, switch point, switch point candidate and switch point upper bound
$\Gamma(t^u_{sw})$	Set of switch point candidates
$\Omega(t_{sw})$, $\Omega(t^c_{sw})$	Set of packets to be scheduled within $[t_{st}, t_{sw}]$ and $[t_{st}, t^c_{sw}]$, respectively
$\Psi(t_{sw})$, $\Psi(t^c_{sw})$	Set of transmissions to be scheduled within $[t_{st}, t_{sw}]$ and $[t_{st}, t^c_{sw}]$, respectively
$\bar{S}$, S	Static schedule and dynamic schedule
$S[t_{st}, t]$	Dynamic schedule within $[t_{st}, t]$
$\bar{S}[t]$, $S[t]$	Assignment of slot t in $\bar{S}$ and S, respectively
Δ^u, $\Delta^u[t_{st}, t]$	The maximum allowed numbers of updated slots for $S[t_{st}, t_{sw}]$ and $S[t_{st}, t]$, respectively
$S[t_{st}, t].\rho$	Number of dropped periodic packets due to $S[t_{st}, t]$
$S[t_{st}, t].\delta$	Number of updated slots due to $S[t_{st}, t]$
$\chi^*_{i,k}(h)$	Ready transmission

and relative deadlines of tasks are presented in columns 2, 3 and 4, respectively, of Table 6.2. Let t^u_{sw} and Δ^u are 18 and 5, respectively. The static schedule $\bar{S}$ for the task set is shown in Figure 6.3(a).

Suppose at time slot $t_{n\to r}$(=9), τ_0 switches its state, and employs $\vec{P}_0 = [3, 6]^T$ and $\vec{D}_0 = [2, 5]^T$ as the vectors of periods and relative deadlines, respectively, when τ_0 is in the rhythmic state. Hence, the rhythmic packet returns to the nominal state at time slot $t_{r\to n}$(=18). Then, it is found that packet $\chi_{0,3}$ which is released at time slot 12 cannot meet its end-to-end deadline 17 if we still use the static schedule $\bar{S}$ after $t_{n\to r}$. Therefore, WNCS cannot make use of the static schedule

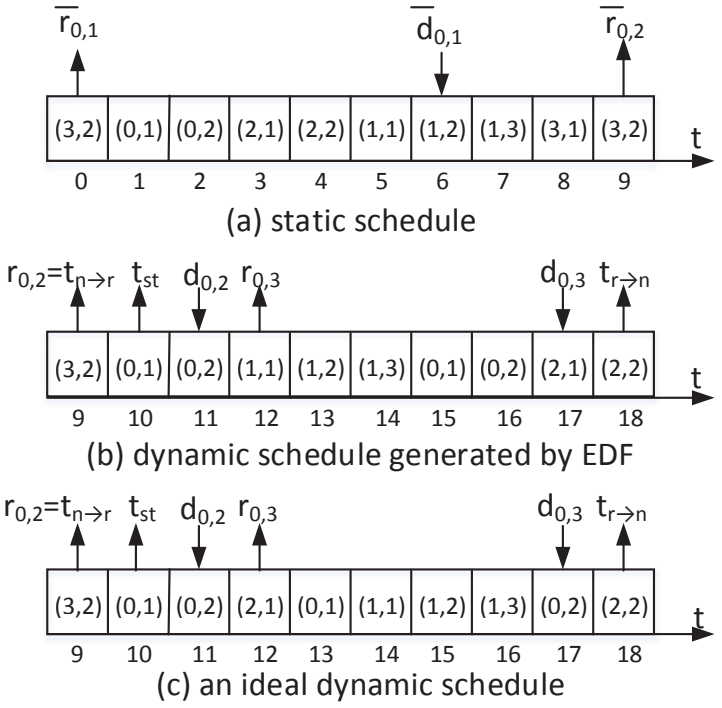


Figure 6.3. Time slot assignment in the static schedule and possible dynamic schedules in the motivational example.

$\bar{S}$ shown in Figure 6.3(a) to adapt to the changes of periods and deadlines for τ_0.

A possible method to make all the rhythmic packets schedulable is to calculate the minimal allowed period of rhythmic task τ_0 while scheduling rhythmic and period tasks under EDF. Then, the WNCS can employ a static schedule $\bar{S}'$ whose size is the least common multiple of rhythmic task and period tasks. In this schedule, rhythmic task τ_0 always transmits its packets with the minimal allowed period, therefore it is not necessary to reserve time slots for broadcast task. According to task set schedulability analysis under EDF in [17, 29, 101], the minimal allowed period of τ_0 is 5, which is larger than the first period 3 in $\vec{P}_0$. Hence, we cannot employ such a method to satisfy the requirement of period when τ_0 enters

the rhythmic state. This approach is also too pessimistic and cannot be scalable since it reserves too much unnecessary resource.

Suppose we set t_{sw} to t_{sw}^{u}, which is equal to 18. An alternative method is to employ EDF on packet set $\Omega(18)$, which generates a dynamic schedule shown in Figure 6.3(b). Although the generated schedule does not drop any packets, it updates 7 time slots, which is higher than $\Delta^{u}(=5)$. To make Constraint 4 satisfied, at least one packet needs to be dropped. In contrast, there exists another dynamic schedule $S[10, 18]$ without dropping any periodic packet and satisfying all the constraints in Problem 1, which is shown in Figure 6.3(c). In the next section, we present an approach on how to design such a dynamic schedule, *i.e.*, to minimize the number of dropped periodic packets while meeting constraints in Problem 1.

6.5 Overall Approach

In order to solve Problem 1, we present an on-line approach in this section, which determines a transient time duration and constructs a dynamic schedule for this time duration with bounded time and schedule update overheads. We first discuss which time slots can be the switch point of the dynamic schedule in Section 6.5.1 and then propose a framework that integrates the determination of possible switch points and the generation of the best dynamic schedule in minimizing the number of dropped periodic packets in Section 6.5.2.

6.5.1 Determining Switch Point Candidates

Choosing the right t_{sw} can be quite challenging as it impacts not only the schedulability of the system but also the running time of generating a dynamic

schedule and the number of updated slots in the dynamic schedule. In addition, $t_{sw} \leq t_{sw}^u$ is required in Constraint 2. To tackle this challenge, we first identify the following sufficient condition under which the static schedule $\bar{S}$ can be employed immediately after t_{sw}.

Theorem 8. *Given time slot t_{sw}, suppose $\bar{r}_{0,p^*+1}$ and r_{0,q^*+1} are the earliest nominal and actual release times of rhythmic packets later than or equal to t_{sw}, respectively, as is shown in Figure 6.2. The static schedule $\bar{S}$ can be employed immediately after t_{sw} without dropping any rhythmic packet if the following three conditions are satisfied by $S[t_{st}, t_{sw}]$.*

Condition 1. *$t_{sw} \geq t_{r \to n}$ and $\bar{r}_{0,p^*+1} = r_{0,q^*+1}$.*

Condition 2. *All the rhythmic packets released earlier than r_{0,q^*} must meet their deadlines.*

Condition 3. *We assume that $\bar{S}[t_{\breve{h}}] = (0, \breve{h})$ and $\bar{S}[t_{H_0}] = (0, H_0)$ are satisfied, where $t_{\breve{h}}$ and t_{H_0} are the earliest time slots reserved for any hop and the H_0-th hop of task τ_0 after t_{sw}, respectively. If $t_{H_0} > d_{0,q^*}$, all the transmissions of χ_{0,q^*} are finished by $\min\{t_{sw}, d_{0,q^*}\}$. If $t_{H_0} \leq d_{0,q^*}$, $\chi_{0,q^*}(\breve{h}-1)$ is finished by t_{sw}.*

Proof. Condition 1 guarantees that τ_0 has returned to the nominal state and all the rhythmic packets released later than or equal to r_{0,q^*+1} can meet their deadlines if the WNCS employs $\bar{S}$ immediately after r_{0,q^*+1}. Condition 2 ensures that all the rhythmic packets released earlier than r_{0,q^*} can meet their deadlines.

Condition 3 guarantees that χ_{0,q^*} can meet its deadline d_{0,q^*} while using $\bar{S}$ immediately after t_{sw}. In the case $t_{H_0} > d_{0,q^*}$, all the transmissions of χ_{0,q^*} are finished by $\min\{t_{sw}, d_{0,q^*}\}$. Hence, χ_{0,q^*} can meet its deadline d_{0,q^*}. If there is a time slot reserved for a hop of τ_0 in $\bar{S}[t_{sw}+1, \bar{r}_{0,p^*+1}]$, the WNCS just keeps idle

at this slot. In the case $t_{H_0} \leq d_{0,q^*}$, $\chi_{0,q^*}(\breve{h}-1)$ is finished by t_{sw} in S. Thus, $\chi_{0,q^*}(\breve{h}), \ldots, \chi_{0,q^*}(H_0)$ can be completed by d_{0,q^*} by following $\bar{S}$. If transmission $\chi_{0,q^*}(h)$ $(h > \breve{h})$ has been completed by t_{sw} and $\bar{S}[t_h] = (0, h)$ is satisfied, node just keeps idle at slot t_h. Hence, $\bar{S}$ can be employed immediately after t_{sw} without dropping any rhythmic packet.

□

Unfortunately, $\bar{r}_{0,p^*+1} = r_{0,q^*+1}$ is not always the case. In this case, we investigate different methods to make the WNCS still employ the static schedule $\bar{S}$ after t_{sw}. The first method is to shorten the time interval between r_{0,q^*} and r_{0,q^*+1}, and shift r_{0,q^*+1} to $\bar{r}_{0,p^*+1}$, which makes Condition 1 in Theorem 8 satisfied. If $d_{0,q^*} > \bar{r}_{0,p^*+1}$ is satisfied, d_{0,q^*} is adjusted to $\bar{r}_{0,p^*+1}$. If the conditions in Theorem 8 are still satisfied, then $\bar{S}$ can be employed after t_{sw} immediately according to Theorem 8. Since $\bar{S}$ is used by the WNCS after t_{sw}, the earliest rhythmic packet released later than or equal to t_{sw} should be released at $\bar{r}_{0,p+1}$. Hence, this method requires $t_{sw} \in [r_{0,q^*} + 1, \bar{r}_{0,p^*+1}]$, as is shown in Figure 6.2.

It is possible that any time slot later than or equal to $t_{r\to n}$ can be a switch point candidate. However, it is very time consuming to check if each time slot later than or equal to $t_{r\to n}$ can be the switch point t_{sw}. According to the following lemma, we only need to check whether the nominal release times of rhythmic packets can be the switch point.

Lemma 11. *Suppose $S[t_{st}, t^*]$ satisfies the conditions in Theorem 8 when we shift r_{0,q^*+1} to $\bar{r}_{0,p^*+1}$ and adjust d_{0,q^*} to $\min\{d_{0,q^*}, \bar{r}_{0,q^*+1}\}$, where $t^* \in [r_{0,q^*}+1, \bar{r}_{0,p^*+1}-1]$ is satisfied. If we select $\bar{r}_{0,p^*+1}$ as the switch point, there exists $S[t_{st}, \bar{r}_{0,p^*+1}]$ satisfying (i) the conditions in Theorem 8, (ii) $S[t_{st}, t^*].\rho = S[t_{st}, \bar{r}_{0,p^*+1}].\rho$ and (iii) $S[t_{st}, t^*].\delta = S[t_{st}, \bar{r}_{0,p^*+1}].\delta$.*

Proof. If we combine $S[t_{st}, t^*]$ and $\bar{S}[t^*+1, \bar{r}_{0,p^*+1}]$, we can construct the dynamic schedule $S[t_{st}, \bar{r}_{0,p^*+1}]$. Since $\bar{r}_{0,p^*+1} = r_{0,q^*+1} > t^* \geq t_{r\to n}$, we have Condition 1 satisfied. Since the combination of $S[t_{st}, t^*]$ and $\bar{S}[t^*+1, \bar{r}_{0,p^*+1}]$ does not change the time slot assignment in $[t_{st}, \bar{r}_{0,q^*}]$, $S[t_{st}, \bar{r}_{0,p^*+1}]$ guarantees all the rhythmic packets released earlier than r_{0,q^*} meet their deadlines, *i.e.*, Condition 2 is satisfied. In addition, when setting switch point to $\bar{r}_{0,p^*+1}$, $t_{H_0} > d_{0,q^*}$ is satisfied. Meanwhile, $S[t_{st}, t^*]$ ensures that all the transmissions of χ_{0,q^*} are completed by d_{0,q^*} according to Condition 3. Therefore, $S[t_{st}, \bar{r}_{0,p^*+1}]$ also satisfies Condition 3. Furthermore, since there is neither packet drop nor updated slot in $\bar{S}[t^*+1, \bar{r}_{0,p^*+1}]$, $S[t_{st}, t^*].\rho = S[t_{st}, \bar{r}_{0,p^*+1}].\rho$ and $S[t_{st}, t^*].\delta = S[t_{st}, \bar{r}_{0,p^*+1}].\delta$ are satisfied. □

Based on Lemma 11, each nominal release time $\bar{r}_{0,p+1}$ of rhythmic packet within $[t_{r\to n}, t^u_{sw}]$ can be a switch point candidate (denoted as t^c_{sw}). All such switch point candidates are grouped to switch point candidate set $\Gamma(t^u_{sw}) = \{t^c_{sw} | t^c_{sw} = \bar{r}_{0,p+1}, \forall \bar{r}_{0,p+1} \in [t_{r\to n}, t^u_{sw}]\}$.

The other method is to shorten the nominal release time interval between $\bar{r}_{0,p^*}$ and $\bar{r}_{0,p^*+1}$ and shift $\bar{r}_{0,p^*+1}$ to r_{0,q^*+1} when $\bar{r}_{0,p^*+1} > r_{0,q^*+1}$. Such a shift also makes Condition 1 in Theorem 8 be satisfied. However, this method needs to adjust and deliver the static schedule $\bar{S}$ to all the affected nodes, which is very complex. There are also other possible methods to make $\bar{r}_{0,p^*+1}$ equal to r_{0,q^*+1}. For example, we can either reduce the periods $P_{0,x}$'s or evenly shorten the constant period P_0 of rhythmic task to make $r_{0,q+1}$ equal to $\bar{r}_{0,p+1}$. Such methods may result in a high computational overhead within $[t_{st}, t_{sw}]$, which not only increase the number of updated slots in S but also make a feasible S hard to be found. Hence, in this work, we will only focus on the first method.

6.5.2 Framework

Algorithm 7 On-Line Scheduling (OLS)

Upon rhythmic task τ_0 enters the rhythmic state:
$\mathbf{S} = \emptyset$
$t_0^r = t_{st} - 1$, $t_1^r = t_{st} - 1$, $S_0^r = NULL$, $S_1^r = NULL$
Determine switch point candidate set $\Gamma(t_{sw}^u)$
for $(\forall t_{sw}^c \in \Gamma(t_{sw}^u))$ **do**
  Construct $\Omega(t_{sw}^c)$ and $\Psi(t_{sw}^c)$
  $b = 1$
  while $(b \geq 0)$ **do**
    if $(b == 1)$ **then**
      Drop periodic packets and periodic transmissions from $\Omega(t_{sw}^c)$ and $\Psi(t_{sw}^c)$, respectively
    end if
    Generate dynamic schedule $S[t_{st}, t_{sw}^c]$ based on $\Psi(t_{sw}^c)$, t_b^r and reusable schedule S_b^r
    while ($S[t_{st}, t_{sw}^c].\delta > \Delta^u$ and at least one periodic packet exist in $S[t_{st}, t_{sw}^c]$) **do**
      Drop the periodic packet causing the maximum number of updated slots and update $\Gamma(t_{sw}^u)$ correspondingly
    end while
    if $(S[t_{st}, t_{sw}^c].\delta \leq \Delta^u$) **then**
      $\mathbf{S} = \mathbf{S} \cup \{S[t_{st}, t_{sw}^c]\}$
      Calculate t_b^r and record $S[t_{st}, t_b^r]$ as S_b^r, where $S_b^r \subseteq S[t_{st}, t_{sw}^c]$ is satisfied
    end if
    $b = b - 1$
  end while
end for
Upon the maximum allowed running time T_{max} is run out:
Stop generating any new dynamic schedule
Select the best dynamic schedule $S[t_{st}, t_{sw}] \in \mathbf{S}$

To solve Problem 1, we design an on-line scheduling (OLS) approach to generate a dynamic schedule. Every time rhythmic task enters the rhythmic state at $t_{n \to r}$, OLS generates a basic dynamic schedule and an ordinary dynamic sched-

ule for each switch point candidate t_{sw}^c in $\Gamma(t_{sw}^u)$. A basic schedule only assigns time slots to rhythmic transmissions by dropping all the periodic transmissions. In contrast, an ordinary schedule aims at making periodic packets as many as possible meet their deadlines. A backup schedule can be used when no ordinary schedule is found. If the number of updated slots in a generated dynamic schedule is smaller than or equal to Δ^u, this schedule is accepted by OLS. Each accepted schedule is stored in the schedule set $\mathbf{S}$. Then, among all the dynamic schedules in $\mathbf{S}$, OLS selects the best dynamic schedule in terms of reducing the fewest number of dropped periodic packets first and the fewest number of updated slots second.

To speed up the search of a best dynamic schedule, OLS can reuse partial slot assignments generated for the previous value of t_{sw}^c after the value of t_{sw}^c is updated. Specifically, every time a basic or ordinary schedule is generated for $\Omega(t_{sw}^c)$, a reusable slot t_b^r is calculated and partial slot assignments $S[t_{st}, t_b^r]$ of $S[t_{st}, t_{sw}^c]$ is stored as reusable schedule S_b^r. Note that t_1^r and S_1^r are used to generate the next basic schedule while t_0^r and S_0^r are used to generate the next ordinary schedule. We calculate t_b^r by using

$$t_b^r = \min\{r_{i,k} | \forall \chi_{i,k} \in \Omega(t_{sw}^c), d_{i,k} \leq t_{sw}^c\} \tag{6.6}$$

when employing the heuristic to be presented in Section 6.6.2 to generate $S[t_{st}, t_{sw}^c]$. Since $S[t_{st}, t_b^r]$ does not schedule any packet $\chi_{i,k}$ satisfying $d_{i,k} > t_{sw}^c$, reusing $S[t_{st}, t_b^r]$ after updating t_{sw}^c has a negligible influence on packets whose deadlines are larger than the previous value of t_{sw}^c. After updating t_{sw}^c, OLS only needs to design dynamic schedule $S[t_b^r + 1, t_{sw}^c]$ and generates $S[t_{st}, t_{sw}^c]$ by combining $S[t_b^r + 1, t_{sw}^c]$ and S_b^r together. The details of OLS are shown in Algorithm 7.

In Algorithm 7, every time rhythmic task enters the rhythmic state at $t_{n \to r}$,

the gateway initializes dynamic schedule set $\mathbf{S}$ (Line 2). Next, OLS sets t_b^r to $t_s - 1$ and assigns S_b^r to $NULL$, where b is 0 and 1 (Line 3). Then, OLS determines switch point candidate set $\Gamma(t_{sw}^u)$ (Line 4). For each t_{sw}^c, the gateway V_g constructs packet set $\Omega(t_{sw}^c)$ and transmission set $\Psi(t_{sw}^c)$ (Line 6), which is described in more details in the end of this section, and sets b to 1 initially (Line 7). Then, OLS first generates a basic schedule and next generates an ordinary schedule for each switch point candidate (Lines 8-21). When b is 1 (Line 9), all the periodic packets and transmissions are dropped from $\Omega(t_{sw}^c)$ and $\Psi(t_{sw}^c)$, respectively (Line 10). Then, the approach generates dynamic schedule $S[t_{st}, t_{sw}^c]$ in Line 12, which will be discussed in Section 6.6 in details. While $S[t_{st}, t_{sw}^c]$ has more than Δ^u number of updated slots and $\Omega(t_{sw}^c)$ has more than one periodic packets (Line 13), OLS keeps dropping the periodic packet resulting in the maximum number of updated slots in $S[t_{st}, t_{sw}^c]$ in Line 14. If $S[t_{st}, t_{sw}^c].\delta \leq \Delta^u$ is satisfied (Line 16), $S[t_{st}, t_{sw}^c]$ will be included in schedule set $\mathbf{S}$ (Line 17). Meanwhile, reusable point t_b^r and reusable schedule S_b^r are updated in Line 18. When a dynamic schedule is generated, b is updated in Line 20. Such a process is repeated until each $t_{sw}^c \in \Gamma(t_{sw}^u)$ has been checked. When T_{max} is used out (Line 23), V_g stops generating any new dynamic schedule, and selects $S[t_{st}, t_{sw}]$ from $\mathbf{S}$ such that

$$S[t_{st}, t_{sw}].\rho \leq S[t_{st}, t_{sw}^c].\rho, \forall S[t_{st}, t_{sw}^c] \in \mathbf{S}. \tag{6.7}$$

Ties are broken in favour of the minimum number of updated slots first and the earliest switch point candidate next.

For each switch point candidate t_{sw}^c in $\Gamma(t_{sw}^u)$, we construct a packet set $\Omega(t_{sw}^c)$ and a transmission set $\Psi(t_{sw}^c)$ in Line 6 of Algorithm 7. It is required that any feasible schedule for $\Psi(t_{sw}^c)$ found by OLS satisfies Conditions 2 and 3 in Theo-

rem 8. Then, the WNCS can reuse $\bar{S}$ immediately after t^c_{sw} when we adopt t^c_{sw} as switch point t_{sw} and shift r_{0,q^*+1} to $\bar{r}_{0,p^*+1}$ according to Theorem 8. Meanwhile, any periodic packet can be dropped although a minimum number of dropped periodic packets is desired. In the rest of the paper, we use $\chi_{i,k}$ to represent either a rhythmic or a periodic packet in $\Omega(t^c_{sw})$ unless we specify the type of packet $\chi_{i,k}$. We assume that $\chi_{i,k}(\tilde{h})$ and $\chi_{i,k}(\hat{h})$ are the first and last transmissions in $\Psi(t^c_{sw})$ for packet $\chi_{i,k}$ $(1 \leq \tilde{h} \leq \hat{h} \leq H_i)$ in $\Omega(t^c_{sw})$, respectively. Any packet $\chi_{i,k}$ in $\Omega(t^c_{sw})$ is associated with an adjusted deadline $d'_{i,k}$ by which $\chi_{i,k}(\hat{h})$ must be completed. Suppose $\chi_{i,k}(\bar{h})$ is the first transmission of $\chi_{i,k}$, which is to be completed later than or equal to t_{st}.

We first determine which rhythmic packets belong to $\Omega(t^c_{sw})$ based on the following lemma.

Lemma 12. *Rhythmic packet $\chi_{0,k}$ belongs to $\Omega(t^c_{sw})$ if and only if $r_{0,k} < t^c_{sw}$ is satisified and $\chi_{0,k}$ has not been completed before t_{st}, $\chi_{0,k}$ belongs to $\Omega(t^c_{sw})$. In addition, we have $d'_{0,k} = \min\{d_{0,k}, t^c_{sw}\}$, $\tilde{h} = \bar{h}$ and $\hat{h} = H_0$. Then, any dynamic schedule $S[t_{st}, t^c_{sw}]$ ensuring all the rhythmic packets in $\Omega(t^c_{sw})$ to meet their adjusted deadlines satisfies Conditions 2 and 3 in Theorem 8 if t^c_{sw} is used as the switch point t_{sw}.*

Proof. We consider two cases of rhythmic packet $\chi_{0,k}$ in $\Omega(t^c_{sw})$, (i) $r_{0,k} < r_{0,q^*}$ and (ii) $r_{0,k} = r_{0,q^*}$, where r_{0,q^*} is the actual release time of rhythmic packet earlier than t^c_{sw}. In case (i), packet $\chi_{0,k}$ satisfies $d'_{0,k} = d_{0,k} < r_{0,q^*} < t^c_{sw}$. Since $\chi_{0,k}$ meets its adjusted deadline $d'_{0,k}$, $\chi_{0,k}$ also meets its actual deadline $d_{0,k}$, *i.e.*, Condition 2 is satisfied. In case (ii), all the transmissions are finished by $d'_{0,k} = \min\{t^c_{sw}, r_{0,q^*}\}$. According to the definition of t^c_{sw}, $t_{H_0} > t^c_{sw} > d_{0,q^*}$ is satisfied. Hence, Condition 3 is satisfied. □

In order to reuse static schedule $\bar{S}$ immediately after t^c_{sw}, we partition the workload of periodic packets proportionally based on the time slot assignment for periodic packets in $\bar{S}$. Specifically, any periodic packet $\chi_{i,k}$ which has at least one transmission to be finished within $[t_{st}, t^c_{sw}]$ according to $\bar{S}$ must belong to $\Omega(t^c_{sw})$. Suppose $\chi_{i,k}(\ddot{h})$ is the last transmission of $\chi_{i,k}$ to be completed within $[t_{st}, t^c_{sw}]$ according to $\bar{S}$. For any periodic packet $\chi_{i,k}$ in $\Omega(t^c_{sw})$, we have $d'_{0,k} = \min\{d_{0,k}, t^c_{sw}\}$, $\tilde{h} = \bar{h}$ and $\hat{h} = \ddot{h}$.

After V_g constructs packet set $\Omega(t^c_{sw})$, it gets transmission set $\Psi(t^c_{sw})$ containing transmissions $\chi_{i,k}(h)$'s, $h = \tilde{h}, \ldots, \hat{h}$, of each packet $\chi_{i,k} \in \Omega(t^c_{sw})$. Each transmission $\chi_{i,k}(h) \in \Psi(t^c_{sw})$ is associated with release time $r_{i,k}(h)$, deadline $d_{i,k}(h)$ and finish time $f_{i,k}(h)$, which are calculated by

$$r_{i,k}(h) = \begin{cases} r_{i,k} & \text{if } h = \tilde{h} = 1 \\ t_{st} - 1 & \text{if } h = \tilde{h} > 1 \\ f_{i,k}(h-1) & \text{if } \tilde{h} + 1 \leq h \leq \hat{h} \end{cases} \tag{6.8}$$

and

$$d_{i,k}(h) = d'_{i,k} - (\hat{h} - h). \tag{6.9}$$

Packet $\chi_{i,k} \in \Omega(t^c_{sw})$ can meet its deadline $d_{i,k}$ if $\chi_{i,k}$ can meet its adjusted deadline $d'_{i,k}$. Meanwhile, packet $\chi_{i,k} \in \Omega(t^c_{sw})$ can meet its adjusted deadline $d'_{i,k}$ if and only if each transmission $\chi_{i,k}(h)$ can meet deadline $d_{i,k}(h)$. If packet $\chi_{i,k}$ has a transmission $\chi_{i,k}(h)$ violate its deadline, $\chi_{i,k}$ is dropped out of $\Omega(t^c_{sw})$, and any transmission of $\chi_{i,k}$ is dropped out of $\Psi(t^c_{sw})$. In the next section, we will propose a heuristic to determine the finish time of each transmission in $\Psi(t^c_{sw})$, which is to be called in Line 12 of OLS.

6.6 Heuristic

In this section, we proposed a dynamic programming based heuristic to assign time slots to $\Psi(t^c_{sw})$, which is called in Line 12 of Algorithm 7. An exact method is to employ dynamic programming, which is presented in Section 6.6.1. Dynamic programming checks an exponential number of schedules to search for a solution, which is very time consuming and unsuitable for on-line use. Our heuristic presented in Section 6.6.2 modifies the structure of dynamic programming to search for solutions, which is called modified dynamic programming (mDP). However, mDP is constrained to check a limited number of schedules such that the time slot assignment can be completed with a pseudo polynomial time complexity.

6.6.1 Dynamic Programming

The key idea of dynamic programming is to construct a dynamic schedule $S[t_{st}, t]$ by determining the assignment of time slot t given a dynamic schedule $S[t_{st}, t-1]$. Schedule $S[t_{st}, t-1]$ is a child schedule of $S[t_{st}, t]$ for slot t while $S[t_{st}, t]$ is a parent schedule of $S[t_{st}, t-1]$ for slot t. If a transmission either obtains a time slot or has been removed from $\Psi(t^c_{sw})$ by t according to $S[t_{st}, t]$, $S[t_{st}, t]$ accommodates this transmission. Note if a transmission is removed from $\Psi(t^c_{sw})$ by t, all of its successor are also removed from $\Psi(t^c_{sw})$ by t and hence $S[t_{st}, t]$ also accommodates all of its successors. Schedule $S[t_{st}, t]$ accommodates transmission subset ψ if $S[t_{st}, t]$ accommodates any transmission in ψ but does not accommodate any transmission outside ψ. If $S[t_{st}, t]$ and $S'[t_{st}, t]$ accommodates the same transmission subset, $S[t_{st}, t]$ is $S'[t_{st}, t]$'s equivalent schedule (denoted to be $S[t_{st}, t] \equiv S'[t_{st}, t]$). We now define a ready transmission of time slot t, which is the candidate to obtain t.

Definition 3. *Given a dynamic schedule $S[t_{st}, t-1]$, transmission $\chi^*_{i,k}(h) \in \Psi(t^c_{sw})$ is a ready transmission of slot t if (i) it is not finished by $t-1$ and (ii) $r_{i,k}(h) < t \leq d_{i,k}(h)$ is satisfied. Transmission subset $\tilde{\psi}$ of $\Psi(t^c_{sw})$ is a ready transmission subset of t if (i) any transmission in $\tilde{\psi}$ is a ready transmission of t and (ii) $\tilde{\psi}$ contains all the ready transmissions of t.*

Let $\Delta^u[t_{st}, t]$ represent the upper bound on the number of updated time slots for $S[t_{st}, t]$. Given dynamic schedule $S[t+1, t^c_{sw}]$, Δ^u can be calculated by

$$\Delta^u[t_{st}, t] = \Delta^u - S[t+1, t^c_{sw}].\delta. \tag{6.10}$$

Suppose that $opt(t, \psi, \Delta^u[t_{st}, t])$ returns $S[t_{st}, t].\rho$, where $S[t_{st}, t]$ drops the fewest number of periodic packets among all the schedules $S[t_{st}, t]$'s accommodating ψ and have $S[t_{st}, t].\delta \leq \Delta^u[t_{st}, t]$ satisfied. The goal of dynamic programming is to find $S[t_{st}, t^c_{sw}]$ to achieve

$$opt(t^c_{sw}, \Psi(t^c_{sw}), \Delta^u[t_{st}, t^c_{sw}]). \tag{6.11}$$

The value of $opt(t, \psi, \Delta^u[t_{st}, t])$ is determined not only by the time slot assignment on t but also by the selection of parent schedule $S[t_{st}, t-1]$. We employ function $\phi(S[t] = (i, h), \psi)$ to return the number of periodic packets which miss their deadlines at t due to the assignment of $S[t] = (i, h)$ while $S[t_{st}, t]$ accommodates ψ. A periodic packet misses its deadline at t when one of its transmission is a ready transmission of t, has its deadline equal to t but is not assigned on t in the dynamic schedule. As soon as a periodic packet misses its deadlines, this packet and all of its unfinished transmissions are dropped from $\Omega(t^c_{sw})$ and $\Psi(t^c_{sw})$, respectively. To calculate $opt(t, \psi, \Delta^u[t_{st}, t])$, we consider two cases, (i) $t = t_{st}$ and

(ii) $t_{st} < t \leq t^c_{sw}$. In case (i), $opt(t_{st}, \psi, \Delta^u[t_{st}, t_{st}])$ is only determined by the time slot assignment of t since there is no parent schedule at t, which is the initial case of $opt(t, \psi, \Delta^u[t_{st}, t])$. In case (ii), if t is set idle, we have

$$opt(t, \psi, \Delta^u[t_{st}, t])|_{(-1,-1)} = opt(t-1, \psi, \Delta^u[t_{st}, t]) + \phi(S[t] = (-1,-1), \psi), t_{st} < t \leq t^c_{sw}, \tag{6.12}$$

where function $opt(t, \psi, \Delta^u[t_{st}, t])|_{(-1,-1)}$ is a special case of $opt(t, \psi, \Delta^u[t_{st}, t])$ by setting t to be idle. If t is assigned to a ready transmission $\chi^*_{i,k}(h)$, we have

$$opt(t, \psi, \Delta^u[t_{st}, t])|_{(i,h)} = opt(t-1, \psi - \{\chi^*_{i,k}(h)\}, \Delta^u[t_{st}, t] - S[t]|_{(i,h)}.\delta) + \phi(S[t] = (i,h), \psi), t_{st} < t \leq t^c_{sw}, \tag{6.13}$$

where function $opt(t, \psi, \Delta^u[t_{st}, t])|_{(i,h)}$ is a special case of $opt(t, \psi, \Delta^u[t_{st}, t])$ by assigning t to $\chi^*_{i,k}(h)$. $S[t]|_{(i,h)}.\delta$ returns 0 if $\bar{S}[t] = (i,h)$ and 1 otherwise. In order to reuse equations (6.12) and (6.13), we have

$$opt(t_{st} - 1, \psi, \Delta^u[t_{st}, t_{st} - 1]) = 0. \tag{6.14}$$

By combining equations (6.12), (6.13) and (6.14), we have

$$opt(t, \psi, \Delta^u[t_{st}, t]) = \min\{opt(t, \psi, \Delta^u[t_{st}, t])|_{(-1,-1)}, \min_{\chi^*_{i,k}(h) \in \psi \cap \tilde{\psi}} \{opt(t, \psi, \Delta^u[t_{st}, t])|_{(i,h)}\}\}, t_{st} \leq t \leq t^c_{sw}. \tag{6.15}$$

In order to satisfy Constraint 4 of Problem 1, $opt(t, \psi, \Delta^u[t_{st}, t])$ has already considered the maximum allowed number of updated slots for $S[t_{st}, t]$. Specifi-

cally, dynamic programming never considers any schedule $S[t_{st}, t]$ which satisfies $S[t_{st}, t].\delta > \Delta^u$. We now define the urgent time slot and favourite time slot to ensure any considered schedule $S[t_{st}, t]$ in dynamic programming satisfy Constraints 1 and 5.

Definition 4. *A time slot t is an urgent time slot of $\chi^*_{0,k}(h)$ if ready transmission $\chi^*_{0,k}(h) \in \tilde{\psi}$ satisfies $d^*_{0,k}(h) = t$.*

An urgent time slot t of $\chi^*_{0,k}(h)$ must be assigned to rhythmic transmission $\chi^*_{0,k}(h)$ since no rhythmic packet can be dropped according to Constraint 1. Hence, we have

$$\phi(S[t] = (i', k', h'), \psi) = +\infty \text{ if } t \text{ is an urgent slot of } \chi^*_{0,k}(h), \text{ and } i' \neq 0. \quad (6.16)$$

Definition 5. *A time slot t is a favourite time slot of $\chi^*_{i,k}(h)$ if ready transmission $\chi^*_{i,k}(h) \in \tilde{\psi}$ satisfies $\bar{S}[t] = (i, h)$.*

A favourite time slot t of $\chi^*_{i,k}(h)$ cannot be set to be idle because an idle slot is not regarded as an update slot. Otherwise, the WNCS still follows $\bar{S}$ to send $\chi^*_{i,k}(h)$ and Constraint 5 is violated. Therefore, we have

$$\phi(S[t] = (-1, -1), \psi) = +\infty \text{ if } t \text{ is a favourite slot of } \chi^*_{i,k}(h). \quad (6.17)$$

Although dynamic programming is optimal in solving Problem 1, it is time consuming and not suitable for the on-line use. In the experiment, we also demonstrate the running time of dynamic programming. To make the search process effective and efficient, we propose a dynamic programming based heuristic in the next section.

6.6.2 Modified Dynamic Programming (mDP)

The dynamic programming approach is very time consuming because it reserves an exponential number of child schedules for time slot t and use them as the parent schedules for time slot $t+1$. Algorithm mDP employs a variable β to judiciously limit the maximum allowed number of reserved child schedules for each time slot in $[t_b^r+1, t_{sw}^c]$ to speed up the generation of dynamic schedule. The critical challenge of designing mDP is how to generate a dynamic schedule which minimizes the number of dropped periodic packets while still ensuring all the rhythmic packets meet their deadlines (Constraint 1) and avoiding any implicit schedule conflict (Constraint 5). The main idea of mDP is that at most β number of child schedules dropping the fewest number of periodic packets are reserved for time slot t and used as the parent schedules for time slot $t+1$. Such a process is repeated from t_b^r+1 to t_{sw}^c.

The details of mDP are summarized in Algorithm 8. The inputs to mDP are transmission set $\Psi(t_{sw}^c)$, start time t_{st}, reusable time slot t_b^r, switch point candidate t_{sw}^c, static schedule $\bar{S}$, reusable schedule S_b^r, the maximum allowed number of updated slots, Δ^u, the number of periodic tasks, n, and the maximum allowed number of reserved schedules for each slot, β. Without loss of generality, a transmission is always associated with its release time, deadline and finish time. Algorithm mDP starts with initializing set of parent schedules, $\mathbf{S_p}$, to be $\{S_b^r\}$ (Line 1). Then, the program enters the main loop of Algorithm mDP (Lines 2-38) to generate child schedules $S[t_{st}, t]$'s based on each parent schedule $S[t_{st}, t-1]$ for each slot t starting from t_b^r+1 to t_{sw}^c. Given the parent schedule $S[t_{st}, t-1]$, the generation of child schedules is composed of two parts. The first part (Lines 4-5) determines if t is an urgent or favourite time slot of some ready transmissions

Algorithm 8 mDP($\Psi(t_{sw}^c)$, t_{st}, t_b^r, t_{sw}^c, $\bar{S}$, S_b^r, n, β)

$\mathbf{S_p} = \{S_b^r\}$
for ($t = t_b^r + 1$; $t \leq t_{sw}^c$; $t++$) **do**
  for ($\forall S[t_{st}, t-1] \in \mathbf{S_p}$) **do**
    $urgent\_flag = Is\_Slot\_Urgent(\Psi(t_{sw}^c), S[t_{st}, t-1], t)$
    $favourite\_flag = Is\_Slot\_Favourite(\Psi(t_{sw}^c), S[t_{st}, t-1], t)$
    for ($i = -1$; $i \leq n$; $i++$) **do**
      if ($urgent\_flag == 1$ and $i \neq 0$) **then**
        continue
      end if
      if ($favourite\_flag == 1$ and $i == -1$) **then**
        continue
      end if
      if (i==-1) **then**
        $(i, h) = (-1, -1)$ //Set t to be idle
      else
        $\chi_{i,k}^*(h) = Get\_Ready\_Tx(\Psi(t_{sw}^c), i, S[t_{st}, t-1])$
      end if
      $S[t] = (i, h)$
      $S[t_{st}, t] = S[t_{st}, t-1] + S[t]$
      $S[t_{st}, t] = Calculate\_Dropped\_Packet\_Number(\Psi(t_{sw}^c), S[t_{st}, t], n)$
      $S[t_{st}, t] = Calculate\_Updated\_Slot\_Number(\bar{S}[t], S[t_{st}, t])$
      $accept\_flag = 1$
      for $\forall S'[t_{st}, t] \in \mathbf{S_c}$ **do**
        if ($S'[t_{st}, t] \equiv S[t_{st}, t]$) **then**
          if ($S[t_{st}, t].\delta \geq S'[t_{st}, t].\delta$ and $S[t_{st}, t].\rho \geq S'[t_{st}, t].\rho$) **then**
            $accept\_flag = 0$
          else if ($S[t_{st}, t].\delta \leq S'[t_{st}, t].\delta$ and $S[t_{st}, t].\rho \leq S'[t_{st}, t].\rho$) **then**
            $\mathbf{S_c} = \mathbf{S_c} - \{S'[t_{st}, t]\}$
          end if
        end if
      end for
      if ($accept\_flag == 1$) **then**
        $\mathbf{S_c} = Insert\_to\_Child\_Schedule\_Set(\mathbf{S_c}, S[t_{st}, t])$ //Insert $S[t_{st}, t]$ to the child schedule list
      end if
    end for
  end for
  $\mathbf{S_p} = Reserve\_Schedules(\mathbf{S_c}, \beta)$
end for
$S[t_{st}, t_{sw}^c] = Select\_A\_Best\_Schedule(\mathbf{S_p})$
return $S[t_{st}, t_{sw}^c]$

according to Definitions 4 and 5, respectively. If t is an urgent slot of $\chi^*_{0,k}(h)$, variable $urgent\_flag$ is set to 1. Similarly, if t is a favourite slot of $\chi^*_{i,k}(h)$, variable $favourite\_flag$ is set to 1. Based on the values of $urgent\_flag$ and $favourite\_flag$ for slot t, the second part (Lines 6-35) generates child schedules $S[t_{st}, t]$'s.

In the second part, mDP first determines the assignment of t, *i.e.*, t can be set idle (Line 14) or assigned to a ready transmission $\chi^*_{i,k}(h)$ (Line 16). If t is an urgent slot of $\chi^*_{0,k}(h)$ ($urgent\_flag = 1$), t can only be assigned to $\chi^*_{0,k}(h)$ to make each rhythmic packet meet its deadline (Lines 7-9). Similarly, if t is a favourite slot of any ready transmission $\chi^*_{i,k}(h)$ ($favourite\_flag = 1$), t cannot be set to be idle to avoid any implicit schedule conflict (Lines 10-12). For each possible assignment of time slot t, mDP records such an assignment in $S[t]$ (Line 18), and attaches $S[t]$ to parent schedule $S[t_{st}, t-1]$ to get a child schedule $S[t_{st}, t]$ (Line 19). Next, mDP calculates the number of dropped packets for $S[t_{st}, t]$ in function $Calculate\_Dropped\_Packet\_Number()$ (Lines 20), which is introduced in more details in Algorithm 9. Similarly, mDP calculates the number of updated slots in $S[t_{st}, t]$ in function $Calculate\_Updated\_Slot\_Number()$ (Line 21), which is introduced in more details in Algorithm 10.

Every time a new child schedule $S[t_{st}, t]$ for slot t is generated, mDP determines which child schedules should be reserved in set of child schedules, $\mathbf{S_c}$ (Lines 22-31). Specifically, mDP initializes $accept\_flag$ to 1 and compares the new child schedule $S[t_{st}, t]$ to any other equivalent child schedule $S'[t_{st}, t]$ already stored in $\mathbf{S_c}$. If a child schedule dropping more periodic packets and updating more time slots than any other equivalent schedule, this child schedule should not be stored in $\mathbf{S_c}$. If such a schedule is the newly generated schedule $S[t_{st}, t]$, $accept\_flag$ is set to 0 in Line 26. If such a schedule is an existing schedule $S'[t_{st}, t]$ in $\mathbf{S_c}$, $S'[t_{st}, t]$ needs

to be moved out of $\mathbf{S_c}$ in Line 28. If $accept\_flag$ is 1 (Line 32), *i.e.*, $S[t_{st}, t]$ is inserted into $\mathbf{S_c}$ in the non-decreasing order of the number of dropped periodic packets first and the number of updated slots next in Line 33. After mDP has obtained $\mathbf{S_c}$, it only reserves the first β number of child schedules in $\mathbf{S_c}$ and put them to $\mathbf{S_p}$ in Line 37. Such a process is repeated for each time slot t from t_{st} to t^c_{sw}.

After the main loop, mDP selects the schedule $S[t_{st}, t^c_{sw}]$ dropping the fewest number of periodic packets among all the schedules in $\mathbf{S_p}$ (Line 39). Ties are broken in favour of the fewest number of updated slots. Finally, mDP returns $S[t_{st}, t^c_{sw}]$ (Line 40). The time complexity of mDP is dominated by the main `for` loop starting at Line 2. For each time slot t, mDP will generate at most $O(n \cdot \beta)$ number of child schedules. Within the main `for` loop, the most time consuming part is comparing the newly generated child schedule $S[t_{st}, t]$ to each existing child schedule $S'[t_{st}, t]$ in $\mathbf{S_c}$ (Lines 23-31). Since there are at most $(n+1) \cdot \beta$ number of child schedules to be generated at slot t, there are $O(n^2 \cdot \beta^2)$ number of comparisons for slot t. Hence, the time complexity of mDP is $O(t^c_{sw} \cdot n^2 \cdot \beta^2)$ since t^r_b may be 0. Suppose there are κ number of switch point candidates in $\Gamma(t^u_{sw})$. Then, OLS-mDP needs $O(\kappa \cdot t^c_{sw} \cdot n^2 \cdot \beta^2)$ time to find the best dynamic schedule $S[t_{st}, t_{sw}]$.

Algorithm 9 presents function $Calculate\_Dropped\_Packet\_Number()$, which starts with initializing $S[t_{st}, t].\rho$ (Lines 1-5) and then updates $S[t_{st}, t].\rho$ (Lines 6-11). Specifically, if a ready transmission $\chi_{i',k'}(h')$ has its deadline $d_{i',k'}(h')$ equal to t but does not obtain t (Line 8), $S[t_{st}, t].\rho$ is incremented by 1 (Line 9) presents function $Calculate\_Updated\_Slot\_Number()$, which initializes $S[t_{st}, t].\delta$ (Lines 1-5), and increments $S[t_{st}, t].\delta$ by 1 (Line 7) if t is not set idle and there is an

assignment conflict between $S[t]$ and $\bar{S}[t]$ (Line 6).

Algorithm 9 Calculate_Dropped_Packet_Number($\Psi(t^c_{sw})$, $S[t_{st}, t]$, n)

1: **if** (t==t_{st}) **then**
2: $S[t_{st}, t].\rho = 0$
3: **else**
4: $S[t_{st}, t].\rho = S[t_{st}, t-1].\rho$
5: **end if**
6: **for** $i' = 1$; $i' \leq n$; $i'++$ **do**
7: $\chi^*_{i',k'}(h') = Get\_Ready\_Tx(\Psi(t^c_{sw}), i', S[t_{st}, t-1])$
8: **if** ($d_{i',k'}(h') == t$ and $S[t] \neq (i', h')$) **then**
9: $S[t_{st}, t].\rho = S[t_{st}, t].\rho + 1$
10: **end if**
11: **end for**
12: return $S[t_{st}, t]$

Algorithm 10 Calculate_Updated_Slot_Number($\bar{S}[t]$, $S[t_{st}, t]$)

1: **if** ($t = t_{st}$) **then**
2: $S[t_{st}, t].\delta = 0$
3: **else**
4: $S[t_{st}, t].\delta = S[t_{st}, t-1]$
5: **end if**
6: **if** ($\bar{S}[t] \neq (i, h)$ and $i \neq -1$) **then**
7: $S[t_{st}, t].\delta = S[t_{st}, t-1].\delta + 1$
8: **end if**
9: return $S[t_{st}, t]$

An astute reader may notice that we do not require $S[t_{st}, t^c_{sw}].\delta \leq \Delta^u$ in mDP. Since the first β child schedules dropping the minimum number of packets for

slot t can be reserved, it is possible that the reserved schedules for t have already updated nearly Δ^u number of slots. Based on such reserved schedules, the child schedules for slots t' larger than t may update more than Δ^u slots. If mDP does not accept a child schedule updating more than Δ^u slots, probably no child schedule can be generated in a time slot larger than t. Hence, we do not employ the constraint $S[t_{st}, t^c_{sw}].\delta \leq \Delta^u$ in mDP. However, the reservation of child schedules (Lines 22-31), updating set of parent schedules (Line 37), and selection of best dynamic schedule (Line 39) still consider the numbers of updated slots in generated child schedules. Furthermore, OLS-mDP ensures that dynamic schedule $S[t_{st}, t^c_{sw}]$ returned by mDP satisfies $S[t_{st}, t^c_{sw}].\delta \leq \Delta^u$ by adjusting $S[t_{st}, t^c_{sw}]$ (Lines 13-15 of Algorithm 7).

6.7 Performance Evaluation

In this section, we evaluate the performance and efficiency of our mDP approach using generated task sets on a generated network topology. We start to evaluate OLS-mDP by calibrating different numbers of reserved schedules, β's, and setting different switch point upper bounds t^u_{sw}'s. Then, we select best parameters for OLS-mDP and compare the performance of OLS-mDP and another efficient approach under different types of workloads. Finally, we apply OLS-mDP on a real-world case study to show its effectiveness of our OLS-mDP approach.

6.7.1 Simulation Setup

There are 36 nodes in total deployed in a 6×6 square mesh grid, as is shown in Figure 6.4. The wireless network consists of gateway V_{35}, 17 sensors and 18 actuators. Both sensors and actuators can serve as relay nodes. The wireless

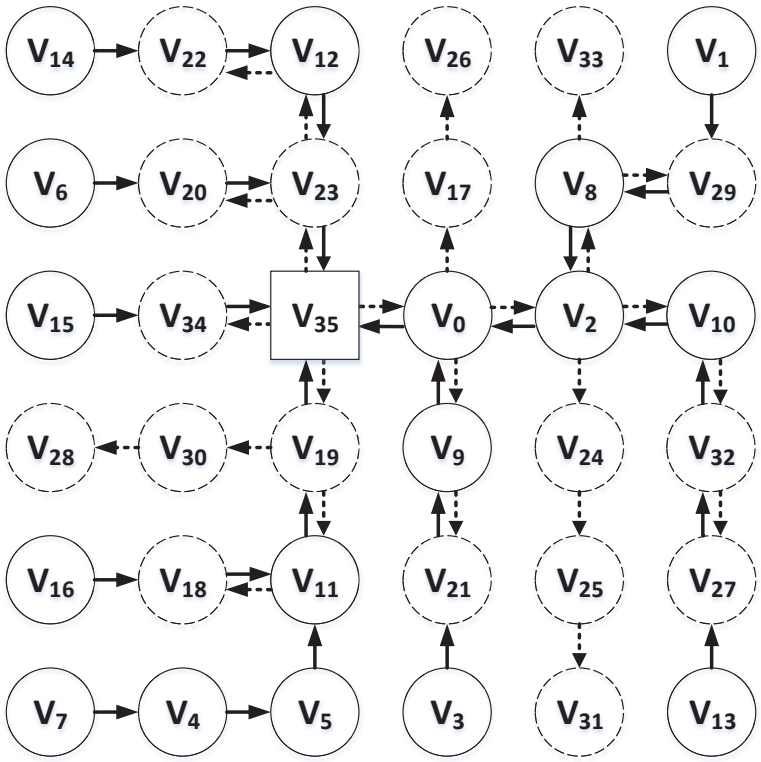


Figure 6.4. Topology of wireless network used in the simulation, which is composed of gateway V_{35}, 17 sensors and 18 actuators, which are represented as a square, solid circles and dashed circles, respectively. A solid and dashed direct link serves the routing path from a sensor to the gateway and from the gateway to an actuator, respectively.

network topology is a connected graph, *i.e.*, the gateway V_g can reach any other node in the graph. Each routing path from a sensor to the gateway or from the gateway to an actuator is pre-determined. In Figure 6.4, a solid and dashed direct link serves the routing path from a sensor to the gateway and from the gateway to an actuator, respectively.

We use 20 groups of task sets to different types of WNCSs and each group is labelled as "(n+1)-u", where (n+1) represents the number of rhythmic and

periodic tasks while u is the utilization level $\sum_{i=0}^{n+1} \frac{H_i}{P_i}$ in a task set. We select (n+1) to be 5, 10, 15 and 20 while assigning utilization levels to be 50%, 60%, 70%, 80% and 90%. The routing path of each rhythmic or periodic task is pre-selected randomly such that each rhythmic or periodic task is composed of a chain of 4 to 13 hops. We use UUnifast algorithm [22] to generate the initial period of each task since UUnifast provides better control on how to assign periods to tasks than a random assignment. The unit of period is one time slot, which represents 10 milliseconds. Similar to the WirelessHART protocol, we then adjust the generated period to the closest value 2^a ($a \leq 11$) in order to limit the size of static schedule $\bar{S}$. Each task set is generated with the guarantee that the total utilization level of task set is always equal to the specified utilization level. A static schedule containing $\mathcal{L}$ slots is designed for each task set.

We compare OLS-mDP to another time slot assignment approach which also uses the framework of OLS but employs a modified EDF to generate dynamic schedules (called OLS-mEDF). For each t_{sw}^c in $\Gamma(t_{sw}^u)$, mEDF first uses On-Line Distributed Algorithm (OLDA)[3] to test the schedulability of $\Omega(t_{sw}^c)$. If $\Omega(t_{sw}^c)$ is found to be unschedulable, some periodic packets are dropped to ensure the schedulability of $\Omega(t_{sw}^c)$ by following packet drop policy MLET presented in the work [65]. Then, mEDF applies EDF on $\Omega(t_{sw}^c)$ to generate a basic dynamic schedule and an ordinary dynamic schedule for each t_{sw}^c. Although OLS-mEDF is very efficient in generating dynamic schedules, mEDF ignores the constraint on the maximum allowed number of updated slots. This may cause OLS-mEDF to drop extra periodic packets to reduce the number of updated slots in Lines 13-15 of Algorithm 7.

[3]OLDA [65] is an on-line algorithm which combines local-deadline assignment with schedulability analysis on the workload in a uniprocessor.

We implement OLS-mDP and OLS-mEDF in C++. Experimental data were collected on a Sun Ultra 20 (x86-64) workstation with Red Hat Enterprise Linux 6.5. We compare the performance of both approaches using two metrics. The first metric is the number of solved task sets by OLS-mDP and OLS-mEDF, and the second metric is the average drop rate for the commonly solved task sets by both approaches. The average drop rate is the ratio between the number of dropped periodic packets and the number of periodic packets in $\Omega(t_{sw}^c)$. It is possible that OLS-mDP and OLS-mEDF may not employ the same switch point for a commonly solved task set. To ensure the fairness of comparison, we use the larger one out of the two switch points selected by two approaches when calculating the number of periodic packets in $\Omega(t_{sw}^c)$. Both metrics indicate the capability of both approaches in responding to the external disturbances.

6.7.2 Parameter Selection for OLS-mDP

Since the performance of OLS-mDP depends on the parameters β and t_{sw}^u, we start by calibrating these parameters to fully exploit the potential of OLS-mDP in reducing the number of periodic packets. As a starting point, we first test OLS-mDP using different values $(10, 20, 30, \ldots, 100)$ for β while fixing Δ^u and t_{sw}^u to $+\infty$ and $+\infty$, respectively. The numbers of dropped packets for groups of 5-task, 10-task, 15-task and 20-task sets are shown in Figures 6.5, 6.6, 6.7 and 6.8, respectively. Our results show that increasing β can reduce the number of dropped periodic packets initially. Then, the number of dropped packets fluctuates for a while with the increasing of β, and finally rises abruptly when β increases to a certain extent. Although the increasing of β may improve the performance of OLS-mDP, it simultaneously results in a larger computational overhead for using each

switch point candidate. Thus, OLS-mDP may try fewer switch point candidates within a maximum allowed running time T_{max} and drops more packets. If no ordinary schedule is found and T_{max} is run out, a backup schedule for a specific t_{sw}^{c} is used and all the periodic packets in $\Omega(t_{sw}^{c})$ are dropped (see Section 6.5.2). We observe that 90, 50, 50 and 40 are the best values of β which make OLS-mDP drop the fewest number of packets for groups of 5-task, 10-task, 15-task and 20-task sets, respectively.

Next, we choose the value of t_{sw}^{u} for OLS-mDP, which is expected to be as small as possible without sacrificing the performance of OLS-mDP. This is because a small t_{sw}^{u} could reduce the computational workload of running OLS-mDP on the gateway such that the gateway could perform other services. Since each switch point candidate is a nominal release time of rhythmic packet in OLS-mDP, we can calculate t_{sw}^{u} as follows:

$$t_{sw}^{u} = t_{r \to n} + (\alpha - 1) \cdot P_0, \tag{6.18}$$

where $\alpha \in N^{+}$ is called switch point scaling factor. By fixing β to the values selected above, we measured the numbers of dropped packets under OLS-mDP with different α values, $1 \leq \alpha \leq 3$, for all the groups of task sets, as is shown in Figure 6.9. It is observed that increasing α from 1 to 2 and from 2 to 3 can reduce the dropped packet number by 16% and 1% on average (56% and 9% at most), respectively. Without sacrificing the performance of OLS-mDP, we set α to 2 to reduce the computational workload on the gateway.

6.7.3 Performance of OLS-mDP against OLS-mEDF

We compared the performance of OLS-mDP with OLS-mEDF in terms of number of solved task sets and number of dropped packets while we employ different values of Δ^u. We set parameters β and α of OLS-mDP to the values selected above for different groups of task sets. To ensure the fairness of comparison, we also set α to 2 for OLS-mEDF. We assume that the payload size of packet in the WNCS is 90 bytes, which is similar to WirelessHART. We can use 3 bytes (24bits) to represent one updated slot when we use 13, 7 and 4 bits for specifying a slot identifier, a task identifier and a hop index, respectively. Therefore, one broadcast packet can accommodate information of at most 30 updated slots, *i.e.*, $\Delta^u = 30$. Suppose other advanced technology allows the payload size of packet in the WNCS is 180 bytes. Then, we also consider the situation when Δ^u is set to 60. Figures 6.10 and 6.11 show the numbers of solved task sets by OLS-mDP and OLS-mEDF when Δ^u is set to 30 and 60, respectively. It is found that OLS-mDP can solve more task sets than OLS-mEDF by 41% and 11% on average (at most 108% and 31%) for $\Delta^u = 30$ and $\Delta^u = 60$, respectively. Out of 2000 task sets, 431 and 1765 task sets are solved by both OLS-mDP and OLS-mEDF for $\Delta^u = 30$ and $\Delta^u = 60$, respectively.

The average drop rates of periodic packets for commonly solved task sets by both approaches are shown in Figure 6.12 and 6.13. It is observed that OLS-mEDF drops more packets than OLS-mDP by 122% and 128% on average (at most 321% and 263%) when setting Δ^u to 30 and 60, respectively. Since more updated slots can be tolerated in a dynamic schedule with the increasing of Δ^u, both approaches can find more task sets and drop fewer dropped packets. When the number of rhythmic and periodic tasks is smaller, fewer updated slots will

be caused by mDP and mEDF. Hence, OLS can avoid dropping extra packets to reduce the number of updated slots (Lines 13-15 of Algorithm 7) for more task sets when Δ^u is set to 60. Since EDF used in mEDF is optimal in terms of meeting packet deadlines for the WNCS containing a single channel, mEDF may drop more packets than mDP. Therefore, OLS-mEDF drops more packets than OLS-mDP for task set groups 5-0.7 and 5-0.9 when Δ^u is set to 60.

6.7.4 Case Study

Using a large number of randomly generated task sets, we have shown that OLS-mDP outperforms OLS-mEDF for most of task sets. However, it is important to validate the performance of OLS-mDP under real-world workload. In this section, we compare OLS-mDP with OLS-mEDF in a bio-reactor application presented in the Hart Communication Protocol WirelessHART Device Specification [1]. The bio-reactor application is composed of 10 periodic tasks measuring physical conditions, 5 periodic tasks actuating regulating valves and 3 aperiodic tasks actuating blocking valves. Tasks are either assigned periods or average response times in the specification. We assume that periods are equal to relative deadlines. To improve the performance of the application, we change the 3 aperiodic tasks to be periodic tasks and set their periods to half of their average response times. We also add a broadcast task to the bio-reactor application such that the dynamic schedule can be broadcast to all the nodes in the WNCS. Suppose that the bio-reactor application is running in the 6×6 square mesh presented in Section 6.7.1. The routing path of each rhythmic or periodic task is randomly selected, which is composed of 1 to 6 hops. In addition, a static schedule is generated off-line for the WNCS.

Assume that the rhythmic task is in the nominal state and the WNCS follows the static schedule from time slot 0. Suppose at time slot 1600, rhythmic task enters the rhythmic state and employs the following vectors of periods and deadlines,

$$\vec{P}_0 = \vec{D}_0 = [6, 12, 25, 50, 100, 150, 200, 400, 600, 800]^T. \tag{6.19}$$

When setting Δ^u to 30, OLS-mEDF cannot find a solution. In contrast, OLS-mDP only drops 5 periodic packets out of 510 periodic packets. When we increase Δ^u to 60, OLS-mDP and OLS-mEDF drops 7 and 1 packets out of 512 periodic packets, respectively.

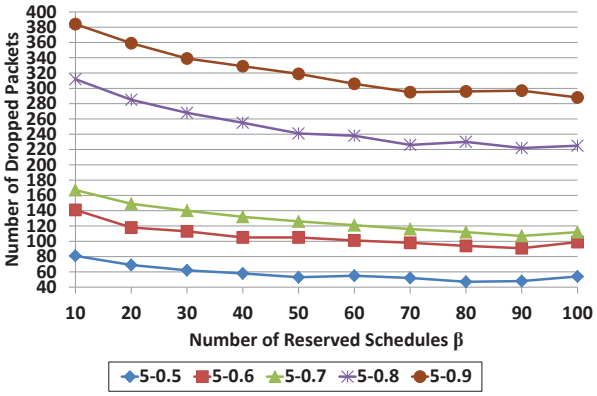


Figure 6.5. Number of dropped periodic packets in different groups of 5-task sets under OLS-mDP with different β values.

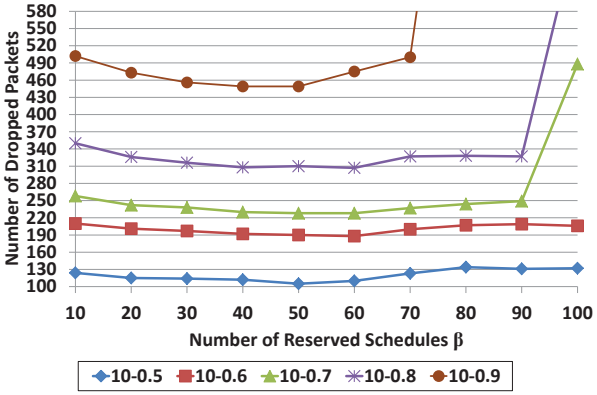


Figure 6.6. Number of dropped periodic packets in different groups of 10-task sets under OLS-mDP with different β values.

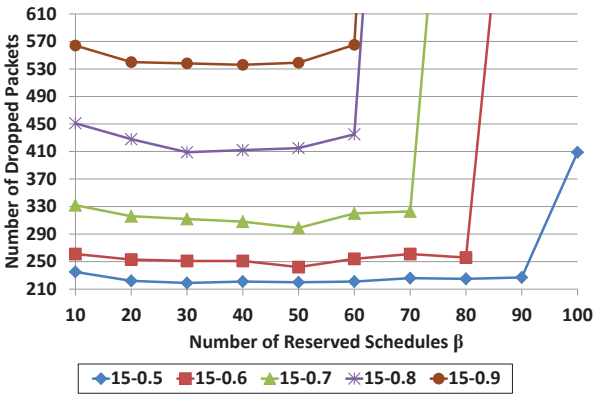


Figure 6.7. Number of dropped periodic packets in different groups of 15-task sets under OLS-mDP with different β values.

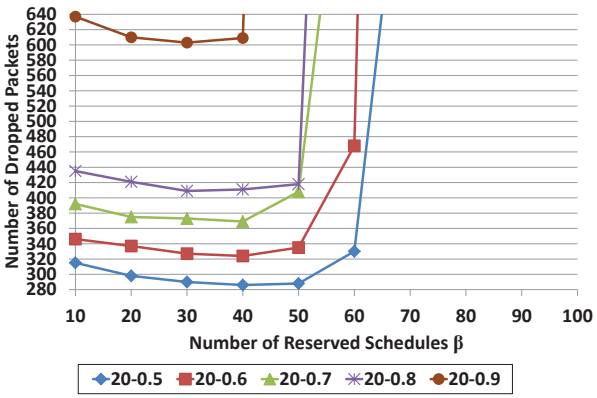


Figure 6.8. Number of dropped periodic packets in different groups of 20-task sets under OLS-mDP with different β values.

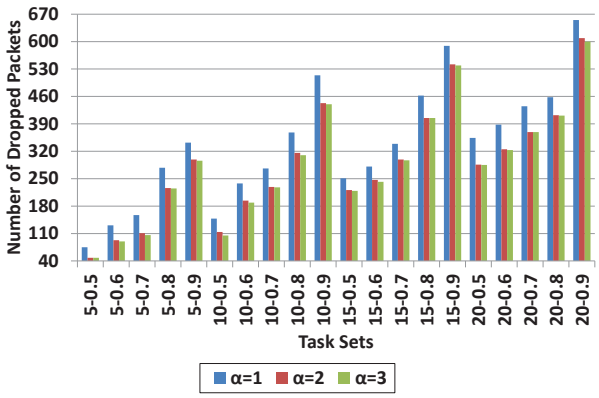


Figure 6.9. Number of dropped periodic packets in different groups of sets under OLS-mDP with different α values.

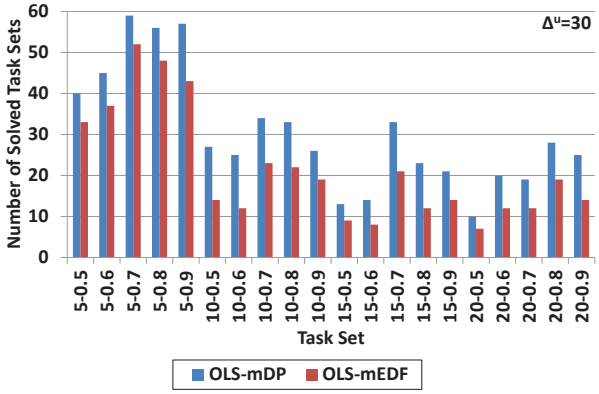


Figure 6.10. Number of solved task sets by OLS-mDP and OLS-mEDF with Δ^u equal to 30.

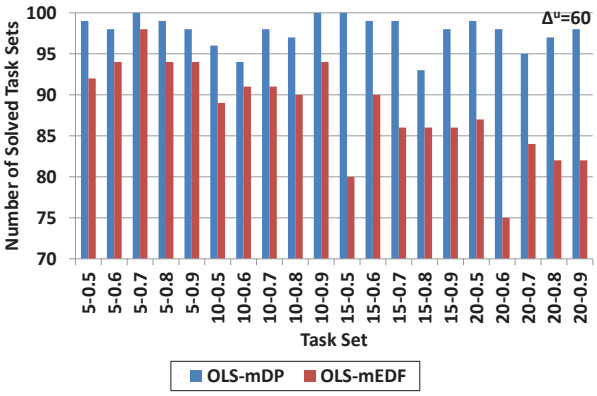


Figure 6.11. Number of solved task sets by OLS-mDP and OLS-mEDF with Δ^u equal to 60.

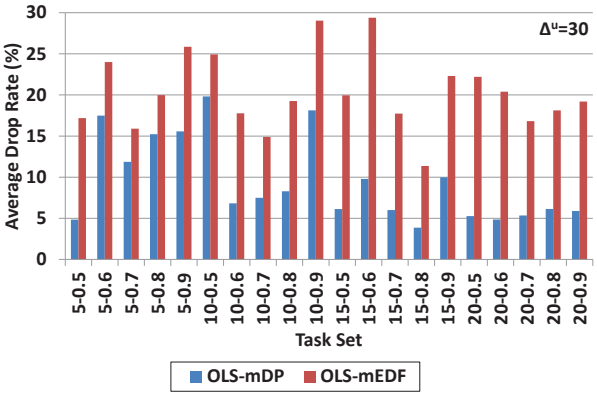


Figure 6.12. Average drop rates of dropped periodic packets for commonly solved task sets by OLS-mDP and OLS-mEDF with Δ^u equal to 30.

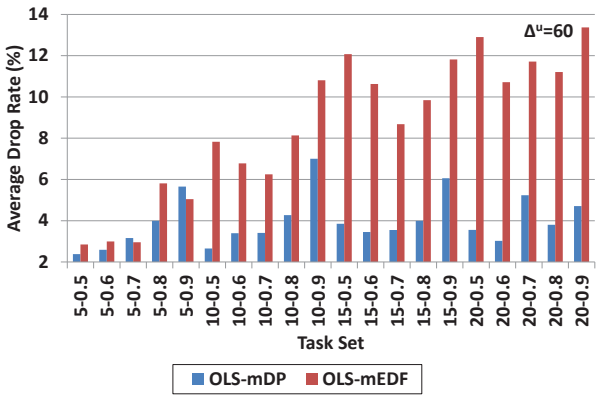


Figure 6.13. Average drop rates of dropped periodic packets for commonly solved task sets by OLS-mDP and OLS-mEDF with Δ^u equal to 60.

BIBLIOGRAPHY

1. Hart communication protocol wirelesshart device specification hcf_spec-290, revision 1.1. May 2008.

2. ISA100. `http://www.isa.org/isa100`.

3. `http://lpsolve.sourceforge.net/5.5/`.

4. M. Z. and E. Modiano. A calculus approach to energy-efficient data transmission with quality-of-service constraints. In *IEEE/ACM Transactions on Networking*, volume 17, 2009.

5. R. Agarwal and A. Goldsmith. Joint rate allocation and routing for multi-hop wireless networks with delay-constrained data. In *Technical Report, Wireless Systems Lab, Stanford University*, 2004.

6. Ahmed Rahni, Emmanuel Grolleau, and Michael Richard. Feasibility Analysis of Non-Concrete Real-Time Transactions With EDF Assignment priority. In *16th International Conference on Real-Time and Network Systems*, pages 109–117, Oct. 2008.

7. P. Albertos, A. Crespo, I. Ripoll, M. Valles, and P. Balbastre. Rt control scheduling to reduce control performance degrading. In *Proceedings of the 39th IEEE Conference on Decision and Control*, volume 5, pages 4889–4894, 2000.

8. A. Antunes and A. Mota. Improving control loop performance using dynamic rate adaptation in networked control systems. In *Proceedings of the 14th IEEE international conference on Emerging technologies & factory automation*, pages 1507–1510, 2009.

9. J. Bai, E. Eyisi, Y. Xue, and X. Koutsoukos. Distributed sampling rate adaptation for networked control systems. In *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 768–773, April 2011.

10. J. Bai, E. Eyisi, F. Qiu, Y. Xue, and X. D. Koutsoukos. Optimal cross-layer design of sampling rate adaptation and network scheduling for wireless networked control systems. In *2012 IEEE/ACM Third International Conference on Cyber-Physical Systems (ICCPS)*, pages 107–116, April 2012.

11. P. Balbastre, I. Ripoll, and A. Crespo. Control tasks delay reduction under static and dynamic scheduling policies. In *Proceedings. of Seventh International Conference on Real-Time Computing Systems and Applications*, page 522, 2000.

12. P. Balbastre, I. Ripoll, J. Vidal, and A. Crespo. A task model to reduce control delays. In *Real-Time Systems*, volume 27, pages 215–236, Sept. 2004.

13. P. Balbastre, I. Ripoll, and A. Crespo. Optimal deadline assignment for periodic real-time tasks in dynamic priority systems. In *18th Euromicro Conference on Real-Time Systems*, pages 65–74, 2006.

14. P. Balbastre, I. Ripoll, and A. Crespo. Minimum deadline calculation for

periodic real-time tasks in dynamic priority systems. In *IEEE Transactions on Computers*, volume 57, 2008.

15. S. Baruah and A. Burns. Sustainable scheduling analysis. In *27th IEEE International Real-Time Systems Symposium*, 2006.

16. S. K. Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks. In *Real-Time Systems*, volume 24, pages 93–128, Jan 2003.

17. S. K. Baruah, L. E. Rosier, and R. R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. In *Real-Time Systems*, volume 2, pages 301–324, Oct. 1990.

18. M. Behnam. Flexible scheduling for real time control systems based on jitter margin. In *Master Thesis, Malardalen Research and Technology Center, Malardalen University*, 2005.

19. M. Behnam and D. Isovic. Real-time control and scheduling co-design for efficient jitter handling. In *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2007.

20. R. Bettati and J. W.-S. Liu. End-to-end scheduling to meet deadlines in distributed systems. In *Proceedings of the 12th International Conference on Distributed Computing Systems,*, pages 452–459, June 1992.

21. E. Bini and G. Buttazzo. The space of edf deadlines: the exact region and a convex approximation. In *Real-Time Systems*, volume 41, 2009.

22. E. Bini and G. C. Buttazzo. Biasing effects in schedulability measures. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, pages 196–203, July 2004.

23. E. Bini. and A. Cervin. Delay-aware period assignment in control systems. In *Real-Time Systems Symposium*, 2008.

24. B. D. Bui, R. Pellizzoni, M. Caccamo, C. F. Cheah, and A. Tzakis. Soft real-time chains for multi-hop wireless ad-hoc networks. In *Proceedings of the 13th IEEE Real Time and Embedded Technology and Applications Symposium*, RTAS '07, pages 69–80, 2007.

25. G. Buttazzo and L. Abeni. Adaptive workload management through elastic scheduling. In *Real-Time Systems*, volume 23, pages 7–24, 2002.

26. G. Buttazzo and A. Cervin. Comparative assessment and evaluation of jitter control methods. In *Proc. of 15th International Conference on Real-Time and Network Systems*, 2007.

27. G. Buttazzo, E. Bini, and Y. Wu. Partitioning parallel applications on multiprocessor reservations. In *Proceedings of the 22nd Euromicro Conference on Real-Time Systems*, pages 24–33, July 2010.

28. G. Buttazzo, E. Bini, and D. Buttle. Rate-adaptive tasks: Model, analysis, and design issues. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–6, March 2014.

29. G. C. Buttazzo. Hard real-time computing systems: Predictable scheduling algorithms and applications. Springer, 2005.

30. S. Cavalieri. Meeting real-time constraints in CAN. In *IEEE Transactions on Industrial Informatics*, volume 1, pages 124–135, May 2005.

31. A. Cervin. Integrated control and real-time scheduling. In *PHD Thesis, Department of Automatic Control, Lund Institute of Technology*, 2003.

32. A. Cervin and B. Lincoln. Jitterbug 1.21 reference manual. Feb. 2006.

33. A. Cervin, B. Lincoln, J. Eker, K.-E. Arzen, and G. Buttazzo. The jitter margin and its application in the design of real-time control systems. In *Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications*, Aug. 2004.

34. T. Chantem, X. S. Hu, and M. Lemmon. Generalized elastic scheduling for real-time tasks. In *IEEE Transactions on Computers*, volume 58, 2009.

35. S. Chatterjee and J. Strosnider. Distributed pipeline scheduling: A framework for distributed, heterogeneous real-time system design. In *The Computer Journal*, volume 38, pages 271–285, 1995.

36. D. Chen, M. Nixon, and A. Mok. *WirelessHART: Real-Time Mesh Network for Industrial Automation*. Springer Publishing Company, Incorporated, 2010.

37. W. Chen, M. Neely, and U. Mitra. Energy-efficient transmissions with individual packet delay constraints. In *IEEE Transactions on Information Theory*, volume 54, 2008.

38. W.-P. Chen, J. Hou, L. Sha, and M. Caccamo. A distributer, energy-aware, utility-based approach for data transport in wireless sensor networks. In *Military Communications Conference*, volume 3, pages 1761–1767, Oct. 2005.

39. H. Chetto and M. Chetto. Scheduling periodic and sporadic tasks in a real-time system. In *Information Processing Letters*, volume 30, pages 177–184, Feb. 1989.

40. H. Chetto, M. Silly, and T. Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. In *Real-Time Systems*, volume 2, pages 181–194, Sept. 1990.

41. O. Chipara, C. Lu, and J. A. Stankovic. Dynamic conflict-free query scheduling for wireless sensor networks. pages 321–331, 2006.

42. O. Chipara, C. Lu, and G.-C. Roman. Real-time query scheduling for wireless sensor networks. In *RTSS*, pages 389–399, 2007.

43. O. Chipara, C. Wu, C. Lu, and W. G. Griswold. Interference-aware real-time flow scheduling for wireless sensor networks. In *ECRTS*, pages 67–77, 2011.

44. T. Chiueh, R. Krishnan, P. De, and J.-H. Chiang. A networked robot system for wireless network emulation. In *Proceedings of the 1st International Conference on Robot Communication and Coordination*, RoboComm '07, 2007.

45. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to algorithms. The MIT Press, 2002.

46. T. L. Crenshaw, S. Hoke, A. Tirumala, and M. Caccamo. *ACM Trans. Embed. Comput. Syst.*, 6(4), Sept. 2007.

47. A. Crespo, I. Ripoll, and P. Albertos. Reducing delays in rt control: The control action interval, decision and control. In *The International Federation of Automatic Control*, November 1999.

48. A. Davare, Q. Zhu, M. D. Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli. Period optimization for hard real-time distributed automotive systems. In *Proceedings of the 44th annual Design Automation Conference*, pages 278–283, June 2007.

49. R. P. Dick, D. L. Rhodes, and W. Wolf. TGFF: task graphs for free. In *Proceedings of the Sixth International Workshop on Hardware/Software Codesign*, pages 97–101, Mar. 1998.

50. T. Facchinetti and M. D. Vedova. Real-time modeling for direct load control in cyber-physical power systems. In *IEEE Transactions on Industrial Informatics*, volume 7, pages 689–698, Nov. 2011.

51. G. Fiore, V. Ercoli, A. Isaksson, K. Landernas, and M. D. Benedetto. Multihop multi-channel scheduling for wireless control in WirelessHART networks. In *IEEE Conference on Emerging Technologies Factory Automation*, pages 1–8, Sept. 2009.

52. D. Fontanelli, L. Greco, and A. Bicchi. Anytime control algorithms for embedded real-time systems. In *Proceedings of the 11th international workshop on Hybrid Systems: Computation and Control*, 2008.

53. A. Fu, E. Modiano, and J. Tsitsiklis. Optimal energy allocation for delay-constrained data transmission over a time-varying channel. In *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications*, 2003.

54. A. E. Gamal, C. Nair, B. Prabhakar, E. Uysal-Biyikoglu, and S. Zahedi. Energy-efficient scheduling of packet transmissions over wireless networks. In *Proceedings of Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, 2002.

55. M. S. Gast. 802.11@wireless networks: The definitive guide creating and administering wireless networks. In *O'REILLY*, 2002.

56. K. Gatsis, A. Ribeiro, and G. Pappas. Optimal power management in wireless control systems. In *American Control Conference (ACC), 2013*, pages 1562–1569, June 2013.

57. S. Gopalakrishnan, L. Sha, and M. Caccamo. Hard real-time communication in bus-based networks. In *Proceedings. of 25th IEEE International Real-Time Systems Symposium*, pages 405–414, Dec. 2004.

58. R. Gupta and M.-Y. Chow. *IEEE Transactions on Industrial Electronics*, 57 (7):2527–2535, 2010.

59. A. Hagiescu, U. D. Bordoloi, S. Chakraborty, P. Sampath, P. V. V. Ganesan, and S. Ramesh. Performance analysis of FlexRay-based ECU networks. In *Proceedings of the 44th annual Design Automation Conference*, pages 284–289, June 2007.

60. S. Han, X. Zhu, A. Mok, D. Chen, and M. Nixon. Reliable and real-time communication in industrial wireless mesh networks. In *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 3–12, April 2011.

61. J. Hatcliff, A. L. King, I. Lee, A. Macdonald, A. Fernando, M. Robkin, E. Y. Vasserman, S. Weininger, and J. M. Goldman. Rationale and architecture principles for medical application platforms. In *2012 IEEE/ACM Third International Conference on Cyber-Physical Systems (ICCPS)*, pages 3–12, April 2012.

62. W. Hawkins and T. Abdelzaher. Towards feasible region calculus: An end-to-end schedulability analysis of real-time multistage execution. In *Proceedings*

of the 26th IEEE International Real-Time Systems Symposium, pages 75–86, Dec. 2005.

63. X. Hei, X. Du, S. Lin, and I. Lee. Pipac: Patient infusion pattern based access control scheme for wireless insulin pump system. In *INFOCOM, 2013 Proceedings IEEE*, pages 3030–3038, April 2013.

64. H. Hoang, G. Buttazzo, M. Jonsson, and S. Karlsson. Computing the minimum edf feasible deadline in periodic systems. In *Proceedings. 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 125–134, 2006.

65. S. Hong, T. Chantem, and X. S. Hu. Meeting end-to-end deadlines through distributed local deadline assignments. In *2011 IEEE 32nd Real-Time Systems Symposium (RTSS)*, pages 183–192, Dec. 2011.

66. T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu. Dynamic voltage scaling in multitier web servers with end-to-end delay control. In *IEEE Transactions on Computers*, volume 56, pages 444–458, April 2007.

67. B. Hu and M. D. Lemmon. Using channel state feedback to achieve resilience to deep fades in wireless networked control systems. In *Proceedings of the 2Nd ACM International Conference on High Confidence Networked Systems*, HiCoNS '13, pages 41–48, 2013.

68. S. Hua, G. Qu, and S. S. Bhattacharyya. *ACM Trans. Embed. Comput. Syst.*, 6(3), July 2007.

69. O. Imer and T. Basar. To measure or to control: Optimal control with scheduled measurements and controls. In *American Control Conference*, 2006.

70. P. Jayachandran and T. Abdelzaher. Transforming distributed acyclic systems into equivalent uniprocessors under preemptive and non-preemptive scheduling. In *Proceedings of the 20nd Euromicro Conference on Real-Time Systems*, pages 233–242, July 2008.

71. P. Jayachandran and T. Abdelzaher. End-to-end delay analysis of distributed systems with cycles in the task graph. In *Proceedings of the 21st Euromicro Conference on Real-Time Systems*, pages 13–22, July 2009.

72. P. Jayachandran and T. Abdelzaher. Bandwidth allocation for elastic real-time flows in multihop wireless networks based on network utility maximization. In *The 28th International Conference on Distributed Computing Systems*, pages 849–857, June 2008.

73. P. Jayachandran and T. Abdelzaher. Delay composition in preemptive and non-preemptive real-time pipelines. In *Real-Time Systems*, volume 40, pages 290–320, Dec. 2008.

74. R. Jejurikar and R. Gupta. Energy-aware task scheduling with task synchronization for embedded real-time systems. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 25, 2006.

75. Y. Jiang and A. Striegel. An exploration of the effects of state granularity through (m, k) real-time streams. In *IEEE Transactions on Computers*, volume 58, pages 784–798, June 2009.

76. J. Jonsson and K. G. Shin. Robust adaptive metrics for deadline assignment in distributed hard real-time systems. In *Real-Time Systems*, volume 23, pages 239–271, Nov. 2002.

77. V. M. Karbhari and F. Ansari. Structural health monitoring of civil infrastructure systems. CRC Press, 2009.

78. M. Khojastepour and A. Sabharwal. Delay-constrained scheduling: power efficiency, filter design, and bounds. In *Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 3, 2004.

79. A. Kim, F. Hekland, S. Petersen, and P. Doyle. When hart goes wireless: Understanding and implementing the wirelessHART standard. In *IEEE International Conference on Emerging Technologies and Factory Automation*, pages 899–907, Sept. 2008.

80. H. Kim and K. G. Shin. Scheduling of battery charge, discharge, and rest. In *IEEE Real-Time Systems Symposium*, pages 13–22, Dec 2009.

81. J. Kim, K. Lakshmanan, and R. Rajkumar. Rhythmic tasks: A new task model with continually varying periods for cyber-physical systems. In *IEEE/ACM Third International Conference on Cyber-Physical Systems (ICCPS)*, pages 55–64, April 2012.

82. N. Kim, M. Ryu, S. Hong, M. Saksena, C.-H. Choi, and H. Shin. Visual assessment of a real-time system design: a case study on a cnc controller. In *17th IEEE Real-Time Systems Symposium*, pages 300 –310, Dec 1996.

83. T. Kim, H. Shin, and N. Chang. Deadline assignment to reduce output jitter of real-time tasks. In *Proceedings of The 16th IFAC Workshop on Distributed Computer Control Systems (DCCS)*, November 2000.

84. H. Kopetz. Real-time systems design principles for distributed embedded applications. Springer, 2011.

85. G. Kulkarni, V. Raghunathan, and M. Srivastava. Joint end-to-end scheduling, power control and rate control in multi-hop wireless networks. In *Global Telecommunications Conference*, volume 5, pages 3357–3362, Nov.-3 Dec. 2004.

86. W.-C. Kwon and T. Kim. Optimal voltage allocation techniques for dynamically variable voltage processors. In *ACM Transactions on Embedded Computing Systems*, volume 4, 2005.

87. J. Lee, I. Shin, and A. Easwaran. *Journal of Systems and Software*, 85(10): 2331–2339, 2012.

88. M. Lemmon and X. S. Hu. Almost sure stability of networked control systems under exponentially bounded bursts of dropouts. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, pages 301–310, 2011.

89. F. Leonardi, A. Pinto, and L. P. Carloni. Synthesis of distributed execution platforms for cyber-physical systems with applications to high-performance buildings. In *2011 IEEE/ACM International Conference on Cyber-Physical Systems (ICCPS)*, pages 215–224, April 2011.

90. H. Li, P. Shenoy, and K. Ramamritham. Scheduling messages with deadlines in multi-hop real-time sensor networks. In *11th IEEE Real Time and Embedded Technology and Applications Symposium*, pages 415–425, March 2005.

91. L. Li and M. Lemmon. Performance and average sampling period of sub-optimal triggering event in event triggered state estimation. In *50th*

IEEE Conference on Decision and Control and European Control Conference (CDC-ECC), pages 1656–1661, Dec. 2011.

92. L. Li, B. Hu, and M. Lemmon. Resilient event triggered systems with limited communication. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pages 6577–6582, Dec 2012.

93. M. Li and F. F. Yao. An efficient algorithm for computing optimal discrete voltage schedules. In *Journal SIAM Journal on Computing*, volume 35, 2005.

94. C. Lin, T. Kaldewey, A. Povzner, and S. A. Brandt. Diverse soft real-time processing in an integrated system. In *27th IEEE International Real-Time Systems Symposium, 2006.*, pages 369–378, Dec. 2006.

95. B. Lincoln and A. Cervin. Jitterbug: a tool for analysis of real-time control performance. In *Proceedings of the 41st IEEE Conference on Decision and Control*, volume 2, pages 1319–1324, Dec. 2002.

96. Q. Ling and M. Lemmon. Input-to-state stabilizability of quantized linear control systems under feedback dropouts. In *American Control Conference (ACC)*, pages 241–246, June 30-July 2, 2010 2010.

97. Q. Ling and M. D. Lemmon. A necessary and sufficient feedback dropout condition to stabilize quantized linear control systems with bounded noise. In *IEEE Transactions on Automatic Control*, volume 55, pages 2590–2596, 2010.

98. C. Liu and J. Anderson. Supporting graph-based real-time applications in distributed systems. In *IEEE 17th International Conference on Embedded*

and Real-Time Computing Systems and Applications (RTCSA), pages 143–152, Aug. 2011.

99. C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. In *Journal of the ACM*, volume 20, pages 46–61, Jan. 1973.

100. D. Liu, X. S. Hu, M. D. Lemmon, and Q. Ling. Firm real-time system scheduling based on a novel qos constraint. In *IEEE Transactons on Computers*, volume 55, pages 320–333, 2006.

101. J. W. Liu. Real-time systems. Prentice Hall, 2000.

102. M. Lluesma, A. Cervin, P. Balbastre, I. Ripoll, and A. Crespo. Jitter evaluation of real-time control systems. In *Proceedings. 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 257–260, 2006.

103. G. Loandpez, V. Custodio, and J. Moreno. Lobin: E-textile and wireless-sensor-network-based platform for healthcare monitoring in future hospital environments. In *IEEE Transactions on Information Technology in Biomedicine*, volume 14, 2010.

104. C. Locke, D. Vogel, and T. Mesler. Building a predictable avionics platform in ada: a case study. In *Proceedings. of Twelfth Real-Time Systems Symposium*, pages 181–189, Dec 1991.

105. W.-J. Ma and V. Gupta. Input-to-state stability of hybrid systems with receding horizon control in the presence of unreliable network packet dropouts. In *American Control Conference (ACC)*, pages 1243–1247, June 2012.

106. S. Manolache, P. Eles, and Z. Peng. Optimization of soft real-time systems with deadline miss ratio constraints. In *10th IEEE Real-Time and Embedded Technology and Applications Symposium.*, May 2004.

107. S. Manolache, P. Eles, and Z. Peng. Task mapping and priority assignment for soft real-time applications under deadline miss ratio constraints. In *Transactions on Embedded Computing Systems*, volume 7, January 2008.

108. J. Mao, C. G. Cassandras, and Q. Zhao. Optimal dynamic voltage scaling in energy-limited nonpreemptive systems with real-time constraints. In *IEEE Transactions on Mobile Computing*, 2007.

109. D. Marinca, P. Minet, and L. George. Analysis of deadline assignment methods in distributed real-time systems. In *Computer Communications*, volume 27, pages 1412–1423, June 2004.

110. S. Matic and T. Henzinger. Trading end-to-end latency for composability. In *Proceedings of the 26th IEEE International Real-Time Systems Symposium*, pages 99–110, Dec. 2005.

111. S. K. Mazumder. Wireless networking based control. Springer New York, 2011.

112. L. Miao and C. G. Cassandras. Optimal transmission scheduling for energy-efficient wireless networks. In *25th IEEE International Conference on Computer Communications*, 2006.

113. P. Naghshtabrizi and J. P. Hespanha. Implementation considerations for wireless networked control systems. In *Wireless Network Based Control*, pages 1–27. Springer, 2011.

114. S. G. Nash and A. Sofer. Linear and nonlinear programming. McGraw-Hill, 1996.

115. L. Niu and G. Quan. A hybrid static/dynamic dvs scheduling for real-time systems with (m, k)-guarantee. In *26th IEEE International Real-Time Systems Symposium, 2005. RTSS 2005.*, pages 356–365, Dec. 2005.

116. M. Nixon, D. Chen, T. Blevins, and A. Mok. Meeting control performance over a wireless mesh network. In *IEEE International Conference on Automation Science and Engineering*, pages 540–547, Aug. 2008.

117. M. Pajic, S. Sundaram, G. J. Pappas, and R. Mangharam. *IEEE Trans. Automat. Contr.*, 56(10):2305–2318, 2011.

118. J. Palencia and M. G. Harbour. Offset-based response time analysis of distributed systems scheduled under EDF. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, pages 3–12, July 2003.

119. P. Park and C. Tomlin. Investigating communication infrastructure of next generation air traffic management. In *Proceedings of the 2012 IEEE/ACM Third International Conference on Cyber-Physical Systems*, pages 35–44, April 2012.

120. P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, SOSP '01, pages 89–102, 2001.

121. B. Prabhakar, E. U. Biyikoglu, and A. E. Gamal. Energy-efficient transmission over a wireless link via lazy packet scheduling. In *Proceedings. of*

Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies, volume 1, 2001.

122. Z. Qin, Q. Li, and M.-C. Chuah. Unidentifiable attacks in electric power systems. In *Proceedings of the 2012 IEEE/ACM Third International Conference on Cyber-Physical Systems*, pages 193–202, April 2012.

123. D. Quevedo and D. Nesic. Input-to-state stability of packetized predictive control over unreliable networks affected by packet-dropouts. In *IEEE Transactions on Automatic Control*, volume 56, pages 370–375, Feb. 2011.

124. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic. Cyber-physical systems: The next computing revolution. In *Proceedings of Design Automation Conference*, pages 731–736, June 2010.

125. M. Saad, A. Leon-Garcia, and W. Yu. Optimal network rate allocation under end-to-end quality-of-service requirements. In *IEEE Transactions on Network and Service Management*, volume 4, pages 40–49, Dec. 2007.

126. A. Saifullah, Y. Xu, C. Lu, and Y. Chen. Real-time scheduling for wirelessHART networks. In *2010 IEEE 31st Real-Time Systems Symposium (RTSS)*, pages 150–159, Nov. 30-Dec. 3 2010.

127. A. Saifullah, Y. Xu, C. Lu, and Y. Chen. Priority assignment for real-time flows in WirelessHART networks. In *23rd Euromicro Conference on Real-Time Systems (ECRTS)*, pages 35–44, July 2011.

128. A. Saifullah, Y. Xu, C. Lu, and Y. Chen. End-to-end delay analysis for fixed priority scheduling in wirelessHART networks. In *2011 17th IEEE*

Real-Time and Embedded Technology and Applications Symposium (RTAS), pages 13–22, April 2011.

129. S. Samii, P. Eles, Z. Peng, and A. Cervin. Quality-driven synthesis of embedded multi-mode control systems. In *Proceedings of the 46th Annual Design Automation Conference*, pages 864–869, July 2009.

130. J. Schlick. Cyber-physical systems in factory automation - towards the 4th industrial revolution. In *2012 9th IEEE International Workshop on Factory Communication Systems (WFCS)*, page 55, May 2012.

131. N. Serreli and E. Bini. Deadline assignment for component-based analysis of real-time transactions. In *2nd Workshop on Compositional Real-Time Systems*, Dec. 2009.

132. N. Serreli, G. Lipari, and E. Bini. The distributed deadline synchronization protocol for real-time systems scheduled by EDF. In *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, Sept. 2010.

133. M. Sha, R. Dor, G. Hackmann, C. Lu, T.-S. Kim, and T. Park. Self-adapting mac layer for wireless sensor networks. In *IEEE 34th Real-Time Systems Symposium (RTSS)*, pages 192–201, 2013.

134. W. Shen, T. Zhang, M. Gidlund, and F. Dobslaw. *Wireless Networks*, 19(6): 1155–1170, 2013.

135. C.-S. Shih and J. W. S. Liu. State-dependent deadline scheduling. In *23rd IEEE Real-Time Systems Symposium*, pages 3–14, 2002.

136. S. Shih and S. Liu. Acquiring and incorporating state-dependent timing requirements. In *Proceedings. 11th IEEE International Requirements Engineering Conference*, volume 9, pages 121–131, 2004.

137. D. Shin, J. Kim, and S. Lee. Intra-task voltage scheduling for low-energy, hard real-time applications. In *IEEE Design Test of Computers*, volume 18, pages 20–30, 2001.

138. P. Soldati, H. Zhang, and M. Johansson. Deadline-constrained transmission scheduling and data evacuation in wirelesshart networks. In *proceeding of the 10th European Control Conference (ECC)*, 2009.

139. J. Song, S. Han, A. K. Mok, D. Chen, M. Lucas, M. Nixon, and W. Pratt. WirelessHART: Applying wireless technology in real-time industrial process control. In *Real-Time and Embedded Technology and Applications Symposium*, 2008.

140. W.-Z. Song, R. Huang, M. Xu, B. Shirazi, and R. LaHusen. Design and deployment of sensor network for real-time high-fidelity volcano monitoring. In *IEEE Transactions on Parallel and Distributed Systems*, volume 21, 2010.

141. M. Spuri. Analysis of deadline scheduled real-time systems. In *Technical Report, INRIA*, 1996.

142. P. Tabuada. Event-triggered real-time scheduling of stabilizing control tasks. In *IEEE Transactions on Automatic Control*, volume 52, pages 1680–1685, Sept. 2007.

143. J. Tang, G. Xue, C. Chandler, and W. Zhang. Link scheduling with power

control for throughput enhancement in multihop wireless networks. In *IEEE Transactions on Vehicular Technology*, volume 55, pages 733–742, May 2006.

144. A. Tarello, J. Sun, M. Zafer, and E. Modiano. Minimum energy transmission scheduling subject to deadline constraints. In *Third International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, 2005.

145. K. W. Tindell, A. Burns, and A. J. Wellings. Allocating hard real-time tasks: an NP-hard problem made easy. In *Real-Time Systems*, volume 4, pages 145–165, May 1992.

146. Y.-H. Wei, Q. Leng, S. Han, A. K. Mok, W. Zhang, and M. Tomizuka. RT-WiFi: Real-time high-speed communication protocol for wireless cyber-physical control applications. In *2013 IEEE 34th Real-Time Systems Symposium (RTSS)*, 2013.

147. Y. Yang, J. Hou, and L.-C. Kung. Modeling the effect of transmit power and physical carrier sense in multi-hop wireless networks. In *26th IEEE International Conference on Computer Communications*, pages 2331–2335, May 2007.

148. F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proceedings. of 36th Annual Symposium on Foundations of Computer Science*, 1995.

149. W. Ye, J. Heidemann, and D. Estrin. Medium access control with coordinated adaptive sleeping for wireless sensor networks. In *IEEE/ACM Transactions on Networking*, volume 12, pages 493–506, June 2004.

150. R. Yedavalli and R. Belapurkar. *Journal of Control Theory and Applications*, 9(1):28–33, 2011.

151. J. Yi. Dynamic reservation medium access for multihop wireless real-time communications. In *PHD Thesis, Department of Computer Science and Engineering, University of Notre Dame*, 2012.

152. J. Yi, C. Poellabauer, X. S. Hu, T. Chantem, and L. Zhang. Dynamic channel reservations for wireless multihop communications. In *SIGMOBILE Mob. Comput. Commun. Rev.*, volume 14, pages 43–45, Dec. 2010.

153. H. Yu and P. Antsaklis. Event-triggered real-time scheduling for stabilization of passive and output feedback passive systems. In *American Control Conference (ACC), 2011*, pages 1674 –1679, June 29-July 1 2011.

154. Y. Yu, B. Krishnamachari, and V. Prasanna. Energy-latency tradeoffs for data gathering in wireless sensor networks. In *Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 1, March 2004.

155. M. Zafer and E. Modiano. Optimal adaptive data transmission over a fading channel with deadline and power constraints. In *40th Annual Conference on Information Sciences and Systems*, 2006.

156. H. Zhang, P. Soldati, and M. Johansson. Optimal link scheduling and channel assignment for convergecast in linear wirelesshart networks. In *Proceedings of the 7th international conference on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, pages 1–8, June 2009.

157. H. Zhang, P. Soldati, and M. Johansson. Time- and channel-efficient link scheduling for convergecast in wirelessHART networks. In *IEEE 13th International Conference on Communication Technology (ICCT)*, pages 99–103, Sept. 2011.

158. Y. Zhang and R. West. End-to-end window-constrained scheduling for real-time and communication. In *IEEE 10th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 143–152, Aug. 2004.

159. Y. Zhang, R. West, and X. Qi. A virtual deadline scheduler for window-constrained service guarantees. In *Proceedings. of the 25th IEEE International Real-Time Systems Symposium*, pages 151–160, Dec. 2004.

160. W. Zheng, M. D. Natale, C. Pinello, P. Giusto, and A. S. Vincentelli. Synthesis of task and message activation models in real-time distributed automotive systems. In *Proceedings of the conference on Design, automation and test in Europe*, pages 93–98, Apr. 2007.

161. W. Zheng, M. D. Natale, C. Pinello, P. Giusto, and A. S. Vincentelli. Definition of task allocation and priority assignment in hard real-time distributed systems. In *28th IEEE International Real-Time Systems Symposium*, pages 161–170, Dec. 2007.

162. X. Zhong and C.-Z. Xu. Online energy efficient packet scheduling with delay constraints in wireless networks. In *The 27th Conference on Computer Communications*, pages 421 –429, 2008.

This document was prepared & typeset with LaTeX 2ε*, and formatted with* NDdiss2ε *classfile (v3.0[2005/07/27]) provided by Sameer Vijay.*