SCHEDULING ALGORITHMS OPTIMIZING THROUGHPUT AND ENERGY FOR NETWORKED SYSTEMS

by

Zhi Zhang
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

_____ Dr. Fei Li, Dissertation Director

_____ Dr. Hakan Aydin, Committee Member

_____ Dr. Songqing Chen, Committee Member

_____ Dr. Roman Polyak, Committee Member

_____ Dr. Dana Richards, Committee Member

_____ Dr. Sanjeev Setia, Department Chair

_____ Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering

Date: _____ Summer Semester 2013
George Mason University
Fairfax, VA

Scheduling Algorithms Optimizing Throughput and Energy for Networked Systems

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Zhi Zhang
Master of Science
Wuhan University, China, 2008
Bachelor of Science
Wuhan University, China, 2006

Director: Dr. Fei Li, Professor
Department of Computer Science

Summer Semester 2013
George Mason University
Fairfax, VA

Dedication

I dedicate this dissertation to my family.

Acknowledgments

My deepest gratitude goes first to my Ph.D. advisor Professor Fei Li. I truly thank him for his guidance and constant encouragement throughout the past 5 years. He let me have the opportunity to pursue my Ph.D. study and access to these interesting research areas. I benefit tremendously from his vision and technical insights. He also helped me come up with many great ideas without which this dissertation will be nowhere at all.

In 2012, I did my summer research intern at Bell Labs. My mentors were Dr. Yigal Bejerano and Dr. Spyridon Antonakopoulos. They inspired me of doing solid research in solving important industrial problems. I worked closely with them at Bell Labs. Their broad knowledge and attitudes toward doing research always encourage me to become an excellent researcher. I thank Yigal and Spyridon for the opportunity of working with them at Bell Labs. Chapter 5 is our joint work.

I would like to thank my Ph.D. dissertation committee members — Professor Roman Polyak, Professor Dana Richards, Professor Hakan Aydin, and Professor Songqing Chen. During my Ph.D. study, I took classes with them and was greatly benefit from their profound knowledge and outstanding teaching. This also led to the publications of some papers: The joint work with Professor Songqing Chen led to part of Chapter 2. The joint work with Professor Hakan Aydin led to the work in Chapter 4 which I applied what I learned in Professor Roman Polyak's class.

Table of Contents

List of Tables

List of Figures

Abstract

SCHEDULING ALGORITHMS OPTIMIZING THROUGHPUT AND ENERGY FOR NETWORKED SYSTEMS

Zhi Zhang, PhD

George Mason University, 2013

Dissertation Director: Dr. Fei Li

Scheduling problems consider allocating limited resources under constraints among competing requests in order to fulfill their obligations. Practical resource management algorithms with provable performance guarantees are of great importance. In this dissertation, we study scheduling algorithms for resource management in networked systems. Mainly, we design, analyze, and implement two types of scheduling algorithms: (1) throughput-aware scheduling algorithms, and (2) energy-aware scheduling algorithms.

Throughput-aware scheduling algorithms. Throughput is a main metric that scheduling algorithms are designed to optimize. We study algorithms for network routers to schedule weighted packets with time constraints over a wireless fading channel. The wireless channel's fading state determines the channel's achievable throughput per unit time. However, the fading state may change over time and is unknown to network routers. We design both offline and online algorithms to maximize weighted throughput, which is defined as the total weight of the packets successfully sent before their respective deadlines. We give theoretical proofs of the offline algorithm's optimality and its running time complexity. For practical considerations, we design online algorithms and analyze their performance using

competitive ratio, which has been widely used as a theoretical metric in evaluating online algorithms' performance. We also develop an online learning algorithm and compare its performance with the best expert in hindsight using external regret.

Energy-aware scheduling algorithms. Energy consumption is very critical to computers and networked systems, which has become an important performance metric in designing scheduling algorithms for such systems. Two advanced hardware techniques, *dynamic power management* (DPM) and *dynamic voltage scaling* (DVS), have been widely employed in designing energy-aware scheduling algorithms.

DPM-based scheduling algorithms. DPM is an architecture-level design based on clock gating or power gating. DPM is commonly used to cut a system's energy cost via eliminating or reducing power consumption of one or more of its components. A constant amount of energy called transition energy is usually associated with power state transition. Thus, suspending a system is beneficial only when the idle and sleep time (without running jobs) is long enough. In this dissertation, we consider DPM-based scheduling algorithms for network routers and Points of Presence (PoPs).

For network routers, we first study a problem of scheduling jobs with values and deadlines to maximize net profit, which is defined as the difference between the revenue obtained from the jobs sent before their respective deadlines and the cost of total energy consumption during this course. This model and its variants are studied in details and we provide this model's hardness analysis and running time complexities. Suspending a router's forwarding mechanisms to save energy incurs jobs' increased delays. Motivated by this, we study the trade-off between energy consumption and jobs' flow time or stretch in an online setting. We design bi-criteria power-down algorithms optimizing both and analyze their performance using competitive ratio.

For PoP design, current IP core networks operate at a nearly constant power rate independent of the traffic load. Thus, the gap between the available network capacity and the temporal traffic demand presents opportunities for reducing network power consumption

by deactivating network components without noticeably affecting network performance. We study a theoretical model for PoP design. In this model, a PoP has multiple chassis and each chassis can accommodate multiple traffic links. A chassis has to be at the active state with energy consumption when one or more of its assigned traffic links have traffic flows. The objective is to find out an assignment between traffic links and chassis such that the total energy cost of the PoP is minimized. We analyze the hardness of this model and design several approximation algorithms with proved near-optimal performance.

DVS-based scheduling algorithms. DVS is a technique that adaptively changes a processor's voltage and frequency levels to meet service performance requirements. Energy consumption can be reduced due to a physical semi-conductor law that in some scenarios, executing a job at a lower clock frequency uses less energy than that at a higher clock frequency.

For a given processor, each speed change involves time and energy overhead, as well as a negative impact on its lifetime reliability. Motivated by this observation, we study theoretical energy-aware scheduling problems by considering the number and the cost of processor's speed changes. Four related problems based on this framework are studied. By using convex programming techniques, we develop polynomial-time algorithms that perform arbitrarily close to the optimal solutions.

Research contributions. The research in this dissertation provides provable algorithmic solutions to several fundamental problems on performance and profitability in networked systems. Novel algorithmic design techniques are developed to optimize throughput and energy cost for networked systems, which provide insights into fundamental problems of optimal resource allocation.

Chapter 1: Introduction

1.1 Scheduling Problems

Scheduling problems consider allocating limited resources among competing requests in order to fulfill their obligations [1]. This kind of problems arises in many areas in our daily lives; for example, scheduling of flights competing for runways, scheduling of assembling parts of automobiles competing for assembling machines, and scheduling of classes competing for classrooms. Such kind of problems requires smart combinatorial optimization techniques rather than simple heuristic approaches when the input sizes grow out of hand. This gives the rise of *scheduling theory*. In this dissertation, we develop effective and efficient scheduling algorithms for some problems in networked systems with great algorithmic interests and practical importance.

1.2 Online Algorithms and Competitive Analysis

In general, scheduling algorithms are designed in two kinds of settings: *offline setting* and *online setting*. In the *offline setting*, the inputs are given completely beforehand and our task is to design algorithms to optimize some objectives subject to the resource constraints. In the *online setting*, the whole input may not be available beforehand and an online algorithm has to make decisions before all the inputs complete. An *online algorithm* makes non-revocable scheduling decisions upon a request's arrival without the information of future input sequence. Some scheduling problems that we consider in this dissertation are intrinsically online. Thus, it is preferred to design online algorithms for them.

Consider an example called the *bin-packing problem* [2, 3]. Given a set of items with their sizes in the range of $(0, 1)$ and all bins with size 1, we put one or more items in one

bin as long as the sum of their sizes is less than or equal to 1. The objective is to minimize the number of bins used to accommodate all the items. If the set of items are given in advance, then this problem is an offline problem. If the items are known one by one when they arrive, then we have to determine in an online manner whether a newly arriving item should be grouped with some already-known items in the bins or this newly item should be put into a new bin. Once a decision of putting an item into a bin is made, it cannot be revoked. This problem is therefore an online problem.

In order to evaluate the worst-case performance of an online algorithm, we compare it with an optimal offline algorithm. The offline algorithm is a clairvoyant algorithm, empowered with the information of the whole input sequence in advance to make its decision. A *competitive online algorithm*, on the contrary, does not know the input sequence beforehand and has a request information only when it arrives. We use *competitive ratio* as the worst-case metric. In contrast to stochastic algorithms that provide statistical guarantees under some mild assumptions over the input sequences, competitive online algorithms guarantee the worst-case performance.

Definition 1.1 (Competitive Ratio [4])**.** *Given a maximization (respectively, minimization) problem, an online algorithm ON is called* k-competitive *if its objective for* any *instance is at least* $1/k$ *(respectively, at most* k *times) of the objective achieved by an optimal offline algorithm for this instance. For a maximization problem,*

$$k := \max_{\mathcal{I}} \frac{OPT(\mathcal{I}) - \delta}{ON(\mathcal{I})},$$

where $OPT(\mathcal{I})$ *is the optimal solution of the input* $\mathcal{I}$ *and* $\delta > 0$ *is a fixed constant (*δ *becomes insignificant when the size of the input* $|\mathcal{I}|$ *increases). The parameter* k *is known as the online algorithm's* competitive ratio*. We also call the optimal offline algorithm* adversary*.*

The *upper bounds* for competitive ratios are achieved by some known online algorithms.

A competitive ratio less than the *lower bound* is not achievable by any online algorithms. An online algorithm is said to be *optimal* if its competitive ratio achieves the lower bound. If the additive constant δ is no larger than 0, then the online algorithm ON is called *strictly k-competitive*. Competitiveness has been widely accepted as the metric to evaluate an online algorithm's worst-case performance in theoretical computer science and operations research [4].

1.2.1 The necessity of employing worst case analysis and extensions

In this dissertation, we primarily use worst-case analysis (such as worst-case asymptotic running time, approximation ratio, and competitive ratio) [4–7] in evaluating the proposed algorithms rather than probabilistic analysis [8,9]. Worst-case analysis is the only choice in situations when the input cannot be modeled or predicted precisely. Compared to probabilistic analysis, worst-case analysis better exposes the fundamental properties of the models and the limitations of the algorithms, and thus has significant implications from theoretical perspectives. In addition, for those mission-critical applications with zero-tolerance of failure, worst-case guarantee is necessary.

Competitive analysis is certainly not the only way to evaluate the performance of an online scheduling algorithm with uncertainty of future input instance. For example, if we have a reasonable approximation of the input probability distribution, *average-case analysis* can be done either analytically [7, 8, 10] or experimentally. However, when such information is unavailable or unreliable, and/or when *analytical worst-case performance guarantees* are needed, competitive analysis is of fundamental importance. Note that in competitive analysis, the input can be intentionally generated in an adversarial manner to worsen the online algorithm's competitive ratio. Thus, sometimes, competitive ratio provides the worst-case guarantees but they are pessimistic.

Note that competitive analysis is sometimes too pessimistic as the adversary is empowered to change the input according to what the online algorithm does over time. Many alternative metrics have been proposed (see [4] and the references therein). In this dissertation,

we also consider *external regret* as a metric in measuring online algorithms' performance using *online learning approaches*. Motivated by Occam's razor (a principle stating that "the simplest explanation is usually the correct one"), we assume that there exist multiple *experts* (i.e., algorithms) following simple but fixed policies for the problem that we consider. The experts' policies are not changed dynamically over time. Then the best expert is defined as the one optimizing the objective in hindsight. When we design online algorithms, instead of comparing them with the optimal offline algorithm, we compare online algorithms with the best expert in hindsight. The difference between the online algorithm and the best expert in hindsight is called *external regret*. Use news-vendor problem [11] as an example. The newspaper boy buys newspapers and resells them to customers on the streets. He does not know future demands on newspapers. The revenue is the gain from the newspapers sold minus the money that he pays for all the newspapers he buys. Let us assume there are multiple experts. Each expert predefines a quota of the newspaper that the expert buys for every day. For a fixed number of days, the best expert will be the one with the maximum revenue in hindsight. The online learning algorithm has the flexibility of changing the newspaper boy's quota each day but lacks the power of foreseeing the future demands.

1.3 Problems Studied

In this dissertation, we study efficient and effective scheduling algorithms for resource management in networked systems and energy-critical systems. The problems studied in this dissertation can be roughly grouped into the following two classes:

1.3.1 Throughput-aware scheduling problems

In the first part of my dissertation (Chapter 2), we study a throughput-aware scheduling problem motivated by providing better Quality-of-Service (QoS) for wireless networked systems. This model generalizes the well-studied model — *bounded-delay model* for QoS

buffer management in network routers [12].

In the model, packets arrive at a router over time and they compete for the outgoing link's bandwidth. Packets are unit-length and each packet has a deadline. A packet is said to be *successfully scheduled* if it is transmitted completely before its deadline. Throughput is determined by the number of packets scheduled successfully. Sometimes, it is more appropriate to consider the differentiation among packets from different types of applications. Therefore, in the bounded delay model, each packet is associated with a non-negative weight which indicates its level of importance. Then the objective becomes to maximize weighted throughput, defined as the total weights of all successfully sent packets. Generalizing this model to the wireless channels, we consider scheduling packets over a fading channel. The fading channel's state changes over time and the success of delivering a packet depends on the channel's signal strength. (If the fading state is perfect all the time, then this model becomes the well-studied bounded delay model.) This additional restriction (fading states) complicates algorithm design and analysis. The objective is to maximize the weighted throughput. Our study on designing efficient scheduling algorithms for this model aims at improving the performance of the wireless networked systems.

1.3.2 Energy-aware scheduling problems

Considering energy issues in designing scheduling algorithm has theoretical and practical significance. Due to the huge operational costs to maintain the sustainability of computing infrastructures and the increasing emission of greenhouse gases such as CO_2 into the global environments, effective power management has become a critical issue [13]. According to the United States Environmental Protection Agency's report [14], "The energy used by the nation's servers and data centers is significant. ..., (in 2006) servers and data centers alone account for approximately 61 billion kilowatt hour, for a total electricity cost of about 450 million annually. ..., (the energy use) is estimated more than doubled the electricity consumed for this purpose in 2006. Under current efficiency trends, national energy consumption by servers and data centers could nearly double again in another five years to

more than 100 billion kilowatt hour". High power consumption also leads to destructive effects of high current density and heat dissipation. If the temperature is raised beyond the safety limit, processors and devices may work erratically or even halt. Any organization running large-scale computing systems or data centers has a strong urge of reducing power from both engineering and economic concerns [15]. Possible strategies towards energy management have been commented by Steven Chu, former U.S. Secretary of Energy (2008) [16], "A dual strategy is needed to solve the energy problem: (1) maximize energy efficiency and decreases energy use — This will remain the lowest hanging fruit for the next few decades; and (2) develop new sources of clean energy." As a successful case, by employing energy-aware scheduling algorithms, Kyoto University has saved 0.2 million dollars per year [15].

In the second part of my dissertation (Chapter 3 — Chapter 5), we design energy-aware scheduling algorithms. We focus our study on operating system level algorithms to maximize energy usage efficiency and to minimize energy consumption, and still maintain the system performance such as throughput, job's flow time, and chip's lifetime reliability. The main hardware techniques supporting algorithms to reduce energy consumption can be broadly classified into two categories: *dynamic voltage scaling* (DVS) and *dynamic power management* (DPM).

- DVS is a technique to dynamically set a processor's voltage and frequency levels to meet service performance requirements. Consider a processor in the DVS setting. The processor has variable clock frequencies. Under frequency s, the processor consumes energy $P(s)$ per unit time. In general, the function $P(\cdot)$ is convex. The total energy consumed by a schedule is the integral of energy consumption over time, i.e., $E = \int P(s(t))dt$ where $s(t)$ is the processor frequency at time t. Thus, energy consumption can be reduced since executing a job at a lower clock frequency uses less energy than that at a higher clock frequency.

- DPM is an architecture-level design based on a function component called *clock gating*

or *power gating*. A system equipped with DPM functionalities has a higher-power ACTIVE state and one or more lower-power SLEEP or STANDBY states. Jobs can be processed only when the system is in ACTIVE state but not in any lower-power state. DPM is commonly used to reduce a system's energy consumption by setting one or more of its components into the lower-power state. A constant amount of energy called *transition energy* is usually associated with transitions between the higher-power state and the lower-power states.

In addition to the challenges of the classic scheduling problems that we have to deal with, the following two challenges complicate the design of job scheduling algorithms when energy consumption is taken into account.

1. Energy is a kind of resource neither sharable nor renewable without extra cost.

   Different from resources such as processor availability or job queue sizes in classic scheduling problems, energy is neither sharable nor renewable without extra cost. This makes allocating energy in an online manner challenging. An energy-efficient algorithm needs to specify not only *when to execute a job* but also *at which speed to run the job* and/or *how much energy is allocated to the job*.

2. Mixed objectives involving job completion time and energy consumption may have inherent conflicts.

   For example, objectives of minimizing flow time and minimizing energy consumption are two orthogonal objectives. If we want to save energy, then we have to use a slower speed which unavoidably extends a job's flow time. On the other hand, if we want to complete job in a shorter time by running the processor at a higher speed, then more energy has to be consumed.

In this dissertation, we design scheduling algorithms for energy-aware scheduling problems in systems equipped with DVS and DPM technologies.

1.4 Thesis Organization

The research in this dissertation provides solutions to several fundamental algorithmic problems on performance and profitability of networked systems and energy-critical systems. The goals are to understand the mathematical structure of these problems, design elegant and easy-to-implement offline and online algorithms, and provide mathematically rigorous analysis on their performance bounds. We expect that this research produces robust and insightful algorithm design and analysis techniques for scarce resource management in throughput-aware scheduling problems and energy-aware scheduling problems.

For throughput-aware scheduling problems, we design and analyze offline and online packet scheduling algorithms over fading channels for better resource utilization in Chapter 2. Furthermore, we design online learning algorithms to evaluate the power of learning fading signal strength in this model.

Discussion of the research related to energy-aware scheduling focuses on systems capable of using DPM and DVS, respectively. In Chapter 3, we consider two models. One is a *net profit* model of scheduling jobs with deadlines in which costs of energy used in completing jobs should be paid. In the other model, to schedule jobs without deadlines, we study the trade-offs between saving energy and hazarding users' experience — flow time or stretch. These problems are for system equipped with DPM techniques. In Chapters 4, we study energy-efficient scheduling problems in a single-machine system equipped with DVS techniques. We mainly focus on investigating energy-aware real-time scheduling algorithms with speed change constraints. In Chapter 5, we design topologies for DMP-based PoPs to save energy cost. We design and theoretically analyze several approximation algorithms with near-optimal performance.

Other research that I have done during my Ph.D. study is included in Appendix C.

Chapter 2: Scheduling Packets over a Wireless Channel

2.1 Motivation

Motivated by providing various Quality-of-Service to Internet users, we consider buffer management within network routers. Figure 2.1 illustrates the functionalities of the buffer management inside a router. A buffer management algorithm is in charge of two tasks — *packet queuing* and *packet delivery*. When new packets arrive, a buffer management algorithm decides which ones to admit and queue for potential deliveries, and which ones already in the buffer to drop permanently due to the constraints from packet deadlines or from the buffer capacity. In each time step the buffer management algorithm selects a packet in the buffer to send.

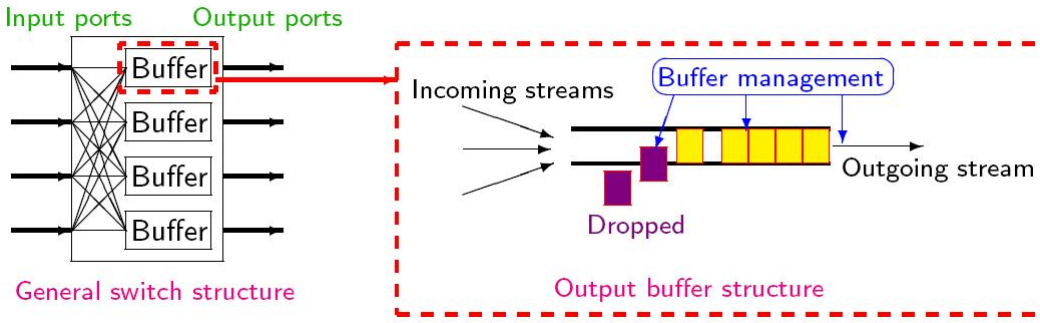

Figure 2.1: Buffer management inside routers.

Consider discrete time. For wired networks, one packet can be delivered in each unit of time. For wireless networks, the throughput rate (in other words, how many packets delivered in a unit of time) depends on the wireless signal strength. This signal strength, called *fading channel state* or *fade state*, is changing over time. In this research, we make no assumptions over the stochastic properties of the fade states. In this chapter, we propose algorithms to schedule weighted packets with time constraints over a wireless fading channel.

2.2 Model Description

In this model, time is assumed discrete. Each unit of time is called a *time step* and a few continuous time steps are called a *time interval*. Packets arrive at a router with buffer size $B \in \mathbb{Z}^+$ over time. All packets are with the same length $l \in \mathbb{R}^+$ (l is a constant). Each packet p has an integer *release time* (*arriving time*) $r_p \in \mathbb{Z}^+$, a positive real value $w_p \in \mathbb{R}^+$ to represent its *weight* (*value*), and an integer deadline $d_p \in \mathbb{Z}^+$. We also use the pair (w_p, d_p) to denote a packet p. The time required to send a packet depends on the *state quality* q_t ($q_t \in [q_{\min}, q_{\max}]$) of the fading channel during a time step t. For simplicity, we assume $q_{\min} = 0$ and $q_{\max} = l$ such that if the fading channel is at its highest quality $q_{\max}$, one packet can be sent in a time step. Without loss of generality, we assume the fade state in a single time step keeps unchanged. A packet has to be sent in consecutive time steps. Completely sending a packet p takes $t(p)$ time steps subject to

$$\sum_{t=t_s}^{t_e} q_t \geq l, \qquad t_s, t_e \in \mathbb{Z}^+,$$

where the packet sending process begins at time t_s and ends at time t_e. Therefore, $t(p) = t_e - t_s + 1$. If $[t_s, t_e] \subseteq [r_p, d_p]$, then we say that the packet p is successfully sent before its deadline d_p, and its weight w_p is contributed to our objective.

A schedule S is a matching between packets and time steps such that these packets are all successfully sent before their respective deadlines subject to the constraint stated above. The weighted throughput W of S is $W = \sum_{p \in S} w_p$.

Here, we enforce that two or more packets cannot be sent in the same time step. For each packet p, we associate an indicator variable $\chi_p(t)$ defined as below.

$$\chi_p(t) = \begin{cases} 1, & \text{if time step } t \text{ is matched to packet } p \\ 0, & \text{otherwise} \end{cases}.$$

For each time step t and all the packets sent in schedule S, we have $\sum_{p \in S} \chi_p(t) \leq 1$. The objective is to maximize the weighted throughput W subject to the deadline constraints of packets and the varying fading channel states. The model can be an *overloaded system* (defined in Appendix A) such that due to packets' deadline constraints, no algorithm can deliver all packets in the input instance, even if the fading channel state is at its maximum all the time. Hence, some packets have to be dropped in this case. Note that in an *underloaded system* (defined in Appendix A), the classic algorithm EDF (Earliest-Deadline-First) delivers all the packets before their deadlines and it is optimal in both offline and online settings.

We design two kinds of algorithms: *offline algorithms* and *online algorithms*. All input information (including the fading channel states, and the packets' release times, deadlines, and weights) is known to an offline algorithm in advance. For an online algorithm, the complete packet input sequence is unknown beforehand and a packet's deadline and weight are known to the algorithm only at the time when it arrives at the router. The fade state of the channel is unknown or partially known to the online algorithm, depending on the assumptions in the variants of the online version of this problem.

We also apply online learning approaches to the online version of this model to examine possible effects on algorithms' performance which learning could bring. The main idea is learning from *experts'* advices, in which an expert fixes one policy at the beginning of the schedule. Instead of designing competitive online algorithms, we design a group of online algorithms and refer them as experts among which the online learning algorithm chooses one to follow in each time step. Instead of comparing the performance with the optimal offline algorithm, we compare an online learning algorithm with the expert having the best performance in hindsight.

2.3 Previous Work

Scheduling packets over fading wireless channels has received much attention (see [17–21] and the references therein). A scheduling algorithm can significantly improve the communication performance by taking advantages of the changing channel states. In previous studies, the objective is usually to maximize the total number of packets delivered by their deadlines. However, for many practical problems, it is more reasonable to differentiate various packets and take into account the amount and/or the significance level of the information associated with the packets. Thus, in this chapter, we address the problem of optimizing *weighted* throughput of packets with time constraints in a fading wireless channel. Our results show that the algorithmic solutions for maximizing weighted throughput as well as their computational complexity are significantly different from those optimizing throughput of packets with identical values.

Resource allocation for fading channels has been a well-studied topic in the area of information theory. The quantity to maximize is often the *Shannon capacity*, defined as the tightest upper bound of the amount of information (i.e., the total number of packets) that can be reliably transmitted per unit time over a communication channel. Tse and Hanly [17] have found capacity limits and optimal resource allocation policies for such fading channels. They also studied the greedy approach for channel allocations in multi-access fading channels, assuming all packets arriving at the router are successfully delivered. Prabhakar et al. [22] have considered proactively adjusting the rate of packet transmission for saving energy where the quality of the fading channel is assumed to be fixed and the energy consumption rate is a convex function of the transmission speed. The discrete version of this algorithm has been proposed in [23] for a more general problem setting. In [18], the authors applied a dynamic programming approach to getting the optimal solution for scheduling uniform-value packets under both time and energy constraints. However, this algorithm in [18] runs in exponential-time in overloaded systems. A polynomial-time optimal offline solution of scheduling packets with hard deadlines was given in [19, 20]. In their problem

settings, energy is minimized under the assumption that all arriving packets are successfully delivered. An optimal offline algorithm maximizing throughput and a heuristic online approach of scheduling identical-value packets with different deadlines were given in [21]. No theoretical analysis has been provided for the heuristic online solution.

Note that in these previous studies, packets have identical values and their arrivals at the router are usually modeled by a Poisson distribution. However, packets from different users and various applications may have different significance levels of embedded information. For the sake of being realistic and practical, we associate packets with *weights* (*values*) to indicate the significance of their embedded information. None of the previous algorithms for delivering packets can be generalized to this problem setting, because a schedule with the maximum throughput does not imply its optimality on maximizing weighted throughput. Moreover, we made no probabilistic or stochastic assumption over the incoming traffic which generalizes the previous work.

We have realized the connection between this problem and the *bounded-delay model* in buffer management. For the bounded-delay model (the buffer size is assumed to be infinite), an optimal offline algorithm has been proposed in [12], running in $O(n \log n)$ time where n is the number of packets released. For online algorithms, the best known lower bound of competitive ratio of deterministic algorithms is $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$ [24–26]; this lower-bound also applies to instances in which the deadlines of the packets (weakly) increase with their release dates. A simple greedy algorithm that always schedules the maximum-value pending packet in the buffer is 2-competitive [12, 24]. For a variant in which the deadlines of the packets (weakly) increase with their release times, Li et al. [27] proposed an optimal deterministic ϕ-competitive algorithm. Using the same analysis, but in a more complicated way, Li et al. provided a ($\frac{3}{\phi} \approx 1.854$)-competitive deterministic algorithm [27] for the general model. Independently, Englert and Westermann presented a 1.894-competitive deterministic memoryless algorithm and a ($2\sqrt{2}-1 \approx 1.828$)-competitive

deterministic algorithm [28]. Closing the gap $[1.618, 1.828]$ of competitive ratio for deterministic algorithms is a difficult open problem. A randomized online algorithm with a competitive ratio of $\frac{e}{e-1} \approx 1.582$ was proposed in [29]. The lower bound of competitive ratio of randomized algorithms is 1.25. How to tighten the gap $[1.25, 1.582]$ in the randomized bounded-delay model remains open. If the buffer size is a finite number, the generalization of the bounded-delay model is called a *bounded-buffer model*. In [30], a 3-competitive deterministic algorithm and a ($\phi^2 \approx 2.618$)-competitive randomized algorithms were given. Fung [31] provided a 2-competitive deterministic algorithm and Li presented an alternative proof [32]. When the number of size-bounded buffers is more than 1, Azar and Levy [33] provided a 9.82-competitive deterministic algorithm and Li [34] improved the competitive ratio to $3 + \sqrt{3} \approx 4.723$. Note that the bounded-delay model [12, 24, 27, 28, 35] implicitly applies an assumption of ideal channel quality such that in every time step, one packet can be delivered. However, things may become much more complicated when channel quality varies over time, and new algorithms need to be built to tackle with this challenge.

In this chapter, we design offline algorithms, competitive online algorithms and online learning algorithms to maximize weighted throughput for packets with time constraints over a fading channel.

2.4 Algorithms and Analysis

We classify our algorithms and present them as offline algorithms, online algorithms and online learning algorithms in Section 2.4.1, Section 2.4.2 and Section 2.4.3, respectively. Note that in designing offline algorithms, there is no difference between these two settings, *non-preemption* and *preemption-restart* (see their definitions in Appendix A). Let the input sequence be $\mathcal{I}$ and $|\mathcal{I}| = n$. All packets have the same length l.

2.4.1 Offline algorithms

In this section, we present a few exact (optimal) algorithms running in polynomial time for several variants, assuming all input information is known beforehand. At first, we note the following result.

Theorem 2.1. *[36] Assume the fading channel has a fixed quality $q \in [q_{\min}, q_{\max}]$ during all time steps. If all packets are with the same value, then there exists an exact polynomial-time optimal algorithm running in time $O(n \log n)$, where n is the number of packets released.*

We then consider an important variant in which packets are with *agreeable deadlines*, i.e., for any two packets i and j, $r_i < r_j$ implies $d_i \leq d_j$. This variant allows an optimal algorithm running in an online manner. Here, we study the Earliest-Deadline-First (EDF) algorithm, which is one of the most studied policies in the area of real-time scheduling. Note that EDF runs in an online manner. We have the following result.

Theorem 2.2. *Assume the fading channel has a fixed quality $q \in [q_{\min}, q_{\max}]$ during all time steps. If all packets are with the same value and if they are with agreeable deadlines, then EDF is an exact polynomial-time optimal algorithm running in linear time $O(n)$.*

Proof. To prove Theorem 2.2, it is sufficient to show that at any time t (t does not have to be an integer), EDF finishes no fewer packets than any algorithm ALG. Let $A(\mathcal{I})$ denote the throughput of an algorithm A with input $\mathcal{I}$. The proof (to show $EDF(\mathcal{I}) \geq ALG(\mathcal{I})$) consists of proving the following two parts:

1. Given any algorithm ALG and the set of packets $\mathcal{I}'$ ($\subseteq \mathcal{I}$) that ALG schedules, we can create an earliest-deadline-first scheduler EDF$'$ finishing all packets in $\mathcal{I}'$ before their deadlines. That is,

$$EDF'(\mathcal{I}') = |\mathcal{I}'| = ALG(\mathcal{I}). \tag{2.1}$$

2. Given the input $\mathcal{I}$ for EDF and the input $\mathcal{I}'$ for EDF$'$, EDF is no worse than EDF$'$

in the number of packets finished before their deadlines at any time t. That is,

$$EDF(\mathcal{I}) \geq EDF'(\mathcal{I}'). \tag{2.2}$$

Equation (2.1) and Equation (2.2) imply $EDF(\mathcal{I}) \geq ALG(\mathcal{I})$.

At first, we note that if the fading channel has a fixed quality q, for any packet p with length l, it takes $\lceil l/q \rceil$ time steps to deliver p. Given the set of packets $\mathcal{I}'$ finished by an algorithm ALG as the input of EDF$'$, we can use the *exchange argument* to show that EDF$'$ can finish the packets in $\mathcal{I}'$. Secondly, since all packets are with the same value and processing time, we can always replace the packets $\in (\mathcal{I}' \setminus \mathcal{I})$ using the packets $\in (\mathcal{I} \setminus \mathcal{I}')$ with no later release times or no later deadlines. Thus, the second part of the proof is true as well.

Finally, we study EDF's running time complexity. If packets are with agreeable deadlines, newly arriving packets can be appended at the end of the packet queue. EDF sends the first pending packet which has not expired yet in the next $\lceil l/q \rceil$ time steps. The scheduling algorithm runs in linear time $O(n)$. □

In the following, we prove that there exists an optimal offline policy for this problem in the general setting. First, we assume that the channel's quality is a fixed constant. Then, we apply this algorithm to the general setting in which the fade states of the channel vary over time.

Theorem 2.3. *Assume the fading channel has a constant quality $q \in [q_{\min}, q_{\max}]$ during all time steps. Then, there exists an optimal algorithm in maximizing weighted throughput.*

Before proceeding to the proof of Theorem 2.3, we would like to point out that since it may not be feasible to deliver all packets ever arrive at the router in an overloaded system, the optimal solutions in the previously studied models in [18, 21, 37] cannot be directly applied to our model. Instead, we design an exact algorithm that depends on the following two critical observations (on the matroidal structure of the model).

Remark 2.1. *Given a set S of packets, any feasible schedule on S can be converted to an earliest-deadline-first schedule in which the earliest-deadline packet $\in S$ is scheduled as long as it is available for the router.*

Remark 2.2. *Denote S^* as both the optimal solution maximizing the weighted throughput and the set of packets delivered. If a packet $p_j \in S^*$ is pending at time t and it is not scheduled at time t, then there must exist a packet $p_i \in S^*$ such that $r_{p_i} \leq t + \lceil \frac{l}{q} \rceil$ and p_i is scheduled at time r_{p_i}.*

Proof of Theorem 2.3. Let the set of packets arriving at the router be $\{p_1, p_2, \ldots, p_n\}$. It takes $\lceil l/q \rceil$ continuous time steps to deliver one packet. The set of time steps that a packet can be sent is a subset of all the time steps T,

$$T := \bigcup_i [r_{p_i}, r_{p_i} + n \cdot \lceil \frac{l}{q} \rceil], \tag{2.3}$$

where q is the constant channel quality. Let the time steps in T be $t_1, t_2, \ldots, t_m$, where $|T| \leq n \cdot n \cdot \lceil \frac{l}{q} \rceil \leq n^2 \cdot \frac{l}{q} + n^2$.

We have a greedy algorithm as follows. Based on Remark 2.1 and Remark 2.2, if there are two pending packets available for delivery, we can always pick one with the earlier deadline to send in a time step $\in T$. We call this order a *canonical order*. Our following algorithm is based on the matroidal property of the model.

Algorithm 1 Offline-Optimal($\mathcal{I}$)

```
Initialize the set of packets to be sent P' = ∅.
Initialize the set of packets to be considered P = I (= {p_1, p_2, ..., p_n}).
while |P'| ≤ n and there are packets left in P do
    remove the maximum-value packet p from P;
    if the set P' ∪ {p} can be feasibly scheduled in T under the canonical order (i.e., all
    packets can be sent before their deadlines) then
        insert the packet into P' and update P' as P' ∪ {p}.
    end if
end while
return P'.
```

The generated schedule in P' is the optimal solution and its correctness is based on the fact that feasible schedules form a matroid. The running time of this algorithm is $O(n \cdot \log n + n \cdot |T|) = O(n \log n + \frac{l}{q} \cdot n^3 + n^3) = O(n^3)$. For each packet p, it takes time $O(|T|)$ to verify the feasibility of adding p into the existing schedule. For this case, our result improves the algorithm in [38], whose running time is $O(n^{10})$ and which also holds when q is fixed but not a constant value. □

Following the proof of Theorem 2.3, we immediately have the following result.

Corollary 1. *Consider scheduling weighted packets with deadlines in a fading channel. There exists an optimal algorithm in maximizing weighted throughput in time $O(n \cdot \log n \cdot m)$, where m is the number of time steps we consider.*

In the model, as long as an interval with time steps $[t_s, t_e]$ has $\sum_{t=t_s}^{t_e} q_t \geq l$, a packet can be sent. For each release time r_p, we seek the following n consecutive time intervals such that for each time interval $[t_s, t_e]$, we have $\sum_{t=t_s}^{t_e} q_t \geq l$. Let the union of all such time steps be T'. Then, the number m in Corollary 1 is $|T'|$.

Note that our proof of Theorem 2.3 depends on the following three assumptions (1) all packets have the same length, (2) packets are sent continuously, and (3) packets do not share a time step. If any one of these assumptions does not hold, we conclude that the offline version of this problem cannot be found in polynomial time unless P = NP[1].

Theorem 2.4. *Consider packet scheduling in fading channels. Assume a packet can be preempted before the router finishes it. Only its unfinished part of the packet is resumed later. Then, maximizing weighted throughput is a NP-hard problem, even if all packets share a common release time and a common deadline.*

[1]A problem D_1 belongs to P if there exists an exact polynomial-time algorithm. A problem D_2 belongs to NP if a given candidate solution of D_2 can be verified in polynomial time. For a *NP-complete problem* D_3, $D_3 \in$ NP and if D_3 can be solved in polynomial time, then so can every problem in NP. Although any given solution to a NP-complete problem can be verified quickly, there is no known efficient way to locate a solution unless P = NP.

Proof. To prove the NP-hardness, it is sufficient to show that we can reduce a well-known NP-complete problem to our problem in polynomial time. To prove Theorem 2.4, the remaining work is to reduce the NP-hard TWO-PARTITION problem to our problem.

The TWO-PARTITION problem is defined as follows. Given an instance that has a finite set $\mathcal{I}$ and a size $s_i \in \mathbb{Z}^+$ for $i \in \mathcal{I}$, the objective is to find out if there exists a subset $\mathcal{I}' \subseteq \mathcal{I}$ such that $\sum_{i\in\mathcal{I}'} s_i = \sum_{i\in\mathcal{I}\setminus\mathcal{I}'} s_i$. This problem is NP-complete [39].

Here is the NP reduction. Given any instance $\mathcal{I}$ of the TWO-PARTITION problem, we normalize $\mathcal{I}$ such that $\sum_{i\in\mathcal{I}} s_i = 2 \cdot l$. Then we generate the channel quality $q_i = s_i$ for each $i \in \mathcal{I}$ and we have two packets whose release times are 0, weights are 1, and deadlines are $\sum_{i\in\mathcal{I}} s_i = 2 \cdot l$. This conversion takes polynomial time. Consider any algorithm ALG. If ALG returns a throughput of 2, then ALG returns two sets of fading states such that each of them is with a total quality $\sum_j q_j = l$. The time steps of delivering one packet (respectively, the other packet) constitute one partition set (respectively, the other partition set) for the TWO-PARTITION problem. Since TWO-PARTITION problem is NP-complete, then ALG cannot be solved exactly in polynomial-time. Hence, maximizing (weighted) throughput with time varying quality, is NP-hard. □

Though it is not possible to build an exact polynomial-time optimal offline algorithm for varying channel quality setting (unless P = NP), a pseudo-polynomial-time optimal offline algorithm exists by reducing the problem to a bipartite matching problem. Before stating the algorithm, we first introduce two concepts.

Definition 2.1 (Matching [7])**.** *Given an undirected graph $G = (V, E)$, a* matching *is a subset of edges $M \subseteq E$ such that for all vertices $v \in V$, at most one edge of M is incident on v.*

Definition 2.2 (Bipartite Graph [7])**.** *Given an undirected graph $G = (V, E)$, a* bipartite graph *is a graph in which the vertex set can be partitioned in to $V = L \cup R$, where L and R are disjoint and all edges in E go between L and R.*

In a maximum weighted bipartite matching problem, each edge $e \in E$ in the graph is associated with a weight function $c(\cdot) : E \mapsto \mathbb{R}$. The *weight of a matching* is the sum of the weight of its edges, i.e., $\sum_{e \in M} c(e)$. A maximum weighted bipartite matching problem can be solved in time $O(n^3)$, where $n = \max\{|L|, |R|\}$, using the Hungarian algorithm [40].

Theorem 2.5. *When the channel reliability varies over time, there exists an optimal offline algorithm running in pseudo-polynomial time.*

If the buffer size B is infinite, then an optimal offline algorithm has been given for the bounded-delay model in [12] with a running time of $O(n \log n)$. If the buffer size B is a constant, then an optimal offline algorithm was proposed in [41] with a running time of $O(n^2 \log \max\{B, n\})$.

Proof. We consider the generalization in which the channel reliability may not always be constant. Given a set of packets $P = \{p_1, \ldots, p_n\}$, each packet p_i is denoted as a 3-tuple (r_i, d_i, w_i). Let $r_{\min} = \min\{r_i | i \in \{1, \ldots, n\}\}$ and $d_{\max} = \max\{d_i | i \in \{1, \ldots, n\}\}$. All possible scheduling time slots are in the range of $[r_{\min}, d_{\max}]$; they are known as *feasible intervals*. Consider these m $(= d_{\max} - r_{\min} + 1)$ time slots. Let the sequence of known channel quality be $\{q_1, \ldots, q_m\}$ respectively.

The reduction from the model with varying channel quality to the maximum weighted bipartite matching problem works as below. We first construct a graph $G = (V, E)$, and $V = L \cup R$ where $L \cap R = \emptyset$. Each $a_i \in L$ represents a packet p_i and each $b_j \in R$ represents a time slot in the feasible intervals. Obviously, we have $|L| = n$ and $|R| = m$. We create an edge connecting each a_i and b_j with $r_{p_i} \leq b_j \leq d_{p_i}$. Each edge has a weight defined as $c(e_{ij}) = q_j \cdot w_{p_i}$, where q_j is the channel quality at time slot represented by vertex b_j and w_{p_i} is the weight of the packet represented by vertex a_i. At this point, our problem could be solved by finding the generalized maximum weighted matching of bipartite graph G and this results in a maximum total weight W.

The running time complexity of this algorithm includes two parts: the construction part and the part of solving the reduced problem. The construction will take a linear time

$O(|V| + |E|)$, and the solving part is mainly the running time in solving a maximum flow problem, which is $O(|E|^3)$, using the Hungarian algorithm. □

2.4.2 Online algorithms

We first investigate the challenge of designing efficient online algorithms for this problem. Without time constraints on packets, weighted throughput is maximized by simply delivering all packets that ever arrive at the router. When time constraints are enforced on identical-value packets, the objective of this problem becomes to send as many packets as possible before their respective deadlines — this variant is the same as the problem of scheduling equal-length jobs [42]. A 2-competitive deterministic algorithm and a 1.5-competitive deterministic algorithm have been given for this variant in the non-preemption setting and the preemption-restart setting respectively [42]. Both online algorithms' competitive ratios are tight (i.e., the lower bounds of competitive ratio).

Though optimal competitive online algorithms have been proposed in [42] for a variant in which throughput (of uniform-value packets) is maximized, this problem is open and becomes more interesting and complicated when packet weights are different. Now we present an example in which the fade state of the channel is ideal (i.e., $q_t = q_{\max} = l, \forall t$) but packets have different weights.

Example 1. Consider two packets p_1 and p_2 with $d_{p_1} = 1 < d_{p_2} = 2$ and $w_{p_1} < w_{p_2}$ at time 1 in an overloaded system. Note that the router has no knowledge of future arriving packets. Sending the packet p_1 in the first time step may cause p_2 not to be sent anymore if we assume that another packet p_3 with $d_{p_3} = 2$ and $w_{p_3} > w_{p_2}$ arrives at time 2 (since p_2 and p_3 cannot be sent simultaneously in step 2 before their deadlines). A better (clairvoyant) way is to send p_2 in the first time step and send p_3 in the second time step. On the other hand, if the online algorithm picks p_2 to send in the first time step, it leads to the expiration of the packet p_1. In the case that p_3 is not released in step 2 in the input sequence, the online algorithm loses the value of p_1 — it is better to send p_1 and p_2 in the first two consecutive time steps clairvoyantly.

In summary, even under ideal fade states, the challenge of designing efficient online algorithms lacking of information about future input is to balance wisely between sending an earliest-deadline packet and a largest-weight packet. Our proposed online algorithms are based on this intuition. Another challenge of this model is due to the uncertainty of the fade states of the wireless channel. We will address more on these challenges and on our solutions later.

We consider non-preemption and preemption-restart settings separately. In both cases, the algorithm gets credit only for packets that are executed continuously from the beginning to the end before its deadline. We also call the optimal offline algorithm *adversary*. Let $w_{\max}$ and $w_{\min}$, respectively, denote the maximum value and the minimum value of a packet in the input sequence $\mathcal{I}$.

In the non-preemption setting

We first show a pessimistic result, and then propose an optimal online algorithm for a variant of this model.

Theorem 2.6. *In the non-preemption setting, no online algorithm has a constant competitive ratio, even if the fade state is a fixed value q (but $q < q_{\max} = l$) and if packets are with agreeable deadlines. The lower bound of competitive ratios can be up to $w_{\max}/w_{\min}$.*

Note that if packets have the same value and if the fading channel has a fixed quality, EDF is 2-competitive [42]. Thus, associating values to packets complicates the model.

Proof. We set the channel's quality $q = 0.5 \cdot l$. Any packet can be sent in consecutive 2 time steps. Let an online algorithm be ON. We use (w, d) to denote a packet with value w and deadline d.

In the first time step, a packet $(w_{\min}, 2)$ is released. The adversary keeps releasing a packet $(w_{\min}, 2 \cdot (i+1))$ in each time step $2 \cdot i + 1$ ($i \in \{1, 2, 3 \ldots\}$) until one of the following events happens: (1) ON picks up a packet $(w_{\min}, 2 \cdot k)$ to send, or (2) the adversary has released one such packet with value $w_{\min}$, and ON does not pick it up to send.

For the second case, the adversary stops releasing new packets and it schedules the packet ever released with a total gain of $w_{\min}$. On the other side, ON gains nothing overall. For the first case, when ON picks up a packet $(w_{\min}, 2 \cdot k)$ to send, the adversary releases a packet $(w_{\max}, 2 \cdot k + 1)$ at time $2 \cdot k$. Note that in the non-preemption setting, ON cannot stop sending the packet $(w_{\min}, 2 \cdot k)$ till the time $2 \cdot k$ when this packet is finished. Thus, ON cannot execute the packet $(w_{\max}, 2 \cdot k + 1)$ at time $2 \cdot k$ to get it finished before its deadline. After releasing the packet $(w_{\max}, 2 \cdot k + 1)$, the adversary releases nothing. Overall, the optimal offline algorithm will send all packets $(w_{\min}, 2 \cdot 1), (w_{\min}, 2 \cdot 2), \ldots, (w_{\min}, 2 \cdot (k-1))$ and $(w_{\max}, 2 \cdot k + 1)$. On the other side, ON executes only one packet $(w_{\min}, 2 \cdot k)$. The competitive ratio is

$$\frac{(k-1) \cdot w_{\min} + w_{\max}}{w_{\min}} = k - 1 + \frac{w_{\max}}{w_{\min}} \geq \frac{w_{\max}}{w_{\min}}.$$

Then, ON is no better than $(w_{\max}/w_{\min})$-competitive. □

To complement Theorem 2.6, we note the following result.

Theorem 2.7. *[43] In the non-preemption setting, no online algorithm has a constant competitive ratio, even if the fade state is ideal ($q = q_{\max} = l$). The lower bound of competitive ratios can be up to $\sqrt{\frac{w_{\max}}{w_{\min}}}$.*

Given the assumptions that the channel state is a fixed value and packets are with agreeable deadlines, we have proved that at any time t, EDF finishes no fewer packets than any algorithm (see the proof of Theorem 2.2). Given an input $\mathcal{I}$, we assume EDF finishes s packets with a total value $W \geq s \cdot w_{\min}$. Any algorithm finishes no more than s packets with a total value $\leq s \cdot w_{\max} \leq W \cdot (w_{\max}/w_{\min})$. Thus, we immediately have the result.

Corollary 2. *In the non-preemption setting, if the fade state is a fixed value and if packets are with agreeable deadlines, then EDF is an optimal online algorithm.*

If the fade state is at its maximum all the time (such that a packet is sent in a single time step), this variant of the online problem is the same as the bounded-delay model [12, 24, 27, 28, 35]. An optimal online algorithm has been proposed for the agreeable deadline case [27]. For the general case, the best known lower bound of competitive ratios is $\phi := \frac{1+\sqrt{5}}{2} \approx 1.618$ [24] and the best known upper bound is 1.832 [28, 35]. Closing the gap $[1.618, 1.832]$ is still an intriguing open problem [44].

In the preemption-restart setting

In the preemption-restart setting, we first provide an example to show that if the fade states are unknown to the online algorithms (there are no stochastic assumptions as well), no online algorithm can have a competitive ratio better than $w_{\max}/w_{\min}$.

Theorem 2.8. *If the fade states are unknown to online algorithms, then no online algorithm can have a competitive ratio better than* $w_{\max}/w_{\min}$.

Proof. Consider time step 1 and two packets are released. We use (w, d) to represent a packet p with value w and deadline d. Let an online algorithm be ON. The fade state at time step 1 is $0.5 \cdot l$. A packet $p_1 := (w_{\min}, 2)$ is released at time step 1.

The fade state keeps its quality $0.5 \cdot l$ from time step 1 to time step 2. At time step 1, a packet $p_2 := (w_{\max}, 3)$ is released. If ON schedules p_1, we keep the fade state at $0.5 \cdot l$ till time step 3 and ON cannot schedule p_2 before its deadline. The optimal offline algorithm will schedule p_2 instead and the competitive ratio is $w_{\max}/w_{\min}$. On the other hand, if ON schedules p_2 at its arrival, the fade state sharply changes to 0 at the end of time step 2 and keeps 0 since then. Thus, even though ON starts to schedule p_2, it cannot finish it. Instead, the optimal offline algorithm schedules p_1 and the competitive ratio is $w_{\min}/0$, which can be arbitrarily large. □

Based on Theorem 2.8, we know that if the fade states are completely unpredictable, without one step of look-ahead, then no online algorithm can have a competitive ratio

better than $w_{\max}/w_{\min}$. Therefore, we consider a practical scenario and make the following assumption that is widely accepted:

Assumption 1. *[17,19,20] The online algorithms have the ability of looking one-step ahead of knowing the fade states of the channel. At the time when an online algorithm starts to schedule a packet, the algorithm knows that this "committed" packet can be sent successfully according to future fading states. However, the online algorithm is allowed to preempt-restart this packet later, and this packet is not guaranteed to be sent successfully if it is preempted.*

Assumption 1 applies to all the following variants that we consider.

Assume that both the fade states and the packet input sequence are unknown to the online algorithms. Two packets are with more interests when neither the fade states nor the input sequence is known to the online algorithm:

1. packet i: the currently running packet. If i is not available, we simply create a virtual packet i with $w_i = 0$.

2. packet h: the packet with the maximum-value among all pending packets in the router.

From previous results, we conclude that always sending the packet with the earliest deadline or with the maximum weight results in a competitive ratio arbitrarily large. Here, we employ the following ideas to design an online algorithm with a better competitive ratio: If the currently sending packet i has a sufficiently large value, then we keep sending it. Otherwise, we let packet h preempt it. The algorithm we study is called SEMI-GREEDY.

Algorithm 2 SEMI-GREEDY($\alpha > 1$)

1: Let the maximum-value pending packet be h, with ties broken in favor of the earliest deadlines.
2: Let the currently being sent packet be i. If h (or i) does not exist, we set $w_h = 0$ (or $w_i = 0$).
3: **if** $w_h \geq \alpha \cdot w_i$ **then**
4: abort i and send h.
5: **end if**

Before we prove the competitive ratio for the algorithm SEMI-GREEDY, we define a concept that is useful for the proof.

Definition 2.3 (Packet Chains)**.** *We define a packet chain C of k packets as $C := \{p_1, \ldots, p_k\}$ with the following property ($\alpha > 1$), $w_{p_i} \leq \frac{w_{p_{i+1}}}{\alpha}$, $\forall i = 1, 2, 3, \ldots, k-1$. We use $W(C)$ to represent the total value of the packets of C.*

Lemma 1. *Given a chain C of $k \geq 2$ packets $p_1, p_2, \ldots, p_k$, we have $W(C) \leq \frac{1}{\alpha-1} \cdot \frac{\alpha^{n+1}-1}{\alpha^n} \cdot w_{p_k}$.*

Proof.

$$\begin{aligned}
\frac{W(C)}{w_{p_k}} &= \frac{\sum_{i=1}^{k} w_{p_i}}{w_{p_k}} = \frac{w_{p_1} + w_{p_2} + \cdots + w_{p_{k-1}} + w_{p_k}}{w_{p_k}} \\
&\leq \frac{w_{p_1} + w_{p_2} + \cdots + w_{p_{k-1}} + \alpha \cdot w_{p_{k-1}}}{\alpha \cdot w_{p_{k-1}}} \\
&= 1 + \frac{1}{\alpha} \cdot \frac{w_{p_1} + w_{p_2} + \cdots + w_{p_{k-1}}}{w_{p_{k-1}}} \\
&\leq \cdots \\
&\leq 1 + \frac{1}{\alpha} + \frac{1}{\alpha^2} + \cdots + \frac{1}{\alpha^{k-2}} + \frac{1}{\alpha^{k-1}} \\
&= \frac{1}{\alpha - 1} \cdot \frac{\alpha^k - 1}{\alpha^{k-1}}
\end{aligned}$$

□

Theorem 2.9. *The SEMI-GREEDY algorithm has a competitive ratio of* $\max\{1+\alpha, \frac{1}{\alpha-1} \cdot \frac{\alpha^n-1}{\alpha^{n-1}}\}$*. It is ($\phi^2 \approx 2.618$)-competitive when $\alpha = \phi \approx 1.618$.*

Proof. We use a charging scheme to prove Theorem 2.9. The idea is: For the packets sent by the adversary, we charge them into different time intervals and we prove that in each pair of corresponding intervals, the value we charge to the adversary in that interval is no more than $\max\{1 + \alpha, \frac{1}{\alpha-1} \cdot \frac{\alpha^{n+1}-1}{\alpha^n}\}$ times of what SEMI-GREEDY achieves.

Let the subset of packets chosen by the adversary (that is, an optimal offline algorithm) (respectively, SEMI-GREEDY) be Π_1 (respectively, Π_2). Without loss of generality, we assume that the adversary sends packets in a *canonical order*, i.e., for any two pending packets p_i and p_j, the adversary sends the packet with an earlier deadline. We are going to prove

$$\frac{\sum_{p_j \in \Pi_1} w_{p_j}}{\sum_{p_i \in \Pi_2} w_{p_i}} \leq \max\{1+\alpha, \frac{1}{\alpha-1} \cdot \frac{\alpha^n - 1}{\alpha^{n-1}}\}$$

The proof depends on the following two observations:

1. Given a set of packets S at time t, we assume that an online algorithm schedules a packet $p_i \in S$. We consider time $t' > t$. Since all packets have the same length, if the packet p_i cannot be finished by time t', any packet in S cannot be finished completely by time t', no matter what the fade states of the channel are.

2. Given a set of packets S at time t, we assume that the SEMI-GREEDY algorithm schedules a packet $p_i \in S$. We have $w_{p_i} \geq \max_{p_j \in S} \frac{w_{p_j}}{\alpha}$.

   If we assume p_i is aborted at time $t' > t$ by a packet p_k, then we have $w_{p_i} < w_{p_k}/\alpha$ and $p_k \notin S$. If the preempting packet p_k is not sent by the algorithm SEMI-GREEDY, then p_k must be aborted by another packet which has the potential of being sent. So on and so forth, we regard all aborted packets and the last-sent packet p_l as a chain. From Lemma 1, all ever-aborted packets have value $\leq w_{p_l} \cdot \frac{1}{\alpha-1} \cdot \frac{\alpha^n-1}{\alpha^{n-1}}$.

   Note that no chains share a same packet, since each preempted packet and its preempting packet are in the same chain.

For any packet $p \in (\Pi_1 \setminus \Pi_2)$ sent only by the optimal offline algorithm, either p expires before SEMI-GREEDY sends it or p was sent, SEMI-GREEDY aborted p before p could be finished, and p is never completed before its deadline. If p expires, any packet that SEMI-GREEDY sends since time r_p has a value $\geq w_p/\alpha$ (from the algorithm). For each time interval in which a single packet is sent, we examine it for both the optimal offline

algorithm and this online SEMI-GREEDY algorithm in a sequential order. Our charging scheme works as follows:

1. For any packet $p \in (\Pi_1 \setminus \Pi_2)$ that SEMI-GREEDY has not ever run, we charge it to the corresponding time interval that SEMI-GREEDY sends a packet.

   We note that SEMI-GREEDY must have one pending packet to send in this interval since this packet p is a candidate. The packet SEMI-GREEDY sends, say p', in this corresponding interval has a value no less than w_p/α. Also, SEMI-GREEDY finishes p' no later than the adversary finishes p since p and p' have the same processing time and p and p' are being executed in corresponding time intervals when both algorithms send packets.

2. For any packet $p \in (\Pi_1 \setminus \Pi_2)$ that SEMI-GREEDY ever sends but aborts it later, we know that (from above observations) p belongs uniquely to a chain and the last element of this chain, say p', is sent by SEMI-GREEDY. Thus, we charge w_p to the time interval when p' is sent by SEMI-GREEDY.

3. For any packet $p \in (\Pi_1 \cap \Pi_2)$, we charge w_p to the time interval when SEMI-GREEDY sends p.

   Clearly, for any packet acting as the last-element of a chain, this charging scheme results that the value ratio is bounded by $\frac{1}{\alpha-1} \cdot \frac{\alpha^n-1}{\alpha^{n-1}}$ (see Lemma 1).

The remaining part of the proof is to argue that when we charge a packet $p \in (\Pi_1 \setminus \Pi_2)$ that SEMI-GREEDY has not ever run yet in the corresponding time interval, SEMI-GREEDY sends a packet p', $w_{p'} \geq w_p/\alpha$. This claim is easy to prove since if $w_{p'} < w_p/\alpha$, p' will be aborted by p immediately at the time when p arrives. Thus, for each packet p that SEMI-GREEDY sends, the charged value to p for the adversary is bounded by $1 + \alpha$ and $\frac{1}{\alpha-1} \cdot \frac{\alpha^n-1}{\alpha^{n-1}}$ times w_p. All packets sent by the adversary have been charged. Theorem 2.9 is proved. □

Closing or shrinking the gap $[2, 2.618]$ for deterministic online algorithms still remains as an open problem.

Assume that the fade states are known to the online algorithms, but the packet input sequence is unknown. Now we consider a variant in which the fade states are known beforehand, but the packet input sequence is unknown. To illustrate the challenge, we present an instance in which packets have the same value and the fade state of the channel is fixed at $q_t = \frac{l}{2}$, $\forall t$.

Example 2. Consider one packet p_1 with deadline 5 at time 1. If an online algorithm executes p_1, the adversary releases another packet p_2 with deadline 3 at time 2. So, the online algorithm cannot finish both jobs and the competitive ratio is 2, given the adversary finishes both packets in the order of p_2 and p_1. If the online algorithm aborts p_1 but executes p_2 at time 2, the adversary releases another packet p_3 at time 2 with deadline 4. Here, the online algorithm cannot finish both p_2 and p_3, but the adversary can finish p_1 and p_3 by their deadlines in order. Thus, the lower bound of competitive ratios for this variant ($w_{p_i} = 1$, $\forall i$ and fade states keep constant $l/2$) is 2.

It is intuitive to abort a running packet if it can be sent later with respect to the given set of pending packets and fade states of the channel. Our proposed online algorithms are based on this intuition. In order to check if a set of packets can be delivered successfully, we define a concept first.

Definition 2.4 (Provisional Schedule [28, 42])**.** *At any time t, a* provisional schedule *is a schedule for the pending packets at time t (assuming no new arriving packets). This schedule specifies the set of packets to be transmitted, and for each packet, it specifies the delivery time. An optimal provisional schedule is the one achieving the maximum total value of packets among all provisional schedules.*

In the following, we provide a modified earliest-deadline-first algorithm called EDF_β. Since the fade states are known, there exists an efficient algorithm in calculating an optimal

provisional schedule for time t. We are interested in two packets in this provisional schedule: the earliest-deadline pending packet e and the packet h with the maximum value. We either schedule e (if e has a sufficiently large value) or another packet f satisfying $w_f \geq \max\{\beta \cdot w_e, \frac{w_h}{\beta}\}$.

Algorithm 3 EDF$_\beta$

1: Abort the currently running packet i only if the new arrival with value $\geq \beta \cdot w_i$, ties are broken in favor of the packet with the earliest deadline.
2: **if** there is no currently running packet **then**
3: calculate the optimal provisional schedule, based on the set of pending packets and the known fade states;
4: **if** $w_e \geq \frac{w_h}{\beta}$ **then**
5: execute e;
6: **else**
7: execute a packet f satisfying

$$w_f \geq \max\{\beta \cdot w_e, \frac{w_h}{\beta}\},$$

where ties are broken in favor of the earliest-deadline packet. Note h itself is a candidate for f.
8: **end if**
9: **end if**

Theorem 2.10. *Assume fade states are known to online algorithms. Algorithm EDF_β is* $\max\{2, \beta, \frac{1}{\beta-1} \cdot \frac{\beta^n-1}{\beta^{n-1}}\}$*-competitive, and it is* 2*-competitive when* $\beta = 2$.

Proof. We use a potential function method and a loop invariant method to prove Theorem 2.10. We compare our algorithm EDF_β with the adversary ADV. Let Φ_t^{ADV} and Φ_t^{EDF} denote the potentials of the adversary and EDF_β respectively. Specifically, Φ_t^{ADV} denotes the total value achieved since time t from the pending packets at time t for the adversary. Let this set of packets be S_t^*. Let Φ_t^{EDF} denote the total value of the optimal provisional schedule of the pending packets at time t for EDF_β. We use p_t and p'_t to denote the t-th packet sent by EDF_β and ADV respectively. If such a packet does not exist, then p_t (p'_t, respectively) is a null packet with value 0. To prove Theorem 2.10, we need to show

that for any t, we always have

$$c \cdot w_{p_t} + \Delta\Phi_t^{EDF} \geq w_{p'_t} + \Delta\Phi_t^{ADV},$$

where $c := \max\{2, \beta, \frac{1}{\beta-1} \cdot \frac{\beta^n-1}{\beta^{n-1}}\}$. We provide the following loop invariants and prove their correctness by case study.

- Denote the pending packets at time t for ADV and EDF_β as $\mathcal{P}'_t$ and $\mathcal{P}_t$. $\mathcal{P}'_t \subseteq \mathcal{P}_t$. Note that EDF_β may not deliver all packets in $\mathcal{P}_t$.
- When a packet is sent, the sum of the actual gain and the credit charge (see below) is called *amortized gain*. We prove that for the i-th packet sent, ADV's amortized gain is no more than c times of EDF_β's amortized gain.

For arrivals, with the first invariant, the invariants are easy to prove. Note $w_{p_t} = w_{p'_t} = 0$. In the following, we consider packet deliveries only. Let the packet EDF_β chooses to send in this time interval be p. One fact that we will use in the proof is: Given two packet p and p^* with $d_p \leq d_{p^*}$, if p is not in the optimal provisional schedule, but p^* is, then $w_{p^*} \geq w_p$. This fact further implies that if p is the packet EDF_β is currently sending, any packet not in the optimal provisional schedule has a value $\leq \beta \cdot w_p$.

1. Assume that ADV sends a packet p'. Assume that p is sent successfully by EDF_β.

   Based on the invariants, $w_{p'}, w_p \leq w_h$. From the algorithm itself, $w_p \geq w_h/\beta$. Since all packets have the same length, under any fade states, EDF_β finishes p no later than ADV finishes p'.

   If $d_{p'} < d_p$, we have $w_{p'} < w_p$ in the optimal provisional schedule. Then we charge $w_{p'} + w_p$ to the adversary and we have $w_{p'} + w_p \leq 2 \cdot w_p$. If $d_{p'} > d_p$, p will not be sent by the adversary. Then we charge $w_{p'}$ to ADV and we have $\beta \cdot w_p \geq w_h \geq w_{p'}$.

2. Assume that ADV sends a packet p'. Assume that p is aborted by EDF_β before it is finished.

   If the adversary will send p, we will charge w_p to the packet that preempts it. Like the chain we have calculated in Lemma 1, the value gained by sending the last packet of the chain is at least $(\beta-1)\cdot\frac{\beta^{n-1}}{\beta^n-1}$ times of the total value we charge for the adversary.

3. Assume that ADV has nothing to send from the currently pending packets for EDF_β.

   We claim that either p has been sent by ADV or ADV must have one new arrival before EDF_β finishes the packet p which it chooses to send. Otherwise, ADV can get more value by delivering p. It does not hurt if we have run p till new arrivals come. This analysis is similar to what we have had in above cases.

□

Theorem 2.10 implies that extra information (for example, the known fade states) helps improve competitive ratios.

Assume that the fade states are unknown, but the packet input sequence is known. We first provide the lower bound $\phi \approx 1.618$ of competitive ratio for deterministic online algorithms for this variant. Then we discuss the relation between this model and another well-studied online problem.

Theorem 2.11. *Consider a variant in which the fade states are unknown, but the packet input sequence is known to online algorithms. The lower bound of competitive ratio for deterministic online algorithms is* $\phi \approx 1.618$.

Proof. Such an instance is easy to construct. Assume there are two packets in the input sequence only. One packet p_1 is with value 1 and deadline 2. The other packet p_2 is with value ϕ and deadline 4. These two packets are released at time step 1. Let an online algorithm be ON.

If ON schedules p_1, the optimal offline algorithm schedules p_2 and the fade states are $0.5 \cdot l$ from time step 1 to 3. Note here the Assumption 1 still holds. Then the competitive

ratio is ϕ. If ON schedules p_2, the optimal offline algorithm schedules both p_1 and p_2 given the fading states are $0.5 \cdot l$ from time step 1 to 4. Thus, the competitive ratio is $\frac{1+\phi}{\phi} = \phi$. $\square$

Next, we reveal the relation between this variant and a well-studied model called the *bounded-delay model* (see [28, 35] and the references therein). In the bounded-delay model, packets are released in an online manner. Each packet is associated with a value and a deadline before which it should be sent. In each time step, a packet can be sent and the goal is to maximize the total value of the packets sent before their respective deadlines.

Theorem 2.12. *Assume that the fade states are unknown, but the packet input sequence is known to online algorithms. A c-competitive algorithm for the bounded-delay model implies a c-competitive algorithm for our model.*

Proof. Consider an input sequence $\mathcal{I}$ for the bounded-delay model. Let the packets sent by an optimal offline algorithm be $\mathcal{O}$ and the algorithm be OPT_d.

Given a time t, we create the fade states such that the optimal offline algorithm OPT_f for this variant achieves the same weighted throughput as OPT_d. Also, for an online algorithm, the extra given information about the whole input sequence cannot circumvent the difficulty brought by the unpredictable fade states. The construction of fade states is as follows.

For the bounded-delay model, let the set of packets $\mathcal{O}$ be $p_1, p_2, \ldots, p_m$ and they are sent in time steps $1, 2, \ldots, m$ respectively. (If there is no packet sent in a step i, we create a dummy packet p_i for step i with $w_{p_i} = 0$.) Without loss of generality, all packets p_i can be sent in the earliest-deadline-first manner. Then we modify the deadlines of the packets in $\mathcal{O}$ such that $d_{p_i} < \min\{d_{p_{i+1}}, \ldots, d_{p_m}\}$, for all $i = 1, 2, \ldots, m-1$. At last, we force the quality of the fade states from time d_{p_i} to $d_{p_{i+1}}$ be $l/(d_{p_{i+1}} - d_{p_i})$. This guarantees that a packet can be sent under such fade states, and if p_i is pending to an online algorithm at time $d_{p_{i-1}}$ and the online algorithm sends any packet other than p_i, p_i cannot be sent by the online algorithm any more. We ensure that the optimal offline algorithm for this variant works the same as the optimal offline algorithm for the bounded-delay model. Also, the extra information about the packet input sequence does not help the online algorithm,

since it has no knowledge about the fade states. With Assumption 1, the online algorithm knows that only one packet can be sent once it is committed and this is exactly the same as what is assumed in the bounded-delay model. □

Closing or shrinking the gap of competitive ratios $[1.618, 1.832]$ for the bounded-delay model is an intriguing problem and thus, from Theorem 2.12, the gap still applies to the variant in which the fade states are unknown, but the packet input sequence is known to online algorithms.

2.4.3 Online learning algorithms

Driven by the observation that adversary is too powerful in generating inputs against online algorithms and competitiveness is sometimes too pessimistic in evaluating an online algorithm, we want to use another metric *external regret* to evaluate the performance of an online algorithm using online learning approaches. We note that competitive online algorithms and online learning algorithms can be both applied as online approaches to this model without making any stochastic or probabilistic assumptions on input arrivals. The main question here will be: Does there exist a finite input instance such that any online learning algorithm cannot achieve a constant external regret?

Theorem 2.13. *Consider a finite but large input sequence. There is no online learning algorithm that can have a constant external regret.*

Proof. In order to show that for a given finite input sequence, no online learning algorithm has a constant external regret, we need to show that compared with the best algorithm in hindsight, any online learning algorithm has a constant competitive ratio which is larger than 1. We consider packets with agreeable deadlines. This example is modified from the one given in [25]. We first show that for any online policy π, its competitive ratio $c \geq \phi - \epsilon$, for any small $\epsilon > 0$.

At each time step t, two packets p_t and p'_t with *span* (the difference between deadline and release time) 1 and 2 respectively are released with values w_t and w_{t+1} respectively. Assume

a deterministic online algorithm runs policy π, and its competitive ratio is c. Initially, we set $\tau = \sqrt{5} - 2$, $w_0 = 1$, $w_1 = \phi + \epsilon$, and $w_{i+1} = \frac{w_i - w_{i-1}}{\tau}$. (Explicitly, we set $w_i = (1-\epsilon)\phi^i + \epsilon(\phi+1)^i$, $\forall i \geq 0$.)

1. Let k be a sufficient large number. If there exists $0 \leq j < k$, π selects p'_j in step j (i.e., time interval $[j, j+1]$), its adversary stops releasing packets after j.

   Policy π does not select p_j to send in step j, and $\forall i < j$, π selects p_i to send in step i. On the contrary, π's adversary delivers p'_i in step i, $\forall i < j$, p_j in step j, and p'_j in step $j+1$. Note that $\lim_{k\to\infty} \frac{w_k}{w_{k-1}} = \phi + 1$ and $\sum_{i=1}^{n} w_i = (w_{n-1} - w_0)/\tau$. The competitive ratio is

   $$\begin{aligned} c &\geq \frac{(w_1 + w_2 + \cdots + w_{j+1}) + w_j}{(w_0 + w_1 + \cdots + w_{j+1}) - w_j} \\ &> 1 + \frac{2\tau}{1-\tau} - \left(\frac{2\tau}{1-\tau}\right)^2 \epsilon \\ &> \phi - \epsilon. \end{aligned}$$

2. Otherwise, the adversary releases all packets p_i and p'_i up to step $k-1$ and at time k, only p_k is released.

   $\forall i < k$, π selects p_i to send in step i. π's adversary delivers all packets p'_i in step i up to step k, where p_k is sent. Assume k is large.

   $$\begin{aligned} c &\geq \frac{w_1 + \cdots + w_k + w_k}{w_0 + w_1 + \cdots + w_k} \\ &= 1 + \frac{w_k - w_0}{w_0 + w_1 + \cdots + w_k} \\ &\xrightarrow{k\to\infty} 1 + \frac{(1+\phi)w_{k-1} - 1}{1 + \phi + \frac{w_{k-1}-1}{\tau}} \\ &\xrightarrow{k\to\infty} 1 + \frac{\phi + 1}{\tau^{-1}} = \phi. \end{aligned}$$

Thus, no online algorithm can achieve a competitive ratio better than $\phi > 1$. Therefore, no online learning algorithm has a constant external regret. The limit of online learning algorithm is due to the fact that the partial input sequence cannot reveal more information on the optimality of each static algorithm until the end of this input sequence. □

Online learning algorithms description

We design online learning algorithms for both the bounded-delay model and its generalization, as well as a few of its variants. We name a static algorithm an *expert*. Based on Occam's razor (a principle stating that "the simplest explanation is usually the correct one"), we interpret that in general, we prefer simpler explanations. Consider the model that we study. Packet values, packet deadlines, and channel quality are the three factors that we need to consider in the online-decision making procedure. Thus, the static algorithms should be simple functions of packet values, packet deadlines, and channel quality only.

First, we design a few experts (static algorithm with hindsight). Then we design online learning algorithms based on the observed performance of these experts. Note that the number of experts cannot be large since this value determines the running time of the online learning algorithm in each time step. Thus, we apply the well-known *geometric rounding technique*. Consider all distinct packet values $w_{\min} = w_1 < w_2 < \ldots < w_n = w_{\max}$. We let $w_{\max} = (1+\delta)^k w_{\min}$, where $k \leq n$. Then for each value w_i, it falls in the range of $[w_{\min}(1+\delta)^{i-1}, w_{\min}(1+\delta)^i)$, $i \in \{1, \ldots, k\}$. Thus, we have $\log_{1+\delta} \frac{w_{\max}}{w_{\min}}$ distinct intervals. Let $M = \lceil \log_{1+\delta} \frac{w_{\max}}{w_{\min}} \rceil$. We first introduce the ways these M experts work. Then we represent our online learning algorithm under various scenarios respectively.

We have two phases of delivering a packet for each expert. In the first phase, the quality of the channel is predicted. In the second phase, the expert chooses one pending packet to send. In selecting a packet to send, packet values play the role and we have two (families of) strategies.

1. (A strategy based on absolute values.)

Sort all pending packets in increasing order of deadlines, with ties broken in favor of the one with larger value. Choose the packet with a value $\geq$ its predefined threshold with absolute value. If such a packet is not available in its buffer, this expert sends nothing. This algorithm is described in Algorithm 4.

Algorithm 4 EBAV

1: Sort packets in the buffer in canonical order, i.e., in increasing order of deadlines with ties broken in favor of larger value.
2: Send the first packet p satisfying $w_p \geq \frac{w_{\max}}{(1+\delta)^j}$. If p does not exist in the buffer, send nothing.

Each expert only admits the packet that it will send eventually. Thus, for the expert EBAV with a parameter j, it only accepts packets with values $\geq w_{\max}/(1+\delta)^j$. If the buffer size is limited, when overflow happens, we apply the greedy approach to filter out the packets that cannot be accommodated. That is, we drop the minimum-value packets when packet overflow happens; ties broken arbitrarily.

2. (A strategy based on relative values.)

   Sort all pending packets in increasing order of deadlines, with ties broken in favor of the one with a larger value. Choose the packet with a value $\geq$ its predefined threshold with relative value defined using the maximum-value packet in the buffer, with ties broken in favor of the earlier released packet. We note that such a packet is always in the buffer.

Algorithm 5 EBRV

1: Sort packets in the buffer in canonical order, i.e., in increasing order of deadlines with ties broken in favor of larger value.
2: Let h be the maximum-value packet in the buffer.
3: Send the first packet p satisfying $w_p \geq \frac{w_h}{(1+\delta)^j}$. The packet p can always be found since either the first satisfying packet or the packet h is a candidate.

In admitting packets, we apply the approach of identifying the *optimal provisional schedule*, which is similar to the one used in competitive online algorithms. Given a set of pending packets P, a *provisional schedule* S specifies which packets in P should be sent in which time step. An *optimal provisional schedule* S^* is the one that achieves the

maximum weighted throughput among all provisional schedules on pending packets P (channel quality is assumed perfect). Clearly, an optimal provisional schedule S^* at time t can be calculated via a maximum-weighted bipartite matching over pending packets in $O(|P|^2)$ (see [30]).

The channel quality is always perfect. This is the typical bounded-delay model. Here, we introduce two learning algorithms based on the same strategy: Follow the 'best expert'. One is simply following the strategy of the expert who has the best gain up to the current time, and the other is following the strategy of the expert who has the best gain after delivering all packets in its buffer successfully. We call these two online learning algorithms 'Follow COPT' and 'Follow OPT', respectively.

The channel quality varies over time. When the channel quality is not a fixed value, we need to consider the channel quality's variability as well. Thus, we employ a set of experts in predicting the channel's quality over time.

In estimating the channel quality, we introduce two experts. (We can adapt our algorithm to multiple experts.) Each expert insists on giving a fixed prediction of the future channel states, which is either H (representing 'high') or L (representing 'low'). These two experts are named EH and EL, respectively. We then use the "weighted majority algorithm" [45, 46] in predicting the state of the channel quality. We associate credits to these experts, which means to how much extend an expert's opinion can be trusted. Let the credit for expert EH at time t be c_t^{EH} and for expert EL be c_t^{EL}, respectively. Initially, we set $c_0^{EH} = c_0^{EL} = 1$. In each time step $1, 2, \ldots, t-1$, we have a label indicting whether the channel quality is 'H' or 'L'. Then we proceed as the following Winnow Algorithm (Algorithm 6).

However, in admitting packets, we are not only predicting the next step's channel quality, we also need to estimate the future channel's state when we calculate an optimal provisional schedule. Hence, we need to study the 'chain effect' of the prediction. Let $E(\cdot)$ be the

Algorithm 6 Winnow(EH, EL)

Initially, set $c_0^{EH} = c_0^{EL} = 1$.
if $\sum c_t^{EH} \geq \sum c_t^{EL}$ **then**
  predict 'H';
else
  predict 'L'.
end if
if the channel's quality was 'high' **then**
  $c_{t+1}^{EH} = 2c_t^{EH}$;
  $c_{t+1}^{EL} = c_t^{EL}/2$;
else
  $c_{t+1}^{EH} = c_t^{EH}/2$;
  $c_{t+1}^{EL} = 2c_t^{EL}$.
end if

predicted value using Algorithm 6.

Lemma 2. *If $E(c_t^{EH}) \geq E(c_t^{EL})$ (respectively, $E(c_t^{EH}) < E(c_t^{EL})$) holds at time slot t, then $E(c_{t+1}^{EH}) \geq E(c_{t+1}^{EL})$ (respectively, $E(c_{t+1}^{EH}) < E(c_{t+1}^{EL})$) holds at time $t+1$.*

Proof. Let η_t denote the predicted quality at time t. We have $\eta_t = \frac{E(c_t^{EH})}{E(c_t^{EH})+E(c_t^{EL})}$. Based on the Winnow Algorithm, we have

$$E(c_t^{EH}) = \eta_{t-1}E(c_{t-1}^{EH}) + (1-\eta_{t-1})\frac{E(c_{t-1}^{EH})}{2}, \tag{2.4}$$

$$E(c_t^{EL}) = \eta_{t-1}\frac{E(c_{t-1}^{EL})}{2} + (1-\eta_{t-1})E(c_{t-1}^{EH}). \tag{2.5}$$

Given the assumption that $E(c_t^{EH}) \geq E(c_t^{EL})$, we have $\eta_t \geq \frac{1}{2}$. Then from Equations (2.4) and (2.5), we have

$$E(c_{t+1}^{EH}) = (1+\eta_t)\frac{1}{2}E(c_t^{EH}) \geq \frac{3}{4}E(c_t^{EH}),$$

$$E(c_{t+1}^{EL}) = \left(1-\frac{1}{2}\eta_t\right)E(c_t^{EL}) \leq \frac{3}{4}E(c_t^{EL}).$$

Lemma 2 is completed. □

Theorem 2.14. *If the initial expected credits of experts satisfies $E(c_0^{EH}) \geq E(c_0^{EL})$ (respectively, $E(c_0^{EH}) < E(c_0^{EL})$), then the quality of channel η_t is non-decreasing (respectively, non-increasing) for t.*

Proof. Theorem 2.14 can be proved using the inductive method. Based on the definition of η and $E(\cdot)$, we have

$$\begin{aligned}
\eta_t &= \frac{E(c_t^{EH})}{E(c_t^{EH}) + E(c_t^{EL})}, \\
\eta_{t+1} &= \frac{E(c_{t+1}^{EH})}{E(c_{t+1}^{EH}) + E(c_{t+1}^{EL})} \\
&= \frac{(1+\eta_t)\frac{1}{2}E(c_t^{EH})}{(1+\eta_t)\frac{1}{2}E(c_t^{EH}) + \left(1-\frac{1}{2}\eta_t\right)E(c_t^{EL})} \\
&= \frac{(1+\eta_t)E(c_t^{EH})}{(1+\eta_t)E(c_t^{EH}) + (2-\eta_t)E(c_t^{EL})}.
\end{aligned}$$

Therefore, we have

$$\begin{aligned}
\frac{\eta_{t+1}}{\eta_t} &= \frac{(1+\eta_t)E(c_t^{EH})}{(1+\eta_t)E(c_t^{EH}) + (2-\eta_t)E(c_t^{EL})}\left(\frac{E(c_t^{EH}) + E(c_t^{EL})}{E(c_t^{EH})}\right) \\
&= \frac{(1+\eta_t)E(c_t^{EH}) + (1+\eta_t)E(c_t^{EL})}{(1+\eta_t)E(c_t^{EH}) + (2-\eta_t)E(c_t^{EL})}.
\end{aligned}$$

Assume that we have the initial case of $E(c_0^{EH}) \geq E(c_0^{EL})$. Based on Lemma 2, we have $E(c_t^{EH}) \geq E(c_t^{EL})$. That is, $\frac{1}{2} \leq \eta_t \leq 1$. So $2-\eta_t \leq \frac{3}{4} \leq 1+\eta_t$, which indicates $(1+\eta_t)E(c_t^{EL}) \geq (2-\eta_t)E(c_t^{EL})$ $(E(c_t^{EL}) > 0)$. Thus, it is obvious that $\frac{\eta_{t+1}}{\eta_t} \geq 1$. Theorem 2.14 is completed. □

Theorem 2.14 indicates that the predicted quality of the channel at a certain time highly depends on the result predicted in the previous step. Thus, the 'predicted state' can be

rolled over along the time we send packets. We have the following Algorithm 7.

Algorithm 7 PCR(EH, EL)

1: Apply the Winnow Algorithm to predict channel's quality at time t.
2: **if** $E(c_t^{EH}) \geq E(c_t^{EL})$ **then**
3: assign all predicted channel state 'High';
4: **else**
5: assign all predicted channel state 'Low'.
6: **end if**

After we identify the channel state over time, we can apply the bipartite matching to find the optimal provisional schedule and arrange packets in canonical order. In summary, we assign all the pending packets to their latest time slots that they can be feasibly sent before deadlines if $E(c_t^{EH}) \geq E(c_t^{EL})$. If $E(c_t^{EH}) < E(c_t^{EL})$, these pending packets are assigned to the earliest time slots for delivery.

A special case of only two experts. When there are two experts, the algorithmic decisions of these experts work the same as there are multiple ones. We consider this variant for analysis only.

Analysis of the online learning algorithms

We apply the local optimization technique to analyze the online learning algorithms that we have. We first consider two packets, one is sent by the best expert and one is sent by our algorithm. Then we show that the regret is bounded by a ratio of these two packets. Finally, we generalize this ratio to the case in which there are multiple values for packets.

Assume that there are only two kinds of packets denoted as p_s and p_b, where $w_s = 1$ and $w_b = \alpha > 1$. In this case, it is obvious that only two experts are needed. The first expert denoted as EA will schedule the earliest deadline packet whose weight is equal to or larger than w_s, but if there exist both p_s and p_b sharing the same earliest deadline, it will schedule p_b. The second expert denoted as EB will schedule the earliest deadline packet whose weight is equal to or larger than α, but if there are only p_s remained in the current pending set, it will schedule p_s anyway. These two experts are not like EBAV or EBRV,

they are for the purpose of analysis only.

Let the best expert among EA and EB be EXP. Let the total gains of the best expert and of the online learning algorithm are G^{EXP} and G^{ON}, respectively. Then the average regret R is expressed as

$$R = \frac{G^{EXP} - G^{ON}}{T}, \tag{2.6}$$

where T is the overall time spent in scheduling all the packets for the online learning algorithm.

We case study the regret R. Given a time step, if the online algorithm has exactly the same behavior (schedules the same packet) with the best expert in hindsight, then the average regret will be zero. Otherwise, if the average regret is larger than zero, then we can claim that there exist a time step t such that the packet scheduled by the online learning algorithm is different from the one scheduled by the best expert.

- Assume that the best expert EXP sends a packet p_s and the online learning algorithm ON schedules a packet p_b.

  We have the claim as below.

  Remark 2.3. *The best expert must send a packet p_b in the future, i.e., after the time the online learning algorithm sends the packet p_s.*

  Proof. We prove Remark 2.3 using the contradiction method. Assume that the claim fails at some time $t' > t$. We know that in any time step $t' > t$, the best expert sends a packet with less or equal weight comparing to the online learning algorithm. Since t is the first time step that the online learning algorithm differs from the best expert, the best expert will have a strict less gain than the online learning algorithm. This is a contradiction to the assumption that the best expert outperforms the online learning algorithm. This claim is proved. □

  In this case, we charge EXP the value $1 + \alpha$ and charge ON the value α. We have the

internal regret of 1 and the ratio of gains is bounded by $(1+\alpha)/\alpha$.

- Assume that the best expert EXP sends a packet p_b and the online learning algorithm ON schedules a packet p_s.

  In this case, the packet p_b is either sent by ON already or it is pending in ON's buffer. If ON has sent this packet, we can charge EXP the value α to the time step when ON sends it. In that time step, EXP sends a packet with value 1; otherwise, EXP will choose to send a large-value packet, say α, and no packet p_b is available for sending now.

  Now, assume that EXP schedules another large-value packet in the future. Instead, ON schedules the packet p_b and may lose the value of p_b. We consider two time steps. Overall, the value is bounded by a ratio of $(\alpha+\alpha)/(1+\alpha)$.

We come to the result that for this variant only, the internal regret is bounded by $\alpha-1$. This analysis approach can be applied to the cases in which packets have more than two values. We skip the details of expanding this analysis.

Example 3. Here, we give a tight example for the online algorithm we present. We have two kinds of packets with values 1 and α respectively. Initially, B packets with value 1 arrive and their deadlines are $1, 2, \ldots, B$. Then, B packets with value α arrive and their deadlines are $B+1, B+2, \ldots, 2B$. For the online learning algorithm EBRV or EBAV, if it chooses the packets with value 1 to send (of course, it plans to send the packets with value α), we generate input of B packets with value α at time $B+1$ and these packets have deadlines of $2B$. If the online learning algorithm chooses packets with value α to send, we stop releasing new packets. In either case, the average regret is $(\alpha-1)/2$.

Simulations

We design simulations to evaluate the performance of online learning algorithms in Appendix B. In summary, both online learning algorithms and competitive online algorithm perform nearly as good as the optimal offline algorithms. The online learning algorithm

has much lower running time complexity ($O(\log n)$ for each arriving packet) compared with competitive online algorithms ($O(n)$ for each arriving packet).

2.5 Conclusions

In this chapter, we design offline and online algorithms to maximize weighted throughput for scheduling packets with values and deadlines over a wireless channel. The achievement of this work are summarized as below.

1. We provide an polynomial-time optimal offline algorithm for channel with constant quality, as well as a pseudo-polynomial-time optimal offline algorithm for channel with varying quality.

2. We present online algorithms and their competitive analysis, as well as the lower bounds of competitive ratio. Without any information about the channel quality and packets, we design a 2.618-competitive online algorithm SEMI-GREEDY. If packets characters are unknown but the fading states are known beforehand, a 2-competitive online algorithm is achieved by EDF_β ($\beta = 2$). If packet release sequence and their characterizes are known a prior, we provide the lower bound 1.618 of competitive ratio for deterministic online algorithms.

3. We apply online learning approaches to this model as well as a few of its variants. We design online learning algorithms and analyze their performance theoretically in terms of external regret. We also measure these algorithms' performance experimentally. We conclude that no online learning algorithms have a constant regret. However, in general, the designed online learning algorithm works as almost well as the best known competitive online algorithm in practice.

Chapter 3: Energy-Aware Scheduling Algorithms in the DPM Setting

Starting from this chapter, we explore energy-efficient algorithms for scheduling jobs in energy-critical systems. Here, the system is equipped with *dynamic power management* (DPM) as the energy-saving technique which is commonly used to cut a system's energy cost via eliminating or reducing power consumption of one or more of its components. The system has a *higher-power* ACTIVE state and one or more *lower-power* SLEEP or STANDBY states. Jobs can be processed only when the system is at its ACTIVE state but not at any lower-power state. Usually, a constant amount of energy called *transition energy* and a constant time called *transition time* are associated with state transition. We consider two models of energy-aware scheduling problems in this chapter.

In the previous throughput-aware scheduling problems (discussed in Chapter 2), weighted throughput is considered as the gain of the system regardless of the energy consumption. However, this may not be the case in an energy-critical system, where energy cost is involved. Therefore, the profit gained by the system is diluted by its energy expense. In one problem discussed in this chapter, we switch the maximization target to *net profit* which is the value gained by scheduling jobs less the cost paid for energy consumption in the course of scheduling.

Based on the concept of DPM, it is better for the jobs to be grouped together from the energy-saving's point of view. That is, without deadlines, earlier released jobs can be delayed and combined with later released jobs in order to save extra energy spent in powering on or spinning. However, from the perspective of users' experience, a job should not be delayed for too long time. Thus, there exists a tradeoff between saving energy and enhancing users' experience. In this chapter, we also investigate the impact on job processing delay introduced by power-down energy-saving mechanisms. We consider two

important user-perspective system performance metrics — *flow time* and *stretch* (see their definitions in Section 3.2). Specifically, we study bicriteria algorithms that minimize the maximum flow time or largest stretch given a fixed energy budget, and minimize the total energy consumption given an upper bound of flow time or stretch. We consider both offline and online algorithms.

3.1 Net Profit Model

In the settings, time is discrete and each time interval $[t, t+1)$ is called a time step t. A job j with processing time $p_j \in \mathbb{Z}^+$ can be finished within p_j time steps.

Machine. There is one machine (for instance, a packet transmitter in a wired/wireless communication system). At any time, at most one job can be run on the machine. For practical considerations of the current hardware support in embedded networks [47], the machine has only two states: ACTIVE and SLEEP. If the machine is currently running a job, it must be at its ACTIVE state and a machine cannot run a job at its SLEEP state. When the machine does not run any job (though there *may* exist pending jobs), it can be at either its ACTIVE or SLEEP state. We call the machine *spinning* when it is at the ACTIVE state but no job is running. When it is at the ACTIVE and SLEEP states, the machine consumes energy $\mu \in \mathbb{R}^+$ and $e(s) \in \mathbb{R}^+$ per unit time, respectively. Without loss of generality, we assume

$$\mu > 0 \text{ and } e(s) = 0. \tag{3.1}$$

In order to power on (respectively, off) the machine from a SLEEP (respectively, an ACTIVE) state to an ACTIVE (respectively, a SLEEP) state, we have to pay transition energy $C(s/a) \in \mathbb{R}^+$ (respectively, $C(a/s) \in \mathbb{R}^+$). We denote

$$C = C(s/a) + C(a/s). \tag{3.2}$$

Jobs. Let the set of jobs released be $\mathcal{I}$. Each job $j \in \mathcal{I}$ is released at an integer time $r_j \in \mathbb{Z}^+$. Any job j has a *reward* (*value*) $v_j \in \mathbb{R}^+$. If j is processed, its reward/value v_j is contributed to our objective. Let V denote the total profit gained by completing a set of jobs $\mathbf{S} \subseteq \mathcal{I}$, we have

$$V = \sum_{j \in \mathbf{S}} v_j, \tag{3.3}$$

Jobs have deadlines. A job j has an integer deadline $d_j \in \mathbb{Z}^+$. Only jobs that can be finished before their respective deadlines contribute their values to our objective. We do not expect to finish all released jobs in the schedule and hence $\mathbf{S} \subseteq \mathcal{I}$. We consider *under-loaded systems* in which all jobs in $\mathcal{I}$ can be finished (before their deadlines) as long as the machine processes jobs immediately at their arrivals or in a non-decreasing deadline order for all pending jobs.

In this chapter, we consider the general case and some important variants of the net profit model as well. One variant is called *s-uniform deadline instance*. We define the *span* of a job j as $d_j - r_j$ and *s-uniform deadline* as below.

Definition 3.1 (s-uniform deadline [29])**.** *Given a constant s, if for any job j we have $d_j - r_j = s$, then we say that this input instance has s-uniform deadlines.*

Energy cost. Let E denote the total energy used by the machine in finishing a set of jobs $\mathbf{S} \subseteq \mathcal{I}$. Let $T(a)$ (respectively, $T(s)$) denote the total amount of time the machine remains at the ACTIVE (respectively, SLEEP) state. Let $m \in \mathbb{Z}^+$ be the number of times that the machine is powered on during the course of scheduling jobs. Without loss of generality, the machine is assumed at its SLEEP state initially and finally. The total energy cost is defined as

$$E = \mu \cdot T(a) + e(s) \cdot T(s) + (C(s/a) + C(a/s)) \cdot m = \mu \cdot T(a) + C \cdot m. \tag{3.4}$$

Since the values we consider include energy costs and jobs' rewards, we need to keep the units of energy and reward consistent. Thus, we define a constant r to denote the value

paid per unit of energy.

In the net profit model, our objective is to maximize the total *net profit* P of scheduling jobs. Here, $P = V - r \cdot E$, where V and E are specified in Equation 3.3 and Equation 3.4, respectively. Without loss of generality, we assume that for any job $j \in \mathcal{I}$,

$$v_j \geq r \cdot \mu \cdot p_j. \tag{3.5}$$

For those jobs not satisfying Inequality 3.5, we simply drop them out of the input sequence since executing such jobs cannot make any positive net profit.

3.1.1 Previous Work

In previous research, many researchers have developed algorithmic solutions either to minimize the total energy consumption while all released jobs are expected to be finished (such as [48, 49]) or to maximize the total reward by completing a selected set of released jobs under a given energy budget (such as [50,51]). Unfortunately, such prior work has not fully addressed those proposed algorithms' efficiency of using energy, with respect to the patterns of the job sequences. For example, let us assume all jobs will have the same reward upon being finished. Then an algorithm spending amount E of energy in finishing n jobs may not be considered more 'efficient' than an algorithm spending amount $E/4$ of energy in finishing $3n/4$ jobs. Unlike what other researchers have worked on, we target on maximizing the difference between the total reward achieved by running jobs and the total energy cost paid during this course. We call this difference *net profit*. As a result, some jobs may not even be considered for scheduling in our settings, due to the considerations of maximizing net profit. The model we propose is called a *net profit model*.

The first online algorithm studying energy efficiency in completing jobs in the DVS setting was proposed in [52]. Instead of having a fixed reward upon completing a job, Pruhs and Stein defined a job's income as a non-increasing function of its flow time. For this problem, the lower bound of competitive ratio of online algorithms cannot be bounded by

any function of the number of jobs. The underlying idea of the proof is to construct such an example: The online algorithm is forced to run a job, however, another job with a higher profit will be released later. In [52], Pruhs and Stein considered resource augmentation approaches in the multi-processor setting. In this chapter, we consider the DPM setting instead.

3.1.2 General net profit model

The net profit model has its unique challenges. Through the following Example 4, we illustrate the difference between the model of minimizing the total energy cost while completing all jobs (the model discussed in [49]) and the model of maximizing the total net profit by selecting a subset of jobs to be completed. Note that these two models differ only by their objectives.

Example 4. The set of 6 jobs $\{j_1, j_2, \ldots, j_6\}$ has the same reward 2, the same processing time 1, and they have agreeable deadlines as shown in Table 3.1. We assume $\mu = 1$ (μ is the energy cost per time unit when the machine is ACTIVE), $C = 4$ (C is the transition cost as that in Equation 3.2), and $r = 1$ (r is the net profit of finishing one job).

Table 3.1: A set of jobs with uniform rewards and agreeable deadlines.

jobs	release time	deadlines	reward/reward
j_1	1	2	2
j_2	2	5	2
j_3	3	6	2
j_4	7	9	2
j_5	8	10	2
j_6	9	11	2

We realize here that even for such a simple instance (with agreeable deadlines and uniform rewards), the scheduling results of minimizing the total energy and maximizing the total net profit are different. As shown in Figure 3.1, in minimizing the total energy, the machine will be spinning for two time steps with a total energy cost of $E = 12$ and hence, the net profit is 0 (note that all jobs are finished). In maximizing the net profit, the

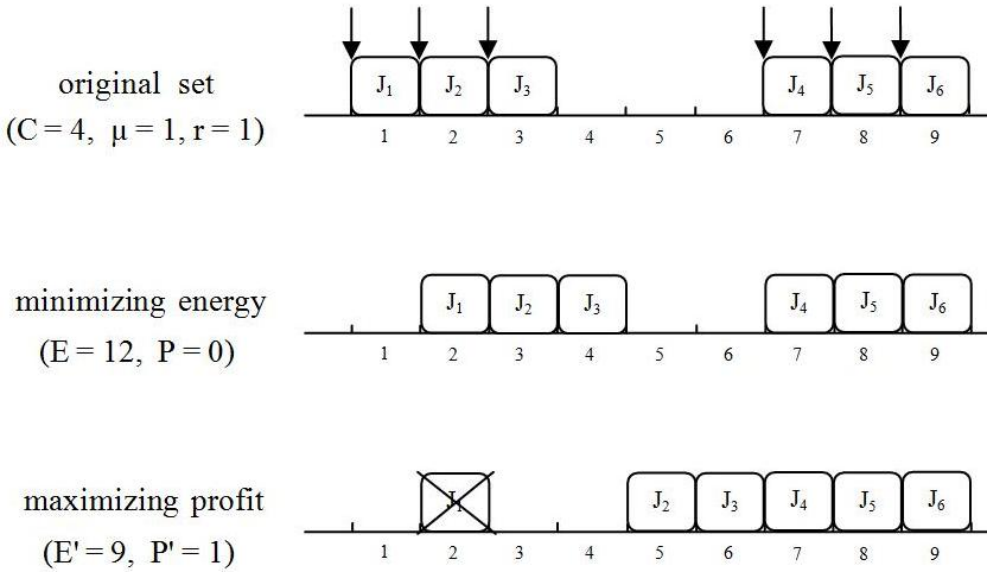

Figure 3.1: Schedules minimizing total energy cost versus maximizing net profit.

machine is allowed to skip running some jobs and thus, job j_1 is skipped and the resulted schedule has a total energy cost of $E' = 9$ and a total net profit $P' = 1$. The energy costs and rewards are calculated as follows (we assume that the machine is at its SLEEP state initially).

$$E = 4 + 8 = 12. \qquad P = 2 \cdot 6 - 1 \cdot E = 0.$$

$$E' = 4 + 5 = 9. \qquad P' = 2 \cdot 5 - 1 \cdot E' = 1.$$

We consider both overloaded systems and underloaded systems in the net-profit model. Remember that in both systems, jobs can be preempted. Based on the definitions given above, we note that preemptive EDF (earliest-deadline-first) policy can identify whether a system is overloaded or underloaded. In a non-preemptive underloaded system and in a preemptive overloaded system, we prove that the problem of maximizing net profit is NP-complete, which implies that it is unlikely to find exact algorithms running in polynomial time unless P = NP. A problem called *strongly NP-complete* indicates that even a pseudo-polynomial time algorithm cannot exist unless P = NP. Our proofs follow polynomial-time reductions from two other proved NP-complete problems.

Definition 3.2 (3-PARTITION [39]). *Given a finite set A of $3m$ elements, a bound $B \in \mathbb{Z}^+$, and a 'size' $s(a) \in \mathbb{Z}^+$ for each $a \in A$, such that each $s(a)$ with $B/4 < s(a) < B/2$ and $\sum_{a \in A} s(a) = m \cdot B$, can A be partitioned into m disjoint sets $S_1, S_2, \ldots, S_m$ such that for $1 \leq i \leq m$, $\sum_{a \in S_i} s(a) = B$?*

Definition 3.3 (SUBSET-SUM [39]). *Given a set of integers and a number M, does the sum of the numbers in some non-empty subset equals exactly to M?*

Theorem 3.1. *Consider a non-preemptive underloaded system. The general net profit problem is a strongly NP-complete problem.*

Proof. Given a subset of jobs, we can verify in polynomial-time whether these jobs can be scheduled with a given target profit. This verifier applies the routine in [49] which has a running time of $O(n^5)$. Note that given a subset of jobs, their total reward is fixed. Thus, minimizing the total energy cost leads to maximizing the total net profit subject to all jobs in this subset are finished.

We reduce the net profit problem from a strongly NP-complete problem 3-PARTITION [39]. The reduction works as follows. Given an instance of 3-PARTITION with $3m$ sizes $s(1), \ldots, s(3m)$ and a target sum B, we create $4m$ jobs such that for $3m$ of them, job i has a processing time $p_i = s(i)$. All these $3m$ jobs share the same release time $R = 0$, a uniform value 1, and a common deadline $D = \sum_{i=1}^{3m} s(i) + \delta \cdot (m-1)$. Also, $\sum_{i=1}^{3m} s(i) = m \cdot B$. There are m other jobs. A job j, $j \in \{1, \ldots, m\}$, has a release time $j \cdot B + (j-1)\delta$, a processing time δ, a value 1, and a deadline $j \cdot B + j \cdot \delta$. We let all energy consumption be zero. Thus, the objective of the net profit model becomes simply to maximize the total value of the schedulable jobs within an interval $[0, D]$. The reduction takes linear time $O(n)$. It is easy to show that we could maximize the total net profit up to $4m$ if and only if there is a positive answer to the 3-PARTITION problem. The proof is completed. □

Theorem 3.2. *Consider an overloaded system. Even if all jobs share the same release time and the same deadline, maximizing net profit is a NP-complete problem.*

Proof. Similarly, we claim that we have a polynomial-time verifier for the net profit problem. We reduce the net profit problem to a NP-complete problem SUBSET-SUM [39] in polynomial time. Given an instance of SUBSET-SUM with n integers $m_1, \ldots, m_n$ and a target sum M, we create n jobs such that a job i has processing time $p_i = m_i$ and a value $v_i = m_i$. All jobs share the same release time $R = 0$ and a common deadline $D = M$. We let all energy consumption be zero. Thus, the objective of the net profit model becomes simply to maximize the total value of the schedulable jobs within an interval $[0, M]$. The reduction takes linear time $O(n)$. It is easy to show that we could maximize the total net profit up to M if and only if there is a positive answer to the SUBSET-SUM problem. The proof is completed. □

3.1.3 Some variants in underloaded systems

In this section, we consider a few important variants of the net profit model. As what we have discussed, given an overloaded system, it is unlikely to get an efficient algorithm running in polynomial time of the input size. Given an under-loaded system, preemptive EDF achieves the optimal total reward gained but it may not lead to an optimal net profit. In this section, we consider underloaded systems only.

Jobs sharing a common deadline

Assume that all jobs share a common deadline. Then for any subset of the jobs, they can be finished in a back-to-back manner. (Remember that we consider an underloaded system.) Thus, only one time of power-on is sufficient. The schedule maximizing the net profit is the same as the one of maximizing the total reward.

Algorithm 8 Greed-Based(D)

```
1: Sort jobs in increasing order of release time.
2: if the total reward is larger than the total energy cost then
3:     run all jobs in the sorted order;
4: else
5:     run nothing.
6: end if
```

Theorem 3.3. *For the variant in which all jobs share a common deadline, Algorithm 8 has a running time complexity of $O(n \log n)$ in maximizing net profit, where n is the number of jobs.*

Proof. Sorting jobs in increasing order of release time takes time $O(n \log n)$. Thus, the algorithm's running time complexity is $O(n \log n)$ where n is the number of jobs. The proof is completed. □

Jobs sharing a common release time

Note that all jobs are available at the same time, thus, any subset of jobs can be run in a back-to-back manner. Similarly, the problem of maximizing net profit could be reduced to maximizing total values under this setting.

Algorithm 9 SRPT-Based(R)

```
1: Sort jobs in increasing order of deadlines.
2: Apply SRPT (shortest-remaining-processing-time first) policy on the jobs.
   {As the system is preemptive and underloaded, all jobs can be finished before their
   deadlines.}
3: if the total reward is larger than the total energy cost then
4:    run all jobs in the sorted order;
5: else
6:    run nothing.
7: end if
```

Similar to the analysis of Theorem 3.3, we have the following result.

Theorem 3.4. *For the variant in which all jobs share a common release time, Algorithm 9 has a running time complexity of $O(n \log n)$ in maximizing net profit, where n is the number of jobs.*

Jobs with agreeable deadlines

We consider the following variant: Jobs may have different release time and different deadlines. The deadlines of jobs satisfy the *agreeable-deadline requirement*. That is, earlier released jobs have no later deadlines. In this variant, we assume that all jobs have integer

release time, integer processing time, and integer deadlines. This assumption implies that each job starts and ends at integer time.

At first, we study the characteristics of an optimal solution and have the following observations for agreeable-deadline instances.

Lemma 3. *In an optimal schedule, the selected jobs can be sorted in increasing order of their release time and they finish before their deadlines. Also, no job is preempted.*

Proof. For an underloaded system, we can use preemptive EDF to schedule all jobs before their deadlines. Note that all jobs are with agreeable deadlines (a later released job has a weakly later deadline), thus, no job is ever preempted in the preemptive EDF schedule in which all jobs are finished. Consider an optimal schedule maximizing net profit. If we ever run a job j partially, we should include j completely in the optimal schedule, since its completion contributes more reward but (under an underloaded system) including this job does not introduce energy cost of powering on the machine. Thus, the optimal schedule maximizing net profit is an arrangement of a subset of the complete jobs. For agreeable-deadline instances, all these jobs are feasibly aligned in increasing order of their release time and finished before their deadlines. The proof is completed. □

We consider a case in which $C = \mu$. For this case, the machine can be regarded as never spinning. At any time it finishes a job and if there is no pending jobs, the machine will shut down and restart to process the next job in the given schedule. We can view a final optimal schedule which maximizes the net profit as multiple, say m, non-overlapping 'blocks of jobs' $B_1, B_2, \ldots, B_m$. Each block B_i consists of jobs consecutively executed in a back-to-back manner without any 'gap' in-between. Given a block B_i, all the jobs in it may be shifted as a whole, but in the same relative order of execution, to the left or to the right subject to the constraints from the jobs' release time and deadlines. If a block cannot move to the left or to the right further without generating a 'gap', we say such a block *fixed at its position.*

Lemma 4. *If $C = \mu$, in one optimal schedule maximizing net profit, all blocks of jobs are*

fixed at their positions.

Proof. From Lemma 3, all selected jobs in the optimal schedule can be arranged in the increasing order of their release time (deadlines). Thus, for each block B_i of jobs, all jobs can keep the fixed relative order when we run this set of jobs. If B_i can be shifted as a whole to its left (or its right), we can proceed to shift it until a 'gap' is generated. If a 'gap' is not generated during this shift, this block must meet another block of jobs to its left or to its right. It is better to combine these two blocks together with one time of transition energy-cost saved. If a 'gap' is generated subject to the release time and deadline constraints, we would better keep this set of jobs as a whole in order to save an extra time of transition cost. The proof is completed. □

Based on Lemma 3 and Lemma 4, we design Algorithm 10 based on the techniques of both dynamic programming and divide-and-conquer to maximize net profit. The high-level ideas come as follows: Consider a job i running continuously from time s_i to $s_i + p_i$ ($[s_i,\ s_i + p_i] \subseteq [r_i,\ d_i]$) in the optimal schedule (based on Lemma 3). Then, there must not exist a block without job i overlaps the range $[r_i, d_i]$. Otherwise, we can shift i and combine it with the other block to save one time of power-on cost. If i must exist in the optimal schedule, then we can force any jobs j having $[r_j, d_j]$ to overlap with the block that i is in to share the same block with i. We consider jobs in decreasing order of deadlines. Let the packet with the latest deadline be l. The optimal schedule with the best net profit is chosen from the following two cases: (1) l not in the optimal schedule; and (2) l in the optimal schedule and in the last block such that the last block cannot shift to the left without generating a 'gap'. If l is in the optimal schedule, the total reward of the block which it resides in (or l itself if this block has only one job) must be $\geq C$.

Theorem 3.5. *For the variant in which all jobs have agreeable deadline and $C = \mu$, Algorithm 10 has a running time complexity of $O(n^2 \log n)$ in maximizing net profit, where n is the number of jobs.*

Proof. We note that if a job l is in an optimal schedule, then all jobs calculated in $B(l)$

Algorithm 10 DP-Based($C = \mu, \mathcal{I}$)

1: Sort all jobs of a given set S in decreasing order of deadlines. Initially, this set is $\mathcal{I}$.
2: Let the last released job in this set S be l.
3: Use $B(i)$ to denote the block containing a job i.
4: Construct the last block including l with the earliest execution time for the whole block. This step can be implemented as follows: Initialize $B(l) = \{l\}$. Insert jobs in decreasing deadline order as long as $B(l)$ can be successfully finished. Shift the block $B(l)$ as a whole to its fixed position at its left side, if necessary.
{Since we consider jobs in decreasing order of deadlines, we move a block to its fixed position on the left. Otherwise (if we sort packets in increasing order of release time), we shift a block to a fixed position on the right.}
5: Let $S(l)$ be the start time to execute the block $B(l)$ containing l.
6: The optimal net profit $P(S)$ is calculated recursively as

$$P(S) = \max\{P(S \setminus B(l)) + \sum_{i \in B(l)} v_i - C, P(S \setminus \{l\})\}. \tag{3.6}$$

7: Apply above formula for each subset of jobs after we identify whether or not to keep the block of jobs $B(l)$ in each iteration.

in Algorithm 10 must be in the optimal schedule as well, since we can always increase the total reward without paying extra energy cost of powering on the machine. From Lemma 4, if l is in the optimal solution, then the calculated $B(l)$ must belong to an optimal schedule as well. These observations imply the correctness of the algorithm. Also, we conclude that in calculating $P(S)$, the results $P(S \setminus B(l))$ and $P(S \setminus \{l\})$ can be stored in a table whose size is $O(n)$. Let the set of jobs sorted in order be $\{j_1, j_2, \ldots, j_n\}$. Let $Z_i = \{j_1, j_2, \ldots, j_i\}$. Then the table contains values of $P(Z_1), P(Z_2), \cdots, P(Z_n)$. We will fill in this table during the execution of Algorithm 10 and the optimal solution will be obtained.

Now, we consider the running time complexity of Algorithm 10. Let $T(n)$ denote the total running time of a given input instance with a size of n. Here we employ a table to memorize all the calculated temporary values (say, rewards $P(S \setminus B(l))$ and $P(S \setminus \{l\})$) after we identify later released jobs (say $B(l)$ and l). Consider the recurrence Equation (3.6). $P(S \setminus B(l))$ is used by both $P(S)$ and $P(S \setminus \{l\})$. From Equation (3.6), we have $T(n) = T(n-1) + T'(S) + C_0$, where $T'(S)$ is the total cost of calculating $B(l)$ and C_0 is the total cost of comparing $P(S \setminus B(l)) + \sum_{i \in B(l)} v_i - C$ with $P(S \setminus \{l\})$. In calculating $T'(S)$, sorting

jobs takes time $O(|S| \log |S|)$. Calculating $B(l)$ takes time $O(|S|)$. (We can run EDF to examine whether or not a packet j can be inserted into $B(l)$ without generating a 'gap'.) C_0 is a constant. Thus, the whole running time of the algorithm is $O(n^2 \log n)$. The proof is completed. □

3.1.4 Conclusions

In this section, we design scheduling algorithms for jobs with release time, processing time, rewards and deadlines. We address a model in which we maximize the difference between the total reward achieved by delivering jobs and the total energy cost paid during this course. We discuss the hardness of the general model and introduce a few polynomial-time optimal algorithms for some important variants. This work lies in the line of research on DPM. In Table 3.2, we summarize our main results on the net profit model.

Table 3.2: Summary of hardness of the net profit model.

Settings	Overloaded	Underloaded
Non-preemptive	Strongly NP-complete (Theorem 3.1)	Strongly NP-complete (Theorem 3.1)
Preemptive	NP-complete (Theorem 3.2)	Some polynomial-time algorithms (Section 3.1.3)

3.2 Tradeoff Model (Between Flow/Stretch and Energy)

In this model, the system that we study is in a one-machine environment. The machine has only two states: ACTIVE and SLEEP. A job is eligible of being processed only when a machine is at its ACTIVE state. Without loss of generality, the machine is assumed to consume energy 1 unit and 0 unit per unit time when it is at the ACTIVE and SLEEP states, respectively. For each time of powering on the machine from the SLEEP state to the ACTIVE state, we have to pay a constant transition energy cost $E_{tran} \in \mathbb{R}^+$. Without loss of generality, the transition energy cost from the ACTIVE state to the SLEEP state can be assumed negligible, since we can always count this cost to the cost of its most recent

previous powering on event.

Jobs arrive at the machine for processing over time. Each job J_j has an arriving time $r_j \in \mathbb{R}^+$ and a processing time $p_j \in \mathbb{R}^+$. Jobs can be preempted and resumed later. There is no cost associated with each job preemption or resumption. Let c_j denote the completion time of J_j. A job J_j's flow time f_j and stretch s_j are defined below.

Definition 3.4 (Flow Time, Stretch)**.** *A job J_j's flow time f_j is defined as the difference between its completion time and release time. J_j's stretch s_j is defined as the ratio between its flow time and its processing time.*

$$\begin{aligned} f_j &= c_j - r_j. \\ s_j &= \frac{f_j}{p_j} = \frac{c_j - r_j}{p_j}. \end{aligned}$$

The maximum flow time and the largest stretch are denoted as $F_{\max} = \max_j f_j$ and $S_{\max} = \max_j s_j$, respectively.

Stretch is a natural and useful metric such that jobs with longer processing time must prepare to tolerate longer waiting time (flow time) [53, 54]. Also, bounding the maximum flow time or the largest stretch for jobs avoids job starvation in multi-task environments.

Let E_{total}^{ALG} denote the total energy consumption by the machine using a power-down strategy ALG. Let T_{active}^{ALG} and T_{sleep}^{ALG} denote the total amount of time that the machine remains at the ACTIVE and SLEEP states, respectively. We omit the superscripts of these notation when the power-down strategy ALG is of no confusion. Let $m \in \mathbb{Z}^+$ denote the total number of times the machine being powered on during the course of scheduling jobs. The total energy cost is then calculated as

$$E_{total} = 1 \cdot T_{active} + 0 \cdot T_{sleep} + m \cdot E_{tran} = T_{active} + m \cdot E_{tran}.$$

Let F_{bound} denote the upper bound of a job's flow time, S_{bound} denote the upper bound

of a job's stretch, and E_{budget} denote the total energy budget. The objectives considered in this model are (recall $F_{\max} = \max_j f_j$ and $S_{\max} = \max_j s_j$):

1. $\min E_{total}$, subject to $F_{\max} \leq F_{bound}$;
2. $\min F_{\max}$, subject to $E_{total} \leq E_{budget}$;
3. $\min E_{total}$, subject to $S_{\max} \leq S_{bound}$;
4. $\min S_{\max}$, subject to $E_{total} \leq E_{bound}$.

3.2.1 Previous Work

We note that stretch is a natural and useful metric such that jobs with longer processing time must prepare to tolerate longer waiting time (flow time or sojourn time) [53, 54]. Also, bounding the maximum flow time or the largest stretch for jobs avoids job starvation in multi-task environments. However, energy consumption has not been considered as a constraint in these models.

There are two research work closely related to mine. Albers and Fujiwara [55] studied an online version of the model in a DVS setting whose objective is to minimize the *sum of total energy cost and total flow time*. Instead, we consider DPM setting here. Baptiste et al. [49] studied offline version of scheduling jobs with release time and deadlines in a DPM setting with the objective of minimizing total energy consumption. However, they did not consider the negative impact of energy-saving on jobs' flow time or stretch.

3.2.2 Offline algorithms

In this section, we introduce four offline algorithms optimizing energy consumption and maximum flow time or largest stretch, respectively. The ideas associated with these algorithms are the greedy approach, the dynamic programming approach, and the doubling technique.

Minimizing total energy consumption subject to an upper bound of flow time

Let an optimal solution minimizing the total energy consumption E_{total} subject to an upper bound of flow time F_{bound} be OPT. We first study a few properties of OPT and then we design OPT based on these properties.

In the single-machine environment, we have the following critical observation.

Lemma 5. *In OPT, the earlier released jobs are finished earlier (no later) than the later released jobs. Also, no jobs are preempted.*

Proof. Lemma 5 can be proved using a simple exchange argument. First, we prove that in OPT, jobs are executed in the order that Lemma 5 specifies. Otherwise, assume that we are given a schedule with two jobs J_i and J_j that are running 'out of the order'. Then we can always swap the order of executing these two jobs, within the time periods that have been allocated to these two jobs for execution in the given schedule, to reduce the maximal flow time among these two jobs. This swapping does not introduce more energy or increase other jobs' flow time. Next, we convert an OPT with possible preemption into one without preemption. As earlier released jobs are to be finished earlier (no later) than later released ones, we can always execute earlier released jobs without preemption until they are completely finished. Adding this restriction does not increase a job's flow time. □

Lemma 5 states that *all the jobs in OPT are executed in a FIFO (First-In-First-Out) order*, though these jobs may not be executed in a back-to-back manner.

To construct a min-energy schedule OPT with an upper bound of flow time F_{bound}, we start with a schedule with a minimum maximal flow time but without energy considerations. Let this schedule be ALG. In ALG, there is a job queue and every newly released job is appended at the end of this queue. The machine is *non-idling* (that is, the machine has to run a job as long as the queue is not empty) and it processes all the pending jobs in a FIFO order. See Algorithm 11 below for the description of ALG.

Based on Lemma 5 and its proof, we immediately have the following result.

Corollary 3. *ALG has the smallest maximum flow time among all algorithms.*

Algorithm 11 ALG

1: Append newly arriving jobs at the end of the pending job queue.
2: **while** there are pending jobs **do**
3: execute jobs in a FIFO order.
4: **end while**

In the schedule constructed by ALG, we cluster jobs into groups according to their execution patterns.

Definition 3.5 (Batch of Jobs)**.** *In the schedule constructed by ALG, all the jobs executed in a back to back and consecutive manner are grouped together as a* batch of jobs. *These batches are indexed in increasing order of the start time of the first jobs in these batches as* $G_1, G_2, \ldots, G_k$.

Lemma 6. *Let the k batches of jobs generated by the algorithm ALG be $G_1, G_2, \ldots, G_k$, respectively, in increasing time order. All the jobs in the same batch G_i are still executed in a back to back manner in OPT.*

Note that it is possible that multiple batches of jobs are grouped in execution as one larger batch of jobs.

Proof. Assume OPT has k^* so-called *optimal batches* $G_1^*, G_2^*, \ldots, G_{k^*}^*$. In the following, we prove that $G_i (i \in \{1, \ldots, k\})$ belongs to exactly one optimal batch, from which Lemma 6 can be proved.

From Lemma 5, we know that all the jobs in OPT are executed in a FIFO order. Thus, the jobs in OPT should follow the same relative order as they are in the algorithm ALG. This observation turns out that if any assumed batch G_i belongs to two different optimal batches in OPT, for example, G_u^* and G_w^* with $u < w$, then we must have $u + 1 = w$ — G_u^* and G_w^* are two neighboring batches in OPT. Denote the part of G_i belongs to G_u^* (G_{u+1}^*, respectively) as $\tilde{G_{i_1}}$ ($\tilde{G_{i_2}}$, respectively). Next we will prove that in one optimal min-energy schedule OPT, this case that G_i consists of two separate groups $\tilde{G_{i_1}}$ and $\tilde{G_{i_2}}$ does not need to happen. Consider any given batch of jobs G_i from the algorithm ALG. Note that if we split G_i into two neighboring parts, the only reason is to save possible energy by merging

the second part with the following batch as G_{i+1}, otherwise, the split is unnecessary upon increased energy consumption or flow time. However, this *'split and merge'* procedure can be saved because the energy reduced through merging will be no more than the extra energy that we have to pay through splitting since the machine either keeps ACTIVE during the gap generated between $\tilde{G_{i_1}}$ and $\tilde{G_{i_2}}$ or the machine has an extra cost of E_{tran}. □

From Lemma 5, Corollary 3, and Lemma 6, we have the following greedy idea to calculate OPT: Start from the batches $G_1, G_2, \ldots, G_k$ generated by the algorithm ALG. For the batch G_i ($i \in \{1, \ldots, k-1\}$), the algorithm postpones the start time of this batch till a time such that some job in this batch has its flow time reach F_{bound}, or till G_i meets the following batch G_{next}. (During the course of this algorithm, G_{next} may not always be G_{i+1}, due to possible merging. See below.) If any job in G_i's flow time violates the constraint F_{bound} before G_i's last job's completion time meets the start time of its following batch, then we denote G_i as G'_i and study the next batch G_{next}. If G_i's last job's completion time meets the start time of its following batch, then we combine these two batches as a single new batch. We still call the newly generated batch G_i and repeat the same process, so on and so forth. The algorithm stops when all the batches are considered and the optimal schedule is consist of a set of batches $G'_1, G'_2, \ldots, G'_{k'}$. The algorithm OPT is described in Algorithm 12.

Algorithm 12 OPT (F_{bound})

```
1: Run ALG and get batches G_1, G_2, ..., G_k.
2: Set i = 1, next = i + 1.
3: while i < k do
4:    postpone the start time of G_i till a time such that some job in G_i has its flow time
      reach F_bound or G_i's last job's completion time meets the following batch G_next's start
      time.
5:    if G_i does not meet G_next then
6:       set G'_i = G_i, i = next;
7:    else
8:       update G_i with G_i ∪ G_next;
9:       set next = next + 1.
10:   end if
11: end while
```

Theorem 3.6. *OPT minimizes total energy consumption subject to each job's flow time bounded by F_{bound}. OPT has a running time of $O(n^2)$.*

Proof. Theorem 3.6 can be proved using an exchange argument. Based on Lemma 5 and Lemma 6, the relative order of $G_1, \ldots, G_k$ remains the same as in OPT's solution $\mathcal{G}' = \{G'_1, \ldots, G'_{k'}\}$. Consider an optimal solution $\mathcal{G}^* = \{G^*_1, \ldots, G^*_{k^*}\}$. If there exists some i such that $G'_i \neq G^*_i$, we can always modify G^*_i to be G'_i without introducing extra flow time or extra energy consumption. Without loss of generality, we assume $i = 1$.

- Assume that G^*_1 contains the same set of jobs as the batch G'_1.

  It is always possible to set G^*_1's start time same as that of batch G'_1. Note that G'_1 has the latest position to be executed without violating the flow time bound F_{bound}, G^*_1 cannot start later than G'_1. We postpone G^*_1, if needed, till the start time of G'_1 without increasing energy cost or violating flow time bound.

- Assume that G^*_1 contains a different set of jobs from the batch G'_1.

  G^*_1 contains fewer number of jobs than G'_1, since from Algorithm 12, G'_1 can not be further merged with any following batches without violating flow time bound. The set of batches $G_{diff} = G'_1 \setminus G^*_1$ must appear in the front of the next batch G^*_2. We append G_{diff} at the end of G^*_1 without increasing energy or violating flow time bound and convert G^*_1 to be G'_1.

Next, we show OPT's running time complexity. The algorithm ALG takes time $O(n)$. For each batch, we examine the last time that it can be postponed, which takes time $O(n)$. For each batch, we need to run this test once, thus OPT has its total running time $O(n^2)$. □

Minimizing maximum flow time subject to a bounded energy consumption

In this section, we consider minimizing the maximum flow time subject to a given energy budget E_{budget}. Note that Lemma 5 and Lemma 6 still hold for this setting. Let the maximum flow time returned by the algorithm ALG (see Algorithm 11) be $\tilde{F}$. Realizing that $\tilde{F}$ is the lower bound of maximum flow time that an algorithm can achieve, we start

from an estimated flow time $\tilde{F}$ and run the algorithm OPT (Algorithm 12) to find out the minimum energy cost $\tilde{E}$. If $\tilde{E} > E_{budget}$, we relax $\tilde{F}$ and employ the doubling technique to estimate $F^*_{\max}$ — the optimal maximum flow subject to a bounded energy consumption E_{budget}. Noticing that the algorithm OPT takes time $O(n^2)$, we thus have the following result.

Theorem 3.7. *There exists an optimal algorithm minimizing the maximum flow time subject to the total energy consumption bounded by E_{budget}. This algorithm has a running time of $O(n^2 \log F^*_{\max})$.*

Minimizing total energy consumption subject to an upper bound of stretch

Consider a job J_j. Its stretch is defined as $s_j = f_j/p_j$, where $f_j = c_j - r_j$ and c_j is the completion time of the job J_j. Let the upper bound of stretch be S_{bound}. For a job J_j, we need to guarantee that $s_j = \frac{f_j}{p_j} = \frac{c_j - r_j}{p_j} \leq S_{bound}$, which is $c_j \leq r_j + p_j \cdot S_{bound}$. Thus, in order to make sure that every job J_j has its stretch bounded by S_{bound}, we associate a variable with each job J_j to denote its *deadline* d_j and set $d_j = r_j + p_j \cdot S_{bound}$.

Now, our objective is to generate a schedule satisfying all the jobs' deadlines and minimizing the total energy consumption. [49] has given a dynamic-programming based solution to minimize the total energy consumption of scheduling all jobs subject to jobs having release time, processing time, and deadlines. The algorithm has a running time of $O(n^5)$. Note that our conversion ($d_j = r_j + p_j \cdot S_{bound}$) takes linear time. We have the following result.

Theorem 3.8. *There exists an $O(n^5)$-time algorithm minimizing total energy consumption subject to each job's stretch bounded by S_{bound}.*

Minimizing largest stretch subject to a bounded energy consumption

At first, we have the following observation.

Remark 3.1. *Given a set of jobs with release time and processing time, an optimal algorithm with energy budget E^1_{bound} has a no-smaller largest stretch than that of an optimal algorithm with energy budget E^2_{bound}, where $E^1_{bound} \leq E^2_{bound}$.*

Then based on Remark 3.1 and Theorem 3.8, using binary search, we easily conclude the following theorem.

Theorem 3.9. *There exists an $O(n^5 \log S_{\max})$-time algorithm minimizing the largest stretch $S_{\max}$ subject to the total energy consumption bounded by E_{budget}.*

3.2.3 Online algorithms

Executing jobs and dynamically powering-on/off the machine is essentially an online decision-making problem. Algorithms for such kind of problems must operate without knowing arriving time or service requirements of future requests. This demand motivates us to study online algorithms for the bicriteria model. In this section, we design two online algorithms with the objectives of minimizing energy consumption subject to an upper bound of flow time and an upper bound of stretch, respectively.

The most widely employed metric in evaluating online algorithms' performance is *competitive ratio* [4]. However, competitive analysis sometimes provides a pessimistic result of an online algorithm's performance since the adversary can adaptively generate the input to beat the online algorithm. Consider the following example.

Example 5. Fix a constant c. We will show that there exists an input instance such that any online algorithm has a competitive ratio (of flow time or total energy consumption) $> c$. In this input instance, all the jobs are unit length. We name an online algorithm ON and an optimal offline algorithm OPT. Initially, a job J_1 is released at time 0. Remember that the machine consumes energy 1 (0, respectively) unit per unit time when it is ACTIVE (SLEEP, respectively).

- Assume that at time $c-1$, ON has not processed J_1.

Then no more jobs are released and OPT processes J_1 at time 0. ON has J_1's flow time $> c$ while OPT has J_1's flow time $= 1$. OPT has energy cost $E_{tran} + 1$, the minimum among all the algorithms processing J_1. The competitive ratio of maximum flow time is $> c$.

- Assume that at time $t < c - 1$, ON processes J_1.

  At time $t + 1$, ON finishes J_1. If ON keeps ACTIVE till time

  $$t' = t + 1 + (c - 1)(E_{tran} + 1),$$

  and ON consumes energy at least

  $$E_{tran} + (t' - t) \cdot 1 = c(E_{tran} + 1),$$

  then OPT does not release any new jobs. OPT executes J_1 at time 0 for one unit time with energy cost of $E_{tran} + 1$ and flow time 1. ON has energy cost of c times of what OPT has. ON's maximum flow time is t times of what OPT has.

  Assume that ON processes J_1 and keeps active before t'. Then at the time when ON powers off the machine, OPT releases one unit-length job. We repeat this pattern: As long as ON powers off its machine after processing the x-th released job at time t, if ON keeps ACTIVE $< c(E_{tran} + x)$ and its maximum duration of being SLEEP is $< c$, then OPT releases the $(x + 1)$-th job at time t. Otherwise, OPT stops releasing new jobs.

  For the first x released jobs, if ON keeps ACTIVE $\geq c(E_{tran} + x)$ or its maximum duration of being SLEEP is $\geq c$, then OPT can execute each of these x jobs immediately at their release time and go to SLEEP after completing them. The ratio of maximum flow time or energy consumption is at least c. If ON keeps ACTIVE $< c(E_{tran} + x)$ and its maximum duration of being SLEEP is $< c$ for the first x released jobs, we keep this pattern repeat for sufficient time. Thus, ON powers on the machine many

times, compared with OPT which powers on the machine only once but processes all the released jobs in a back-to-back manner. The energy consumption ratio can be larger than any given constant c. (However, OPT has a larger maximum flow time.)

Due to the pessimistic results returned by the standard competitive analysis (see Example 5), in the area of scheduling algorithms, people usually consider the *resource augmentation approach* [56] as an alternative analysis method for online algorithms. In the framework of resource augmentation, online algorithms are given extra resources, such as faster machines or more number of machines, than their adversaries to compensate the lack of future input instances. In this section, we study online algorithms in a framework similar to that of resource augmentation. Instead of speeding up online algorithms in executing jobs, we compare online algorithms with extra flow time $\hat{F}$ or with α times of optimal stretch against non-idling adversaries.

Consider a non-idling adversary OPT. OPT cannot be in the SLEEP state if there are pending jobs. Let OPT's maximum flow time be F^*, largest stretch be L^*, and energy consumption be E^*. Consider an online algorithm ON which is allowed to be in the SLEEP state even though there are pending jobs.

Definition 3.6 (Weak Competitive Ratio with respect to Flow Time)**.** *Let ON's maximum flow time be F^{ON} and energy consumption be E^{ON}. The* weak competitive ratio with respect to extra flow time *is E^{ON}/E^* subject to $F^{ON} \leq F^* + \hat{F}$.*

Definition 3.7 (Weak Competitive Ratio with respect to Stretch)**.** *Let ON's largest stretch be L^{ON} and energy consumption be E^{ON}. The* weak competitive ratio with respect to larger stretch *is E^{ON}/E^* subject to $L^{ON} \leq \alpha L^*$, where α is a given number.*

In following, we introduce two online algorithms with extra flow time and larger stretch, respectively. We also analyze their weak competitive ratios.

An online algorithm ONF in minimizing total energy consumption subject to an upper bound of extra flow time $\hat{F}$

In OPT, jobs are not preempted and they are executed in the FIFO order (see Lemma 5). In the scope of discussing and designing algorithms below, we only consider algorithms running jobs in the FIFO order since this consideration does not lose OPT's optimality. Consider a non-idling algorithm ALG. The non-idling adversary OPT executes the same job at the same time as what ALG does. Remember that OPT optimizes its energy consumption and sets either a SLEEP or an ACTIVE state when there are no jobs to run, since it is capable of foreseing the next release time of jobs.

The algorithm ONF. The algorithm ONF's idea has the flavor of "lazy scheduling". The machine is at its SLEEP state initially. If the machine is at its ACTIVE state, it will execute all the pending jobs in the FIFO order. After finishing all the pending jobs, the machine goes to its SLEEP state immediately. Newly arriving job will be appended at the end of the pending job queue. The machine is powered on to its ACTIVE state only when the current time reaches the first pending job J_j's $r_j + \hat{F}$. This algorithm is described in Algorithm 13.

Algorithm 13 ONF($\hat{F}$)

```
1: Append newly arriving jobs at the end of the pending job queue.
2: while the machine is at its ACTIVE state do
3:    if there are pending jobs then
4:       execute jobs in a FIFO order;
5:    else
6:       go to the SLEEP state.
7:    end if
8: end while
9: while the machine is at its SLEEP state do
10:   if the current time reaches r_j + F̂ for the first pending job J_j then
11:      power on the machine to its ACTIVE state.
12:   end if
13: end while
```

The analysis of ONF.

Theorem 3.10. *The algorithm ONF finishes all the jobs arriving at the machine. Each job J_j has its flow time bounded by $f_j^* + \hat{F}$. ONF has a running time complexity of $O(n)$ where n is the number of jobs in the input instance.*

Proof. ONF's running time complexity is linear $O(n)$ of the number of jobs since we process each arriving job in constant time. Now, we prove that each job J_j has its flow time f_j^{ONF} bounded by $f_j^* + \hat{F}$, where f_j^* is the flow time of J_j in the non-idling adversary OPT. Consider a job J_j.

Assume that J_j is not the first job to run after the machine sleeps for some time. At the time when ONF runs J_j, we consider the most recently executed job J_f since the last SLEEP state. Note that J_f is executed at time $r_f + \hat{F}$. Remember that all the jobs are executed in the FIFO order, J_j is executed at time $\hat{F}$ after $r_j + f_j^*$. Assume that J_j is the first job to run after the machine sleeps for some time. We have $f_j^{\text{ONF}} = \hat{F} + p_j$. □

Remember our assumption of consuming energy 1 unit and 0 unit per unit time being ACTIVE and SLEEP, respectively.

Theorem 3.11. *The algorithm ONF has a weak competitive ratio of $\frac{2 \cdot E_{tran}}{E_{tran} + \hat{F}}$ with respect to extra flow time $\hat{F}$.*

Proof. Our proof employs a *phase-based* charging scheme. We first define non-overlapping *phases* created by the online algorithm ONF. We then prove that in each phase or in multiple consecutive phases, with an appropriate charging scheme, ONF is $\left(\frac{2 \cdot E_{tran}}{E_{tran} + \hat{F}}\right)$-competitive. This directly results that ONF's competitive ratio is $\left(\frac{2 \cdot E_{tran}}{E_{tran} + \hat{F}}\right)$ over the whole schedule.

Let $\mathcal{I}$ be a sequence of jobs. Note that ONF does not spin, that is, ONF does not keep ACTIVE but run no jobs. Then, ONF's schedule consists of ACTIVE and SLEEP periods alternatively. We define each active period as a *phase* of processing jobs. For the

set of jobs in each phase defined by ONF, we compare ONF with an optimal offline non-idling algorithm OPT. (OPT has $\hat{F} = 0$.) Consider a phase and the set of jobs P that is processed by ONF. ONF consumes energy $E_{tran} + \sum_{J_j \in P} p_j$. Now, we study OPT's energy consumption for the same set of jobs. From Lemma 5, OPT runs this set of jobs in the same order as ONF's schedule. Also, note that ONF executes a job no earlier than OPT does for the same job.

1. Assume that OPT executes the job J_f at time $> r_f$.

   Then ONF executes the job which is released before J_f no earlier than OPT does. When J_f is released, both OPT and ONF are ACTIVE and thus, J_f should not be the first job to execute after a SLEEP period.

2. Assume that OPT executes the job J_f at time $= r_f$.

   Then OPT consumes energy $\min\{E_{tran}, T'_{sleep}\} + \sum_{J_j \in P} p_j$, where T'_{sleep} is the idle time from the last active time to r_f in OPT's schedule.

If $T'_{sleep} < \hat{F}$, then ONF should have been ACTIVE at time r_j and thus, J_f is not the first job to execute in P. If $T'_{sleep} > \hat{F}$, OPT consumes energy $\min\{E_{tran}, \hat{F}\} + \sum_{J_j \in P} p_j$. If $\min\{E_{tran}, \hat{F}\} = E_{tran}$, for this set of jobs P, the cost ratio is 1. If $\min\{E_{tran}, \hat{F}\} = \hat{F}$, we understand that OPT keeps ACTIVE just before running the first job J_f in P. We then trace back to the immediate proceeding period in which OPT and ONF run the same set of jobs P'. We keep tracing back till we locate a period in which OPT starts to run a job from its SLEEP state. This period must exist since we assume both OPT and ONF start from the SLEEP state. Let the set of jobs to be run during these periods be $\bigcup P$. Note that OPT keeps ACTIVE in all these periods. We observe that for ONF's each time of power-on to run a set of jobs P_i, if OPT consumes energy less than $\hat{F}$ before OPT runs P_i, then this set of jobs P_i should not be delayed for $\hat{F}$ units. Thus, there is no energy cost associated with powering on the machine to run P_i. Let the immediate proceeding set of jobs to run

be P_{i-1}. Combining these two neighboring periods, we have ONF's competitive ratio as

$$\frac{2 \cdot E_{tran} + \sum_{J_j \in P_i \cup P_{i-1}} p_j}{\min\{E_{tran}, \hat{F}\} + \sum_{J_j \in P_i \cup P_{i-1}} p_j} \leq \frac{2 \cdot E_{tran}}{E_{tran} + \hat{F}}.$$

□

When $\hat{F}$ is set 0, from Theorem 3.11, we immediately have the following result.

Corollary 4. *When online algorithms are enforced to be non-idling, the algorithm ONF is* 2*-competitive.*

Corollary 4 is the standard result that has been presented in [57]. Note that when $\hat{F} = 0$ for online algorithms, the lower bound of competitive ratio of ONF is 2. Thus, in the bicriteria model, increasing $\hat{F}$ beats *any* deterministic non-idling online algorithms. Let the optimal non-idling offline algorithm be OPT.

Next, we present the lower bound of competitive ratio β with respect to flow time for deterministic online algorithms against non-idling offline algorithms for the bicriteria model. Here, we consider an *adaptive offline adversary*, who generates a job sequence based on the past behavior of the online algorithm. We refer to the optimal offline algorithm as OPT and online algorithm as ONF, respectively.

Note that from Theorem 3.11, if $\hat{F} \geq E_{tran}$, we know that there exists an optimal online algorithm against non-idling offline algorithms. Thus, we only consider the lower bound of competitive ratio when $\hat{F} < E_{tran}$. The main technical contribution here is to show that any online algorithm cannot perform arbitrarily close to the optimal offline algorithm. *This lower bound even holds for scheduling unit-length jobs*, that is, all the jobs have processing time 1.

Theorem 3.12. *Assume $\hat{F} < E_{tran}$. The lower bound of weak competitive ratio for deterministic algorithms is $\geq \min\left(\frac{3+2\cdot E_{tran}}{2+2\cdot E_{tran}}, \frac{2+2\cdot E_{tran}}{2+E_{tran}+\hat{F}}\right)$.*

Proof. Initially, we assume that both OPT and ONF have their machines at the SLEEP states.

Let the adversary release a job J_0 at the beginning of step 1. OPT powers on the machine to process J_0. For ONF, there are two options now: (1) letting the machine go to its ACTIVE state to process J_0, or (2) buffering J_0 into the queue till some time $\hat{F} - 1$.

- Assume that ONF powers on the machine to the ACTIVE state immediately at J_0's arrival

  Then after completing J_0 at time 1, ONF either keeps the machine ACTIVE (spinning) or turns to the SLEEP state.

  - Assume that ONF keeps in the ACTIVE state.

    The adversary then releases another job J_1 at time $E_{tran} + 2$. There are no more jobs released.

    In this case, OPT powers off the machine after finishing J_0 at the end of step 1 and then powers on to an ACTIVE state at time $E_{tran} + 2$ to process job J_1. The total cost paid by OPT is $E^1_{\text{OPT}} = 2 + 2 \cdot E_{tran}$.

    On the other hand, ONF either keeps the machine ACTIVE till J_1 arrives with a total cost of $E^1_{\text{ONF}} = 3 + 2 \cdot E_{tran}$, or ONF lets the machine SLEEP at some time before $E_{tran} + 2$ and powers it on to the ACTIVE state at some time before the deadline of job J_1 to process it with a total cost $\geq E^2_{\text{ONF}} = 3 + 2 \cdot E_{tran}$.

    The lower bound of competitive ratio β_1 is

    $$\beta_1 \geq \frac{\min(E^1_{\text{ONF}}, E^2_{\text{ONF}})}{E^1_{\text{OPT}}} = \frac{3 + 2 \cdot E_{tran}}{2 + 2 \cdot E_{tran}} \tag{3.7}$$

  - Assume that ONF powers off the machine to its SLEEP state at the end of time 1.

The adversary releases a job J_2 at the beginning of step 2. There are no more jobs released.

OPT keeps the machine ACTIVE to finish job J_2 with a total cost of $E^2_{\text{OPT}} = 2 + E_{tran}$.

On the other hand, ONF has chosen to power off the machine to SLEEP. To get J_2 finished, it will need to switch back to the ACTIVE state again at some time $t \leq \hat{F} + 2$, the latest time for J_2 to be executed. The total cost of ONF is $E^3_{\text{ONF}} \geq 2 + 2 \cdot E_{tran}$.

The lower bound of competitive ratio β_2 is

$$\beta_2 \geq \frac{E^3_{\text{ONF}}}{E^2_{\text{OPT}}} = \frac{2 + 2 \cdot E_{tran}}{2 + E_{tran}}. \tag{3.8}$$

- Assume that ONF buffers the job J_0 into the queue.

  ONF must switch the machine to ACTIVE to process J_0 at some time $t \leq \hat{F} + 1$. After time $t+1$, ONF will either choose to power off the machine to the SLEEP state or keep it spinning. We consider the end of step $t + 1$ now.

  – Assume that ONF keeps ACTIVE.

    The adversary adopts a strategy similar to the strategy used above. It releases another job J_3 at time $t + E_{tran} + 2$. There are no more jobs released.

    In this case, OPT powers off the machine to the SLEEP state after finishing job J_0 at time 1. Then it switches to the ACTIVE state at time $t + E_{tran} + 2$ to process job J_3. The total cost paid by OPT is $E^3_{\text{OPT}} = 2 + 2 \cdot E_{tran}$.

    On the other hand, ONF will either keep in the ACTIVE state till the job J_3 finished with a total cost of $E^4_{\text{ONF}} = 3 + 2 \cdot E_{tran}$, or switch to the SLEEP state at some time before $t + E_{tran} + 2$ and power on the machine at some time before $t+E_{tran}+2+\hat{F}$ to process the job J_3. The total cost of ONF is $E^5_{\text{ONF}} \geq 3+2\cdot E_{tran}$.

The lower bound of competitive ratio β_3 is

$$\beta_3 \geq \frac{\min(E^4_{\text{ONF}}, E^5_{\text{ONF}})}{E^3_{\text{OPT}}} = \frac{3 + 2 \cdot E_{tran}}{2 + 2 \cdot E_{tran}}. \tag{3.9}$$

– Assume that ONF powers off the machine at time $t + 1 \leq \hat{F} + 1$ after finishing the job J_0. (Note $t \leq \hat{F}$.)

Only in this case, we need the assumption in Theorem 3.12 that $\hat{F} < E_{tran}$.

The adversary releases a job J_4 at time $t + 1$. There are no more jobs released.

OPT keeps the machine ACTIVE till finishing job J_4 at time $t + 2$. The total cost of OPT is $E^4_{\text{OPT}} = 2 + E_{tran} + t$.

On the other hand, ONF powers the machine off and it needs to turn it on again before time $t + \hat{F} + 1$ to process job J_4. The total cost of ONF is $E^6_{\text{ONF}} \geq 2 + 2 \cdot E_{tran}$.

The lower bound of competitive ratio β_4 is

$$\beta_4 \geq \frac{E^6_{\text{ONF}}}{E^4_{\text{OPT}}} = \frac{2 + 2 \cdot E_{tran}}{2 + E_{tran} + t} \geq \frac{2 + 2 \cdot E_{tran}}{2 + E_{tran} + \hat{F}}. \tag{3.10}$$

Combine Inequalities (3.7), (3.8), (3.9), and (3.10), we have the lower bound β of competitive ratio for the bicriteria model: $\beta \geq \min(\beta_1, \beta_2, \beta_3, \beta_4) = \min\left(\frac{3+2\cdot E_{tran}}{2+2\cdot E_{tran}}, \frac{2+2\cdot E_{tran}}{2+E_{tran}+\hat{F}}\right)$.

□

An online algorithm ONS in minimizing energy consumption subject to an upper bound of largest stretch α

For online algorithms minimizing the largest stretch in the one-machine environment, we have a pessimistic result (the lower bound of competitive ratio), even though we do not consider energy constraints.

Theorem 3.13. *[53] On a single machine, no preemptive online algorithm is $\sqrt[3]{\Delta}$-competitive for minimizing largest stretch, where Δ is the ratio of the maximum processing time to the minimum processing time.*

The adversary constructed in the proof of Theorem 3.13 is non-idling. We cannot expect to have an online algorithm whose largest stretch is at most a constant times of that of a non-idling offline algorithm. However, it is reasonable (and there exists hope) to develop online algorithms whose total energy consumption is bounded by a constant times of that of an optimal offline algorithm.

The algorithm ONS. Our online algorithm ONS (Algorithm 14) is motivated by the offline algorithm in minimizing total energy consumption subject to an upper bound of stretch we have given in Section 3.2.2. For each job J_j, at time t, we assign it a deadline d_j. Since we do not know the exact upper bound of stretch for the optimal offline algorithm, we apply the following method to calculate each job's deadline: Consider the input instance given so far at time t. We calculate the optimal stretch L_t^*, assuming no future arrivals. Then we apply $L_t^* \cdot \alpha$ as the estimation of the upper bound of stretch for the optimal schedule over the complete input instance and calculate each job's deadline (at time t) as $d_j^t = r_j + p_j \cdot L_t^* \cdot \alpha$.

After identifying the tentative deadlines d_j^t for all the jobs, we run the pending jobs in increasing deadline order. Our idea is the same as the one presented in [53]. In Algorithm 14, we concrete our idea and detail the solution with a procedure of calculating L_t^*.

The analysis of ONS. Directly from the analysis in [53], we have the following result.

Theorem 3.14. *[53] Algorithm 14 is Δ-competitive in terms of largest stretch when $\alpha = 1$ and $O(\sqrt{\Delta})$-competitive in terms of largest stretch when $\alpha = O(\sqrt{\Delta})$.*

We analyze Algorithm ONS's competitiveness in terms of the total energy consumed over the whole input instance.

Algorithm 14 ONS(α)

1: For each newly arriving job at time t, calculate L_t^*, the optimal stretch for all the released jobs so far.
2: In calculating L_t^*, apply the following way over the input instance so far: For the pending jobs J_j with remaining processing time p'_j at time t', run the job J_i where

$$i = \arg\max \left(\frac{t' + p'_j - r_j}{p_j} \right).$$

After executing all the released jobs at time t, return the largest stretch as L_t^*.
3: For the pending job J_j at time t, define

$$d_j^t = r_j + p_j \cdot L_t^* \cdot \alpha.$$

4: Execute jobs in increasing order of deadlines d_j^t.
5: Power off the machine after it processes the last pending job and keeps ACTIVE for E_{tran} time units.

Theorem 3.15. *Algorithm 14 is* 2*-competitive in terms of total energy consumption.*

Proof. Let NOPT denote an optimal non-idling offline algorithm with the optimal largest stretch and optimal energy consumption. Note that ONS and NOPT both are non-idling. Hence, at any time when ONS executes a job, NOPT must execute some job (may be a different one) as well. ONS may keep ACTIVE at the time when NOPT is SLEEP.

Let E_i denote the i-th *time period* in which ONS keeps ACTIVE while NOPT keeps SLEEP. From Algorithm 14, we have $E_i \leq E_{tran}$, $\forall i$. We define a *charging interval* starting from the time when both ONS and NOPT power on the machine and ending at the immediate following time when ONS and NOPT power on the machine again. We count energy consumption for NOPT and ONS in each interval. For each interval, either ONS and NOPT cost the same amount of energy, or ONS keeps ACTIVE for at most E_{tran} units of time before it powers off but NOPT powers off immediately. Since $E_i \leq E_{tran}$, we conclude that ONS consumes at most 2 times of energy than NOPT does. □

3.2.4 Conclusions

Along with the benefit of saving energy, an effective energy-saving power-down strategy, in general, has an adverse impact on user-perspective performance metrics such as *flow time*

and *stretch*, which are two of the most important factors to optimize in the literature of queueing theory and scheduling theory. Intuitively, the transition energy cost associated with powering on the system from a lower-power state to the ACTIVE state can be considerably decreased if the earlier released jobs are postponed in execution and grouped with the later released jobs to be processed together. However, saving energy in such a way results in increasing these earlier released jobs' flow time and stretch. Algorithms optimizing the two objectives, energy consumption and flow time (or energy consumption and stretch), conflict with each other. In this chapter, we design bi-criteria power-down strategies that optimize both.

Table 3.3: Summary of our results on trad-off between energy consumption and flow time or stretch (n is the number of jobs released).

setting	restrictions	minimizing	performance
offline	bounded maximum flow time	energy consumption	$O(n^2)$
offline	bounded energy consumption	maximum flow time	$O(n^2 \log F^*_{\max})$; $F^*_{\max}$ is the optimal maximum flow time
offline	bounded largest stretch	energy consumption	$O(n^5)$
offline	bounded energy consumption	largest stretch	$O(n^5 \log S_{\max})$; $S_{\max}$ is the optimal largest stretch
online	bounded flow time	energy consumption	no worse than 2-competitive against non-idling adversaries
online	bounded stretch	energy consumption	2-competitive against adversaries

Chapter 4: Energy-Aware Scheduling Algorithms in the DVS Setting

4.1 Motivation

The rapid progress of processor design technologies provides faster processors. Modern processors, such as those supported by Intel's "SpeedStep" and AMD's "PowerNOW" technologies, have been equipped with a feature to vary the clock frequencies dynamically. The operating system is able to adjust the processor's *clock frequency* (*speed*) on the fly to execute jobs and reduce energy consumption at lower speeds [58]. We call this functionality *speed-scaling technology* or *dynamic voltage scaling* (DVS). Speed scaling is expected to satisfy some quality-of-service measures as well as to reduce overall energy cost, by adaptively manipulating modern processors' multiple speeds.

In this chapter, we study energy-aware scheduling problems equipped with DVS functionality. Existing studies considered energy minimization through speed scaling without much attention to the impact of speed changes. Such changes typically involve time and energy overhead. Moreover, recent studies indicate that the *lifetime reliability* of a CMOS circuit is directly related to the number and span of speed changes. For example, in [59], it is reported that *(hardware) failures, such as cracks and fatigue failures, are created not by sustained high temperatures, but rather by the repeated heating and cooling of sections of the processor. This phenomenon is referred to as 'thermal cycling'.* Thermal cycling is caused by the large difference in thermal expansion coefficients of metallic and dielectric materials, and leads to cracks and other permanent failures. Using Mean-Time-To-Failure (MTTF) to describe the expected processor's life, the following Coffin-Manson formula [60] is used

to characterize a processor's lifetime reliability:

$$\text{MTTF} \propto \frac{1}{C_o(\Delta T_{mp} - \Delta T_o)^q x}, \tag{4.1}$$

where C_o is a material dependent constant, ΔT_{mp} is the entire temperature cycle-range of the device, ΔT_o is the portion of the temperature range in the elastic region, q is the Coffin-Manson exponent, and x is the frequency (number of occurrences per unit time) of thermal cycles [60]. Typically, $\Delta T_o \ll \Delta T_{mp}$ and $6 \leq q \leq 9$ for silicon materials. By simplifying Equation (4.1), we have

$$\text{MTTF} \propto \frac{1}{C_o \cdot \Delta T_{mp}^q \cdot x}. \tag{4.2}$$

Equation (4.2) clearly indicates that an algorithm which frequently changes the processor's speeds results in large x and ΔT_{mp}. Thus, such a schedule may introduce a large temperature cycle-range and therefore significantly impair the processor's reliability. Simulations in [59] have confirmed that various speed-scaling energy-aware policies have different impacts on processor's reliability in terms of MTTF. The number of speed changes (x in Equation (4.2)) is a critical factor in determining a processor's reliability (MTTF in Equation (4.2)) under the thermal cycling effect.

In this chapter, we investigate energy-aware real-time scheduling algorithms with speed change constraints.

4.2 Model Description

We consider a single-processor setting. The processor has variable *clock frequencies* (*speeds*). Under a speed f, the processor consumes energy $e(f)$ per unit time and we simply assume that the function $e(\cdot)$ is convex and $e(0) = 0$. This setting is a generalization of the power model used in Yao's paper [23] and its successors [61, 62].

We note that in scaling speeds, the processor's frequency and supply voltage are both

adjusted (dynamic voltage/frequency scaling). Hence, in the rest of the chapter, we will assume that frequency/speed change always involves the corresponding voltage change [23].

We consider scheduling a set of n given jobs $\mathbf{J} = \{J_1, J_2, \ldots, J_n\}$. Each job J_j has a *release time* $r_j \in \mathbb{R}^+$, a *processing time* (also called *worst-case execution time*) $p_j \in \mathbb{R}^+$ and a *deadline* $d_j \in \mathbb{R}^+$. Under the speed f, it takes time p_j/f to complete the job J_j. We consider preemptive scheduling and we assume that the cost of preemption is negligible.

The objective in this study is to design scheduling algorithms to finish all the jobs before their deadlines in minimizing the energy consumption, as well as the *number* and *cost* of speed changes.

Definition 4.1 (Speed Schedule)**.** *A* speed schedule *can be viewed as a piece-wise constant curve, specifying that at which speed the processor runs during which time interval. Assume that the CPU speed changes m times during the execution. Then the speed schedule Ψ is defined by m intervals and the speed used in each of these intervals. Let the m time intervals be:*

$$I_1 := (t_0, t_1], I_2 := (t_1, t_2], \ldots, I_m := (t_{m-1}, t_m].$$

Each triplet (t_{i-1}, t_i, s_i) corresponds to the i-th time interval $I_i = (t_{i-1}, t_i]$ ($0 < i \leq m$ and $t_0 = 0$) in which the processor keeps running at a speed $s_i \geq 0$. The speed scheduler Ψ is

$$\Psi : \{(t_0, t_1, s_1), (t_1, t_2, s_2), \ldots, (t_{m-1}, t_m, s_m)\}.$$

Figure 4.1 illustrates an example schedule. The schedule employs 4 distinct speeds f_1, f_2, f_3, f_4 in 6 time intervals, where $s_1 = s_4 = f_1$, $s_2 = s_6 = f_4$, $s_3 = f_2$, and $s_5 = f_3$.

We call time t_i a *speed switching point*. Without loss of generality, we assume that the processor is in *idle* state initially at time 0 ($s_0 = 0$) and gets back to the idle state after processing all the jobs ($s_{m+1} = 0$). Thus, a schedule with m time intervals have $m+1$ speed switching points $t_0, t_1, \ldots, t_m$.

The total energy consumption of such a schedule is calculated as: $E^\Psi = \sum_{i=1}^{m} e(s_i)(t_i -$

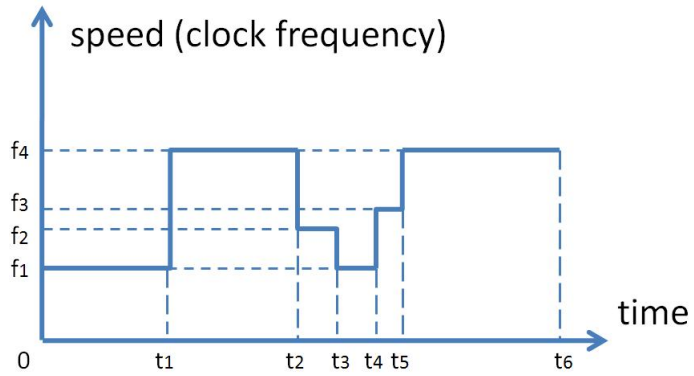

Figure 4.1: A piecewise curve describing the time intervals and the processor's speeds in each interval.

t_{i-1}). For the example in Figure 4.1, we have

$$\begin{aligned} E^{\Psi} &= e(f_1)(t_1 - 0) + e(f_4)(t_2 - t_1) + e(f_2)(t_3 - t_2) + e(f_1)(t_4 - t_3) \\ &\quad + e(f_3)(t_5 - t_4) + e(f_4)(t_6 - t_5) \\ &= e(f_1)(t_1 + t_4 - t_3) + e(f_4)(t_2 - t_1 + t_6 - t_5) + e(f_2)(t_3 - t_2) + e(f_3)(t_5 - t_4). \end{aligned}$$

To incorporate the penalty of changing CPU frequencies, we consider that each speed change from the frequency s_i (in interval I_i) to the frequency s_{i+1} (in interval I_{i+1}) involves a *cost* $c_{i,i+1} \in \mathbb{R}^+$. ($c_{i,i+1}$ reflects the speed change's negative impact on the processor's lifetime reliability.) In [63], analysis and simulation results show the exponential (s^{α} with $\alpha > 2$) and super-linear ($s^{1+\epsilon}$) dependencies of the power-on voltage (speed) s and temperature.

Along with the fact that the processor's reliability is a convex function of the temperature change (see Equation (4.2)), we assume that the reliability cost of a speed change is a convex function of the difference between the speed values of neighboring time intervals. For instance, switching from f_4 to f_1 may be more costly than switching from f_4 to f_3, providing $f_1 < f_3 < f_4$. Consequently, the function $c(\cdot)$ is convex and $c_{i,i+1}$ is the value of $c(|s_i - s_{i+1}|)$.

$$c_{i,i+1} := c(|s_i - s_{i+1}|), \text{where } s_0 = s_{m+1} = 0. \tag{4.3}$$

Let $\mathbf{J}$ denote a set of jobs and let Ψ denote the speed schedule that we are going to design. We formulate four optimization problems as follows.

$\mathcal{P}_1$. [Minimizing sum of energy consumption and costs of speed changes]

Let E^Ψ denote the total energy consumed by the schedule Ψ to complete a set of jobs $\mathbf{J}$ before their deadlines. Assume Ψ has m time intervals. The total cost associated with all the clock speed changes during this schedule is $\sum_{i=0}^{m} c_{i,i+1}$ where s_0 and s_{m+1} are defined as 0 (see Equation (4.3)). In this problem, we want to minimize $E^\Psi + \beta \sum_{i=0}^{m} c_{i,i+1}$, where β is a given constant. After normalizing $c_{i,i+1}$, we remove β and formulate our problem as $\min . \left(\sum_{i=1}^{m} e(s_i)(t_i - t_{i-1}) + \sum_{i=0}^{m} c'_{i,i+1} \right)$, where $s_0 = s_{m+1} = 0$ and $c'_{i,i+1} = c_{i,i+1}/\beta$. (Note that $c'(\cdot) = c(\cdot)/\beta$ is still a convex function.) We rename $c'_{i,i+1}$ as $c_{i,i+1}$.

$\mathcal{P}_2$. [Under a fixed number of speed changes]

Let E^Ψ and m denote the total energy consumed and the total number of speed changes in the schedule Ψ to complete the jobs in $\mathbf{J}$ before their deadlines, respectively. Let M be the upper bound on the *number* of speed changes. The objective is to minimize E^Ψ subject to $m \leq M$. That is, $\min . \sum_{i=1}^{m} e(s_i)(t_i - t_{i-1}),$ subject to $m \leq M$.

The problem $\mathcal{P}_2$ considers the *number* of speed changes as a constraint.

$\mathcal{P}_3$. [Under a fixed energy budget]

Let E^* denote the optimal (minimum) energy consumption required to complete all the jobs $\mathbf{J}$ before their deadlines. (We do not have to know E^* beforehand.) Let E^b denote the energy budget that we are given. In this problem setting, we always have $E^b \geq E^*$. Let the schedule Ψ have m time intervals. The objective

is to minimize the total number of speed changes m during the schedule subject to the constraint that total consumed energy E^{Ψ} does not exceed E^b. That is, $\min .m,$ subject to $\sum_{i=1}^{m} e(s_i)(t_i - t_{i-1}) \leq E^b$.

$\mathcal{P}_4$. [Under a bound of span of frequencies used]

Let E^{Ψ} and m denote the total energy consumed and the total number of time intervals of an schedule Ψ to complete the jobs $\mathbf{J}$ before their deadlines, respectively. Let $s^{\max} = \max_{1 \leq i \leq m} s_i$ and $s^{\min} = \min_{1 \leq i \leq m} s_i$, with $s^{\max} \geq s^{\min} \geq 0$. In other words, s^{max} and s^{min} are the maximum and minimum speed used in Ψ while executing the jobs. The difference $s^{\max} - s^{\min}$ is defined as the *span* of the frequencies used in Ψ.

Let Q be the given upper bound on the span of the speeds which we do not want to exceed. The objective is to minimize E^{Ψ} subject to $s^{\max} - s^{\min} \leq Q$. That is, $\min . \sum_{i=1}^{m} e(s_i)(t_i - t_{i-1}),$ subject to $s^{\max} - s^{\min} \leq Q$.

For the problem $\mathcal{P}_4$, we bear the understanding that running the processor with a smaller speed span (small $s^{\max} - s^{\min}$ difference) results in less fluctuation of temperature (reduced ΔT_{mp} in Equation (4.2)), and thus, better chip's lifetime reliability (improved MTTF in Equation (4.2)).

In the following, we present convex-programming-based algorithmic solutions for the problems $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$, and $\mathcal{P}_4$. We analyze their performance as well. Note that Yao et al. [23] and Li et al. [61] presented algorithms minimizing total energy consumption. The algorithms in [23] and [61] have no restrictions over the number of processor's speed changes. The models that we discussed above have their own algorithmic challenges. As we shall see, our solutions are totally different from [23, 61] that had the objective of minimizing energy alone. The problem $\mathcal{P}_2$ generalizes the well-studied Yao et al.'s model in [23], when we set the upper bound of speed changes M to a very large number.

4.3 Previous Work

The first theoretical energy-efficient job scheduling model is studied by Yao et al. [23]. In this model, jobs have release times and deadlines, and a continuous spectrum of speeds is available. This framework is by far the most extensively studied algorithmic speed scaling problem. A straightforward implementation has running time of $O(n^3)$. Li et al. [61] improved this result, giving a $O(n^2 \log n)$-time algorithm, and this is by far the best algorithm. Li et al. [61] and Kwon and Kim [64] also studied the discrete setting in which the processor has k discrete speeds. Kwon and Kim [64] achieved a $O(n^3)$-time algorithm. Li et al. [61] got a $O(k \cdot n \log n)$ algorithm in minimizing the total consumed energy. Many settings and metrics based on this framework are studied in literature, such as maximizing throughput [65], or minimizing the sum of energy consumption and (weighted) flow time of jobs [55,66]. Competitive online algorithms for these models are also studied. Yao et al. [23] provided two natural online algorithms. Bansal et al. [67] improved the competitive ratios. For online algorithms, their competitive ratios, in general, depend on the convex function $c(\cdot)$. In above min-energy models, the number or cost of frequency changes is unrestricted. Our work studies min-energy speed schedulers with considerations of number and cost of frequency changes. It is a natural next step in this line of research.

Many researchers have addressed how to minimize energy consumption for real-time systems in which periodic jobs are considered. In [68], an optimal static algorithm is given, assuming each job has its worst-case workload at each instance. In [69], the authors proposed an algorithm to update the running frequency based on the workload and the periodic deadlines. A model in which each frequency transition is associated with energy cost and delay is considered in [70]. The policy EDF with realistic assumptions is considered. Feedback control has been introduced for EDF scheduling of real-time jobs in [71]. Our work falls along the line of Yao et al.'s algorithm in that we do not assume any specific arrival pattern for real-time jobs, but aim to maximize our metrics for the most general case.

4.4 Algorithms and Analysis

In this section, we provide algorithmic solutions for the problems $\mathcal{P}_1 - \mathcal{P}_4$.

4.4.1 Minimizing the sum of energy consumption and costs of speed changes

Assume the schedule Ψ has m time intervals $I_i = (t_{i-1}, t_i]$ $(1 \leq i \leq m)$ and within each interval I_i, the schedule keeps running at constant speed $s_i \geq 0$. The system does not consume any energy after finishing the last job. The objective is

$$\min . \left(\sum_{i=1}^{m} e(s_i)(t_i - t_{i-1}) + \sum_{i=0}^{m} c_{i,i+1} \right), \tag{4.4}$$

where $c_{i,i+1}$ is scaled by a factor β from its definition in Equation (4.3).

Define OPT as an optimal algorithm minimizing the sum of energy consumption and the costs of speed changes (Equation (4.4)). Our job is to determine all the candidate values that t_i in Equation (4.4) can take. We note that the function $c(\cdot)$ is assumed to be a general convex function, hence determining the optimal schedule's speed switching points heavily depends on the function $c(\cdot)$ itself. Instead of designing algorithms for some specific functions $c(\cdot)$, we study a large class of algorithms called *event-driven DVS algorithms*. The purpose of introducing an event-driven DVS algorithm for the problem $\mathcal{P}_1$ is to show that there exists an optimal convex-programming-based solution, and this solution's framework can be proved to generate optimal solutions for the other three problems $\mathcal{P}_2$, $\mathcal{P}_3$, and $\mathcal{P}_4$.

Definition 4.2 (Event-Driven DVS Algorithm)**.** *For* event-driven DVS algorithms*, speed changes (speed switching points) only happen at jobs' release times and/or deadlines.*

We note here that event-driven DVS algorithms have the distinct advantage of keeping the run-time overhead due to DVS low, as opposed to DVS algorithms that require speed change at arbitrary points during execution. The CPU scheduler, that is invoked at task

release times and deadlines, can also regulate the frequency according to the pre-determined speed schedule during the same invocation. As a result, we believe that our results regarding to the optimality of event-driven DVS algorithms will be also very useful in practice.

Let $\mathbf{J} = \{J_1, J_2, \ldots, J_n\}$ denote the n jobs to be scheduled. A job J_j is represented by a triplet (r_j, d_j, p_j). Let $R = \{r_1, r_2, \ldots, r_n\}$ and $D = \{d_1, d_2, \ldots, d_n\}$. We use $Z = R \cup D$ to denote the union of all the release time and deadlines of jobs. Note $|Z| = |R \cup D| \leq |R| + |D| \leq 2n$. We sort all the values in Z in increasing order and index them as $z_1, z_2, \ldots, z_{n'}$ where $n' \leq 2n$. Without loss of generality, we assume $z_1 = 0$. Before the last deadline $z_{n'}$, the time range is divided into $n' - 1$ non-overlapping intervals $(z_i, z_{i+1}]$, $\forall 1 \leq i \leq n' - 1$. We name the interval $T_{i,i'} := (z_i, z_{i'}]$ $(i' \geq i + 1)$ as a *scheduling interval*. There are *at most* $\binom{n'}{2} = \frac{n'(n'-1)}{2} = O(n^2)$ such scheduling intervals. For each scheduling interval $T_{i,i'}$, we can compute its corresponding *workload* denoted by a variable $P_{i,i'}$ and *processing capacity* denoted by a variable $W_{i,i'}$ as follows.

$$P_{i,i'} = \frac{\sum_j p_j}{z_{i'} - z_i}, \text{ where } z_i < r_j < d_j \leq z_{i'}, \tag{4.5}$$

$$W_{i,i'} = \sum_{l=i+1}^{i'} s'_l, i < l \leq i', \tag{4.6}$$

where s'_l, $l \in \{i + 1, \ldots, i'\}$, is the *speed variable* to denote at which speed the processor runs in the interval $(z_{l-1}, z_l]$ $(s'_1 = s'_{|Z|+1} = 0)$. In order to complete all the jobs before their deadlines, the processing capacity should be at least as large as the workload requirement for each time interval.

The remaining task is to determine s'_l such that all the jobs in $\mathbf{J}$ can be finished before their deadlines and the objective $\sum_{l=2}^{|Z|} e(s'_l)(z_l - z_{l-1}) + \sum_{l=2}^{|Z|+1} c(|s'_l - s'_{l-1}|)$ is minimized.

We formulate this problem using a convex program CP_1 as below:

$$\begin{aligned} \min. \quad & \sum_{l=2}^{|Z|} e(s'_l)(z_l - z_{l-1}) + \sum_{l=2}^{|Z|+1} c(|s'_l - s'_{l-1}|) \\ \text{subject to} \quad & W_{i,i'} \geq P_{i,i'}, \forall 1 \leq i < i' \leq |Z| \\ & s'_l \geq 0. \end{aligned}$$

We have Algorithm 15 to compute an event-driven DVS schedule for the problem $\mathcal{P}_1$.

Algorithm 15 Convex-Programming-Based Solution

Require: A job set $\mathbf{J} := \{J_1, J_2, \ldots, J_n\}$ where J_j denoted by $\{r_j, d_j, p_j\}$

Ensure: A piecewise curve describing the time intervals and the speeds at which the processor runs in these intervals

1: Let $R = \{r_1, r_2, \ldots, r_n\}$, $D = \{d_1, d_2, \ldots, d_n\}$, and $Z = R \cup D$.

2: Sort Z in increasing order. Let the distinct values be z_i, $1 \leq i \leq |Z|$. (Without loss of generality, assume Z has exactly $|Z|$ distinct elements and $z_1 = 0$.)

3: Define $T_{i,i'} = (z_i, z_{i'}]$, for any $1 \leq i < i' \leq |Z|$. Define s'_l for each $(z_{l-1}, z_l]$. Set $s'_1 = 0$ and $s'_{|Z|+1} = 0$.

4: **for** each pair (i, i') with $1 \leq i < i' \leq |Z|$ **do**

5: calculate

$$P_{i,i'} = \frac{\sum_j p_j}{z_{i'} - z_i}, z_i < r_j < d_j \leq z_{i'};$$

6: define

$$W_{i,i'} = \sum_{l=i+1}^{i'} s'_l, i < l \leq i'.$$

7: **end for**

8: Solve the convex program CP_1:

$$\begin{aligned} \min. \quad & \sum_{l=2}^{|Z|} e(s'_l)(z_l - z_{l-1}) + \sum_{l=2}^{|Z|+1} c(|s'_l - s'_{l-1}|) \\ \text{subject to} \quad & W_{i,i'} \geq P_{i,i'}, \forall 1 \leq i < i' \leq |Z| \\ & s'_l \geq 0. \end{aligned}$$

9: **return** a schedule running jobs in a canonical order using speeds s'_l.

We provide the analysis of correctness and running time for Algorithm 15. For a given set of real-time jobs with known processing times, preemptive EDF is optimal in the sense

that any feasible job set can be also scheduled in a feasible manner by EDF policy [72]. We have the following result.

Lemma 7 (EDF Optimality)**.** *An optimal preemptive algorithm can have its jobs executed in a* canonical (deadline) order*, that is, for any two or more jobs ready to be executed for a given time, the job with the earliest deadline has the highest priority, with ties broken arbitrarily.*

Lemma 8 (Convexity of CP_1)**.** *The formulation CP_1 in Algorithm 15 is a convex program.*

Proof. We study the objective first. Note that all the z-values are chosen from $Z = R \cup D$. Thus, $e(s'_l)(z_l - z_{l-1})$ is a non-negative linear combination of the convex function $e(s'_l)$, which is convex as well. The norm $|s'_l - s'_{l-1}|$ of an affine function $s'_l - s'_{l-1}$ is a convex function. Note that all the variables $|s'_l - s'_{l-1}|$ are non-negative, so, by the definition of function $c(\cdot)$, $c(|s'_l - s'_{l-1}|)$ is a non-decreasing function. As the cost function $c(\cdot)$ is assumed convex, the composition $c(|s'_l - s'_{l-1}|)$ preserves convexity.

We then study the constraints. All the constraints are convex ones (actually, linear ones). (Note that all the calculated values $P_{i,i'}$ are constants in our formulation CP_1. We can rewrite $W_{i,i'} \geq P_{i,i'}$ as $W_{i,i'} - P_{i,i'} - \delta_{i,i'} = 0$, where $\delta_{i,i'} \geq 0$.) □

Let $G(\cdot)$ denote the running time of evaluating $e(\cdot)$ and $c(\cdot)$ and their first and second derivatives for all the constraints in the convex program CP_1.

Theorem 4.1 (Optimal Event-Driven Schedule for Problem $\mathcal{P}_1$)**.** *Algorithm 15 generates an optimal event-driven speed schedule and has a running time of $O(\max\{n^4, n \cdot G\})$, where n is the number of jobs to be scheduled and G is the time of evaluating $e(\cdot)$ and $c(\cdot)$ and their first and second derivatives for all the constraints.*

Proof. The correctness of Theorem 4.1 depends on Lemma 7 and Lemma 8. First, Lemma 7 guarantees that running jobs in the canonical manner does not hurt the optimality. Second,

we claim that $W_{i,i'} \geq P_{i,i'}$ is the necessary condition to ensure a feasible schedule. This has been proved in [73]. Combining these two observations and Lemma 8, we conclude that Algorithm 15 generates an optimal event-driven schedule.

Now, we analyze the running time complexity. Note $|Z| \leq 2n$ and thus, $|Z| = O(n)$. Sorting Z takes time $O(n \log n)$. The number of scheduling intervals $T_{i,i'}$ is $O(n^2)$. There are $O(n^2)$ variables for $P_{i,i'}$ and $W_{i,i'}$. Calculating $P_{i,i'}$ takes time $O(n^2)$. (A straightforward way of calculating $P_{i,i'}$ takes time $O(n^3)$. Here is one alternative way with faster running time: We map each pair (r_j, d_j) of a job J_j to each corresponding time in Z and result in a convex bipartite graph. Working on this convex bipartite graph improves the calculation time of getting $P_{i,i'}$ to be $O(n^2)$.) Before we get to solve the convex program CP_1, we pay time $O(n^2)$ for the preliminary work. Note that CP_1 has $O(n^2)$ constraints. We use the interior-point method [74] to solve CP_1 optimally (arbitrarily close to optimal). Thus, it takes time $O(\max\{n^3, n^2 \cdot O(n^2), n \cdot G(e(\cdot), c(\cdot))\}) = O(\max\{n^4, n \cdot G(e(\cdot), c(\cdot))\})$, where $G(\cdot)$ is the time of evaluating $e(\cdot)$ and $c(\cdot)$ and their first and second derivatives for all the constraints [75]. □

4.4.2 Minimizing energy consumption under limited number of speed changes

In this section, we consider speed schedules under a limited number of speed changes. Let M be the upper bound of the number of speed changes that a speed schedule Ψ is allowed to schedule jobs. We study the problem $\mathcal{P}_2$: minimizing the total energy consumption subject to satisfying all the jobs' deadline constraints and bounding the number of speed changes by M.

Again, let $\mathbf{J} = \{J_1, J_2, \ldots, J_n\}$ denote the n jobs to be scheduled. A job J_j is represented by a triplet (r_j, d_j, p_j). Let $R = \{r_1, r_2, \ldots, r_n\}$ and $D = \{d_1, d_2, \ldots, d_n\}$. We use $Z = R \cup D$ to denote the union of all release time and deadlines of jobs. Note $|Z| \leq 2n$. This problem $\mathcal{P}_2$ has the following property which guarantees that the number of constraints of our convex

program is polynomial in the number of jobs in **J**. Also, Lemma 9 ensures that an optimal schedule for $\mathcal{P}_2$ can be an event-driven DVS one.

Lemma 9 (Upper Bound of # of Scheduling Intervals)**.** *An optimal algorithm OPT has its speed changes only happen at time points in the set* $Z = R \cup D$.

Proof. We prove Lemma 9 using a contradiction method. Without loss of generality, assume that OPT has M line-segments to indicate the time intervals in which the processor executes jobs at different speeds. Let t be the first (earliest) time such that t is a speed switching point and t is neither a release time nor a deadline of a job. Thus, at time $t - \epsilon$, either the processor is *idle* (that is, no job is being executed) or some job, say J_j. has not been finished and it is being executed at this point.

(1) Assume the processor is idle at time $t - \epsilon$. At time t, a new job must be released, since otherwise, in order to save more energy without violating the bound of number of speed changes, the processor can keep the same speed till the next speed change (if the next speed at time t is larger than the one at time $t - \epsilon$) or the processor can immediately slow down its speed at the time when the processor finished the last job before time $t - \epsilon$ (if the next speed at time t is smaller than this one at time $t - \epsilon$). This contradiction shows that t must be some release time.

(2) Assume the processor is executing some job J_j at time $t - \epsilon$ and $t \neq d_j$. From Lemma 7, we know that in this OPT, all the jobs are executed in a canonical order. Particularly, if no job is released at time t, then the processor should keep the same speed or fasten (respectively, slow down) its speed before t in order to reduce the total energy consumption. Thus, we have that either at time t some job is released or all the jobs have been finished before time t.

Based on the above discussion, we claim that Lemma 9 holds, due to the convexity of the energy consumption function $e(\cdot)$ with $e(0) = 0$. □

In the following, we design algorithms for the problem $\mathcal{P}_2$. Similar to the case of $\mathcal{P}_1$, we sort all the values in Z in increasing order and index them as $z_1, z_2, \ldots, z_{n'}$ where

$n' \leq 2n$. Thus, the whole time is divided into $n'-1$ non-overlapping intervals $(z_i, z_{i+1}]$ in which the processor runs at possible positive speeds, $\forall 1 \leq i \leq n'-1$. The interval $T_{i,i'} := (z_i, z_{i'}]$ $(i' \geq i+1)$ is a scheduling interval. There are *at most* $\binom{n'}{2} = \frac{n'(n'-1)}{2} = O(n^2)$ such scheduling intervals. For each scheduling interval $T_{i,i'}$, we calculate its corresponding workload $P_{i,i'}$ and processing capacity $W_{i,i'}$ as those in Equation (4.5) and Equation (4.6), given the speed s'_l assumed for each interval $(z_{l-1}, z_l]$.

The remaining task is to determine s'_l such that all the jobs in $\mathbf{J}$ can be finished before their deadlines and the objective is to bound the number of speed changes by M. Note that the following piece-wise function is a convex one.

$$c_{i,i+1} = c(|s_i - s_{i+1}|) = \begin{cases} \epsilon, & \text{if } |s'_i - s'_{i+1}| < \delta \\ H, & \text{otherwise} \end{cases},$$

where H is a large positive number, and ϵ is a small positive constant. $c(\cdot)$ is convex since it can be represented by $c(x) = \max\{y_1(x), y_2(x)\}$ for $x \in \mathbb{R}^+$, where $y_1(x) = \epsilon$, $\forall x \geq 0$, $y_2(x) = H$, $\forall x \geq \delta$ and $y_2(x) = 0$, $\forall 0 \leq x < \delta$.

By using the above function $c(\cdot)$, we set the objective to minimizing the energy consumption $\sum_{l=1}^{|Z|} e(s'_l)(z_l - z_{l-1})$ subject to the total cost of speed changes bounded by the sum of $M \cdot H$ and the costs associated with those intervals running at similar frequencies (where the frequency difference is bounded by a specified input value δ). To ensure that the final schedule has no more than M speed changes, we need to set $H \gg \epsilon \cdot |Z|$.

We present the algorithm (Algorithm 16) for this problem using the convex programming technique.

Theorem 4.2 (Optimal Schedule for $\mathcal{P}_2$)**.** *Algorithm 16 generates a schedule arbitrarily close to the optimal and has a running time of* $O(\max\{n^4, n \cdot G\})$*, where* G *is the time of evaluating* $e(\cdot)$ *and its first and second derivatives for all the constraints.*

Algorithm 16 Bounded # of Speed Changes (M)

Require: A job set $\mathbf{J} := \{J_1, J_2, \ldots, J_n\}$ and $J_j = \{r_j, d_j, p_j\}$. The upper bound of number of speed changes M

Ensure: A piecewise curve describing the time intervals and the speeds at which the processor runs in these intervals

1: Let $R = \{r_1, r_2, \ldots, r_n\}$, $D = \{d_1, d_2, \ldots, d_n\}$, and $Z = R \cup D$.

2: Sort Z in increasing order. Let the distinct values be z_i, $1 \leq i \leq |Z|$. (Without loss of generality, assume Z has exactly $|Z|$ distinct elements and $z_1 = 0$.)

3: Define $T_{i,i'} = (z_i, z_{i'}]$, for any $1 \leq i < i' \leq |Z|$. Define s'_l for each $(z_{l-1}, z_l]$. Set $s'_1 = 0$ and $s'_{|Z|+1} = 0$.

4: **for** each pair (i, i') with $1 \leq i < i' \leq |Z|$ **do**

5: calculate

$$P_{i,i'} = \frac{\sum_j p_j}{z_{i'} - z_i}, z_i < r_j < d_j \leq z_{i'};$$

6: define

$$W_{i,i'} = \sum_{l=i+1}^{i'} s'_l, i < l \leq i'.$$

7: **end for**

8: Solve the convex program CP_2:

$$\begin{aligned} \min . \quad & \sum_{l=2}^{|Z|} e(s'_l)(z_l - z_{l-1}) \\ \text{subject to} \quad & W_{i,i'} \geq P_{i,i'}, \forall 1 \leq i < i' \leq |Z| \\ & \sum_{l=2}^{|Z|+1} c(|s'_l - s'_{l-1}|) \leq M \cdot H + \epsilon \cdot |Z| \\ & s'_l \geq 0. \end{aligned}$$

9: **return** a schedule running jobs in a canonical order using speeds s'_l.

Proof. It is obvious that $C\mathcal{P}_2$ in Algorithm 16 is a convex program and it can be solved to provide values for the variables s'_l (up to arbitrarily small errors). In the following, we illustrate that Algorithm 16 solves our problem $\mathcal{P}_2$ correctly and we analyze its running time complexity.

Lemma 9 guarantees that for $\mathcal{P}_2$, we only need to consider a speed schedule's speed switching points from the time points in Z. Similar to the analysis of $\mathcal{P}_1$, we conclude that Algorithm 16 generates an optimal solution. The running time complexity of Algorithm 16 is similar to that of Algorithm 15. □

4.4.3 Minimizing number of speed changes subject to bounded energy consumption

In this section, we study the problem of minimizing the number of speed changes subject to satisfying all jobs' deadline constraints and energy consumption bounded by a given budget. This problem is motivated by budgeting energy consumption during execution in energy-constrained environments.

Let E^* denote the optimal (minimum) energy consumption required to satisfy all the jobs' time constraints in $\mathbf{J}$. Let E^b denote the energy budget that we are given. In our problem setting for $\mathcal{P}_3$, we always have $E^b \geq E^*$. The objective is to minimize the total number of speed changes m subject to keeping the total consumed energy E^{Ψ} bounded by E^b. That is, $\min .m,$ subject to $\sum_{i=1}^{m} e(s_i)(z_i - z_{i-1}) \leq E^b$.

Actually, $\mathcal{P}_3$ is the dual problem of $\mathcal{P}_2$. We can use $\mathcal{P}_2$'s objective as our $\mathcal{P}_3$'s constraint and $\mathcal{P}_2$'s constraint $\sum_{l=1}^{|Z|} c(s'_l, s'_{l-1}) \leq M \cdot H + \epsilon|Z|$ as $\mathcal{P}_3$'s objective. We still use the function

$$c_{i,i+1} = c(|s_i - s_{i+1}|) = \begin{cases} \epsilon, & \text{if } |s'_i - s'_{i+1}| < \delta \\ H, & \text{otherwise} \end{cases},$$

where H is a large positive number, and ϵ is a small positive constant. We immediately

have the following algorithm.

Algorithm 17 Bounded Energy Budget (E^b)

Require: A job set $\mathbf{J} := \{J_1, J_2, \ldots, J_n\}$ and $J_j = \{r_j, d_j, p_j\}$. The energy budget E^b

Ensure: A piecewise curve describing the time intervals and the speeds at which the processor runs in these intervals

1: Let $R = \{r_1, r_2, \ldots, r_n\}$, $D = \{d_1, d_2, \ldots, d_n\}$, and $Z = R \cup D$.

2: Sort Z in increasing order. Let the distinct values be z_i, $1 \leq i \leq |Z|$. (Without loss of generality, assume Z has exactly $|Z|$ distinct elements and $z_1 = 0$.)

3: Define $T_{i,i'} = (z_i, z_{i'}]$, for any $1 \leq i < i' \leq |Z|$. Define s'_l for each $(z_{l-1}, z_l]$. Set $s'_1 = 0$ and $s'_{|Z|+1} = 0$.

4: **for** each pair (i, i') **do**

5: calculate

$$P_{i,i'} = \frac{\sum_j p_j}{z_{i'} - z_i}, z_i < r_j < d_j \leq z_{i'};$$

6: define

$$W_{i,i'} = \sum_{l=i+1}^{i'} s'_l, i < l \leq i'.$$

7: **end for**

8: Solve the convex program CP_3:

$$\begin{aligned} \min. \quad & \sum_{l=2}^{|Z|+1} c(s'_l, s'_{l-1}) \\ \text{subject to} \quad & W_{i,i'} \geq P_{i,i'}, \forall 1 \leq i < i' \leq |Z| \\ & \sum_{l=2}^{|Z|} e(s'_l)(z_l - z_{l-1}) \leq E^b \\ & s'_l \geq 0. \end{aligned}$$

9: **return** a schedule running jobs in a canonical order using speeds s'_l.

Theorem 4.3 (Optimal Schedule for $\mathcal{P}_3$)**.** *Algorithm 17 generates a schedule with objective value close to optimal value arbitrarily and has a running time of $O(\max\{n^4, n \cdot G\})$, where G is the time of evaluating $e(\cdot)$ and its first and second derivatives for all the constraints.*

Proof. Note that Lemma 9 holds for the problem $\mathcal{P}_3$ as well. The proof of Theorem 4.3 is almost the same as the one for $\mathcal{P}_2$ (see Theorem 4.2). □

4.4.4 Minimizing energy consumption subject to bounded span of frequencies

In this section, we study the problem of minimizing the energy consumption subject to the span of frequencies used in the schedule bounded by a given value. Let E^{Ψ} and m denote the total energy consumed and the total number of speed changes by the schedule Ψ to complete the jobs in $\mathbf{J}$ before their deadlines, respectively. Let Q be a given upper bound of the span of the speeds used by Ψ. Let $s^{\max} = \max_{1 \le i \le m}\{s_i\}$ and $s^{\min} = \min_{1 \le i \le m}\{s_i\}$. The objective is to minimize E^{Ψ} subject to $s^{\max} - s^{\min} \le Q$. We note that $\mathcal{P}_4$ generalizes the problem $\mathcal{P}_2$, given Q is allowed to be sufficiently large.

Let us consider the following idea for $\mathcal{P}_4$: Assume that we have the min-energy schedule for a given job set $\mathbf{J}$. (This min-energy schedule can be achieved by the algorithms in [23,61] in time $O(n^2 \log n)$.) We then realize that the processor has to run at a speed $s^{\max}$ during an interval $[z, z']$; otherwise, some job belonging to this interval cannot be finished before its deadline. This speed $s^{\max}$ ensures the highest speed to guarantee the feasibility of finishing $\mathbf{J}$. According to $\mathcal{P}_4$, all the speeds of a schedule that we are going to design should be in the range of $[s^{\max} - Q, s^{\max}]$. This range indicates that the processor should keep running at speed $\ge s^{\max} - Q$ all along the schedule. We thus proceed as if we have "two virtual processors" for the single variable-speed processor: One (called A), which is running at speed $s^{\max} - Q$, and the other one (called B) which is running with variable speeds within range $[0, Q]$.

Based on above ideas, to solve $\mathcal{P}_4$, we partition the job set into two parts: One part of jobs that can be finished by running on A with constant speed $s^{\max} - Q$, and the other part of jobs can be finished by the variable speed processor B. For these two processors, we apply two algorithms.

(1) First, we run EDF (earliest-deadline-first policy) over all the jobs using speed $s^{\max} - Q$, assuming $s^{\max} - Q > 0$. (If $s^{\max} \le Q$, we simply use Yao et al.'s algorithm [23,61] as our solution.) If a job J_j cannot be finished before its deadline, we simply cut the unfinished

processing time part at its deadline d_j and name it J'_j with remaining processing time p'_j.

(2) Second, for those unfinished processing time parts of jobs, we run an optimal algorithm to find out min-energy schedule. This step can be solved using a convex program.

The idea of partitioning the jobs into finished part and unfinished part using speed $s^{\max} - Q$ provides us the feasibility of calculating the schedule's speed switching points using a convex program. Algorithm 18 shows the details of the algorithm. The analysis is similar to that of Theorem 4.2.

Algorithm 18 Bounded Span of Frequencies Used (Q)

Require: A job set $\mathbf{J} := \{J_1, J_2, \ldots, J_n\}$ and $J_j = \{r_j, d_j, p_j\}$. The bounded span of frequencies used Q

Ensure: A piecewise curve describing the time intervals and the speeds at which the processor runs in these intervals

1: Let $R = \{r_1, r_2, \ldots, r_n\}$, $D = \{d_1, d_2, \ldots, d_n\}$, and $Z = R \cup D$.
2: Sort Z in increasing order. Let the distinct values be z_i, $1 \leq i \leq |Z|$. (Without loss of generality, assume Z has exactly $|Z|$ distinct elements.)
3: Define $T_{i,i'} = (z_i, z_{i'}]$, for any $1 \leq i < i' \leq |Z|$. Define s'_l for each $(z_{l-1}, z_l]$. Set $s'_1 = 0$ and $s'_{|Z|+1} = 0$.
4: Calculate the min-energy E^* and the maximum speed $s^{\max}$ that E^* has.
5: **if** $s^{\max} \leq Q$ **then**
6: **return** E^* as our solution;
7: **else**
8: simulate the the jobs in $\mathbf{J}$ over a machine with a constant speed $s^{\max} - Q$. For any unfinished job J_j, cut it by the deadline. Name it $J_{j'}$. $J_{j'}$ has a remaining processing time p'_j;
9: solve the convex program CP_4:

$$\begin{aligned} \min . \quad & \sum_{l=2}^{|Z|} e(s'_l)(z_l - z_{l-1}) \\ \text{subject to} \quad & W_{i,i'} \geq P_{i,i'}, \forall 1 \leq i < i' \leq |Z| \\ & 0 \leq s'_l \leq Q. \end{aligned}$$

{If J_j is an unfinished job $J_{j'}$ after we finish step 8, then the remaining processing time p'_j is used in calculating $P_{i,i'}$ as in Equation (4.5).}
10: **return** a schedule running jobs in a canonical order using speeds $s'_l + s^{\max} - Q$.
11: **end if**

4.5 Conclusions

Motivated by minimizing negative effects on processor's lifetime reliability from the perspective of designing speed-scaling algorithms, we investigate energy-aware scheduling algorithms in this chapter. Our contributions include a few scheduling algorithms for one model and three variants, optimizing energy consumption and number/cost of frequency changes. We apply the convex programming techniques to the general model. Based on this framework, we develop three polynomial-time optimal solutions for three important variants. The algorithms that we provide are proved to have objective values arbitrarily close to optimal values. We consider four optimization problems and develop their corresponding solutions. We also analyze the running time complexities. The results in this chapter are summarized in Table 4.1.

Table 4.1: Summary of the results in speed scaling algorithms with speed change constraints. (All the algorithms run in polynomial time.)

algorithms	objectives to minimize	constraints
Algorithm 15	sum of energy consumption and costs of speed changes	-
Algorithm 16	energy consumption	bounded number of speed changes
Algorithm 17	number of speed changes	bounded energy consumption
Algorithm 18	energy consumption	bounded span of frequencies used

We note that the results remain valid for arbitrary convex energy consumption functions $e(\cdot)$ with $e(0) = 0$. We do not require that $e(\cdot)$ should be in some single closed function form; it may be given by various closed formulas in different frequency ranges. A recent study considered the energy minimization for settings in which the frequency of the bus and the memory can be adjusted independently, but without considering the speed change constraints [76]. In this dissertation, we assume that only the CPU's clock frequency can be adjusted.

Chapter 5: Power-Aware Design of IP Core Networks under General Traffic Demands

This chapter is the joint work with Yigal Bejerano at Bell Labs and Spyridon Antonakopoulos at Google. This work was done when I worked as a summer intern at Bell Labs in 2012.

5.1 Introduction

As the access rate of telecommunication networks grows, an increasing number of devices and links are deployed to accommodate such growth. As a result, the energy consumption of the network gradually becomes a big concern [77–80]. Designing and implementing efficient and effective energy management methods have drawn lots of interest both from industrial and academic areas [79, 80]. An important discovery of recent studies is that current networks operate at a nearly constant power rate independent of the traffic load [81,82]. Studies also show that the temporal traffic demand is substantially smaller than the overall network capacity. In [83], the average link utilization in commercial network backbones is reported around $30 - 40\%$. Some studies [84, 85] present even worse statistics that the average link utilization may be as low as $10 - 15\%$. It is widely accepted that the significant discrepancy between traffic demand and network capacity is a consequence of considerable variations in network traffic over time as well as capacity over-provisioning by network service providers. This gap between the available network capacity and the temporal traffic demand presents opportunities for reducing network power consumption by deactivating network components without noticeably affecting network performance [81, 86–88]. Furthermore, this deactivation mechanism can be easily incorporated into the hardware of current networking routing devices [80]. Motivated by the potential for significant energy savings, we consider the problem of designing power-efficient network topologies and auto-configuration algorithms.

In particular, we consider the network design issue within an IP core network composed of multiple Points of Presence (PoPs), which is a common architecture in modern IP networks. A PoP is a collection of multiple core and access routers that are physically located in the same place (e.g., a room) and are connected to several regional networks as well as several other distinct PoPs. While there are several architectures for designing a PoP, we consider a contemporary design based on multi-chassis routing systems [89, 90]. In this architecture, the access routers serve as the end-points of the regional networks and connect to one or two multi-chassis routing systems. The latter are composed of several line-card chassis connected to a switching matrix that switches data packets between the line-card chassis. The main reasons to consider PoPs for power optimization are: 1) IP core routers have notoriously high energy consumption; 2) sleep-mode capability is easy to implement in hardware; and 3) each PoP is managed by a single entity, which simplifies deployment of new designs.

5.1.1 Impact of uncorrelated traffic

An earlier work [91] considered the design problem under correlated traffic demands, however uncorrelated traffic is more common in networks[1]. In [92], the links carrying international traffic of opposite directions (U.S. to U.K. and U.K. to U.S.) have a 5-6 hours time shift on their traffic patterns. Even within a regional network, the different traffic classes (P2P and HTTP) have very different peak hours within a day [93]. Additional evidence can be found in [94], [95] and [96]. Example 6 shows the impact of uncorrelated traffic in PoP design.

Example 6. We consider a single PoP with three chassis $\{C1, C2, C3\}$, each of which has 2 ports connecting to two regional networks, termed RN_x and RN_y. We assume each link connecting with a port carries one unit of traffic, and the maximal traffic demand at any time is three units. We want to design an optimal connection between RNs and the PoP such that, for given traffic demands, the number of active chassis is minimized. Figure 5.1(a)

[1]We use "correlated" and "uncorrelated" terms here with loose relation to their statistical meanings.

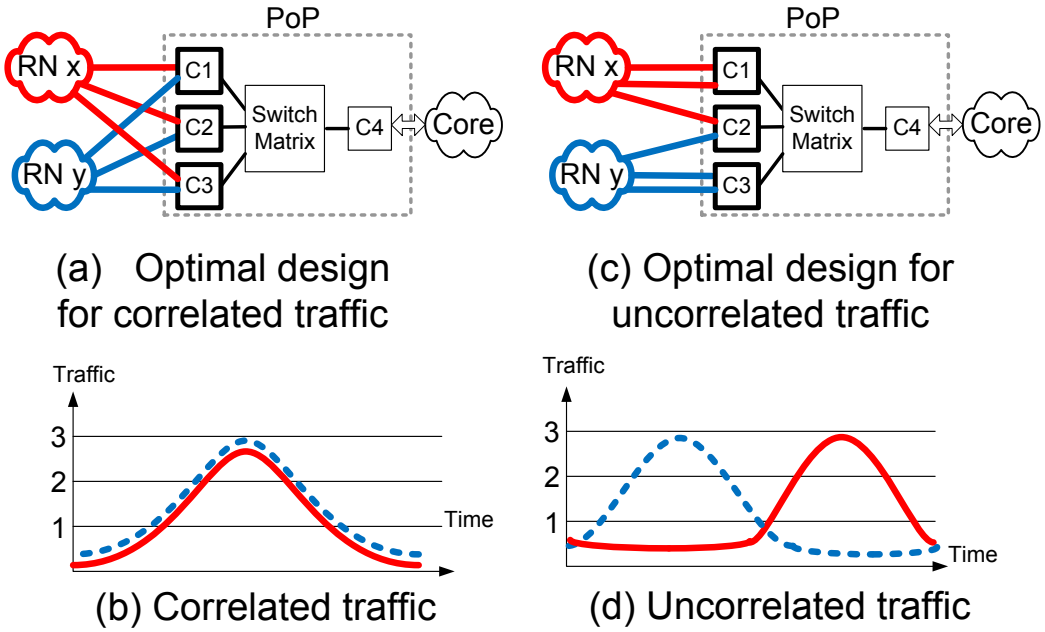

Figure 5.1: Optimal network designs for correlated and uncorrelated traffic demands, respectively.

is the optimal design when the traffic demands over the two RNs are correlated. The traffic is shown in Figure 5.1(b). We can see that as RN_x and RN_y's traffic demands grow/drop together, chassis will be active/sleep in order of $\{C1, C2, C3\}$. Such a design is obtained from the Port-Sorting algorithm presented in [91]. When the traffic demands are uncorrelated as shown in Figure 5.1(d), this connection is no longer optimal because of the different peak periods of RN_x and RN_y. This happens when two RNs serve different types of customers [96] (e.g., residential customers and commercial customers). If we use the same connection as that in Figure 5.1(a), all three chassis have to be active during both peak periods. However, the optimal solution as shown in Figure 5.1(c) has at most 2 chassis active during each peak period. Thus, the optimal solution for correlated traffic cannot be extended to uncorrelated traffic directly. New algorithms need to be developed.

5.1.2 Our contributions

1) Given the typical time-varying traffic demands between the network nodes, we consider the problem of minimizing the overall power consumption of an IP network by a careful design of the link connection and dynamic activation of only the required components, referred to as *auto-configuration*. We show that finding an optimal design and auto-configuration solution requires a complete network-wide view of the traffic demands, since the optimal auto-configuration solution for one PoP may not be optimal for another PoP. We also show that even for a given network design, an optimal auto-configuration that minimizes the overall network energy consumption has to consider the end-to-end routing of all the temporal traffic demands, which may require considerable management overhead.

2) In order to provide a simple and easily deployable solution, we consider a variant of the above optimization problem in which the temporal traffic demands of the links between two adjacent PoPs are known, however, not the end-point chassis of the links. We show that for this variant, the energy consumption of the network can be optimized by considering the connection of each PoP independently.

3) We prove that the problem of minimizing the power consumption of individual PoP with given traffic demands is NP-complete. We then develop an optimal algorithm, Matching, for a variant where each chassis has only 2 ports. For general cases, we design two approximation algorithms, Iterative Matching and Disjoint Set Cover, and provide their approximation guarantees, i.e., the worst case ratio of each algorithm's solution to the optimal solution. We also propose an efficient greedy algorithm with a low running time complexity. The main algorithmic results in this chapter are summarized in Table 5.1, where N is the number of chassis within a PoP and P is the number of ports per chassis. We show that these algorithms are optimal when traffic demands are correlated. We also extend our schemes to support links with various capacities, and address the related network reliability issues.

4) Our extensive simulations demonstrate $20 - 60\%$ energy savings on line cards and chassis due to our algorithms, outperforming several other candidate solutions. In some

Table 5.1: Summary of the results. Note that if either $P = 1$ or $N = 1$, the problem is trivial.

P	$N \geq 2$
2	Matching: an optimal algorithm
≥ 3	Iterative Matching: a $P/2$-approximation algorithm ($P = 2^\kappa$, $\kappa \in \mathbb{Z}^+$)
	Disjoint Set Cover: a $2 \ln N$-approximation algorithm

instances, our algorithms achieve almost twice as much energy savings as that of the Port Sorting algorithm [91] when the traffic is uncorrelated. They also perform substantially better than some simple strategies deployed in current networks. We also test our algorithms' sensitivity to the inaccuracy of the traffic estimation and show that our algorithms' performance is stable.

5.2 Related work

In 2003, Gupta and Singh [97] examined the excessive energy usage of the wired Internet, and first proposed the strategy to save energy by selectively putting network interfaces and components into sleep mode. Since then, more and more emphasis has been put on power management of the Internet and the devices connected to it. Recent surveys on this topic are given in [80, 98]. In this section, we discuss only the studies that are most relevant to our work.

Chabarek et al. [81] considered the power-efficient network design problem as a multi-commodity flow problem and derived an LP formulation. Similarly, Christensen et al. [87,99] proposed heuristic methods in powering down idle links or nodes. Heller et al. [100] studied the problem of minimizing energy consumption in data center networks. They exploited the unique structure of data center topologies in their proposed solution. However, all these works assumed that the topology is provided as an input to the problem.

The problem of power-efficient routing in wired networks has also received considerable attention in recent years. Andrews et al. [86] studied the problem of minimum energy routing in wired networks using the *speed scaling model*. In this model, the processing power

of a network element is adjusted in proportion to its traffic. In [101], the authors considered the same problem assuming that the network elements have *power down* capability. The impact of switching off links on the connection of the network was examined in [88].

Antonakopoulos et al. [102] investigated the potential power savings that are obtained using a combination of rate-adaptive network elements and power-aware routing. Like [86, 101], the authors employed a power model in which only the power consumption of the links is considered while the power consumption of routers and switches is ignored. In this model, the authors proposed a heuristic routing scheme and demonstrated the potential power savings by simulations.

A novel technique to reduce network energy consumption has been proposed in [103]. Their approach buffered traffic at the edges and introduced burst to allow network devices to sleep for a longer period of time. However, this work did not consider the network design problem, which is of interest in this chapter. Restrepo et al. [104] studied the minimum energy routing by taking the energy profile of devices into account. However, this work required detailed power models of the various network elements as input, and it did not consider the network design problem neither.

The PoP design problem was later studied in [91,105]. The main objective in [105] is to reduce the equipment cost in a PoP, as opposed to reducing power consumption (which is the main objective of our work). In [91], the authors studied a similar model to minimize the PoP energy consumption by carefully designing the link connection. However, their solution is optimal for only the fully correlated traffic demands. On the contrary, our work deals with uncorrelated traffic demands.

5.3 Model and Assumptions

5.3.1 The network model

We consider a two-level IP core network. The higher level, referred to as the *network level*, consists of multiple *points-of-presence* (PoPs). Each PoP, e.g., PoP A, provides communication services to *regional networks*[2] (RNs) and is also connected to several other PoPs. We refer to these PoPs and regional networks as the *adjacent nodes* (ANs) of PoP A and denote this set as R_A. Every adjacent node and PoP A are connected with one or more IP *links* that can meet the maximal traffic demand between the two nodes. We first assume that all the links have the same fixed capacity, e.g., 100Gbps. This constraint is relaxed later in Section 5.5.6. We refer to the graph that represents the network of PoPs as the *network graph*, where every node represents either a PoP or a RN and each edge is a hyper-link that represents all the IP links between two adjacent nodes.

The IP links between adjacent nodes and the internal topology of each PoP constitutes the lower level of the IP core network. Though there are several possible PoP architectures (see [89,90,106]), we consider only a contemporary PoP architecture of *Single Multi-Chassis System* [89, 90], in which a PoP contains a multi-chassis router that allows multiple chassis to be clustered together to form a single *logical router*. The latter consists of several *line card chassis* (LLCs) that are connected to a non-blocking scalable switch fabric matrix via their backplanes. Every chassis supports several *line cards*, each with a fixed number of *ports*. Each port is then connected to one link. We refer to the graph that represents the specific link connection between the chassis of different nodes as the *complete graph*. The two graphs of an IP core network are illustrated in Example 7.

We use the power consumption model identical to the one in [81]. We defer the formal definition of the power consumption of a PoP to Section 5.4.2.

Example 7. Figure 5.2 is an example of an IP core network with two POPs and four regional networks (RNs). Figure 5.2(a) shows only the network graph while Figure 5.2(b)

[2] A peering IP core network is also considered as a RN.

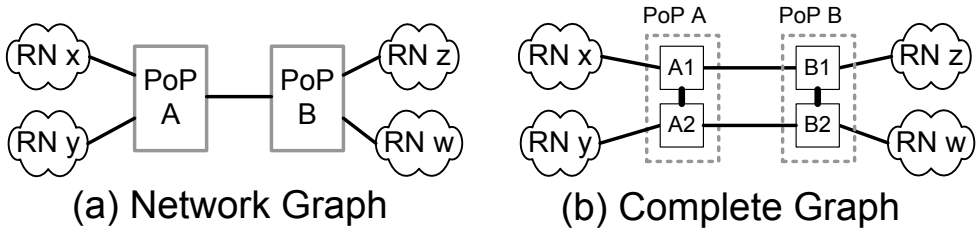

Figure 5.2: An example of an IP core network's network graph and complete graph.

provides the complete graph with detailed connection. The PoPs are denoted as POP A and POP B, each with two line card chassis, marked as A_1, A_2, B_1, B_2, accordingly. The two chassis of each PoP are connected via the PoPs' switch matrix (represented in the figure by a link between the two chassis of a given PoP). In addition, each chassis is connected by a link to a regional network and to a chassis at the other PoP. Although there is a single path between any pair of RNs at the network level as shown in Figure 5.2(a), Figure 5.2(b) demonstrates that the network provides several paths between any pair of RNs. For instance, RN_x and RN_w are connected by the following paths: $P_1 = \{RN_x, A_1, A_2, B_2, RN_w\}$ and $P_2 = \{RN_x, A_1, B_1, B_2, RN_w\}$.

5.3.2 Traffic demands

We consider a general model of time varying traffic between any pair of RNs, which may be routed along a single path or multiple paths between the two nodes at the network level. We do not impose any constraints on the routing mechanism nor the traffic patterns. Instead, we assume that the *temporal traffic pattern* between two adjacent nodes has periodic behavior (e.g., a week), which can be predicated[3]. Given a specific PoP, e.g., PoP A, we denote by $f_i(t)$ the *temporal traffic demand* function between PoP A and each of its adjacent nodes $i \in R_A$ at any time t during a given period of time. The temporal traffic demands are given using units of link capacity. Thus, at any time t, we have $0 \leq f_i(t) \leq K_i$, where K_i is the number of links between PoP A and AN i, and $\lceil f_i(t) \rceil$ is the minimal number of links that

[3]Our simulations in Section 5.6 show that our scheme can tolerate significant errors in the traffic demand predications.

needs to be active between PoP A and AN i. We divide the time into T slots with fixed duration and the traffic demands are given in granularity of a time slot[4].

In our work, we consider the *correlation*[1] between the traffic demand functions $f_i(t)$, $i \in R_A$, of PoP A. Let $u_i(t) = f_i(t)/K_i$ be the normalized traffic demand between PoP A and AN $i \in R_A$. We consider the maximal and minimal values of these functions at any given time t, specified by the following functions, $u_{max}(t) = \max_{i \in R_A} u_i(t)$ and $u_{min}(t) = \min_{i \in R_A} u_i(t)$. The correlation of the traffic demands is specified by a *correlation* parameter $\alpha \in [0, 1]$ such that,

$$\alpha = \max_t \{u_{max}(t) - u_{min}(t)\}. \tag{5.1}$$

Consequently, each temporal demand function $f_i(t)$ can be written as a combination of a *correlated* and an *uncorrelated* components, defined by the functions $h(t)$ and $g_i(t)$, accordingly, such that:

$$f_i(t) = K_i \cdot [(1 - \alpha) \cdot h(t) + \alpha \cdot g_i(t)]. \tag{5.2}$$

If $\alpha = 0$, then $u_{max}(t) \equiv u_{min}(t)$ and for every $i \in R_A$ we have $f_i(t) = K_i \cdot u_{min}(t)$, i.e, $h(t) = u_{min}(t)$ and $g_i(t)$ is ignored. In this case, we say that the traffic demands are *fully-correlated*. If $\alpha = 1$, then $f_i(t) = K_i \cdot g_i(t)$ and $h(t)$ is ignored, and the traffic demands are termed *uncorrelated*. If $0 < \alpha < 1$, the traffic demands are termed *quasi-correlated* and the two components of $f_i(t)$ can be calculated by

$$h(t) = \frac{u_{min}(t)}{1 - \alpha}, \qquad g_i(t) = \frac{u_i(t) - u_{min}(t)}{\alpha}.$$

As an example, we consider the traffic demands given in Figures 5.1(b) and 5.1(d). In the former case $\alpha = 0$, which indicates that the traffic demands are correlated, while for the later case $\alpha \approx 1$, and hence the traffic demands are uncorrelated. Unlike [91] that considered only correlated traffic demands, our study addresses both uncorrelated and quasi-correlated

[4]The duration of each time slot is in the order of minutes, which is substantially longer than time overhead of active/sleep transitions of devices.

demands.

5.4 Problem Statement

We now define the network design and auto-configuration problems addressed in this study.

5.4.1 Network design and auto-configuration

Definition 5.1 (Network Design and Auto-Configuration Problem)**.** *Given a network graph with temporal traffic demands of each PoP, the goal of the* Network Design and Auto-Configuration (NDA) *problem is to find a complete graph and a schedule that specifies the active and sleep time of each chassis, line card and port, such that the traffic demands are satisfied by the active components at any time, while minimizing the overall power consumption of the network.*

In the NDA problem, we (1) design the connection of the links, and (2) determine the time periods for each component in which it is active. As we show in Example 8, finding optimal design and auto-configuration solution requires a complete network-wide view of the traffic demands since the optimal auto-configuration solution for one PoP may not be optimal to another PoP.

Example 8. Consider the network presented in Figure 5.2. Let us assume that at some time slot, the only traffic in the network is between RN_x and RN_w. By considering each PoP independently, it seems sufficient to keep only one chassis active at each PoP to support this traffic, while in practice at least one of the PoPs has *two* active chassis. As described in Example 7, there are two possible paths between RN_x and RN_w: $P_1 = \{RN_x, A_1, A_2, B_2, RN_w\}$ and $P_2 = \{RN_x, A_1, B_1, B_2, RN_w\}$. Note that at each path, the traffic traverses through two chassis of one of the PoPs, thus the energy consumption of this PoP is not optimal.

Example 8 shows that even for a given network design, an optimal auto-configuration that minimizes the overall energy consumption of the network has to consider the end-to-end routing of all the temporal traffic demands. Such approach raises not only a challenging

algorithmic problem, but also a complex management problem of determining the end-to-end routing of each flow and active components at each time. To simplify the problem and to reduce the management complexity of the network, we consider a variant of the NDA problem that allows us to find an optimal solution, while designing the connection and auto-configuration of each PoP separately.

5.4.2 The PoP design problem

We consider a specific PoP A and an adjacent node $i \in R_A$ that are connected with K_i links indexed from 1 to K_i. In the revised NDA problem, we assume that link j, $1 \leq j \leq K_i$, is activated only when the traffic demands $f_i(t)$ between the two nodes exceed the capacity of the first $j-1$ links, i.e., $f_i(t) > (j-1)$. Revisiting Example 8 and assuming that the link (A_1, B_1) has an index 1, then the traffic from RN_x to RN_w is forwarded only along the path $P_2 = \{RN_x, A_1, B_1, B_2, RN_w\}$.

The following two essential properties are direct results from the above modification:

1. *The active time of each link depends only on the traffic demand function $f_i(t)$ and its index*, regardless of the specific end-point chassis of the link. So the active time slots of a link can be represented by a vector of $\{0, 1\}$ of size T.

2. Since the active time of each link is independent of its specific end-point chassis, *the energy consumption of the entire network can be optimized by minimizing the power consumption of each PoP independently.*

In the following, we consider a PoP with $M = \sum_{i \in R_A} K_i$ links, each identified by a unique index in the range $L_M = \{1, \ldots, M\}$. We represent the temporal traffic demands of the PoP by a demand matrix $F_{[M,T]}$ of $\{0, 1\}$ with M rows and T columns, such that row $l \in \{1, \ldots, M\}$ specifies the active slots of link l. We assume that the PoP has N identical chassis, with D line cards per chassis and P ports per line card, such that $M = N \cdot D \cdot P$. The link assignment $\mathcal{L}$ of the PoP matching its M incoming links to its M ports is defined

as follows: Each chassis $i \in \{1, .., N\}$ is associated with a set L_i of $D \cdot P$ links, and each line card $j \in \{1, \ldots, N \cdot D\}$ is associated with a set $\hat{L}_j$ of P links.

We now give a formal definition of the power consumption of the PoP. Considering any set L of links, we denote by $H(L)$ the number of time slots in which at least one of the links in L is active: (Note that $\bigvee$ is an OR operation.)

$$H(L) = \sum_{t=1}^{T} \bigvee_{l \in L} F_{l,t}. \tag{5.3}$$

Observe that any component (the switch matrix, a chassis, a line card or a port) is active only when at least one of its associated links is active, otherwise it is switched off for power saving. Thus, *a given link assignment $\mathcal{L}$ explicitly defines the time slots in which a component is active.* Consequently, the power consumption of the PoP is given by:

$$Power = H(L_M) \cdot W_{sw} + \sum_{i=1}^{N} H(L_i) \cdot W_{ch} + \sum_{j=1}^{N \cdot D} H(\hat{L}_j) \cdot W_{lc} + \sum_{l=1}^{M} H(\{l\}) \cdot W_p, \tag{5.4}$$

where W_{sw}, W_{ch}, W_{lc}, and W_p denote the power consumption of the switch fabric and each chassis, line card, and port, respectively[5]. The power aware PoP design problem is defined as follows:

Definition 5.2 (PoP Design Problem)**.** *Given a PoP with N chassis, D line cards per chassis, P ports per line card, and temporal traffic demand matrix $F_{[M,T]}$, where $M = N \cdot D \cdot P$, the* PoP Design (PD) *problem seeks for a link assignment $\mathcal{L}$ for the M links of the PoP to its M ports and a corresponding schedule, such that the overall power consumption of the PoP is minimized.*

Equation (5.4) shows that the energy consumptions of the switch fabric and the ports are fixed for a certain input traffic $F_{[M,T]}$, regardless of the link assignment $\mathcal{L}$. Therefore,

[5]Assume that the energy cost of switching on/off any component is negligible.

we can remove the energy consumption of these components from our consideration. In real products [107], these components contributes $20 - 30\%$ of total power consumption of a PoP. By carefully observing the problem definition, we see that the problem can be solved in a two-iteration way. In the first iteration, we only consider how to match M links to N different chassis regardless of the line cards assignment. Next, for each chassis, we iteratively match links with line cards in the same way that we solve the first iteration. Based on this observation, without loss of generality, we can assume $D = 1$ and we have $M = N \cdot P$ links.

After this simplification, we consider a link assignment $\mathcal{L} = \{L_1, \ldots, L_N\}$ with links in L_i being assigned to chassis i. Assuming that *a chassis consumes one unit of energy for one time slot when it is active*, the PoP's power consumption for the assignment $\mathcal{L}$ over a traffic matrix $F_{[M,T]}$ is given by:

$$E(\mathcal{L}) = \sum_{i=1}^{N} H(L_i). \tag{5.5}$$

In spite of this simplification, we prove that the problem is still NP-complete in the next section. A summary of the main notations used throughout this chapter is given in Table 5.2.

5.4.3 NP-completeness of the problem

Theorem 5.1. *The PoP Design Problem is NP-complete.*

Proof. Given a connectivity scheme, we can verify in polynomial time that whether the total energy consumption under this assignment is larger than E or not by using Equation (5.5). Thus, the problem is in NP. Then, we will prove its NP-completeness by reducing from 3-PARTITION problem.

In a 3-PARTITION problem [39], there is a set A of $3m$ elements $\{a_1, a_2, \ldots, a_{3m}\}$, and a bound $B \in \mathbb{Z}^+$. Each element a in A has a size $s(a) \in \mathbb{Z}^+$, such that $B/4 < s(a) < B/2$,

Table 5.2: Main notation used in this chapter.

Symbol	Semantics
AN	Adjacent node (a reginal network or a PoP)
RN	Regional network
α	The correlation parameter
C	A chassis within the PoP
D	The number of line cards within a chassis
K_i	The number of links between the PoP and the i-th AN
l	A link or link index
L	A set of incoming links
L_M	The set of all incoming links
$\mathcal{L}$	A link assignment
M	The number of links connecting to the PoP
N	The number of chassis within the PoP
P	The number of ports over a single line-card chassis
t	Time or time slot index
T	The number of total time slots of traffic
$F_{[M,T]}$	The $\{0,1\}$ matrix of traffic demands
$F_{l,t}$	=0, when link l has traffic at time t; =1, otherwise.
$H(L)$	The number of active time slots of L
$E(\mathcal{L})$	The PoP's power consumption given $\mathcal{L}$

and $\sum_{a\in A} s(a) = mB$. The 3-PARTITION problem wants to partition elements in A into m disjoint set $A_1, A_2, \ldots, A_m$ such that $\sum_{a\in A_i} s(a) = B$, $\forall i \in [1, m]$.

Given the above 3-PARTITION problem, we define the following instance in our problem. We have m chassis and each of which has B ports, and there are mB links to be assigned. Here, we consider the traffic demands over the time interval $[1, 3m]$, that is, $T = 3m$. For each time step $t \in [1, 3m]$, we define a group g_t of links corresponding to the element a_t. This group contains $s(a_t)$ distinct links which are active only within this time step t. We assume that the chassis spends one unit of energy while keeping itself active within one time step.

Now we claim that 3-PARTITION problem has a solution if and only if there is an assignment of links with total energy consumption $3m$.

If there is a partition $A_1, A_2, \ldots, A_m$ such that A_i contains three elements a_{i1}, a_{i2} and

a_{i3}, then we just assign the links in groups g_{i1}, g_{i2} and g_{i3} to the ith chassis. In this way, we will end up with m chassis. Because the total size of three elements in each A_i, $i \in [1, m]$, is B, each chassis will have B links correspondingly. Based on the way we generate the links, we also know that all links in a chassis are active within only three time steps. Therefore, the total energy consumption of all chassis is $3m$. In this way, we successfully find an assignment of links with total energy consumption $3m$.

Now we prove from another direction. Due to the constraint $B/4 < s(a) < B/2$, the total number of links within each group g will be in the range $[B/4, B/2]$. So we know that a chassis can accommodate at most 3 complete groups of links. We also know that the links within the same group are active within the same single time step and different group of links will be active in different time steps. Then, if there exists an assignment for links with total energy consumption $3m$, each chassis must be active in exactly 3 time steps. Otherwise, the chassis which is active more than 3 time steps must contain links from more than 3 groups. That is, some groups have their links separated over different chassis. This will obviously cause the total energy consumption growing over $3m$. Overall, each router will contain exactly three complete groups of links which sum up to B, and this indicates a partition for the original 3-PARTITION problem. □

5.4.4 An illustrating example

Throughout this chapter, we use the following example with uncorrelated traffic demands to demonstrate the problem, the optimal solutions, and the link assignment $\mathcal{L}$ obtained from the proposed algorithms.

Example 9. We consider $M = 8$ links, denoted by $\{l_1, \ldots, l_8\}$, to be connected to a PoP. The traffic of these 8 links are given over a time span of $[1, 9]$. Each link is active for several consecutive[6] time slots as shown in Figure 5.3(a). The rectangle bar with index of the link indicates the time slots in which this link is active. For example, link l_1 is active within time $[1, 3]$. Given these information, we consider the following two problems: (1) assign

[6]In general, the active slots of a line are not required to be consecutive.

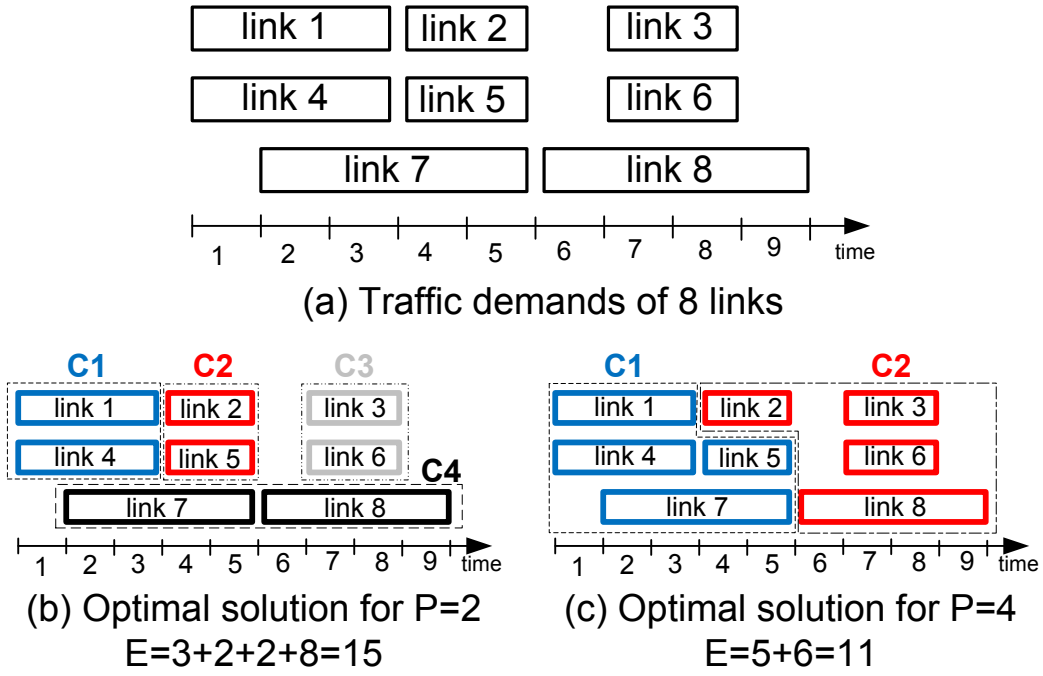

Figure 5.3: An example of traffic demands for a PoP with $M = 8$ and the optimal solutions for $P = 2$ and $P = 4$.

these link to $N = 4$ chassis $\{C_1, C_2, C_3, C_4\}$ (i.e. $P = 2$); and (2) assign these links to $N = 2$ chassis $\{C_1, C_2\}$ (i.e. $P = 4$), such that the total energy consumption of the PoP is minimized.

Figure 5.3(b) and Figure 5.3(c) show optimal solutions for $P = 2$ and $P = 4$, respectively. In these figures, links assigned to the same chassis have the same color and are framed together by black dashed lines. For example, in the case $P = 4$, links l_1, l_4, l_5, and l_7 are connected to the same chassis C_1, and links l_2, l_3, l_6, and l_8 are assigned to the chassis C_2. Remember that a chassis is active when any of its links has traffic, and is asleep when none of its links has traffic. Then, the optimal energy cost, denoted by E, is 15 for the case $P = 2$ and 11 for the case $P = 4$.

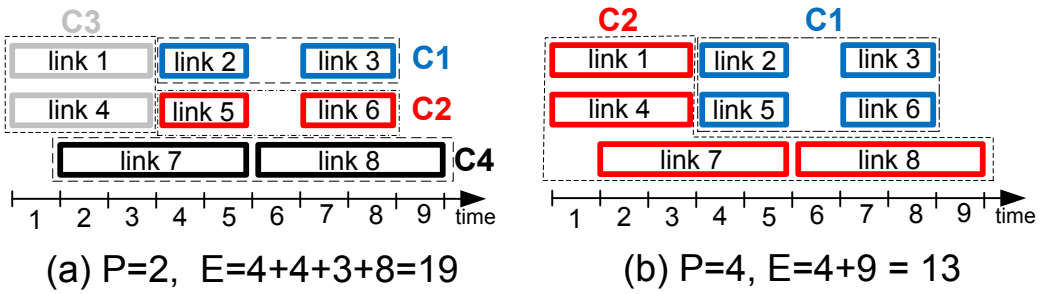

Figure 5.4: The calculated solutions by the Port Sorting algorithm when $P = 2$ and $P = 4$.

5.5 Algorithms

5.5.1 Optimal algorithms for some variants

Port sorting for correlated traffic

In [91], an exact optimal algorithm, termed *Port Sorting* is given for solving our problem when traffic is fully correlated.

The Port Sorting algorithm takes advantage of special coverage relationship that fully correlated traffic have. The *coverage relationship* describes the relation of traffic patterns of a pair of links. Given two links l_1 and l_2, we say that l_1 *covers* l_2, if at any time slot t when l_2 is active, l_1 is also active, i.e., when $F_{l_2,t} = 1$, we have $F_{l_1,t} = 1$ as well. In [91], the authors proved that when the traffic is fully correlated any link which has longer active time must cover those links with shorter active time. We call this special coverage relationship *nesting property*. Therefore, the algorithm sorts the links in increasing order of the amount of active time slots, and assigns links in such order to chassis.

However, in the case of uncorrelated traffic, the optimality of Port Sorting algorithm is not maintained. Here, we use the instance described in Example 9 to explain the sub-optimality of Port Sorting under uncorrelated traffic.

Example 10. Consider the instance presented in Example 9. The Port Sorting algorithm arranges the lines according to their active period and get the following order: $l_2, l_3, l_5, l_6, l_1, l_4, l_7$, and l_8. Then it iteratively associates a set of P consecutive lines to

each chassis. Figure 5.4 shows the results. Note that, the indices of the chassis reflect the order in which the algorithm assigns links to chassis. For example, the Port Sorting algorithm first assigns the links l_2 and l_3 to chassis C_1 in the case $P = 2$, because these two links have the least active durations. Given the assignments, the energy costs of the solutions are 19 for $P = 2$ and 13 for $P = 4$.

We can see that the Port Sorting does not provide the optimal solutions here, because the traffic of the links in Example 10 do not have the "nesting property" as described above. In Section 5.6 we demonstrate that in general the Post Sorting algorithm provides worse designs than the ones provided by our algorithms, presented later, for uncorrelated traffic.

Optimal solution for general traffic

Algorithm 19 Matching ($P = 2$)

1: **Input:** The traffic demand matrix $F_{[M,T]}$ of M links.
2: Create a graph $G(V, E)$, $V \leftarrow \emptyset, E \leftarrow \emptyset$.
3: For Each link l_i, create a node $v_i \in V$.
4: Connect any two nodes $v_i, v_j \in V$ with a link $e_{i,j} \in E$.
5: For any edge $e_{i,j}$, associate a weight $w_{i,j} = H(\{l_i, l_j\})$.
6: Find a minimum weighted perfect matching Y on G.
7: **if** v_i and v_j are matched in Y **then**
8: assign links l_i and l_j to the same chassis.
9: **end if**
10: **return** the assignment.

For the special case $P = 2$, we show that there exists an optimal algorithm running in polynomial time. This algorithm is referred to as *Matching* and is given in Algorithm 19. We prove its optimality in Theorem 5.2.

Theorem 5.2. *The* Matching *algorithm provides the optimal solution in polynomial time under general traffic when* $P = 2$.

Proof. Based on the way that we transform our input M and F to a graph G, we know that G is a complete graph. Thus, any connection scheme has a corresponding perfect matching in G. We also know that the total weight of a perfect matching in graph G is the same

as the corresponding total energy consumption of the connection scheme, because the edge weight $w_{i,j}$ in G is set equal to the energy consumption by grouping links l_i and l_j together.

Assume that the assignment obtained from the algorithm for $P = 2$ is not optimal. Let $Match$ denote the link assignment obtained from *Matching*, and OPT denote the optimal link assignment. Then let the corresponding perfect matching of these two link assignments in graph G be Y_{Match} and Y_{OPT}. Based on the property of *Matching*, Y_{Match} is the minimum weighted perfect matching returned by our algorithm. However, given the above assumption, we know that Y_{OPT} has less total weight than Y_{Match}, which contradicts with the fact that Y_{Match} is the minimum weighted perfect matching of graph G.

Now, we analyze the running-time complexity of *Matching*. For the case $P = 1$ or $N = 1$, the algorithm runs in time $O(M)$. By using the Edmond's Blossom algorithm [108] [109], minimum weighted perfect matching in a general graph $G(V, E)$ could be solved within time $O(|V|^4)$. This time complexity could be further lowered to $O(|V|^3)$ due to Lawler [110] and Gabow [111]'s work. Since $|V| = M$ in our problem, we know that *Matching* has a running time complexity $O(M^3)$.

Therefore, *Matching* is an optimal algorithm for $P = 2$ running in polynomial time. This completes the proof. □

5.5.2 The Iterative Matching algorithm

Algorithm 20 Iterative-Matching ($P = 2^\kappa$)

1: **Input:** The traffic demand matrix $F_{[M,T]}$ of M links.
2: Let L denote the set of links (super-links) in the current iteration. Initially, $L = \{1, 2, \ldots, M\}$, where $l_i \in L$ is the index of a link.
3: **while** $|L| > N$ **do**
4: pairing up links in set S using the *Matching* algorithm;
5: merge any paired-up links into a new super-link with traffic pattern as the union of these two;
6: add this super-link into set L, and remove the two paired-up links.
7: **end while**
8: Assign all links contained in one super-link to the same chassis.
9: **return** the assignment.

Motivated by the *Matching* algorithm, we design an approximation algorithm naturally

derived from it, termed *Iterative-Matching*. That is, we can do *Matching* iteratively for cases in which $P = 2^\kappa$, $\kappa \in \mathbb{Z}^+$. This *Iterative-Matching* algorithm is described in Algorithm 20. Remember that when $P = 2$, we have an optimal algorithm *Matching* by transforming the problem to a minimum weighted perfect matching problem. Here, we just use *Matching* as a subroutine to iteratively build up the solution. Combining with the case $\kappa = 1$, we have Theorem 5.3.

Theorem 5.3. *For $P = 2^\kappa$ where κ is a non-negative integer, the* Iterative-Matching *algorithm has an approximation ratio of $P/2$.*

Before proving Theorem 5.3, we introduce several useful claims.

Claim 1. *For any two sets of links L_i and L_j, we have*

$$H(L_i \cup L_j) \leq H(L_i) + H(L_j) \leq 2H(L_i \cup L_j).$$

Proof. Recall that for any set L of links the value of $H(L)$ is the number of time slots in which at least one of the links in L is active. The first inequality in Claim 1 is true since any time slot t that contributes 1 to $H(L_i \cup L_j)$ also contributes 1 to $H(L_i)$ or $H(L_j)$ or both. The same reason can be used to prove the second inequality, because t may increase $H(L_i) + H(L_j)$ by at most 2. □

Claim 2. *For any two sets of links L_i and L_j with $L_i \subset L_j$, we have $H(L_i) \leq H(L_j)$.*

Proof. The proof is similar to the proof of Claim 1. □

Let IM_i be the calculated sets of links after the i-th iteration of the *Iterative-Matching* algorithm assuming that each chassis supports 2^i links. In addition, let OPT_i be the optimal solution when the number of ports per chassis is 2^i.

Claim 3. *For any $i \in [1, \kappa]$, $E(OPT_i) \leq 2 \cdot E(OPT_{i+1})$.*

Proof. For any $i \in [1, \kappa]$, we let L_j be the set of links associated with the j-th chassis in OPT_{i+1}. Note that $|L_j| = 2^{i+1}$. We can create a new solution, termed $Rand_i$, for $P = 2^i$. This new solution randomly divides each set L_j into two subsets L_j^1 and L_j^2, each having 2^i links. From Claim 1, it holds that $H(L_j^1) + H(L_j^2) \leq 2 \cdot H(L_j^1 \cup L_j^2) = 2 \cdot H(L_j)$. Thus,

$$\begin{aligned} E(Rand_i) &= \sum_{j=1}^{M/2^{(i+1)}} (H(L_j^1) + H(L_j^2)) \\ &\leq 2 \cdot \sum_{j=1}^{M/2^{(i+1)}} H(L_j) \\ &= 2 \cdot E(OPT_{i+1}). \end{aligned}$$

Assume for contradiction that $2 \cdot E(OPT - i + 1) < E(OPT_i)$. Then we have $E(Rand_i) < E(OPT_i)$, which contradicts the assumption that OPT_i is the optimal solution for $P = 2^i$. □

Claim 4. *For any* $1 \leq i \leq \kappa$, $E(IM_{i+1}) \leq E(IM_i)$.

Proof. The proof is similar to the proof of Claim 3 and it is a direct result of Claim 1. □

We now use Claims 3 and 4 to prove Theorem 5.3.

Proof of Theorem 5.3. From Theorem 5.2, we know that after the first iteration, our algorithm achieves the optimal solution for $P = 2$. Thus, $E(IM_1) = E(OPT_1)$. From Claim 4, we know $E(IM_\kappa) \leq E(IM_1) = E(OPT_1)$. From Claim 3, we have $E(OPT_1) \leq 2E(OPT_2) \leq \ldots \leq 2^{\kappa-1}E(OPT_\kappa)$. Combining all these inequalities, we get $E(IM_\kappa) \leq 2^{\kappa-1}E(OPT_\kappa)$. Therefore, our algorithm has an approximation ratio $P/2$. □

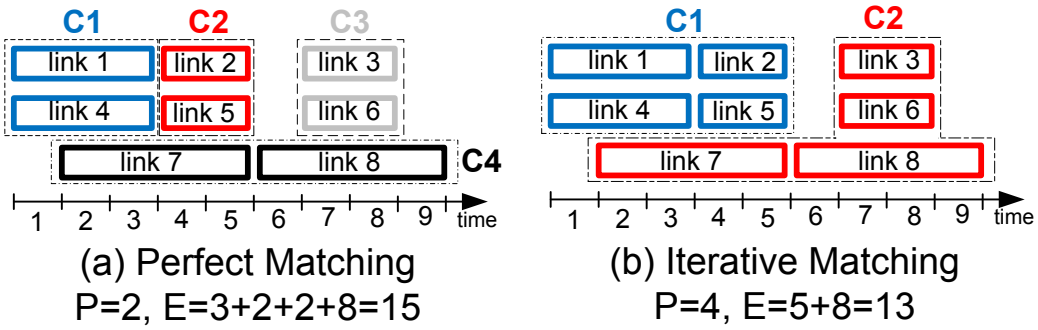

Figure 5.5: The solutions obtained from the Iterative Matching algorithm when $P = 2$ and $P = 4$.

Example 11. We use Example 9 to illustrate how *Iterative-Matching* works and compare its performance with the optimal solutions. The results are shown in Figure 5.5. Note that *Iterative-Matching* is actually *Matching* when $P = 2$, and it gives the optimal solution. So Figure 5.5(a) shows the same results as Figure 5.3(b). For $P = 4$, the algorithm iteratively runs *Matching* on the result given in Figure 5.5(a) by considering the two links in the same chassis as a new link with modified traffic pattern. For example, links l_1 and l_4 are considered as a single new link which is still active during time $[1, 3]$, while links l_7 and l_8 are considered as a new link which is active during time $[2, 9]$. Therefore, the algorithm outputs an assignment with energy cost 13 when $P = 4$. We note that in this case the solution is not optimal.

5.5.3 The Disjoint Set Cover algorithm

For the cases in which P takes arbitrary value rather than power of 2, we give a pseudo-polynomial approximation algorithm, termed *Disjoint Set Cover* (DSC). This algorithm adopts ideas of greedy approximation algorithm for weighted set cover problem.

In the weighted set cover problem [112], there is a universe set U of elements to be covered. We also have a set $\mathcal{S}$ which contains a collection of subsets of U. That is, $\mathcal{S} = \{S_1, \ldots, S_k\}$, where $S_i \subseteq U$ and $\bigcup_i S_i = U$, for $i \in [1, k]$. Each subset S_i has a weight $w(S_i)$. The objective is to find a sub-collection $\mathcal{C} \subseteq \mathcal{S}$, $\mathcal{C} = \{S'_1, \ldots, S'_{k'}\}$, with the minimum total

weight $W' = \sum_{j=1}^{k'} w(S'_j)$ that covers all elements in U.

In our problem, the universe set U will be all links to be assigned, and set $\mathcal{S}$ is all possible P-combinations of links in U. Then the weight of the set S_i is the energy consumption of the chassis after assigning links in set S_i onto it, i.e., $w(S_i) = H(S_i)$ where $H(.)$ is defined by Equation 5.3.

For the weighted set cover problem, there is a greedy approximation algorithm [112], which iteratively finds the most cost-effective subset and removes all elements in this subset from U till U becomes empty. Here, the cost-effectiveness is defined as the average weight of uncovered elements in a subset. Recall that in weighted set cover there is no guarantee that each subset picked will not have overlap with other selected subsets. In our algorithm, we enforce that *the picked subsets do not overlap with each other* by doing the following. At each iteration, we only consider the subsets with P links that do not contains any covered links and we pick such a subset with the minimum weight. The algorithm is formally described in Algorithm 21, where U denotes the set of all links and Z denotes the set of links already covered.

Algorithm 21 Disjoint Set Cover

1: **Input:** traffic demand matrix $F_{[M,T]}$ of M links.
2: $Z \leftarrow \emptyset$, $U = L_M$
3: **while** $Z \neq U$ **do**
4: find the subset S of size P in $U \setminus Z$ which has the minimum weight;
5: assign all links in S to a new chassis;
6: $Z \leftarrow Z \cup S$.
7: **end while**
8: **return** the assignment.

Since we select only disjoint subsets, we cannot claim anymore that at each iteration our algorithm selects the most cost effective subset. Therefore, we cannot use the proof in [112] to guarantee a logarithmic approximation ratio to our algorithm. Nevertheless, several properties of our solution allow us to provide similar approximation ratio.

Theorem 5.4. *The Disjoint Set Cover algorithm provides a* $2 \ln N$*-approximation solution.*

Let DSC and OPT be the solutions calculated by DSC and the optimal algorithm

accordingly. Let $E(DSC)$ and $E(OPT)$ be the cost of these solutions. We denote by L_i the subset picked at the i-th iteration by DSC. We now introduce a property that is essential for proving Theorem 5.4.

Claim 5. *At each iteration i, $1 \leq i \leq (N-1)$:*

$$H(L_i) \leq (\frac{1}{N-(i-1)} + \frac{1}{N-i}) \cdot E(OPT)$$

Proof. Consider the set $U \setminus Z$ of uncovered links at the beginning of the i-th iteration. We now consider the following two most cost effective subsets, say S_1 and S_2, each one with *at most P links* selected as follows: The subset S_1 is the most cost effective set for covering the links in $U \setminus Z$ with the largest size, while S_2 is the next most cost effective subset for covering the links in $U \setminus Z \setminus S_1$. Recall that the cost of each set is $H(S_j)$, $j = \{1, 2\}$. We argue that $P < |S_1| + |S_2|$. Let us assume for contradiction that $|S_1| + |S_2| \leq P$. Then from Claim 1 it holds that $H(S_1 \cup S_2) \leq H(S_1) + H(S_2)$, which contradicts the assumption that S_1 is the most cost effective subset with largest size of at most P links.

We now find bounds on the cost of the sets S_1 and S_2. S_1 is the most cost effective subset for the collection $U \setminus Z$ with $(N-(i-1)) \cdot P$ uncovered links. The optimal cost of covering these links with subsets of P or less links is at most $E(OPT)$ and there are at least $N-(i-1)$ such subsets in any such coverage. Therefore, $H(S_1) \leq \frac{E(OPT)}{N-(i-1)}$. S_2 is the most cost effective subset for the collection $U \setminus Z \setminus S_1$ with at least $(N-i) \cdot P$ uncovered links. By using similar arguments we get that $H(S_2) \leq \frac{E(OPT)}{N-i}$.

Since L_i is the most cost effective subset of $U \setminus Z$ with P links, it holds that its cost is upper bounded by the cost of any subset of P links of $S_1 \cup S_2$. Thus, from Claims 1 and 2 we get that $H(L_i) \leq H(S_1) + H(S_2)$. By applying the upper bounds on the cost of S_1 and S_2 we get the inequality of Claim 5 and this completes the proof. □

We now use Claim 5 to prove Theorem 5.4.

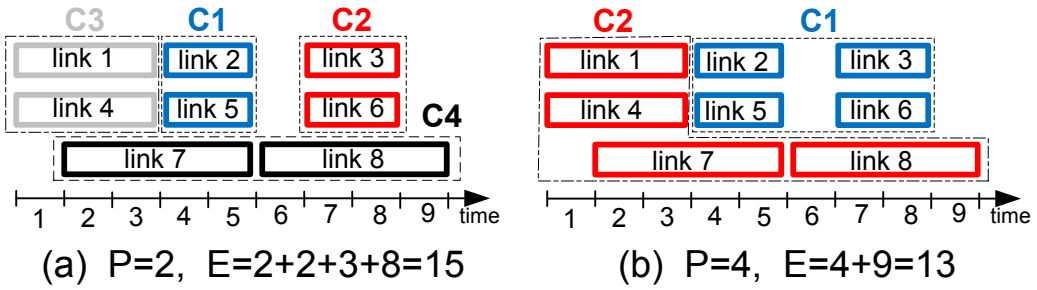

Figure 5.6: The calculated solutions by the Disjoint Set Cover algorithm when $P = 2$ and $P = 4$.

Proof of Theorem 5.4. By applying Lemma 5 on the $N-1$ first calculated subsets L_i, and the fact that $H(L_N) \leq E(OPT)$, we get the following inequality.

$$
\begin{aligned}
E(DSC) &= \sum_{i=1}^{N} H(L_i) \\
&\leq \sum_{i=1}^{N-1} \left(\frac{1}{N-(i-1)} + \frac{1}{N-i}\right) \cdot E(OPT) + E(OPT) \\
&\leq 2 \cdot \sum_{i=1}^{N} \frac{1}{i} \cdot E(OPT) \\
&\leq 2 \cdot \ln N \cdot E(OPT)
\end{aligned}
$$

The third inequality results from rearranging the components and getting the N-th Harmonic sequence. This completes our proof. □

Example 12. We apply the Disjoint Set Cover algorithm to solve the instances of Example 9. The results are presented in Figure 5.6. The order of the subsets chosen by the algorithm is reflected by the indices of the chassis in the figure. When $P = 2$, the first subset chosen is the set containing links l_2 and l_5, because they have the least total amount of energy cost among all 2-combinations of $\{l_1, \ldots, l_8\}$. In the remaining links, the second subset of $\{l_3, l_6\}$ is chosen based on the same reason. Then $\{l_1, l_4\}$ is chosen in the

third round and $\{l_7, l_8\}$ is chosen at last. Consequently, the energy cost of the solution for case $P = 2$ is 15. We get the assignment for the case $P = 4$ using Disjoint Set Cover in Figure 5.6(b). The energy cost for this case is 13.

Note that in the DSC algorithm, we have to enumerate all possible P-combinations of links in the worst case, which will take up to $O(M^P)$ time. While this calculation may be a time consuming task it is substantially shorter than check all possible solutions which is $O(M!)$. In our simulations we were able to calculate solutions for systems with $M = 96$ links and $P = 16$ ports per chassis within several minutes using a personal computer. In the following section, we will give a simple heuristic algorithm which works more efficiently and gives satisfactory performance as well.

5.5.4 The Largest Overlap First algorithm

The greedy algorithm introduced in this section implements an intuitive approach, inspired by the Port-Sorting algorithm in [91]. In this algorithm, we fill up one chassis at a time using un-assigned links. Whenever there is any extra port on a chassis, we add an un-assigned link to it, which has the largest active-time overlap with the links already on this chassis. By adding such a link, intuitively, we save as much energy as possible. We refer this algorithm as Largest Overlap First (LOF), described in Algorithm 22.

Algorithm 22 Largest Overlap First

```
1: Input: traffic demand matrix F_[M,T] of M links.
2: All chassis are empty and all links are un-assigned.
3: while there is an empty chassis C do
4:    pick an un-assigned link with the largest number of active time slots and assign it to
      C;
5:    while chassis C has available ports do
6:       assign the link with the largest overlap with links on C to chassis C. {Ties are
         broken in favor of links with the least energy increment on the current chassis.}
7:    end while
8: end while
9: return the assignment.
```

Example 13. We still use Example 9 to explain our algorithm. For the case $P = 2$, the Largest Overlap First algorithm chooses link l_7 which has the longest active duration for the

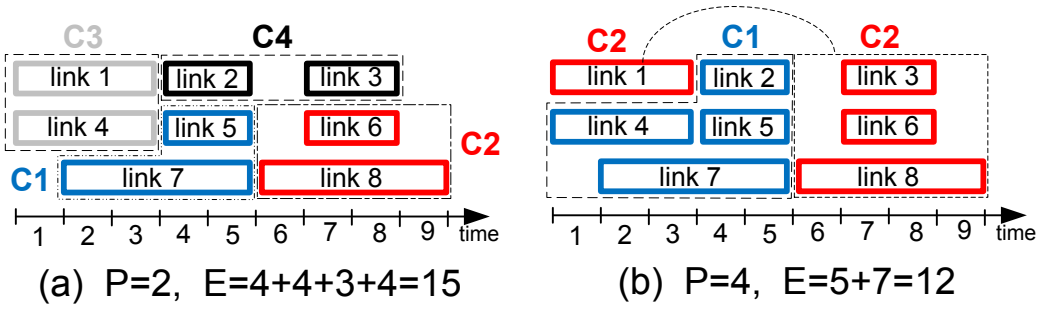

Figure 5.7: The calculated solutions by the Largest Overlap First algorithm when $P = 2$ and $P = 4$.

first empty chassis $C1$ in Figure 5.7(a). For the remaining port of $C1$, the algorithm picks link l_5 which has the largest overlap with l_7 and, at the same time, has the least energy increment to $C1$. Then we consider the second empty chassis. However, for the case $P = 4$, the algorithm continues seeking for the next link for chassis $C1$. In this way, links l_2 and l_4 are picked in order as shown in Figure 5.7(b). Given the assignments, the energy costs is 15 for the case $P = 2$ and is 12 for the case $P = 4$.

5.5.5 Optimality under correlated traffic

Theorem 5.5. *Iterative Matching, Disjoint Set Cover, and Largest Overlap First are optimal algorithms when the given traffic are fully correlated.*

Proof. From previous work [91], when the traffic are correlated, then the optimal solution has the following property. If we sort all M links in an increasing order of their total active time slots as $L = \{l_1, \ldots, l_M\}$, and we sort all N chassis in the optimal solution in an increasing order of their respective energy costs as $\{C_1, \ldots, C_N\}$, then the i-th chassis C_i will include the i-th groups of P-links $\{l_{(i-1)\cdot P+1}, \ldots, l_{(i-1)\cdot P+P}\}$. Therefore, to prove that our algorithms give the optimal solution when the traffic are fully correlated, it is enough to prove that our algorithms maintain this property in their solutions.

Iterative Matching. Given a sorted list of links L as mentioned above, then matching every consecutive 2 links will give us an optimal solution for $P = 2$. If this matching is not the same as what Iterative Matching gives us at the first round, we claim that we can

always swap links to make them the same without increasing the total energy cost. This is due to the nesting property of correlated traffic. After each iteration of matching, we combine all links within the same chassis as a super-link (which should be the same as the longest active link in that chassis). These super-links will still be in increasing order of the number of active time slots. We consider the next step of matching as a matching of these super-links, then consecutive 2 super-links are matched due to the same reason above. Then at the end of the algorithm, we will have the first P links in L assigned to the first chassis, the second P links to the second chassis, and so on, where the optimal solution's property is maintained.

Disjoint Set Cover. Given the sorted list of links L as above, we claim that the Disjoint Set Cover will pick the first P links as the first set. Due to the nesting property, any chassis' energy cost will be the same as the longest active link's energy cost within that chassis. So the first P links will give us the smallest total energy cost among all possible P-combinations, and this is exactly what Disjoint Set Cover looks for. With the same reason, we can see that Disjoint Set Cover will continuously pick consecutive P links from the list in order to make a smallest energy cost chassis based on the remaining links. In this way, we proved that Disjoint Set Cover maintains the same property as the optimal solution.

Largest Overlap First. The proof for Largest Overlap First is quite similar to the proof for Disjoint Set Cover instead that we are looking at the links in a reverse order. Given the sorted list of links L, the algorithm will first pick the last link l_M which has the largest number of active time slots for the first chassis. Due to the nesting property, the link with the largest overlap with l_M will be l_{M-1} and so on. So for the first chassis, we assign the last P links in list L to it. Then we iteratively do this by assigning the last consecutive P links to the next chassis. At last, our solution maintains the optimal solution's property. □

5.5.6 Extensions – Various Link Types and Network Reliability

Various Link Types: In a real multi-chassis router, each chassis may contain different types of line cards, which in turn, support different link capacities. For example, one kind

of line card may have only a single port that supports a 100Gbps link while another type of line card may have 10 ports, each of which supports traffic up to 10Gbps. Here, we consider the case in which each line card supports links of the same type, and the overall capacity of each line card is the same. Our algorithms can be easily extended to this case as below:
Step 1. For each supported link type, we separately assign links of that type to the corresponding line cards by using one of the above algorithms.
Step 2. Now we have a bunch of line cards with links assigned to them, and we consider each line card as a new logical link. Thus we have a set of logical links with equal amount of capacities. Then we do the assignment of line cards to chassis again using the algorithms in this chapter. Similar methods can be used to analyze the algorithms' performance.

The problem remains open when the chassis or line cards are not identical in terms of supporting traffic capacities.

Network Reliability: One of the key concerns of ISPs is preserving the service under component failures. In our scheme, the fault resiliency can be provided in one of two ways:

1. Traffic protection at the network level by provisioning alternative routes.
2. Augmenting each PoP with redundant chassis that are activated after a line or a chassis failure.

For the latter, we observed that our scheme is more efficient than other PoP connection solutions, e.g., the link aggregation technique [83], since it spreads the links between the considered PoP and each one of the ANs on multiple chassis. More specifically, let b_i^{Alg} denote the maximal number of links between any chassis in PoP A and AN $i \in R_A$ according to a link assignment provided by any algorithm Alg. The number of additional chassis that are required to provide protection against any possible single chassis failure is $B^{Alg} = \frac{1}{P} \cdot \sum_{i \in R_A} b_i^{Alg}$. Since our scheme connects each AN i's links to multiple chassis, the values of b_i^{IM} and b_i^{DCS} are small. Therefore, the number of required redundant chassis B^{IM} and B^{DSC} is also small.

5.6 Simulation Results

5.6.1 Simulation Setup

We consider a single PoP, say PoP A, for energy savings. This PoP connects to 8 adjacent regional networks (RNs) and 4 adjacent PoPs. These 12 ANs (8 RNs + 4 PoPs) are the sources and destinations of all generated traffic in our setup, all of which is assumed to transmit through PoP A. There are 6 links associated with each RN and 12 links associate with each adjacent PoP, so there are total $M = 96$ links ($6 \times 8 + 12 \times 4$) in our setting.

There are N identical chassis in A, each having P ports. In our simulations, we choose $P \in \{2, 4, 6, 8, 12, 16\}$. Since we have assumed $M = N \cdot P$, the corresponding N are chosen from $\{48, 24, 16, 12, 8, 6\}$.

The traffic over PoP A is generated by Equation (5.2). As [113] segments the traffic into a predictable component and a stochastic component, we use a sinusoidal function as the correlated component $h(t)$ and a random function uniformly distributed in $[0, 1]$ as the uncorrelated component $g_i(t)$. We control the functions to let links have average utilizations among $35 - 40\%$. We consider correlation parameter $\alpha \in \{1, 0.75, 0.5, 0.25\}$

Note that the traffic over ANs is uncorrelated when $\alpha = 1$ and becomes increasingly correlated when α decreases. We consider a time range of $T = [1, 72]$, which is equivalent to sampling traffic every 20 minutes within 24 hours. Therefore, one traffic instance is an 96×72 ($M \times T$) matrix. To obtain an average performance for each algorithm, we generate 10 traffic instances for each α.

5.6.2 Candidate algorithms and the metric

We implement the following algorithms in our simulations.

1. Algorithm proposed in this chapter: **Iterative Matching**, **Disjoint Set Cover**, **Largest Overlap First**.

2. An algorithm from previous work [91]: **Port Sorting**.

3. Three simple heuristic algorithms:

   - **Least Increasing First.** This algorithm is inspired by constructing a chassis with the smallest energy cost in a heuristic way. In this algorithm, we consider one chassis at a time. For each chassis, we pick a link with the shortest active time at first, then we fill this chassis by adding links which yield the least increase of energy cost.
   - **Aggregate.** This algorithm is also referred as bundled link technique [83]. It considers physical links from the same AN as a single logical link and assigns them onto consecutive chassis.
   - **Random.** A naive method just assigns each link to an arbitrary chassis.

In our simulations, we consider only the chassis whose energy consumption is under the algorithms' control, as described in Section 5.4. We observed that the power consumptions of the components we considered here will contribute up to $70 - 80\%$ of a PoP's power consumption [107]. Therefore, the total energy cost of the PoP could be obtained from our results easily. We assume that each chassis consumes one unit of energy for each time slot. We set the maximum power consumption for A as $N \cdot T$. This is the energy cost of "**always-on scheme**", in which each chassis within A is always active.

Given a pair of α and P, we run each candidate algorithm and record the average energy cost over 10 traffic instances. The difference between the maximum power consumption $N \cdot T$ and the average energy cost of a certain algorithm is the average energy saving achieved by this algorithm. The metric we use to compare candidate algorithms is the *average percentage of energy saving*. This value is the ratio between the average energy saving and the maximum power consumption.

5.6.3 Results

In Figure 5.8, we show the average percentage of energy saving of our candidate algorithms as a function of a parameter P, for the traffic instances of correlation factor $\alpha = 1$ (uncorrelated traffic). In a similar manner, Figure 5.9, Figure 5.10 and Figure 5.12 show the average percentage of energy saving of our algorithms when taking traffic instances with correlation factor α as 0.75, 0.5 and 0.25 respectively. Figure 5.11 and Figure 5.13 are actually the zoom images of the upper parts of Figure 5.10 and Figure 5.12 respectively, which contains the performance of those 5 algorithms.

The energy savings of algorithms

1. For all different values of α and P, algorithms *Iterative Matching* and *Disjoint Set Cover* achieve $20-60\%$ energy savings on the line-card chassis. They significantly outperform all other candidate solutions. Among all benchmark algorithms, we observe that Port Sorting has the best performance while Aggregate and Random perform quite poorly. So we compare our algorithms with Port Sorting in terms of average percentage of energy savings. When the traffic is uncorrelated ($\alpha = 1$), our algorithms achieve almost twice as much savings as that of Port Sorting when $P \geq 6$. For instance, when $P = 16$, Disjoint Set Cover has 20.6% energy savings while Port Sorting has only 10.3% energy savings. The difference becomes smaller as the correlation of the traffic increases (α decreases). This is not surprising, because all our algorithms along with Port Sorting converge to the optimal solution when the traffic is fully correlated ($\alpha = 0$).

2. For all values of α, *Iterative Matching* performs better than *Disjoint Set Cover* when P is relatively small (≤ 6). In fact, it is optimal when $P = 2$. As P goes large, *Disjoint Set Cover* outperforms *Iterative Matching* gradually. This is consistent with their proved approximation ratios.

3. The Largest Overlap First algorithm has comparable performance with Iterative

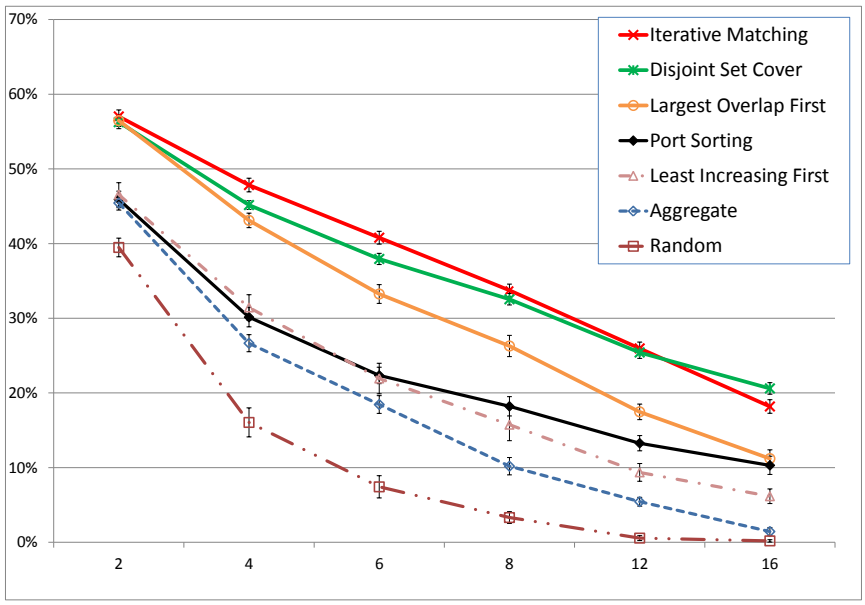

Figure 5.8: Average percentage of energy savings of candidate algorithms for the traffic of $\alpha = 1$.

Matching and Disjoint Set Cover when P is relatively small (< 6), and converges to Port Sorting when P becomes larger. This provides us an efficient solution of satisfactory performance in the cases where chassis does not have many ports. Surprisingly, Least Increasing First scheme has quite poor performance, which is worse than Port Sorting.

For each pair of value P and α, we also calculate the two-sided confidence interval of the average performance of each algorithm at confidence level 95%. These confidence intervals are shown in the above figures using error bars. The results show that for nearly two thirds of the cases, the maximum errors are less than 1%, and for almost all cases, the maximum errors are within 2%.

The Sensitivity of algorithms to traffic estimation error

To show that the performance of our algorithms is not sensitive to traffic estimation error, we add a Gaussian noise with zero-mean and different standard deviation σ to the input

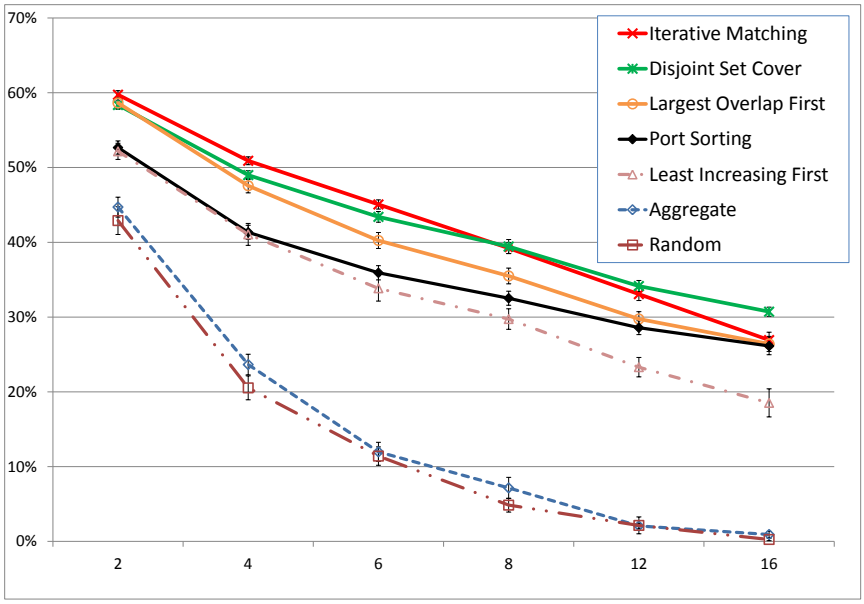

Figure 5.9: Average percentage of energy savings of candidate algorithms for the traffic of $\alpha = 0.75$.

traffic. For these unmatched traffic, we obtain the power consumption of the assignments derived from the algorithms using the original traffic ($\sigma = 0$).

Firstly, our algorithms take the original traffic and generate assignments. Secondly, for each value of $\sigma \in \{0.1, 0.2, \ldots, 1\}$, we generate 10 instances of estimation error and add them to the original traffic. At last, we evaluate each algorithm for the assignment obtained in the first step and the traffic with estimation error generated in the second step. Again, we use the percentage of energy saving as our metric. As an example, we show the results for $\alpha = \{1, 0.75, 0.5, 0.25\}$ and $P = 8$ in Figure 5.14, Figure 5.15, Figure 5.16 and Figure 5.17 respectively.

From the figures, we see that the performance of our algorithms are relatively stable for traffic with estimation error. The relative performance of each algorithm follows the same pattern as described in the previous section. Our algorithms still outperform other candidates in a same manner.

We note that the estimation error has stronger impact on the algorithms' performance

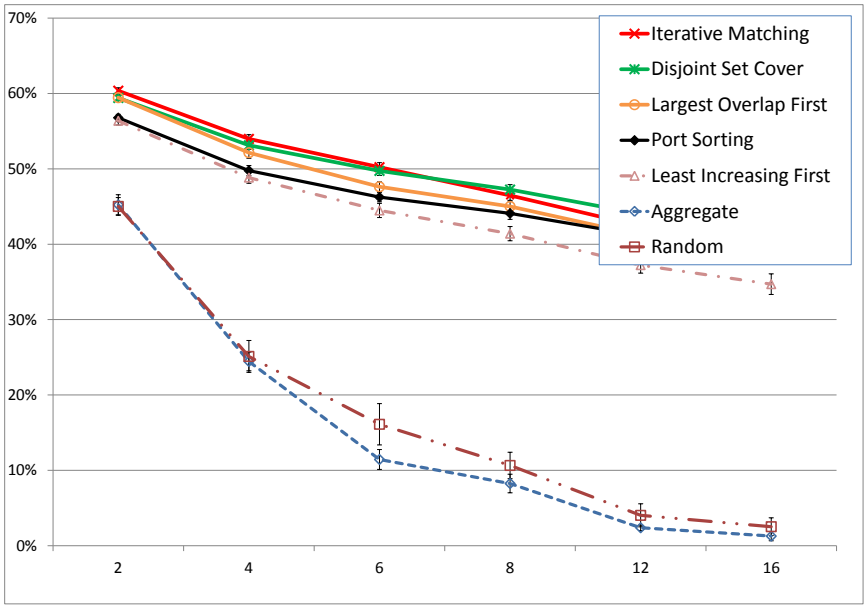

Figure 5.10: Average percentage of energy savings of candidate algorithms for the traffic of $\alpha = 0.5$.

when the original traffic is more correlated ($\alpha = 0.25$).

This is because our algorithms actually benefit from the correlation of the traffic. By adding estimation error to such traffic, the correlation is weakened and the traffic demands are more like uncorrelated ones.

Two sided confidence intervals with confidence level 95% are also calculated. Since all maximum errors are within 1%, the error bars are barely noticeable in the figures.

5.7 Conclusions

We consider the problem of minimizing the power consumption of Internet Points of Presence (PoPs) given the general traffic demands. We show that this problem is NP-complete when number of ports on each chassis exceeds 2. We then provide an optimal Matching algorithm for an important variant where the number of ports on each chassis is 2. For the general case, we design several approximation algorithms and establish their approximation

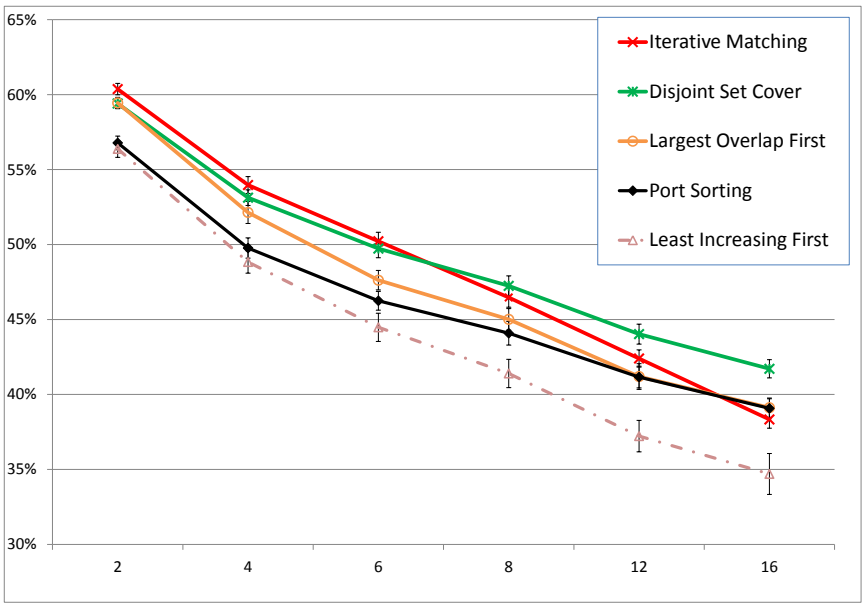

Figure 5.11: Zoom on the average percentage of energy savings with error bars for the traffic of $\alpha = 0.5$.

guarantees. A simple greedy algorithm is given for its time efficiency in finding a satisfactory solution in most cases. We also prove that all our algorithms are optimal when traffic is correlated.

Finally, our extensive simulations demonstrate significant advantages of our algorithms over other candidate solutions. Our proposals are particularly attractive when traffic is uncorrelated, where they outperform other algorithms substantially. For all cases, we observe that the naive algorithms do not provide satisfactory performance. Furthermore, we show that our algorithms have stable performance under traffic with noise, which makes them practical in real applications.

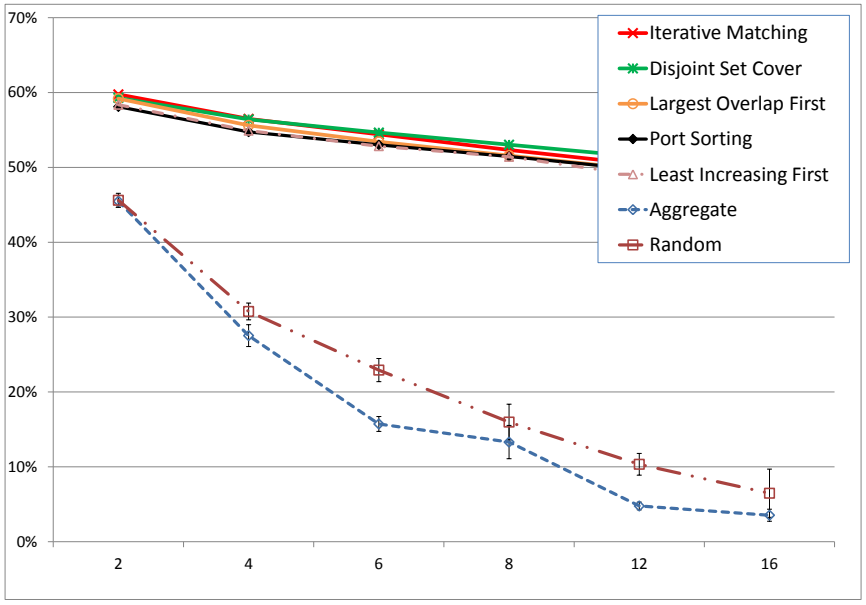

Figure 5.12: Average percentage of energy savings of candidate algorithms for the traffic of $\alpha = 0.25$.

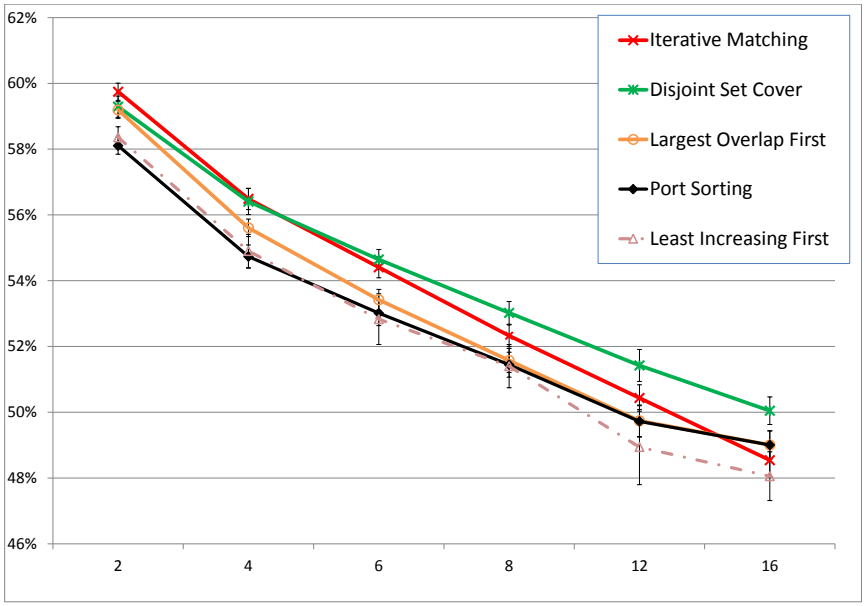

Figure 5.13: Zoom on the average percentage of energy savings of 5 algorithms for the traffic of $\alpha = 0.25$.

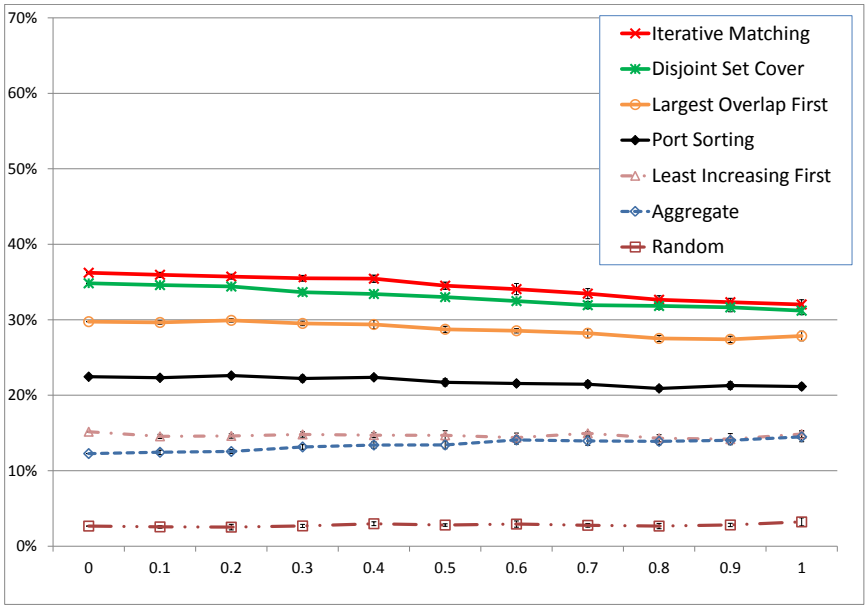

Figure 5.14: Average percentage of energy savings under traffic with noise when $\alpha = 1$ and $P = 8$.

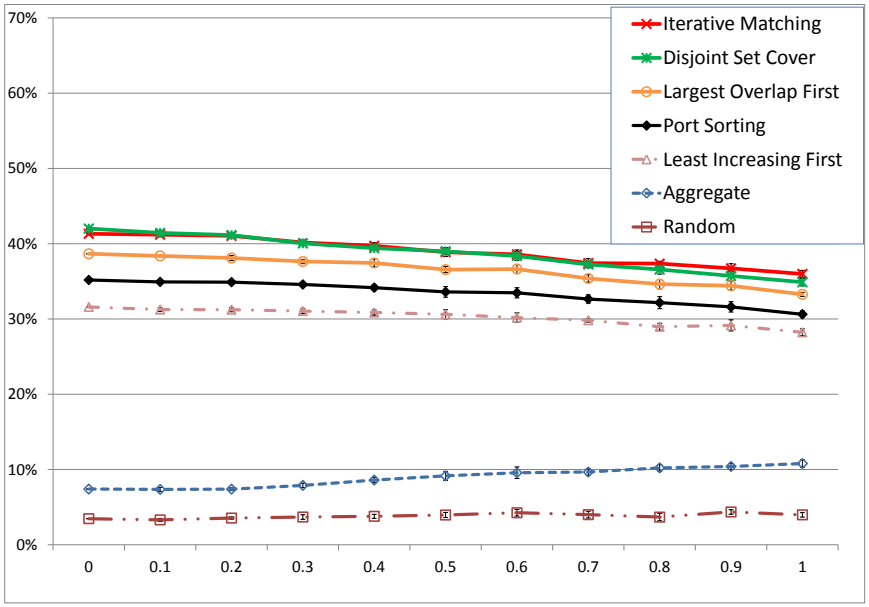

Figure 5.15: Average percentage of energy savings under traffic with noise when $\alpha = 0.75$ and $P = 8$.

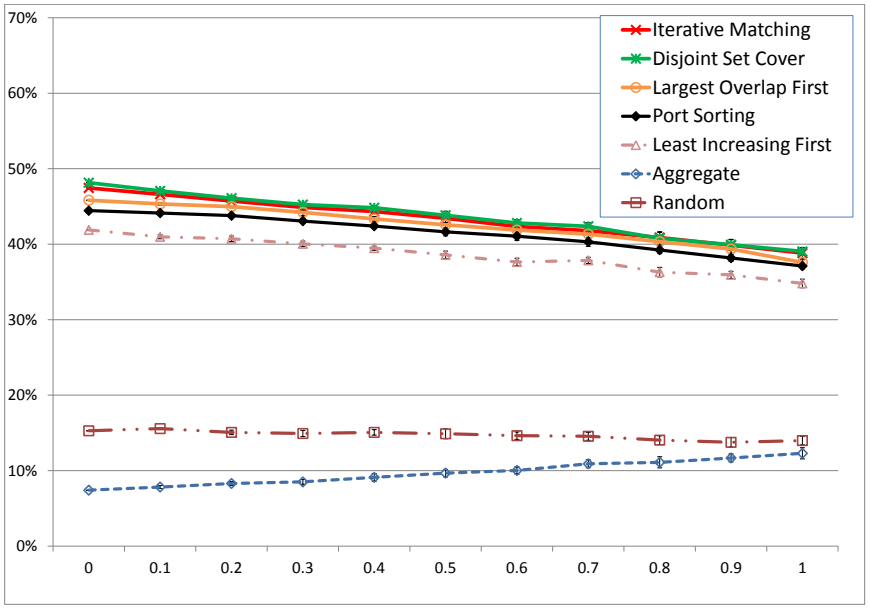

Figure 5.16: Average percentage of energy savings under traffic with noise when $\alpha = 0.5$ and $P = 8$.

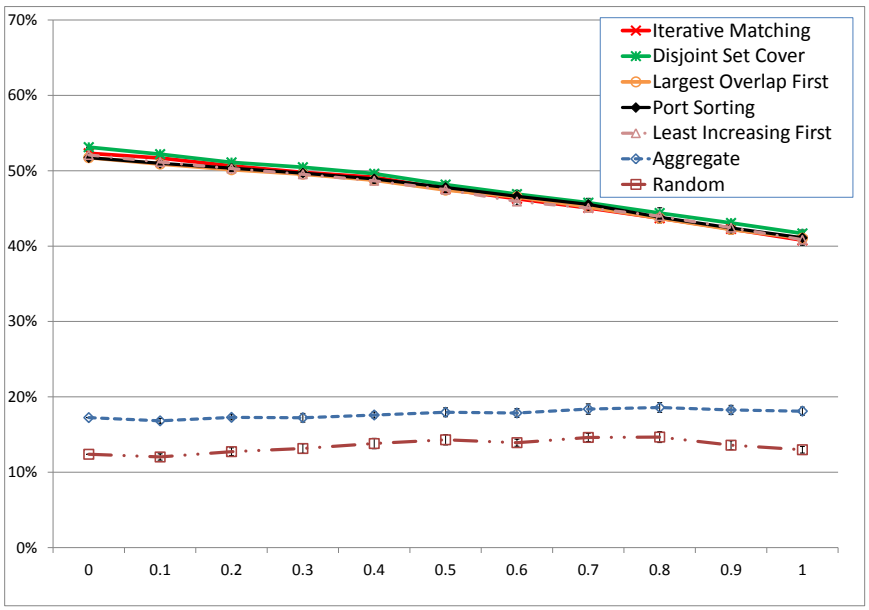

Figure 5.17: Average percentage of energy savings under traffic with noise when $\alpha = 0.25$ and $P = 8$.

Appendix A: Important Definitions

We list several important definitions that frequently appear in the dissertation.

Definition A.1 (Overloaded System)**.** *Given an input instance $\mathcal{I}$, a system is called* overloaded *if there does not exist a schedule to complete all jobs in $\mathcal{I}$ before their deadlines.*

Definition A.2 (Underloaded System)**.** *Given an input instance $\mathcal{I}$, a system is called* underloaded *if there exists a schedule to complete all jobs in $\mathcal{I}$ before their deadlines.*

Definition A.3 (Non-preemption Setting)**.** *In the* non-preemption setting*, a packet/job that is being sent should not be preempted before it is finished.*

Definition A.4 (Preemption Setting)**.** *In the* preemption setting*, an online algorithm is allowed to abort a packet/job during its transmission, and the aborted packet can be resumed and completed later.*

Definition A.5 (Preemption-Restart Setting)**.** *In the* preemption-restart setting*, an online algorithm is allowed to abort a packet/job during its transmission, and the aborted packet can be restarted (from scratch) and completed later.*

Definition A.6 (Agreeable Deadline [27])**.** *If for any two jobs i and j, $r_i < r_j$ implies $d_i \leq d_j$, then we say that they have* agreeable deadlines*.*

Appendix B: Simulation of Online Learning Algorithms

We develop a prototype to simulate the performance of online learning algorithms.

There is a packet generator. This generator generates packets with release time, deadlines and values. In total, there are 2000 packets generated. The range of the release time is $[0, 999]$. The range of the deadlines is $[0, 999]$. The range of the packet values is $[0, 100]$. The maximum and the minimum packet values ($v_{\max}$ and $v_{\min}$) are not specified by us, but the ones generated by the procedure. We generate all the parameters in a uniformly randomized manner. In our simulation, $v_{\max} = 99.951$ and $v_{\min} = 0.009$. For the current version of the simulation, the channel state is fixed. We design the offline algorithm as follows. We assume the channel reliability is always perfect. We implement all the online learning algorithms mentioned in Section 2.4.3 of Chapter 2. Actually, they are EBAV and EBRV. In geometric rounding, we set $\delta = 0.5$. Thus, in total, there are 23 experts $(0.009(1 + 0.5)^{23} \approx 99.951)$.

In our simulated results, the best expert obtains a total gain of 72096.615. It sends 1000 packets and it is the $(j = 2)$-th expert. That is, in each time step, the expert chooses a packet with a value $\geq \frac{v_{\max}}{(1+\delta)^2 v_{\min}}$ to send. The worst expert yields a total gain of 99.951 with only 1 packet scheduled — the most valuable packet. The expert who sets the threshold as the minimum value packet yields a total gain of 56711.178 and 1000 packets are scheduled successfully. The online learning algorithm who follows the best expert so far gains a profit as 72050.509 with 999 packets scheduled. Thus, the regret of online learning algorithm is bounded by 46 and it performs very close to the optimal expert.

Figure B.1 shows the simulated results we have. In Figure B.1, the dotted line represents the performance of our online learning algorithm. In this figure, we take experts with index j of $0, 1, 2, 3, 4, 5, 10, 20$ among all 23 experts such that the remaining experts' performance is bounded by these given ones.

We also develop the prototype to simulate the performance of the optimal offline algorithm as well as two competitive online algorithms: SEMI-GREEDY and EDF_β. The

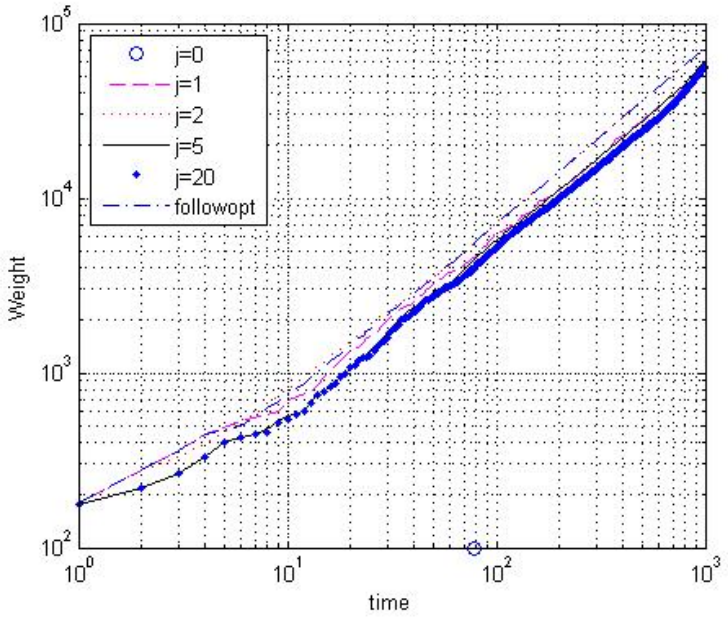

Figure B.1: Simulated results of the online learning algorithms.

simulation is run on the same data set used above. In the simulated results, the optimal offline algorithm has a total gain of 73619.174. It sends 1000 packets. Since we consider the perfect channel state, SEMI-GREEDY works simply the same as greedy algorithm EDF, and the algorithm yields a near optimal result 73330.578. EDF_β earns a total gain of 72145.685, where $\beta = 2$. We notice that both online algorithms' performances are very close to the optimal offline solution.

Figure B.2 shows the simulated results we have. In Figure B.2, the black line represents the performance of our optimal offline algorithm, the green line represents the performance of SEMI-GREEDY and the red line represents the performance of EDF_β.

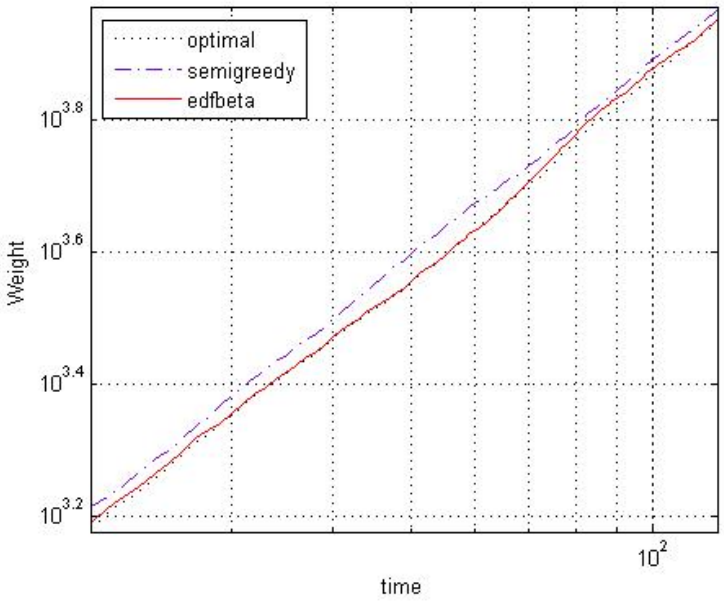

Figure B.2: Simulated results of the optimal offline algorithm and online algorithms.

Appendix C: Scheduling Unit-length Packets with Soft Deadlines

C.1 Motivation

We consider video-on-demand systems as a motivating example. In a timely manner, images and video frames are delivered from a server to a client across the network. Under resource-limited scenarios, not all packets queued in the network switches can reach the client on time. Some packets have to be delayed due to packet overflow. However, video frames, even arriving beyond some timeline, may still contribute some quality-of-service (QoS) for the client. Another motivating example comes from a kind of modern computing platforms, like VNC [114] and other pixel-based remote computing [115–117]. The screen updates are encoded at the server side and the client side decodes it for recovering computed results. With various coding methods, the temporal redundancy of the screen updates allows us dropping some packets. How to select packets to be delivered and dropped, with a progressive QoS, is a very challenging task for clients. In this chapter, we model these applications and consider scheduling *delay-bounded* packets with soft deadlines.

With widely foreseen next-generation networks providing guaranteed differentiated service, most researchers are working on online algorithms for scheduling packets with hard deadlines under various settings and constraints [27–31, 35]. In this chapter, we consider scheduling unit-length packets with soft deadlines such that a packet may contribute different values when it is sent at different time. The model generalizes the one discussed in [30] and [31].

C.2 Model Description

Time is discrete. There is a buffer. All packets in the buffer are *available/pending* for delivery. There is one output link, being able to send one pending packet from the buffer per

unit time. Packets are released over non-trivial integer time. With *soft deadline constraints*, each packet j is released at $r_j \in \mathbb{Z}^+$ and it may have different weights in different time slots from r_j onwards. A packet j has a *weight function* $w_j(t) \in \mathbb{R}^+$ at time t: If j is sent at time t, it contributes weight $w_j(t)$ to the objective function. A packet j's *slack* time is defined as $s_j = d_j - r_j$, where for any $t \geq d_j$, $w_j(t) = 0$. Our objective is to maximize the total weight of the packets delivered.

There is no restriction on the sequence of sending packets. Considering the motivating applications, we study the following two models.

- *General model*. The slack s_j can be arbitrarily large.
- *B-bounded model*: Considering the critical requirements on server response time, we let $s_j \leq s$ for all packets j, where s is the maximal server response time. With $s \leq B$, the buffer has only s positions for buffering packets.

We list three general weight functions for packets with soft deadlines in the B-bounded model. Each packet j contributes a value $w_j(t)$ if being sent at time t ($t \leq r_j + B$). The range $[r_j, r_j + B]$ is called *weight range* of a packet j. Outside of this range, packet j has zero weight. Within its weight range, a packet can have its weight functions defined as follows.

Definition C.1 (Linear Decreasing Weighted)**.** *A packet j is said to be* linear decreasing weighted *if there exists some $\alpha_j > 0$ such that*

$$w_j(t) = \max\{w_j - \alpha_j(t - r_j), 0\},\ \alpha_j > 0.$$

Definition C.2 (Monotonic Decreasing Weighted)**.** *A packet j is said to be* monotonic decreasing weighted *if for any $r_j \leq t_1 \leq t_2 \leq r_j + B$, there exists*

$$w_j(t_1) \geq w_j(t_2).$$

Definition C.3 (Periodic Weighted.)**.** *A packet j is said to be* periodic weighted *if its weight function is periodic within its* periodic range *($R_j \leq B$). For any t, there exists an integer A, such that*

$$w_j(t) = w_j(t - A \cdot R_j),$$

where $A \cdot R_j \leq t < (A+1) \cdot R_j$.

C.3 Offline Optimal Algorithm for the B-bounded Model

The following terms are used in the optimal offline algorithmic description. Note that the algorithm calculating the optimal offline algorithm for the general model is a subroutine of the algorithm for the B-bounded model.

- B: the output buffer size. In the B-bounded model, every packet has its slack no more than B.
- OPT: for the B-bounded model, an optimal offline algorithm with the constraint of buffer size B;
- $\mathbf{G}(t)$: the group of packets released at time t. $\mathbf{G}(t) = \{j | r_j = t\}$;
- $w_j(t)$: the weight of a packet j at time t.

Assume there are k distinct release time: $T_1, \cdots, T_k$, then the groups of packets released are $\mathbf{G}(T_1), \cdots, \mathbf{G}(T_k)$, respectively.

OPT has the property that it can be calculated in an offline manner to get the set of packets to be delivered. Let OPT have a large enough buffer used to calculate the set of packets. Procedure OPT-SET calculates the set of packets with the maximal total weights among all possible selections from the whole set of packets released.

Let $\mathbf{S}^O$ denote the set of packets selected by OPT-SET. In procedure OPT-SET, it assumes that no buffer constraint is applied. Therefore, we have the following lemma.

Algorithm 23 OPT-SET

Let $\mathbf{O}_L$ be a set containing all the packets released. $\mathbf{O}_L = \bigcup \mathbf{G}(t)$. $|\mathbf{O}_L| = n$;
Let $\mathbf{O}_R$ denote set of no more than $k \cdot n$ time slots, containing possible time steps in which OPT sends its packets.
Construct a bipartite graph $G = (V, E)$, using $e(i, j) \in E$ to represent the weight of the edge (i, j). Edge (i, j) connects node i in the node set at the left hand side and node j in the node set at the right hand side. The left side is set $\mathbf{O}_L$ and the right side is set $\mathbf{O}_R$. $\mathbf{O}_L \cup \mathbf{O}_R = V$.
for $j \in \mathbf{O}_L$ and $i \in \mathbf{O}_R$ **do**
    if $i \geq r_j$ **then**
        set $e(j, i) = w_j(i)$.
    end if
end for
Get the maximum weighted matching via the Hungarian algorithm.
Every packet j matched is allocated into its matched time slot t_j.

Lemma 10. *For any instance with soft deadlines, OPT-SET selects the set of packets and their matched time slots in which they are sent, achieving the upper bound of the maximum of total weights gained. OPT-SET has its running time complexity of $O(n^3)$, where n is the number of packets released.*

We devise an optimal offline algorithm, called OPT. During the course of OPT, at any time, only the packets in $\mathbf{S}^O$ are to be kept in the buffer, where the buffer size is B, and at time t_i, packet i is sent. (Assume that (i, t_i) is a matched pair.)

Algorithm 24 OPT

Run OPT-SET to get $\mathbf{S}^O$ and any packet $j \in \mathbf{S}^O$ has its matched time slot t_j;
Sort all matched packets in decreasing matched-value order, that is, in decreasing order of $w_j(t)$ for $j \in \mathbf{S}^O$.
Initialize $O = \emptyset$ and insert packets in $\mathbf{S}^O$ in decreasing matched-value order.
while $\mathbf{S}^O \neq \emptyset$ **do**
    if packet overflows upon inserting j **then**
        drop j;
    end if
    remove j from $\mathbf{S}^O$.
end while
Send the packets in O in those time steps as what OPT-SET does in $\mathbf{S}^O$.

Based on the matroid property of the model, we have the following result.

Theorem C.1. *In the B-bounded model, for any instance with soft deadlines, OPT gets the optimal solution, achieving the maximum weight gained in an offline manner. OPT is*

a polynomial-time algorithm.

C.4 Online Algorithms for the B-bounded Model

In this section, we design two simple online algorithms. One is called MATCH, which is motivated by the optimal offline algorithm for the B-bounded model. The other one is called GREED, which uses the greedy method. Let $\mathbf{B}^A(t)$ denote the set of packets in the buffer at time t for an algorithm A, before the set of packets $\mathbf{G}(t^+)$ are released ($t^+ > t$).

C.4.1 The algorithm MATCH

Algorithm MATCH runs MATCH(t) at every time t. Motivated by the offline OPT algorithm for the B-bounded model, at any time t, MATCH arranges all packets available to the output buffer through getting a maximal weighted matching. Similar to OPT-SET, in algorithm MATCH(t), we construct a bipartite graph $G = (V, E)$, using $e(i, j) \in E$ to represent the weight of the edge (i, j). Edge (i, j) connects node i in the node set at the left hand side and node j in the node set at the right hand side. The left side is set $\mathbf{S}_L$ and the right side is set $\mathbf{S}_R$. $\mathbf{S}_L \cup \mathbf{S}_R = V$. Only matched packets are kept in the buffer and we drop all unmatched packets.

Algorithm 25 MATCH(t)

```
1: S_L = B^M(t) ∪ G(t).
2: S_R = [1, ..., B].
3: for J_j ∈ S_L do
4:    for i = 1 to B do
5:       e(J_j, i) = w_j(t + i).
6:    end for
7: end for
8: Get a maximum weighted matching using the Hungarian algorithm.
9: Drop all unmatched packets.
10: Send the packet matched with the time slot t.
```

C.4.2 The algorithm GREED

Algorithm GREED runs GREED(t) at every time t. In each time slot, GREED selects and arranges the packet available with the maximal weight at that time slot. In identifying the optimal set of packets in the buffer at any time, we also employ the Hungarian algorithm.

We also use 2 sets to represent the packets available and the time slots in algorithm GREED(t).

- $\mathbf{X}_L$: $\mathbf{X}_L$ is the packet set $\mathbf{B}^G(t) \cup \mathbf{G}(t)$, which includes all available packets at time t. Index all available packets such that $\mathbf{X}_L = \{J_1, \cdots, J_l\}$, $l = |\mathbf{B}^G(t) \cup \mathbf{G}(t)|$. The input buffer contains all packets in $\mathbf{G}(t)$ and should be vacated for later arrival packets after time t.

- $\mathbf{X}_R$: let the time slots (positions) in the output buffer are indexed from 1 to B, where the packet in the first time slot is the one being sent out if all packets are delivered sequentially. $\mathbf{X}_R$ is a set of B time slots.

Algorithm 26 GREED(t)

```
1: if G(t) ≠ ∅ then
2:   X_L = B^G(t) ∪ G(t).
3:   for i = 1 to max{B, |X_L|} do
4:     for J_j ∈ X_L do
5:       e(J_j, i) = w_j(t + i);
6:       allocate packet J_j with max{e(J_j, i)} in position i;
7:       remove J_j from X_L.
8:     end for
9:   end for
10: end if
11: Drop unselected packets.
12: Send the packet in time slot t.
```

C.5 Analysis

In this section, we consider competitive ratios of MATCH and GREED for the B-bounded model. We analyze their performance in scheduling the packets with some specific weight functions.

C.5.1 Analysis of MATCH

Theorem C.2. *The running time complexity of MATCH is $O(n^3)$.*

Proof. For the online algorithm MATCH, at any time t, let $n_t = |\mathbf{G}(t) \cup \mathbf{B}^M(t)| \leq |\mathbf{G}(t)| + B$. n_t is the number of the packets available. The Hungarian algorithm is running on a bipartite graph with $2n_t$ nodes to find out the maximal weighted matching with running time complexity of $O(n_t^3)$. With $2n_t$ assignment operations, the running time complexity of MATCH(t) is $O(n_t^3)$. The total running time complexity of MATCH is at most $O(\sum(\mathbf{G}(t)+B)^3) = O((\sum \mathbf{G}(t))^3) = O(n^3)$. Theorem C.2 is proved. □

To our surprise, the idea leading to an optimal office algorithm does not have a constant competitive ratio in an online setting.

Theorem C.3. *For MATCH, there is no constant competitive ratio for linear decreasing weighted packet scheduling, periodic weighted packet scheduling, or monotonic decreasing weighted packet scheduling.*

Proof. Table C.1 provides one instance of monotonic decreasing weighted packets and one instance of periodic weighted packets to prove Theorem C.3.

Table C.1: Instances showing MATCH over 2 kinds of weight functions.

weight function	MATCH	ADV	ratio
linear decreasing	L_2, L_1	L_1, L_2 L_1, L_2 are released in $t+1$	$\frac{\rho^{2t}}{\rho^{2t-1}} = \rho$
periodic ($R_j \leq B$)	P_2, P_1	P_1, P_2 P_1, P_2 are released in $t+1$	$\frac{\rho^{2t}}{\rho^{2t-1}} = \rho$

Let $\rho > 1$ be a constant, $0 < \epsilon < 1$. We define 4 kinds of packets at time t.

- Linear decreasing weighted packets:

$$\begin{aligned} L_1 &= < \rho^{2t}, \rho^{2t} - \epsilon > \\ L_2 &= < \rho^{2t-1}, \rho^{2t-1} - 2\epsilon > \end{aligned}$$

- Periodic weighted packets:

$$\begin{aligned} P_1 &= < \rho^{2t}, \rho^{2t} - \epsilon > \\ P_2 &= < \rho^{2t-1}, \rho^{2t-1} - 2\epsilon > \end{aligned}$$

Let the buffer size $B = 2$. Initially, the output buffer is empty. Two packets are released.

- For the linear decreasing weighted case: L_1 and L_2 are released at time 1. MATCH will schedule L_2 first and its adversary will schedule L_1 first;
- For the periodic weighted case: P_1 and P_2 are released at time 1. MATCH will schedule P_2 first and its adversary will schedule P_1 first.

Based on what MATCH selects, its ADV acts as what is listed in Table C.1. In every time slot t, both MATCH and ADV have the new released L_1 and L_2 or P_1 and P_2. So, in each time slot, the weight ratio between ADV and MATCH is ρ. We can repeat this period many times such that the gains of both schedules in the last step are negligible compared with all other gains in the previous steps. As ρ can be arbitrarily large, Theorem C.3 is proved. □

Remark C.1. *Though the weighted matching algorithm (the Hungarian algorithm) works well in finding the optimal solution in scheduling packets with soft deadlines in the offline setting, it does not work well in the online setting in B-bounded model, even in scheduling packets with some simple weight functions.*

C.5.2 Analysis of GREED

Theorem C.4. *The running time complexity of GREED is $O(n^2)$ for each time step.*

Note that for each time step in the tentative schedule, we choose the packet with the maximum-value for that time step. Thus, the correctness of Theorem C.4 is obvious.

Theorem C.5. *For GREED, there is no constant competitive ratio for periodic weighted packet scheduling.*

Proof. Let $\rho > 1$ be a constant and the buffer size is B. B is even and we define,

- $P_1 =< 1, \rho, 1, \rho, \cdots, 1, \rho >$;
- $P_2 =< 0, 1, 0, 1, \cdots, 0, 1 >$.

GREED will schedule P_1 and P_2 sequentially, who gets a total weight of 2 in two neighboring time slots. However, its adversary will schedule P_2 and P_1 sequentially in these 2 time slots, who gets a total weight of ρ. In every two time slots, 2 packet, P_1 and P_2, are released. We can repeat this period many times such that the gains of both schedules in the last step are negligible compared with all other gains in the previous steps. The competitive ratio is $\rho/2$. As ρ can be arbitrarily large, Theorem C.5 is proved. □

Lemma 11. *For GREED, it has a competitive ratio of* 3 *in scheduling monotonic decreasing weighted packets.*

Proof. The proof of Theorem 11 follows the same idea presented in [30]. This theorem also holds for the general case where the buffer size is bounded.

Let $\mathbf{S}^O(t)$ be the set of packets sent out in GREED's adversary by time t (let GREED's adversary be OPT) and $\mathbf{S}^G(t)$ be the set of packets sent by GREED by time t. Denote the two sets as $\mathbf{S}^O(t) = \{1^O, \cdots, t^O\}$ and $\mathbf{S}^G(t) = \{1^G, \cdots, t^G\}$ in the indexed order of time slots. Let r_j denote the release date of packet j and w_{j^A} denote the weight of packet j at time t in algorithm A, where A $= \{\text{GREED}(G), \text{OPT}(O)\}$. We also use $\mathbf{S}$ to denote the set of packets sent by OPT.

Let $\mathbf{V}_1$ be the set of the time slots, in which GREED sends a packet with no less value than the packet sent by OPT. Regard the time slot having a packet with value 0, if GREED has no packet in that time slot. $\mathbf{V}_1 = \{j | w_{j^G} \geq w_{j^O}\}$, and $\mathbf{V}_2 = \mathbf{S} - \mathbf{V}_1$. Note that $\mathbf{S}^O(t)$ is partitioned into the set of packets scheduled in $\mathbf{V}_1$, and the set of packets scheduled in $\mathbf{V}_2$.

Consider the time slot $j \in \mathbf{V}_2$ in time indexed order, where $w_{j^O} > w_{j^G}$. We want to show a mapping from OPT's packets in $\mathbf{V}_2$ to packets in $\mathbf{S}^G(t)$ with the following property: There must exist one corresponding packet $i^G \in \mathbf{S}^G(t)$ for at most two packets in OPT's $\mathbf{V}_2$: j^O and k^O, which is sent out no later than time slot j and k in GREED, such that $\max\{r_{j^O}, r_{k^O}\} \leq i \leq \min\{j, k\}$ and $w_{i^G} \geq \min\{w_{j^O}, w_{k^O}\}$. If we can find such a mapping, then Lemma 11 is proved. In the following, we claim that this mapping does exist.

First of all, we know that for every i^G ($i \in \mathbf{V}_1$), i^G must be scheduled in OPT. Otherwise, scheduling i^G to replace i^O at time slot i gains more value to OPT. For any $j \in \mathbf{V}_2$, either packet j^O has been scheduled before time slot j in GREED or packet j^O is preempted by another packet in time slot i in GREED due to the buffer constraint, $i < j$ and $w_{i^G} > w_{j^O}$.

We make the mapping from time slot $j \in \mathbf{V}_2$ to time slot $i \in \mathbf{S}$, $i < j$ and $r_{j^O} \leq i < j$: Choose i^G such that (1) $i^G = j^O$; or (2) the packet preempts j^G in GREED due to the buffer constraint in the case that GREED drops j^G. For the first case, no two packets in both $\mathbf{V}_2$ and $\mathbf{S}^G(t)$ share the same time slot. For the second case, notice that GREED can preempt at most one packet in each time slot such that no two packets in OPT's $\mathbf{V}_2$, but not in $\mathbf{S}^G(t)$ share one packet in GREED. Therefore, at most two packets in OPT's $\mathbf{V}_2$ share one packet in GREED (One is in case (1) and one is in case (2); Two packets cannot be in the same case). This results in two times of value in GREED which is more than those packets in OPT's $\mathbf{V}_2$.

The mapped packet, i^G, is called the *corresponding packet* of j^O. Therefore, using our method, for every $j^O \in \mathbf{V}_2$, there must exist one i^G to correspond it and at most two packets in $\mathbf{V}_2$ have the same corresponding packets in $\mathbf{S}^G(t)$. Let $\mathbf{V}_3$ denote the set of all such J_i^G. Of course, $\mathbf{V}_1 \subset \mathbf{S}^G(t)$ and $\mathbf{V}_3 \subset \mathbf{S}^G(t)$.

$$\sum_{j \in \mathbf{S}^O(t)} w_j = \sum_{j \in \mathbf{V}_1} w_j + \sum_{j \in \mathbf{V}_2} w_j \leq \sum_{j \in \mathbf{S}^G(t)} w_j + 2 \sum_{j \in \mathbf{V}_3} w_j \leq 3 \sum_{j \in \mathbf{S}^G(t)} w_j.$$

Lemma 11 is proved. □

In scheduling packets with soft deadlines, we observe that GREED has the same worst-case competitive ratio for scheduling linear decreasing weighted packets and scheduling packets with hard deadlines. We can improve the analysis of Lemma 11 and conclude Theorem C.6 (from [31]).

Theorem C.6. *For GREED, it has competitive ratios* 2 *and* 3 *in scheduling linear decreasing weighted packets and monotonic decreasing weighted packets, respectively, in the* B*-bounded model.*

C.5.3 A tight example for GREED

Let $\rho > 1$ be a constant and the buffer size is B. Two packets are released: L_1 and L_2 at time t.

- $L_1 =< \rho, \rho, 0, \cdots, 0 >$;
- $L_2 =< \rho - \epsilon, 0, 0, \cdots, 0 >$.

GREED schedules L_1 first, then L_2, and gets a total weight of ρ in the neighboring 2 time slots. However, OPT schedules L_2 first, then L_1, and gets a total weight of $2\rho - \epsilon$ in the neighboring 2 time slots. At every two time slots, 2 packets, L_1 and L_2, are released. We can repeat this period many times such that the gains of both schedules in the last step are negligible compared with all other gains in the previous steps. Let $\epsilon \approx 0$. The competitive ratio of GREED is $(2\rho - \epsilon)/\rho \approx 2$.

C.5.4 Extension: bounded linear decreasing function

In a specific case of weight function, bounded linear decreasing weight function, all packets have their weight function $w_j(t) = \max\{w_j - \alpha_j(t - r_j), 0\}$, where $0 \leq L \leq \alpha_j \leq U$, L and U are two constants. This model is motivated by [118].

Theorem C.7. *For GREED, it has a competitive ratio of* $2 - L/U$ *in scheduling bounded linear decreasing weighted packets.*

Proof. The proof of Theorem C.7 is similar to the proof of Lemma 11. We know that for the mapping from packets in OPT's $\mathbf{V}_2$ to GREED, the value reduces at least $(j-i)L$ and $(k-i)L$ for packet j and packet k in OPT's $\mathbf{V}_2$, which map to i in GREED, $\max\{r_{j^O}, r_{k^O}\} \leq i < \min\{j,k\}$.

We know $w_{i^G} \geq w_{j^O} + (j-i)L$ and $w_{j^G} \geq w_{k^O} + (k-j)L$ if $k > j$. As $w_{j^O} \neq 0$ and $w_{j^G} = 0$, $L \leq w_{i^G} \leq U$. Therefore, the amortized value ratio for these packets for the two algorithms is at most

$$\rho \leq \frac{w_{i^O} + w_{j^O} + w_{k^O}}{w_{j^G} + w_{i^G}} \leq 2 - \frac{L}{U}.$$

□

C.6 Conclusions

In this dissertation, we consider scheduling unit-length packets with soft deadlines. Based on different application requirements, we propose two models of scheduling soft deadline packets: The general model and the B-bounded model. We design and analysis two simple online algorithms: MATCH and GREED. The competitive ratios for two online algorithms are listed in Table C.2.

Table C.2: Competitive ratios for online scheduling algorithms MATCH and GREED.

weight function	MATCH	GREED
linear decreasing	*non*	2
monotonic decreasing	*non*	3
periodic	*non*	*non*
any	*non*	*non*

Bibliography

[1] D. Karger, J. Wein, and C. Stein, *Scheduling algorithms, Chapter 20 of Algorithms and Theory of Computation Handbook*, 2nd ed. Chapman & Hall/CRC, 2010.

[2] E. G. Coffman, M. R. Garey, and D. S. Johnson, *Approximation Algorithms for NP-Hard Problems: Approximation Algorithms for Bin Packing: A Survey.* PWS Publishing Company, 1997.

[3] N. Buchbinder and J. Naor, "The design of competitive online algorithms via a primal-dual approach," *Foundations and Trends in Theoretical Computer Science*, vol. 3, no. 2-3, pp. 92–263, 2009.

[4] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[5] J. Kleinberg and E. Tardos, *Algorithm Design*, 1st ed. Addison-Wesley, 2005.

[6] S. Dasgupta, C. Papadimitriou, and U. Vazirani, *Algorithms.* McGraw Hill, 2008.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. MIT Press, 2009.

[8] J. S. Vitter and P. Flajolet, *Average-Case Analysis of Algorithms and Data Structures, Chapter 9 in Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Elsevier, 1990, vol. A: Algorithms and Complexity.

[9] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, 2005.

[10] W. Szpankowski, *Average case analysis of algorithms*, 2nd ed. Chapman & Hall/CRC, 2010, vol. 1, ch. 11 of Algorithms and Theory of Computation Handbook.

[11] S. Neil and A. Chaudhary, "Comparing online learning algorithms to stochastic approaches for the multi-period newsvendor problem," in *Proceedings of the 10th ACM-SIAM Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2008, pp. 49–63.

[12] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko, "Buffer overflow management in QoS switches," *SIAM Journal on Computing (SICOMP)*, vol. 33, no. 3, pp. 563–583, 2004.

[13] D. J. Brown and C. Reams, "Toward energy-efficient computing," *Communications of the ACM (CACM)*, vol. 53, no. 3, pp. 50–58, 2010.

[14] U. E. P. Agency, "Report to congress on server and data center energy efficiency public law 109-431," http://hightech.lbl.gov/documents/data_centers/epa-datacenters.pdf, 2007.

[15] J. Hikita, A. Hirano, and H. Nakashima, "Saving 200 kw and \$200 k/year by power-aware job/machine scheduling," in *Proceedings of IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2008, pp. 1–8.

[16] S. Chu, "The energy problem and Lawrence Berkeley National Laboratory," Talk given to the California Air Resources Board, February 2008.

[17] D. N. Tse and S. V. Hanly, "Multiaccess fading channels: Polymatroid structure, optimal resource allocation and throughput capacities," *IEEE Transactions on Information Theory (IEEEIT)*, vol. 44, no. 7, pp. 2796–2815, 1998.

[18] A. Fu, E. Modiano, and J. Tsitsiklis, "Optimal transmission scheduling over a fading channel with energy and deadline constraints," *IEEE Transactions on Wireless Communications*, vol. 6, no. 1, pp. 630–641, 2006.

[19] A. Tarello, J. Sun, M. Zafer, and E. Modiano, "Minimum energy transmission scheduling subject to deadline constraints," *Wireless Neworks*, vol. 14, no. 5, pp. 633–645, 2007.

[20] M. Zafer and E. Modiano, "Optimal rate control for delay-constrained data transmission over a wireless channel," *IEEE Transactions on Information Theory*, vol. 54, no. 9, pp. 4020–4039, 2008.

[21] W. Chen, M. J. Neely, and U. Mitra, "Energy-efficient transmission with individual packet delay constraints," *IEEE Transactions on Information Theory (IEEEIT)*, vol. 54, no. 5, pp. 2090–2109, 2008.

[22] A. E. Gamal, E. Uysal, and B. Prabhakar, "Energy-efficient transmission over a wireless link via lazy packet scheduling," in *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, vol. 1, 2001, pp. 384–394.

[23] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 1995, pp. 374–382.

[24] B. Hajek, "On the competitiveness of online scheduling of unit-length packets with hard deadlines in slotted time," in *Proceedings of the 35th Annual Conference on Information Sciences and Systems (CISS)*, 2001, pp. 434–438.

[25] F. Y. L. Chin and S. P. Y. Fung, "Online scheduling with partial job values: Does timesharing or randomization help?" *Algorithmica*, vol. 37, no. 3, pp. 149–164, 2003.

[26] N. Andelman, Y. Mansour, and A. Zhu, "Competitive queuing polices for QoS switches," in *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2003, pp. 761–770.

[27] F. Li, J. Sethuraman, and C. Stein, "An optimal online algorithm for packet scheduling with agreeable deadlines," in *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2005, pp. 801–802.

[28] M. Englert and M. Westermann, "Considering suppressed packets improves buffer management in QoS switches," in *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2007, pp. 209–218.

[29] F. Y. L. Chin, M. Chrobak, S. P. Y. Fung, W. Jawor, J. Sgall, and T. Tichý, "Online competitive algorithms for maximizing weighted throughput of unit jobs," *Journal of Discrete Algorithms (JDA)*, vol. 4, no. 2, pp. 255–276, 2006.

[30] F. Li, "Competitive scheduling of packets with hard deadlines in a finite capacity queue," in *Proceedings of the 28th IEEE International Conference on Computer Communications (INFOCOM)*, 2009, pp. 1062–1070.

[31] S. P. Y. Fung, "Bounded delay packet scheduling in a bounded buffer," *Operations Research Letters (ORL)*, vol. 38, no. 5, pp. 396–398, 2010.

[32] F. Li, "Packet scheduling in a size-bounded buffer," arXiv:0909.3637[cs.DS], July 2009.

[33] Y. Azar and N. Levy, "Multiplexing packets with arbitrary deadlines in bounded buffers," *Lecture Notes in Computer Science (SWAT)*, vol. 4059, pp. 5–16, 2006.

[34] F. Li, "Improved online algorithms for multiplexing weighted packets in bounded buffers," *Lecture Notes in Computer Science (AAIM)*, vol. 5564, pp. 265–278, 2009.

[35] F. Li, J. Sethuraman, and C. Stein, "Better online buffer management," in *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2007, pp. 199–208.

[36] M. Garey, D. Johnson, B. Simons, and R. Tarjan, "Scheduling unit-time tasks with arbitrary release times and deadlines," *SIAM Journal on Computing (SICOMP)*, vol. 10, no. 2, pp. 256–269, 1981.

[37] T. Heikkinen and A. Hottinen, "Delay-differentiated scheduling in a fading channel," *IEEE Transactions on Wireless Communications*, vol. 7, no. 3, pp. 848–856, 2008.

[38] Baptiste, "Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times," *Journal of Scheduling*, vol. 2, pp. 245–252, 1999.

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

[40] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly (NRLQ)*, vol. 2, pp. 83–97, 1955.

[41] F. Li and Z. Zhang, "Scheduling weighted packets with deadlines over a fading channel," in *Proceedings of the 43rd Annual Conference on Information Sciences and Systems (CISS)*, 2009, pp. 707–712.

[42] M. Chrobak, W. Jawor, J. Sgall, and T. Tichý, "Online scheduling of equal-length jobs: Randomization and restart help?" *SIAM Journal on Computing (SICOMP)*, vol. 36, no. 6, pp. 1709–1728, 2007.

[43] W. Aiello, R. Ostrovsky, E. Kushilevitz, and A. Rosen, "Dynamic routing on networks with fixed-size buffers," in *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms*, 2003, pp. 771–780.

[44] M. Chrobak, "2007 — An offine perspective," *SIGACT News Online Algorithms*, vol. 13, pp. 96–121, 2008.

[45] N. Littlestone, "Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm," *Machine Learning*, vol. 2, pp. 285–318, 1988.

[46] N. Littlestone and M. Warmuth, "The weighted majority algorithm," *Information and Computation*, vol. 108, no. 2, pp. 212–261, 1994.

[47] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall, "Reducing network energy consumption via sleeping and rate-adaptation," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2008, pp. 323–336.

[48] S. Irani, G. Singh, S. K. Shukla, and R. K. Gupta, "An overview of the competitive and adversarial approaches to designing dynamic power management strategies," *IEEE Transactions on Very Large Scale Integration Systems (IEEE TVLSI)*, vol. 13, no. 2, pp. 1349–1361, 2005.

[49] P. Baptiste, M. Chrobak, and C. Durr, "Polynomial time algorithms for minimum energy scheduling," in *Proceedings of the 15th Annual European Symposium on Algorithms (ESA)*, 2007, pp. 136–150.

[50] C. Rusu, R. Melhem, and D. Mosse, "Maximizing rewards for real-time applications with energy constraints," *ACM Transactions of Embedded Computing Systems (TECS)*, vol. 2, no. 4, pp. 537–559, 2003.

[51] V. Devadas, F. Li, and H. Aydin, "Competitive analysis of energy-constrained real-time scheduling," in *Proceedings of the 21st Euromicro Conference on Real-Time Systems (ECRTS)*, 2009, pp. 217–226.

[52] K. Pruhs and C. Stein, "How to schedule when you have to buy your energy," in *Proceedings of International Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX and RANDOM)*, 2010, pp. 352–365.

[53] M. A. Bender, S. Chakrabarti, and S. Muthukrishnan, "Flow and stretch metrics for scheduling continuous job streams," in *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1998, pp. 270–279.

[54] M. A. Bender, S. Muthukrishnan, and R. Rajaraman, "Improved algorithms for stretch scheduling," in *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2002, pp. 762–771.

[55] S. Albers and H. Fujiwara, "Energy-efficient algorithms for flow time minimization," *ACM Transactions on Algorithms (TALG)*, vol. 3, no. 4, p. Article No. 49, 2007.

[56] B. Kalyanasundaram and K. Pruhs, "Speed is as powerful as clairvoyance," *Journal of the Association for Computing Machinery (JACM)*, vol. 47, pp. 617–643, 2000.

[57] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki, "Competitive randomized algorithms for nonuniform problems," *Algorithmica*, vol. 11, pp. 542–571, 1994.

[58] M. Krishna and Y.-H. Lee, "Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems," in *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2000, pp. 156–165.

[59] A. K. Coskun, R. Strong, D. M. Tullsen, and T. S. Rosing, "Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors," in *Proceedings of the 11th ACM International Joint Conference on Measurement and Modeling of Computer Systems (SIGMETRICS/Performance)*, 2009, pp. 169–180.

[60] "Failure mechanisms and models for semiconductor devices, JEDEC publication JEP122C," http://www.jedec.org.

[61] M. Li, A. C. Yao, and F. F. Yao, "Discrete and continous min-energy schedules for variable voltage processors," *Proceedings of the National Academy of Sciences of the USA*, vol. 103, no. 11, pp. 3983–3987, 2005.

[62] M. Li, B. J. Liu, and F. F. Yao, "Min-energy voltage allocation for tree-structured tasks," *Journal of Combinatorial Optimization*, vol. 11, no. 3, pp. 305–319, 2006.

[63] H. Su, F. Liu, A. Devgan, E. Acar, and S. Nassif, "Full chip leakage-estimation considering power supply and temperature variations," in *Proceedings of the 2003*

International Symposium on Low Power Electronics and Design (ISLPED), 2003, pp. 78–83.

[64] W.-C. Kwon and T. Kim, "Optimal voltage allocation techniques for dynamically variable voltage processors," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 4, no. 1, pp. 211–230, 2005.

[65] H.-L. Chan, W.-T. Chan, T.-W. Lam, L.-K. Lee, K.-S. Mak, and P. W. H. Wong, "Energy efficient online deadline scheduling," in *Proceedings of the 18th Annual ACM-SIAM symposium on Discrete algorithms (SODA)*, 2007, pp. 795–804.

[66] K. Pruhs, P. Uthaisombut, and G. Woeginger, "Getting the best response for your erg," *ACM Transactions on Algorithms (TALG)*, vol. 4, no. 3, p. Article No. 38, 2008.

[67] N. Bansal, T. Kimbrel, and K. Pruhs, "Speed scaling to manage energy and temperature," *Journal of the Association for Computing Machinery (JACM)*, vol. 54, no. 1, p. Article No. 3, 2007.

[68] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Power-aware scheduling for periodic real-time tasks," *IEEE Transactions on Computers*, vol. 53, no. 10, pp. 584–600, 2004.

[69] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proceedings of the 18th ACM symposium on Operating Systems Principles (SOSP)*, 2001, pp. 89–102.

[70] E. Bini, G. Buttazzo, and G. Lipari, "Minimizing CPU energy in real-time systems with discrete speed management," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 8, no. 4, p. Article No. 31, 2009.

[71] Y. Zhu and F. Mueller, "Feedback EDF scheduling of real-time tasks exploiting dynamic voltage scaling," *Real-Time Systems*, vol. 31, no. 1-3, pp. 33–63, 2005.

[72] M. Dertouzos, "Control robotics: the procedural control of physical processes," in *Proceedings of the IFIP Congress*, 1974, pp. 807–813.

[73] S. Baruah, "A general model for recurring real-time tasks," in *Proceedings of the 19th IEEE International Real-Time Systems Symposium (RTSS)*, 1998, pp. 114–122.

[74] Y. Nesterov and Nemirovskii, *Interior-Point Polynomial Methods in Convex Programming*. Society for Industrial and Applied Mathematics (SIAM), 1994.

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

[76] H. Yun, P.-L. Wu, A. Arya, T. Abdelzaher, C. Kim, and L. Sha, "System-wide energy optimization for multiple DVS components and real-time tasks," in *Proceedings of the 22nd Euromicro Conference on Real-Time Systems (ECRTS)*, 2010, pp. 133–142.

[77] G. Epps, D. Tsiang, and T. Boures, "System power challenges," in *Cisco Routing Research Seminar*, August 2006.

[78] D. Cooperson, J. J. Mazur, and M. Walker, "Increased focus on network power consumption to lower opex, go green," Tech. Rep., 2009.

[79] O. Tamm, C. Hermsmeyer, , and A. M. Rush, "Eco-sustainable system and network architectures for future transport networks," *Bell-Labs Technical Journal (BLTJ)*, vol. 14, no. 4, pp. 311–3S28, February 2010.

[80] R. Bolla, R. Bruschi, F. Davoli, and F. Cucchietti, "Energy efficiency in the future internet: a survey of existing approaches and trends in energy-aware fixed network infrastructures," *IEEE Communication Surveys and Tutorials*, vol. 13, no. 2, pp. 359–372, 2011.

[81] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, and S. Wright, "Power awareness in network design and routing," in *IEEE INFOCOM*, 2008, pp. 457–465.

[82] C. Lange, D. Kosiankowski, R. Weidmann, and A. Gladisch, "Energy consumption of telecommunication networks and related improvement options," *IEEE Journal of Selected Topics in Quantum Electronics*, 2010.

[83] W. Fisher, M. Suchara, and J. Rexford, "Greening backbone networks: reducing energy consumption by shutting off cables in bundled links," in *Green Networking*, 2010, pp. 29–34.

[84] A. Odlyzko, "Data networks are lightly loaded and will stay that way," *Review of Network Economics*, vol. 2, no. 3, 2003.

[85] S. Bhattacharya, C. Diot, and J. G. Jetcheva, "Pop-level and access-link-level traffic dynamics in a tier-1 pop," in *Internet Measurement Workshop*, 2001, pp. 39–53.

[86] M. Andrews, A. F. Anta, L. Zhang, and W. Zhao, "Routing for energy minimization in the speed scaling model," in *IEEE INFOCOM*, 2010.

[87] L. Chiaraviglio, M. Mellia, and F. F. Neri, "Energy-aware backbone networks: a case study," in *IEEE GreenComm*, 2009, pp. 1–5.

[88] F. Cuomo, A. Abbagnale, A. Cianfrani, and M. Polverin, "Keeping the connectivity and saving the energy in the internet," in *IEEE INFOCOM Workshop on Green Comm. and Networking*, 2011, pp. 319–324.

[89] "Cisco carrier routing system," 2011, http://www.cisco.com/en/US/prod/collateral /routers/ps5763/prod_brochure0900aecd800f8118.pdf.

[90] "T series core router architecture overview," 2012, http://www.juniper.net/us/en/local/pdf/whitepapers/2000302-en.pdf.

[91] Y. Bejerano and S. Vasudevan, "Power efficient pop design and auto-configuration," in *ACM e-Energy*, 2012, pp. 1–10.

[92] K. Thompson, G. J. Miller, and R. Wilder, "Wide-area internet traffic patterns and characteristics," *IEEE Network*, vol. 11, no. 6, pp. 10–23, 1997.

[93] W. John, S. Tafvelin, and T. Olovsson, "Trends and differences in connection-behavior within classes of internet backbone traffic," in *the 9th International Conference on Passive and Active Network Measurement (PAM)*, 2008, pp. 192–201.

[94] A. M. Odlyzko, "The internet and other networks: utilization rates and their implications," *Inf. Economics and Policy*, vol. 12, no. 4, pp. 341–365, 2000.

[95] "Cisco ubr-mc28c and cisco ubr-mc28c-bnc docsis line cards," 2011. [Online]. Available: http://www.cisco.com/warp/public/cc/pd/ifaa/ubmc28y/prodlit/an28c_an.pdf

[96] A. Aurelius, C. Lagerstedt, M. Kihl, M. Perényi, I. Sedano, and F. M. Marcos, "A traffic analysis in the tramms project," *Telekomunikacije*, vol. 4, pp. 29–37, 2009.

[97] M. Gupta and S. S. Singh, "Greening of the internet," in *ACM SIGCOMM*, 2003, pp. 19–26.

[98] R. Bolla, F. Davoli, R. Bruschi, K. Christensen, F. Cucchietti, and S. Singh, "The potential impact of green technologies in next-generation wireline networks: Is there room for energy saving optimization?" *IEEE Communications Magazine*, vol. 49, no. 8, pp. 80 –86, August 2011.

[99] L. Chiaraviglio, M. Mellia, and F. F. Neri, "Reducing power consumption in backbone networks," in *IEEE ICC*, 2009, pp. 2298–2303.

[100] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "Elastictree: Saving energy in data center networks," in *NSDI*, 2010, pp. 249–264.

[101] M. Andrews, A. F. Anta, L. Zhang, and W. Zhao, "Routing and scheduling for energy and delay minimization in the powerdown model," in *IEEE INFOCOM*, 2010.

[102] S. Antonakopoulos, S. Fortune, and L. Zhang, "Energy-aware routing with rate adaptive networking elements," in *IEEE GreenCom*, 2010.

[103] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall, "Reducing network energy consumption via sleeping and rate-adaptation," in *NSDI*, 2008, pp. 323–336.

[104] J. C. C. Restrepo, C. G. Gruber, and C. M. Machuca, "Energy profile aware routing," in *IEEE GreenComm*, 2009, pp. 1–5.

[105] S. Chamberland, "On the design problem of two-level ip networks," in *the 14th International Telecommunications Network Strategy and Planning Symposium (NETWORKS)*, Sept. 2010, pp. 1 –6.

[106] G. Iannaccone, C.-N. Chuah, S. Bhattacharyya, and C. Diot, "Feasibility of ip restoration in a tier 1 backbone," *IEEE Network*, vol. 18, no. 2, pp. 13–19, 2004.

[107] "T4000 dc power requirements, Juniper Networks," 2012. [Online]. Available: http://www.juniper.net/techpubs/en_US/release-independent/junos/topics/reference/specifications/power-t4000-dc-requirements-consumption.html

[108] J. Edmonds, "Maximum matching and a polyhedron with 0,1-vertices," *Journal of Research of the National Bureau of Standards-B. Mathematics and Mathematical Physics*, vol. 69B, no. 1 and 2, pp. 125–130, 1965.

[109] ——, "Paths, trees and flowers," *Canandian Journal of Mathematics*, vol. 17, pp. 449–467, 1965.

[110] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids.* New York: Holt, Rinehart, and Winston, 1976.

[111] H. N. Gabow, "A scaling algorithm for weighted matching on general graphs," in *26th Annual Symposium on the Foundations of Computer Science*, 1985, pp. 90–100.

[112] V. V. Vazirani, *Approximation Algorithm.* Springer, 2003.

[113] M. Roughan, A. Greenberg, M. R. C. Kalmanek, J. Yates, and Y. Zhang, "Experience in measuring internet backbone traffic variability: models, metrics, measurements and meaning," in *the 2nd ACM SIGCOMM Workshop on Internet measurment*, 2002.

[114] T. Richardson, Q. S. Fraser, K. R. Wood, and A. Hopper, "Virtual network computing," *IEEE Internet Computing*, vol. 2, no. 1, pp. 33–38, 1998.

[115] F. Li and J. Nieh, "Optimal interpolation coding for server-based computing," in *Proceedings of the 37th Annual IEEE International Conference on Communications (ICC)*, 2002, pp. 2542–2546.

[116] ——, "Low-complexity interpolation coding for server-based computing," in *Proceedings of the 12th Annual IEEE Data Compression Conference (DCC)*, 2002, p. 461.

[117] Q. Li and F. Li, "A fast content expression for server-based computing," in *Proceedings of the 39th Annual IEEE International Conference on Communications (ICC)*, 2004, pp. 1426– 1430.

[118] P. Richardson, L. Sieh, and A. Ganz, "Quality of service support for multimedia applications," *Real-time Systems*, vol. 21, pp. 269–284, 2001.

Curriculum Vitae

Zhi Zhang received her B.S. and M.S. degrees in Computer Science from Wuhan University in China in 2006 and 2008, respectively. Zhi's research interests are scheduling algorithm design and analysis, specifically, scheduling algorithms and green algorithms for networked systems. Zhi worked as a summer research intern in the Algorithm Research Group at Bell Labs from June to August in 2012.