ONLINE SCHEDULING IN SMART MANUFACTURING

A Dissertation

by

JIN XU

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2020

Major Subject: Industrial Engineering

ABSTRACT

Empowered by data collection entities and computing entities, smart manufacturing systems are much more advanced than traditional manufacturing systems in terms of information availability, systems reliability and productivity. Considered as the key to Industry 4.0, smart manufacturing also brings research questions such as how to improve information availability and how to utilize information in production planning, maintenance scheduling and other operational decisions. This dissertation research thus focuses on several optimization problems that are or envisioned to be prevalent in smart manufacturing systems, including joint production and maintenance decision making problem, polling system scheduling problem, and age of information based scheduling problem. In solving these optimization problems, we provide policies and algorithms from both online scheduling and queueing control perspectives, and develop advanced mathematical frameworks to evaluate the performance of these policies. While this research focuses on a small part of the vast smart manufacturing domain, its scope is wide enough to cover many important problems that exist in both physical and cyber layers of smart manufacturing systems. We expect our research to contribute significantly to the advancement of smart manufacturing, and our models and analysis to also contribute to development of online optimization, queueing theory and information theory. Our models and methodologies can also be adapted to improve system efficiency and reliability, in many other domains which are equipped with system intelligence.

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to Dr. Natarajan Gautam for his care, support, guidance and mentoring during my Ph.D study. Working with Dr. Gautam is one of the luckiest and most enjoyable things in my life. His wisdom and unwavering enthusiasm for research always teach me how to become a good researcher.

I would like to thank Dr. Satish Bukkapatnam, for his support, collaboration and guidance in this dissertation work. I am grateful to Dr. I-Hong Hou for being my research committee member, and also for his advice and collaboration in this dissertation work. I also would like to sincerely thank Dr. Lewis Ntaimo, for his help in my Ph.D study.

I want to express my gratitude to Dr. Yu Ding for his advice and help in my Ph.D study. I also want to thank Dr. Bimal Nepal, for his support in Summer 2018.

I would like to thank my parents for supporting me all the way. Also I would like to thank Dr. Yu Chen, for being my harbor and my light during my darkest time.

Special thanks to Dr. Hoang M. Tran, Dr. Min Kong and Dr. Ping-Chun Hsieh, for being my research collaborators. I also want to recognize my friends in college station for making my life meaningful.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supervised by a dissertation committee consisting of committee chair, Professor Natarajan Gautam, and committee members, Professors Satish Bukkapatnam and Lewis Ntaimo from the Department of Industrial and Systems Engineering, and Professor I-Hong Hou from the Department of Electrical and Computer Engineering.

Part of this dissertation (Chapter 2) was conducted in collaboration with Dr. Hoang M. Tran from Esmart Systems AS, Halden, Norway. All other work for the dissertation was completed by the student independently.

Funding Sources

There are no outside funding contributions to acknowledge related to the research and compilation of this document.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

1. INTRODUCTION

One of the most significant advances in the development of data science, communication technologies, artificial intelligence, and manufacturing technologies is represented by smart manufacturing. Incorporated with sensor networks and computing entities, and empowered by the latest and foreseeable further development of computer science and communication technology, smart manufacturing is considered as the key to the 4th industrial revolution, frequently denoted as industry 4.0 (see [1]). Based on cyber-physical production systems, smart manufacturing differs dramatically from traditional manufacturing in terms of information accessibility, information timeliness and information utilization. It also brings in research questions in smart manufacturing such as how to guarantee information freshness, how to make decisions under certainty or uncertainty, and how to utilize available information for production and maintenance planning.

Motivated by this, in this dissertation research we consider several important optimization problems that are or envisioned to be prevalent in smart manufacturing systems. These problems occur at either the physical layer (which contains human, machines, resources, products, etc.) or cyber layer (which contains communication network, computers, cloud, etc.) of smart manufacturing systems. Specifically, in the first part of this dissertation including Chapter 2 and Chapter 3, we focus on online scheduling problems that occur at the physical layer of smart manufacturing systems. Different from other static systems where information is known and resources are determined before decision making, smart manufacturing systems are quite dynamic in terms of resource accessibility and information availability. Although system information such as machine status, workload of jobs, and processing progress can be monitored in real-time or accurately estimated by close collaborations among smart sensors, communication systems and computing entities in a smart manufacturing system, decision makers still need to deal with the uncertainty that exists in supply, demand and other uncontrollable factors in production such as human behavior or machine breakdown. Further, in a smart manufacturing system that exchanges its physical entities and information with the outside world over time, the status of the system is time-varying. Therefore, there are many challenges existing in smart manufacturing systems in terms of decision making, including 1) how to utilize available information when making decisions; 2) how to make decisions in real-time without knowing the future information; and 3) how to evaluate the performance of a decision. In the first part of this dissertation, we aim to overcome these challenges by designing online scheduling policies that balance information availability and future uncertainty, and evaluate the performance of those policies from an online-scheduling perspective.

In Chapter 2 we consider a joint production and maintenance decision making problem in a custom manufacturing system. In this system, jobs arrive over time but the pattern could be general. A job reveals its workload upon its arrival at the system, while information about future jobs is completely unknown. There is a degrading server in the system whose remaining useful life depends on the processing speed of each job. If the server runs too fast, the server will soon reach a degraded stage that requires a long maintenance time or replacement time. On the other hand, a low processing speed will prolong the processing of jobs. In Chapter 2 we will focus on how to model this problem and design online scheduling policies to minimize the makespan of arbitrary job instances that occur in such a system. Moreover, we aim to evaluate the performance of these online policies by comparing them with the offline optimal solution.

In Chapter 3 we consider a problem in which a flexible machine (which we call server) receives multiple classes of jobs that are stored in input buffers or queues, one for each class. Whenever the server starts processing a job from a different class (i.e., switching queues), a setup time is incurred. This system is modeled as a multi-queue polling system. In this problem, the server needs to decide when to switching queues and in which order to serve jobs so that to minimize the total completion time for an arbitrary job instance. In Chapter 3, we aim to provide online policies that utilize the available job information and also incorporate future uncertainty. We will focus on evaluating the performance of these online policies by providing their worst-case performance bounds.

In addition to scheduling problems that exist in the physical layer of a smart manufacturing system, in the second part of this dissertation research (Chapter 4 and Chapter 5), we consider scheduling problems that occur at the cyber layer. Information timeliness plays a crucial role in smart manufacturing systems, especially when decisions have to be made timely or in real-time. Any information delay or loss may undermine the quality of real-time decision making. As the cyber layer of smart manufacturing systems is built upon communication networks and computing entities, besides improving the performance of hardware, many challenges still exist in system design and policy design. Age of information (AoI) is a metric that has been studied recently as it characterizes the information freshness or timeliness by measuring the time elapsed from the most recent updated data packet. In the second part of this dissertation research, we focus on research questions such as how to design information transformation policies, and how to design communication systems to optimize the AoI performance in smart manufacturing systems.

In Chapter 4 we consider the age performance of a system with a single server and multiple data sources. In smart manufacturing systems where some data sources contain age sensitive information or emergency information such as high temperature and high pressure, static priority are usually specified for data sources to guarantee the data packets from these age-sensitive sources to be processed once the server becomes available. Motivated by this, we study a priority queue system where job priority is pre-determined and job service order within the same queue is to be decided to minimize the peak age of information (PAoI). In this research, we aim to answer the question whether First Come First Serve (FCFS) or Last Come First Serve (LCFS) is optimal in terms of minimizing PAoI. Also, it is unknown if keeping one packet at each queue is more efficient than keeping all packets in terms of reducing age. In this chapter, we will focus on providing a closed-form expression for PAoI of such a system under different service disciplines and buffer management strategies. We aim to explore the advantage of each service discipline and buffer management strategy to provide guidance in minimizing PAoI in such a system.

In Chapter 5 we consider a system where the server takes vacations over time. This research is motivated by applications in smart manufacturing networks and other communication networks where the server is not constantly available. In communication systems where the server can be put into sleep or wake modes, the sleeping time can be modeled as a vacation time. Also in a system where the server only collects or processes information from time to time, the absence period of the server can also be regarded as a vacation time. In Chapter 5, we will focus on packet management strategies to minimize the AoI and PAoI in such a vacation server system. We aim to provide closed-form expressions of AoI and PAoI for systems with different packet management strategies, and explore the advantage of each strategy.

Figure 1.1 shows a general framework of a smart manufacturing system which contains a physical layer and a cyber layer. In the physical layer, there could be human, machines, warehouses, etc. Physical entities such as resources and products are exchanged in this layer. In the cyber layer, there are communication networks, computing entities such as computers and smart phones, etc. Information is usually exchanged in this layer. Notice that the physical layer and the cyber layer are also connected through communication networks. Data collected by sensors and cameras in the physical layer are sent to the cyber layer for processing, storing or sharing, and the computing results and feedback obtained through the cyber layer are sent to the physical layer for improving system performance. In Chapter 2 and Chapter 3, we mainly focus on scheduling problems at the physical layer where machine and jobs are located. In Chapter 4 and Chapter 5 we mainly focus on scheduling problems at the cyber layer where data packets and information are transmitted. The main objectives of this research are summarized in Table 1.1. Note that this manuscript consists of one published journal article [2] which corresponds to Chapter 2. The copyright and publication information will be provided on the first page of that chapter. This manuscript also consists three papers under review [3, 4, 5], corresponding to Chapters 3, 4 and 5 respectively. Each chapter of this dissertation consists of individual sections including introduction, literature review, detailed description of problem, solution approach, numerical experiments, and conclusion. In Chapter 6 we will conclude this dissertation research, and discuss future work as well as research opportunities.

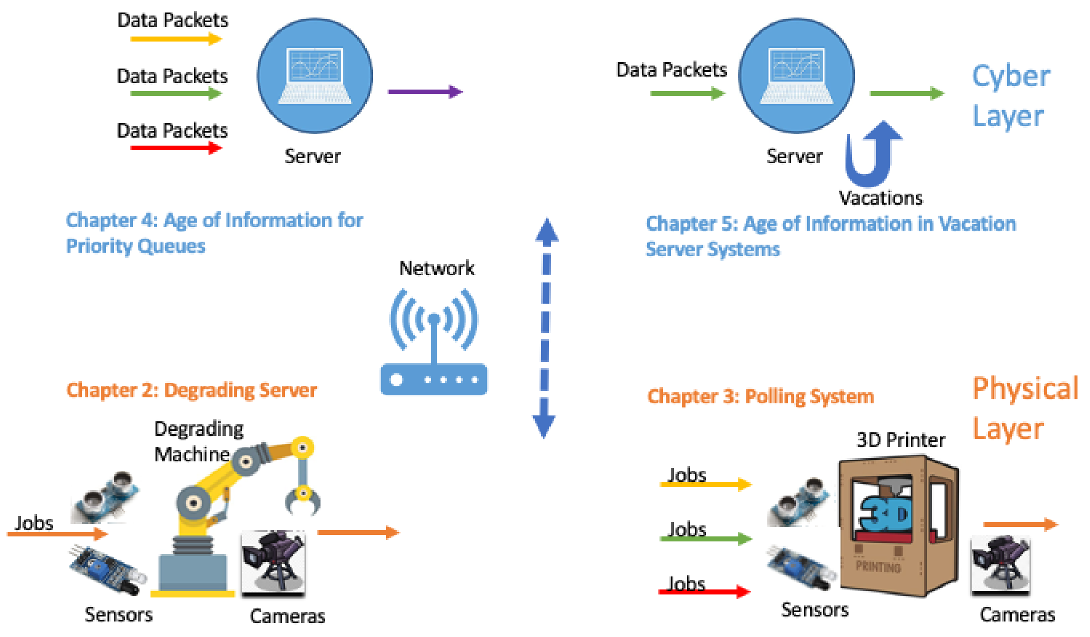


Figure 1.1: A General Framework of This Dissertation

Chapter	Type	Objectives
Chapter 2	Physical Layer	Decide processing speeds for jobs and maintenance for tools to minimize total makespan of a job instance
Chapter 3		Decide processing sequences for jobs and queues to minimize total completion time of a job list
Chapter 4	Cyber Layer	Decide data packets transmission strategies to minimize PAoI
Chapter 5		Decide buffer management strategy for minimizing AoI and PAoI in vacation-server systems

Table 1.1: Summary of Research Objectives of Each Chapter

2. JOINT PRODUCTION AND MAINTENANCE OPERATIONS IN SMART CUSTOM-MANUFACTURING SYSTEMS[*]

2.1 Introduction

In the coming years it is envisioned that there would be a significant rise in the number of custom manufacturing facilities (see [6]). These facilities are expected to have highly versatile machines and actuators that are interconnected enterprise-wide through Internet of Things (IoT) ([1]). IoT enables universal manufacturing resource availability and accessibility by integrating and connecting physical assets into an information network ([7]). From a production and maintenance operations standpoint, there are several benefits of IoT for custom manufacturing such as reducing variability and uncertainty as well as jointly performing production and maintenance activities in a single framework which significantly improves productivity and efficiency of the system as a whole ([7, 8]). However, these necessitate the creation of new models and methods for decision-making and control.

As a small step in that direction, we consider a custom manufacturing machine which is a more generic version of a machine tool employed for flexible manufacturing operations (e.g., [9]) and could use a much broader set of subtractive and/or additive manufacturing operations. The machine is embedded with sensors and analytics to determine and communicate its internal state (e.g., tool condition), gather statuses of various IoT-enabled jobs across the enterprise, and is capable of autonomous operations, especially in terms of making and executing production planning and control decisions.

Traditionally, such custom manufacturing systems have been analyzed either under a completely stochastic framework (with random inter-arrival times and service times of jobs, and random breakdown and repairs of tools) (e.g., [10]), or a completely deterministic framework where all information is available at the beginning of a day and a production plan is created (e.g., [11]).

[*]This is an Accepted Manuscript of an article published by Taylor & Francis in IISE Transactions on January 2, 2019, available online at the Taylor & Francis Ltd web site: www.tandfonline.com with the link to the article - https://www.tandfonline.com/doi/full/10.1080/24725854.2018.1511938

With IoT for custom manufacturing, a middle ground is more appropriate but under a more generic and realistic setting. For example, jobs arrive arbitrarily into the system (not necessarily according to a renewal process) at random times but once they arrive, their processing requirements (or workloads) are revealed (i.e., known deterministically). That is because a job communicates its dimensions, specifications and requirements (i.e., as-is and to-be states) to the machine, and the machine develops a process plan through interactions with databases in a cloud. The plan would include a set of sub-tasks to be performed by the machine and each sub-task is characterized by the amount of workload. The machine also interacts with the tool to get updates on the remaining life and assesses if another sub-task could be performed or would the tool have to be replaced (see [7, 12]).

We keep our production decision fairly simplistic in that we wish to determine the speed at which the tool operates for each job. We note that the approach is quite general to be applicable to emerging custom manufacturing scenarios with IoT and smart manufacturing machines in that sense. It can be used in processes that involve complex chain of sub-processes, such as multiple steps of machining (e.g., drilling) operations. This scenario, for example, can reflect hole drilling operations specified on custom integral (near-net shape) components employed in aerospace, defense, energy and biomedical implant industry (e.g., integral aircraft or hull panels, oil pumps, hip joints, or artificial hearts). It could also be used in milling, turning and drilling where material is removed. We use the term “speed" as opposed to the more common term MRR (material removal rate) or the deposition rate. The faster the speed is, the faster the job is finished, albeit the tool degrades faster as well. We assume replacing a tool when it degrades (e.g., wears out) completely is time-consuming. Hence it is important to plan both speeds and replacements in advance so that tool replacements are scheduled between jobs, and not when a job is being executed, thereby efficiently operating the machine. For that, the objective considered in this paper is to maximize the effective processing capacity of the machine.

This paper is organized as follows. In Section 2.2 we describe the problem statement and related work. In Section 2.3, we introduce notations and modeling details of our problem. We explain analytic approaches to solve the problem and describe the asymptotic properties of the solutions in Section 2.4. We provide numerical results in Section 2.5 and state concluding remarks as well as future work in Section 2.6.

2.2 Problem Statement and Related Work

As described in Section 2.1, we consider a smart machine that can execute different custom jobs using a single process type (e.g., drilling of custom panels). A list L of $n(L)$ sub-tasks $L = (w_1, w_2, ..., w_{n(L)})$ are performed on the machine, where w_i is the quantity of work that needs to be performed for sub-task i. We assume that each w_i is a random variable and when the sub-task arrives, the random variable is realized. Let s_i be the speed for processing sub-task i. The time it takes to complete sub-task i is $\frac{w_i}{s_i}$. Now, let $D(s)$ be the tool degradation rate when it processes a sub-task at speed s. After processing sub-task i, the tool degrades by amount $D(s_i)\frac{w_i}{s_i}$. The decision policy would determine the speed to process each sub-task and replacing the tool $(x_i = 1)$ or not $(x_i = 0)$ before processing sub-task i. Here we assume that replacing a tool takes τ time units.

The objective is to make a decision about tool replacement x_i and machine speed s_i for all i so that in the long run the total time spent per sub-task on average (this includes tool replacement time, if any, and processing time) is minimized. This would be a good indication of the capacity of the machine. It is important to note the implication of having a replacement time τ as it creates a trade off since high processing speeds would imply too many tool changes and low processing speeds would result in too few parts being processed. While we do not explicitly model job arrival processes, it is important to note that we place no restriction on it either. Jobs can arrive individually or in batches, the inter-arrival times can be correlated (i.e. non IID), and even time-varying. The same is the case with the workloads. This brings a tremendous amount of generality and can be used to conveniently decompose the decisions at each machine. Moreover, we relax the definition of job arrivals. In our study, a job “arrival" does not need to be at the machine. Once the workload of a job is revealed deterministically, we regard this job as an arrival. This is because it would be possible to extract full information about a job upstream to a machine with the help of

IoT (see [7]).

An important aspect of this study is to make online decisions based on revealed information. We consider the entire spectrum of revealed workload all the way from $n(L) = 0$ corresponding to no available workload information to $n(L) \to \infty$ corresponding to the offline case in which full information about workload is known before processing. Then we show that as more information is available, the online algorithms perform better.

Manufacturing control has been discussed for decades and there is a rich literature related to it. Many works such as [13, 14, 15, 16] and [17] consider the workload flow controlling problem with continuous workload and objective of minimizing holding (i.e., inventory) cost. [16] modeled a machine with two statuses (up and down), and [17] extend this to arbitrary number of machine statuses. [14] considers the case where machine failure rate depends on processing rate. Discreteness of workload brings randomness in workload processing. Works such as [18, 19, 20, 21, 22] and [10] consider problems with discrete workload and objective of minimizing processing cost. In these works, workload would not be revealed until the end of processing. The idea of revealing workload upon arrival is shown in online machine scheduling problems such as [23] and [24], in which job sequence is considered to minimize mean completion times (sojourn times) of jobs. However, speed controlling and machine maintenance are not considered in these models. [25] address a makespan minimization problem in which workload is revealed upon arrival, and machines are reconfigurable so that processing speed can be adjusted by changing different machine configurations. However, the setup times for reconfiguration is assumed to be negligible and tool degradation is not considered in [25]. Offline machine scheduling problems with minimizing makespan are also discussed, such as [26] and [27]. In these problems, workload is known deterministically at the beginning. Job arrival and machine status are not considered in these works.

In summary, to the best of our knowledge, there aren't any articles in the open literature that consider the following four aspects in a single framework: **a)** Discrete part manufacturing; **b)** Machine with degrading tools and degradation is a function of processing speed; **c)** Jobs where workload is revealed upon “arrival"; and **d)** Objective of minimizing completion time.

Specifically, while we consider a manufacturing problem that has been discussed for decades, the aformentioned unified framework is novel. The jobs are discrete which is different from the settings in [14] as well as [15]. We incorparate the phenomenon where high processing speed usually results in fast tool degradation, which is absent in [28, 29, 30, 31] and[25]. We show that traditional approaches such as [31, 22, 32] and [33] are no longer applicable to our problem as the uncertainty of tool breakage and workload is eliminated in our model. The greedy algorithm used by [10] are not as effective as the online algorithms that we develop. Moreover, we show that knowing the distribution of workload beforehand and revealing workload upon arrival are crucial for decision making. We demonstrate the improvement by knowing workload distribution in Section 5 of this paper. A comparison of relevant literature is provided in Table 2.1 below.

In addition, we provide a new problem setting where job arrivals do not have to be at the machine to determine the workload. With the advent of IoT it would be possible to reveal workload well before the job physically arrives at a machine. In that way our model is adaptable to futuristic settings as compared to traditional control and scheduling models.

Paper	Objective	Job Arrival	Speed Control	Maintenance	Tool Degradation	Workload	Algorithms
[15, 16, 13, 14, 17]	Holding Cost	Yes	Yes	Yes	Random	Continuous	Online
[32]	Holding Cost	Yes	Yes	Yes	Random	Random	Online
[18, 20, 10, 22, 21, 19]	Processing Cost	Yes	Yes	Yes	Random	Random	Online
[34]	Processing Cost	Yes	Yes	No	No	Deterministic	Offline
[35]	Processing Cost	No	Yes	Yes	Random	No	Online
[23, 24]	Sojourn time	Yes	No	No	No	Revealed deterministically upon Arrival	
[26, 27]	Makespan	No	Yes	No	No	Deterministic	Offline
[28]	Makespan	No	No	Yes	No	Deterministic	Offline
[31]	Makespan	No	No	Yes	Random	Random	Online
[25]	Makespan	Yes	Yes	No	No	Revealed Deterministically upon Arrival	Online
Our Paper	Makespan	Yes	Yes	Yes	Deterministic	Revealed deterministically upon Arrival	Online&Offline

Table 2.1: Comparison of related literature

2.3 Model and Notations

Assume workload of job i is upper bounded by w_u. Notice that in this paper we characterize tool degradation by a physics-based deterministic model and we introduce remaining tool level as the measure of its remaining processing capacity. For example, in drilling and other material removal processes, the remaining level of a tool is the amount of wear, measured in terms of length of the wear h it can sustain before it needs to be replaced. We assume all the *new* tools are of the same level h^* and their wear is governed by the same degradation function $D(s)$. For convenience, we provide a list for notations, which is shown in Table 2.2. When job i is about to be processed, it observes the remaining tool level h_i at that time. After the tool processes sub-task i at speed s for time $\widehat{t}$, its remaining level is $h_i - D(s)\widehat{t}$. In our paper, we mainly use Taylor's formula to characterize tool degradation (see [36]), since it perfectly describes the relation of tool degradation and processing speed for jobs. Taylor's formula is given by $sT^n = C^*$, where T is the tool life (i.e., the duration of the time the tool can be used in the specified processing condition), $n \in (0,1)$ and C^* are constants which are experimentally determined. For convenience, we let $\nu = \frac{1}{n}$ (hence $\nu > 1$), and Taylor's formula can be written as

$$s^\nu T = C, \quad \nu > 1 \tag{2.1}$$

where $C = (C^*)^\nu$. When processing with speed s, a tool of remaining level h^* is of lifetime T_s, thus the degradation rate is given by

$$D(s) = \frac{h^*}{T_s} = \frac{s^\nu}{C} h^*, \quad \nu > 1. \tag{2.2}$$

Assume changing of speed and replacing the tool are not allowed during processing ([37]), thus for job i of workload w_i, it takes $\frac{w_i}{s_i}$ to finish processing. Therefore h_i can be updated by $h_{i+1} = h_i - D(s_i)\frac{w_i}{s_i}$. Since tool level cannot be negative, we have for each job i: $h_i - D(s_i)\frac{w_i}{s_i} > 0$. When a tool degrades to a status in which $h_i = 0$, it has to be replaced by a new tool and the new tool will be used to process job i, then we have $h_i = h^*$. Note that we also allow replacing a tool even if has not reached the final status. Since $x_i = \{0, 1\}$ is a binary variable denoting replacing decision before processing job i, we can write $h_i \triangleq h^* x_i + (1 - x_i)\left(h_{i-1} - D(s_{i-1})\frac{w_{i-1}}{s_{i-1}}\right)$ where $x_i = 1$ indicates replacing the tool before job i and $x_i = 0$ for not.

Parameters

w_i: Revealed workload of job i

$w_b = \left((\nu - 1)\tau C^{\frac{1}{\nu-1}}\right)^{\frac{\nu-1}{\nu}}$: Optimal tool level

w_u: Upper bound for the workload w_i

w_c: Tool/buffer capacity, the maximal workload a tool/buffer can serve/hold

W_i: Random variable denoting workload of future job i before revealing

h_i: Remaining tool level before processing job i

h^*: Tool level for a new tool

T_s: Tool life time for a new tool with processing speed s

τ: Replacement time for one tool

$\nu = \dfrac{1}{n}$, C: constants experimentally determined for Taylor's formula

$D(s)$: Degradation rate for processing speed s

η_j: Processed/packed workload by tool/buffer j

Variables

s_i: Processing speed for job i

x_i: Decision variable for tool replacement before processing job i

Problems and Approaches

OF: Offline Problem

OFR: Offline Problem with Rearranging

BC: Best Case Problem

BC*: Modified Best Case Problem

FS: Fixed Speed Approach

FB: Fixed Buffer Approach

SOP: Stochastic Optimal Policy

SA: Simple Approach

Table 2.2: Notations Used

Note that the workload of a job is only revealed when the IoT system gathers sufficient information about it. We first describe an offline problem in which we assume that all the information about jobs in the future is obtained before processing, the objective is to find the optimal decision $\mathbf{s} = \{s_1, s_2, ..., s_{n(L)}\}$ and $\mathbf{x} = \{x_1, x_2, ..., x_{n(L)}\}$ to minimize makespan. The **Offline Problem**

(**OF**) can be described as follows:

Problem 1 (**Offline Problem**).

$$T_{OF}(L) = \min_{\boldsymbol{x},\boldsymbol{s}} \frac{1}{n(L)} \sum_{i=1}^{n(L)} \left(\frac{w_i}{s_i} + \tau x_i \right) \tag{2.3}$$

$$s.t. \quad h_{i+1} \triangleq h^* x_{i+1} + (1 - x_{i+1}) \left(h_i - D(s_i) \frac{w_i}{s_i} \right), \tag{2.4}$$

$$h_i - D(s_i) \frac{w_i}{s_i} \geq 0, \tag{2.5}$$

$$s_i > 0, \tag{2.6}$$

$$x_i = \{0, 1\} \; for \; i = 1, 2, ..., n(L). \tag{2.7}$$

The objective function (2.3) is to minimize the average cycle time of each job that consists of processing time and tool replacement time. The first constraint (2.4) is the update function, showing that before processing job i, the remaining tool level is either h^* if replacement happened or the remaining level $h_{i-1} - D(s_{i-1}) \frac{w_{i-1}}{s_{i-1}}$ if replacement did not happen. Constraint (2.5) indicates that tool level cannot be negative. Our decision variable speed s_i should be positive, and replacing decision variable x_i, which is binary, are shown in constraints (2.6) and (2.7).

If we also consider the order of service, we have the **Offline Problem with Rearranging** as **Problem 2**, where $R(L)$ is the set of lists in which each list is made by rearranging the order of list L. Any list in $R(L)$ is given by $\{w_{(1)}, w_{(2)}, ..., w_{(n(L))}\}$, which is a permutation of list $\{w_1, w_2, ..., w_{n(L)}\}$.

Problem 2 (**Offline Problem with Rearranging (OFR)**).

$$T_{OFR}(L) \triangleq \min_{L^0 \in R(L)} \frac{1}{n(L^0)} \sum_{i=1}^{n(L^0)} \left(\frac{w_{(i)}}{s_i} + \tau x_i \right) \tag{2.8}$$

$$s.t. \; h_{i+1} \triangleq h^* x_{i+1} + (1 - x_{i+1}) \left(h_i - D(s_i) \frac{w_{(i)}}{s_i} \right),$$

$$h_i - D(s_i) \frac{w_{(i)}}{s_i} \geq 0,$$

$$s_i > 0, \; x_i = \{0, 1\}, \textit{ for } i = 1, ..., n(L^0).$$

Notice we can also write **OFR** as

$$T_{OFR}(L) \triangleq \min_{L^0 \in R(L)} T_{OF}(L^0) \tag{2.9}$$

The permutation set $R(L)$ has $n(L)!$ elements. We want to choose the best order from $R(L)$, say L^*, whose optimal value of L^* is less than that of any other lists in set $R(L)$. When job list L has finite number of sub-tasks, then we have $L^* = arg\min_{L^0 \in R(L)} T_{OF}(L^0)$ such that $T_{OFR}(L) = T_{OF}(L^*)$. If we add a constraint on the **Offline Problem with Rearranging** that processing speed is a constant, the modified problem is to find the minimal number of tool replacements, which is a bin packing problem and known to be NP-hard (see [38]).

We select makespan as objective function because it is a good indication for machine capacity. It allows for general arrival processes (i.e, correlated, time varying or in batches). Besides, arrival does not need to be at the machine. As long as a job has revealed its workload, we regard it as an arrival. Moreover, if there is a cost rate for machine in operation, say K, and cost for tool replacement, say R, then the makespan minimization problem can be written as a cost minimization problem. Notice that since we do not allow tool breakage during processing, every job will be processed successfully. The **Cost Minimization Problem (CM)** can be written as follows:

$$\begin{aligned}
\textbf{Cost Minimization}: \quad & \min \sum_{i=1}^{n(L)} \left(K\frac{w_i}{s_i} + Rx_i \right). \\
s.t. \quad & h_{i+1} \triangleq h^* x_{i+1} + (1 - x_{i+1}) \left(h_i - D(s_i)\frac{w_i}{s_i} \right), \\
& h_i - D(s_i)\frac{w_i}{s_i} \geq 0, \\
& s_i > 0,\ x_i = \{0, 1\}\ for\ i = 1, 2, ..., n(L).
\end{aligned}$$

The setting of this problem relaxes the one investigated in [10] by considering that workload is revealed on upon arrival. We will show in **Section 5** of the paper that if we use the greedy approach in [10], it would result a much weaker solution than ours. Notice **CM** is different from **OF** only in the objective function. If $\tau = \dfrac{R}{K}$, then these two problems are identical.

2.4 Online Algorithms

Since the workload of a job is not revealed until sufficient information is gathered, to solve **OF** we must look through all $n(L)$ jobs. A scenario we observe is that once new jobs are revealed we can update our information about jobs dynamically. As more information becomes available, better decisions can be made. Thus our approaches will focus on how to deal with the information updating process. When future arrivals have not been revealed, we make decisions using online algorithms. We use a behavior-imitating way to construct the online algorithms and we want the online algorithms to behave similarly in some aspects as the optimal policy. We first characterize some useful properties that our algorithms can imitate. Also we provide a lower bound of **OF** that can be used to characterize the performance of approaches that we propose. After that we provide an approach where we process jobs with a fixed speed, finally we introduce an approach which solves **OF** part by part by choosing the size of sub-problems dynamically and show that rearranging is unnecessary for online algorithms.

2.4.1 Properties of the Offline Problem

The main idea of constructing the online algorithms is to find some unique properties of **OF** that online algorithms can imitate. So in this subsection we mainly discuss those properties of **OF**. We first have Theorem 2.4.1 in the following to characterize the optimal solution to **OF**. Note that in Theorem 2.4.1 we assume $\frac{D(s)}{s}$ is convex and increasing in s. This is true when using Taylor's formula (Equation (2.1)) to describe the degradation process, in which case degradation rate is given by $\frac{s^\nu}{C}h^*$, as shown in Equation (2.2). Since $\nu > 1$ in Taylor's formula, we have $\frac{D(s)}{s} = \frac{s^{\nu-1}}{C}h^*$ to be convex and increasing when $s > 0$. However Theorem 2.4.1 is not confined to using Taylor's formula to define tool degradation rate function $D(s)$.

Theorem 2.4.1. *On solving **OF**, if $D(s)$ is the degradation rate such that $\frac{D(s)}{s}$ is convex and increasing when $s > 0$ with $\frac{D(s)}{s}|_{s=0} > 0$, then the optimal processing speeds for jobs processed by the same tool are identical, i.e. $s_i = s_j$ if job i and j are processed by the same tool.*

Proof. We know that the optimal solution is a series of x_i's which are either 1 or 0. Without loss of generality, assume that $h_1 = h^*$ thus $x_1 = 1$. Say $x_2 = x_3 = \cdots = x_{m-1} = 0$ and $x_m = 1$. Thus there are m jobs served by this tool, and total time working on these m jobs is $\sum_{i=1}^{m} \left(\frac{w_i}{s_i}\right) + \tau$. Since the same tool is used and no replacement happens during processing, tool level cannot be negative in the end. Suppose ϵ of tool level remains when finishing these m jobs. From the constraint (2.5) and (2.6), we have $\sum_{i=1}^{m} \frac{w_i}{s_i} D(s_i) = h^* - \epsilon$. We want to prove that processing time is minimal when $s_1 = s_2 = ... = s_m$.

First consider the following sub-problem with only the first m jobs of L. Notice its constraints are specified in terms of the update function of Equation (2.4) and no replacement happens during processing of these m jobs. Hence

$$\begin{aligned}
\min \quad & \sum_{i=1}^{m} \left(\frac{w_i}{s_i}\right) + \tau \\
s.t. \quad & h_2 - h^* + \frac{w_1}{s_1} D(s_1) = 0, \\
& \vdots \\
& h_m - h_{m-1} + \frac{w_{m-1}}{s_{m-1}} D(s_{m-1}) = 0, \ \textit{and} \\
& \epsilon - h_m + \frac{w_m}{s_m} D(s_m) = 0.
\end{aligned}$$

Write the constraints of above problem as $\mathbf{g}(s) = (g_1(s), g_2(s), ..., g_m(s))^T = 0$, where $g_i(s)$ is the LHS of the ith constraint above. The Lagrange function of above problem can be written as

$$\begin{aligned}
\mathcal{L}(s, \lambda) \ = \ & \sum_{i=1}^{m} \left(\frac{w_i}{s_i}\right) + \tau - \lambda_1 (h_2 - h_1 + \frac{w_1}{s_1} D(s_1)) - ... \\
& -\lambda_{m-1} (h_m - h_{m-1} + \frac{w_{m-1}}{s_{m-1}} D(s_{m-1})) \\
& -\lambda_m (\epsilon - h_m + \frac{w_m}{s_m} D(s_m)).
\end{aligned}$$

From the first order condition of the Lagrange function we have $-\frac{w_i}{s_i^2} - \lambda_i \frac{\partial}{\partial s_i} \left(\frac{w_i}{s_i} D(s_i)\right) = 0$ for all $i \in \{1, ..., m\}$. It is easy to show that the point $s^* = \{s_1^*, s_2^*, ..., s_m^*\}$ which satisfies $s_1^* = s_2^* = ... = s_m^*$ is a stationary point. Thus we know that $Z(s^*) = \{w_2, -w_1, 0, ..., 0\}$ is a null space of $\nabla \mathbf{g}(s^*) = 0$. From the first order condition we have $\lambda_i = \dfrac{-w_i}{\frac{\partial}{\partial s_i}(\frac{D(s_i)}{s_i})s_i^2} |_{s_i=s_i^*}$. Note that $\lambda^* = (\lambda_1^*, ..., \lambda_m^*)$, then we know $\nabla_{ss}\mathcal{L}(s^*, \lambda^*)$ is a diagonal matrix with the ith diagonal element $\left(\dfrac{2w_i}{s_i^3} + \dfrac{w_i \frac{\partial}{\partial s_i^2}(\frac{D(s_i)}{s_i})}{\frac{\partial}{\partial s_i}(\frac{D(s_i)}{s_i})s_i^2}\right)\Bigg|_{s_i=s_i^*}$. Since $\dfrac{D(s)}{s}$ is increasing, we have $Z(s^*)^T \nabla_{ss}\mathcal{L}(s^*, \lambda^*)Z(s^*)$ to be positive definite. Since we know s_i's being identical is a solution of above Lagrange function, from convexity this solution is optimal (see [39]). □

Corollary 2.4.2. *One solution to* **OF** *is such that when the tool is about to be replaced, its remaining level is zero. That is, when $x_{i+1} = 1$, we have that $h_i - D(s_i)\frac{w_i}{s_i} = 0$.*

Proof. Suppose m jobs are processed by tool j and ϵ tool level is left when tool j is about to be replaced. Then we have that $\sum_{i=1}^m \dfrac{w_i}{s_i} D(s_i) = h^* - \epsilon$, and time between replacements $\sum_{i=1}^m \left(\dfrac{w_i}{s_i}\right) + \tau$, which is the portion corresponding to this tool in (2.3). From **Theorem 2.4.1** we see that in the optimal solution, tool j processes its jobs with a single speed, say s_j^*. If $\widehat{w} = \sum_{i=1}^m w_i$, we have the time between two replacements equal to $\frac{\widehat{w}}{s_j^*} + \tau$, with the constraint $\widehat{w}\frac{D(s_j^*)}{s_j^*} = h^* - \epsilon$. Since $\frac{D(s)}{s}$ is increasing with respect to s, then the solution of $\widehat{w}\frac{D(s_0)}{s_0} = h^*$, say s_0, is greater than s_j^*. Then, since $\frac{\widehat{w}}{s_0} < \frac{\widehat{w}}{s_j^*}$, we can see $\epsilon = 0$ will result in minimal time between replacements. □

So far we have seen that in the optimal solution of **OF**, jobs served by the same tool have identical processing speed. We want our online algorithms to imitate this property. However, to see how good it would be to fix the processing speed, we need a benchmark to compare our algorithms against. For that, we solve another system which serves as the lower bound of **OF**. If we find an algorithm that gets a result larger than but close to this lower bound, we can assure that this result is close to what **OF** gets. Since we know $T_{OFR} \leq T_{OF}$, then we can use the optimal solution of **OFR** as the lower bound of **OF**. We introduce the following problem to get the solution of **OFR** analytically.

Problem 3 (Best Case Problem (BC)).

$$
\begin{aligned}
& T_{BC}(L) \triangleq \min \frac{1}{n(L)} \sum_{i=1}^{n(L)} \left(\frac{w_i}{s_i} + \tau x_i \right) \\
s.t. \quad & \sum_{i=1}^{n(L)} w_i \frac{D(s_i)}{s_i} \leq \sum_{i=1}^{n(L)} x_i h^* \\
& s_i > 0,\ x_i = \{0, 1\}, for\ i = 1, ..., n(L).
\end{aligned} \tag{2.10}
$$

Notice the **BC** differs from **OF** only in the constraints. **BC** provides the number of tools needed for serving a given total workload, without considering feasibility.

Lemma 2.4.3. *For the same job list L, **BC** always gets optimal value no larger than **OFR**.*

Proof. We only need to show for any order $L^0 \in R(L)$, $T_{BC}(L^0) \leq T_{OF}(L^0)$. Since the objective functions of both **BC** and **OF** are the same, we only need to prove the feasible region of the **OF** is contained in that of **BC**. For any feasible solution of **OF**, $\mathbf{x} = (x_1, x_2, ...)$ is made of a series of ones and zeros. Suppose x_{l_m} and $x_{l_{m+1}}$ are two adjacent 1 in the solution, then $\sum_{i=l_m}^{l_{m+1}-1} w_i \frac{D(s_i)}{s_i} \leq h^*$. Thus $\sum_{i=1}^{n(L)} w_i \frac{D(s_i)}{s_i} \leq h^* \sum_{i=1}^{n(L)} x_i$, we can get inequality (2.10). Then its feasible region is contained in that of **OF** for any list L. From Equation (2.9) we get $T_{BC}(L) \leq T_{OFR}(L)$. □

When the degradation rate is given by Equation (2.2), the next lemma gives a bound for $T_{BC}(L)$.

Lemma 2.4.4. *For **BC** with finite $n(L)$, when $D(s) = h^* \frac{s^\nu}{C}$, if $w_b = \left((\nu - 1)\tau C^{\frac{1}{\nu-1}}\right)^{\frac{\nu-1}{\nu}}$ and $w_i \leq w_b$, given that $\widehat{w} = \sum_{i=1}^{n(L)} w_i$, then its optimal solution is given by $s = \left(\dfrac{Cy^*}{\widehat{w}}\right)^{\frac{1}{\nu-1}}$, $y^* = \sum_{i=1}^{n(L)} x_i^* = \lfloor \dfrac{\widehat{w}}{w_b} \rfloor$ or $\lceil \dfrac{\widehat{w}}{w_b} \rceil$. We have either $\dfrac{\tau^{\frac{1}{\nu}}(\nu-1)^{\frac{1}{\nu}-1}\nu\widehat{w}}{C^{\frac{1}{\nu}} n(L)} \leq T_{BC}(L) \leq \dfrac{\lfloor \frac{\widehat{w}}{w_b} \rfloor \tau\nu}{n(L)}$ or $\dfrac{\tau^{\frac{1}{\nu}}(\nu-1)^{\frac{1}{\nu}-1}\nu\widehat{w}}{C^{\frac{1}{\nu}} n(L)} \leq T_{BC}(L) \leq \dfrac{\lceil \frac{\widehat{w}}{w_b} \rceil \tau\nu}{n(L)}$. Also, if $\frac{\widehat{w}}{w_b}$ is an integer, then $y^* = \frac{\widehat{w}}{w_b}$.*

Proof. Let $y = \sum_{i=1}^{n(L)} x_i$. We first show that the optimal solution always guarantees that all s_i are identical, say $s_i = s_1^*$, and $\sum_{i=1}^{n(L)} w_i \frac{s_1^{*\nu-1}}{C} = y$. The Lagrange function of **BC** is

$$\mathcal{L}(s, x, \lambda, \mu) = \sum_{i=1}^{n(L)} \left(\frac{w_i}{s_i} + \tau x_i \right) - \lambda \sum_{i=1}^{n(L)} \left(w_i \frac{s_i^{\nu-1}}{C} - x_i \right).$$

By first and second order conditions we have that $s_i = s_1^*$ (for $\forall i$) is the optimal solution. From the constraint $\frac{\widehat{w} s^{\nu-1}}{C} = y$, we have that $s = (\frac{Cy}{\widehat{w}})^{\frac{1}{\nu-1}}$. Then **BC** can by written as:

$$\min_{y \in \mathbb{Z}_+} \quad \frac{\widehat{w}}{(\frac{Cy}{\widehat{w}})^{\frac{1}{\nu-1}}} + y$$

Then the results in the theorem follow. □

Notice that **Lemma 2.4.3** and **Lemma 2.4.4** hold for any lists L. From **Lemma 2.4.4** we have if $\frac{\widehat{w}}{w_b}$ is an integer, then exactly $\frac{\widehat{w}}{w_b}$ tools are used for processing $\widehat{w}$ amount of workload, with each tool processing w_b amount of workload. In this case w_b gives the optimal workload a tool processes in **OFR** based only on tool parameters $C,\ \nu$ and τ. If we only have one job whose workload is w_b, both **OFR** and **BC** have the same optimal value. If we are able to rearrange a job list so that each tool processes exact w_b amount of workload, then **BC** and **OFR** gets the same optimal makespan for this list. But on what condition can we guarantee the existence of such a list?

Notice that if we add a constraint that specifies a maximal amount of workload a tool can process, say w_c, then this constrained **OFR** can be described as follows: we want to allocate jobs to tools, with each tool getting w_c workload so that the total makespan is minimized. This idea is quite similar to the bin packing problem (see [40]). In the bin packing context, we have several pieces to be packed into a few identical bins. In the one dimensional bin packing problem, we have several pieces with different lengths and bins of identical capacity/length. We want to pack pieces with minimal number of bins. In our paper, we can regard each tool as a bin with a specific capacity, where the capacity is given by the maximal workload it can process. We also regard jobs as pieces to be packed by bins, where the length of each piece is given by the workload. We will discuss how to find the optimal w_c later on, but once we fix w_c, the problem now is to allocate jobs of L to tools so as to use the least number of tools. As we can see, if all the tools are fully ‘‘packed" by jobs, which means each tool processes exact w_c amount of workload, we have the minimal number of tool replacement. Suppose $k_{opt}(L)$ is the minimal number of tools we use to process jobs in list L with rearranging under a fixed speed s^* where s^* is the solution of $w_c\frac{D(s^*)}{s^*} = h^*$. We say perfect packing happens when $w(L) = k_{opt}w_c$. We have the following theorem to show that **OFR** is asymptotically close to **BC** in a special case.

Theorem 2.4.5. *If workload w_i is drawn independently from a uniform distribution within $[0, w_u]$ and $w_u \leq w_b$, then as $n(L) \to \infty$, $\dfrac{\boldsymbol{E}[T_{OFR}(L)]}{\boldsymbol{E}[T_{BC}(L)]} \to 1$.*

Proof. Suppose we have a finite job list L. We fix $w_c = w_b$ and process all the jobs at speed s^*, where s^* is the solution of $w_b\frac{D(s^*)}{s^*} = h^*$. Once the speed is fixed, we only need to allocate jobs of L into tools whose capacities are w_b. In the end we find the minimal number of tools we used is $k_{opt}(L)$, in which $k_0(L)$ tools are fully packed. Let $k_1(L) = k_{opt}(L) - k_0(L)$. We know $k_0(L) = 0$ if no tool is fully packed. From **Lemma 2.4.3** we have $\dfrac{\mathbf{E}[k_0(L)]w_b}{s^*} + \mathbf{E}[k_0(L)]\tau \leq n(L)\mathbf{E}[T_{BC}(L)] \leq n(L)\mathbf{E}[T_{OFR}(L)] \leq \dfrac{\mathbf{E}[k_0(L)]w_b}{s^*} + \mathbf{E}[k_0(L)]\tau + \mathbf{E}[k_1(L)]\dfrac{w_b}{s^*} + \mathbf{E}[k_1(L)]\tau$. The last equality holds because rearranging guarantees the optimal decision among all the possible permutations. Thus

$$\frac{\mathbf{E}[T_{OFR}(L)]}{\mathbf{E}[T_{BC}(L)]} \leq 1 + \frac{\mathbf{E}[k_1(L)]\dfrac{w_b}{s^*} + \mathbf{E}[k_1(L)]\tau}{\dfrac{\mathbf{E}[k_0(L)]w_b}{s^*} + \mathbf{E}[k_0(L)]\tau} = 1 + \frac{\mathbf{E}[k_1(L)]}{\mathbf{E}[k_0(L)]}$$

We now prove that $\lim_{n(L)\to\infty} \dfrac{\mathbf{E}[k_1(L)]}{\mathbf{E}[k_0(L)]} = 0$ by contradiction. From [40] we know that once w_i is uniformly distributed within $[0, w_u]$, we have that $\lim_{n(L)\to\infty} \dfrac{\mathbf{E}[w(L)]}{\mathbf{E}[k_{opt}(L)w_b]} = 1$. Suppose that

$\lim_{n(L)\to\infty} \dfrac{\mathbf{E}[k_1(L)]}{\mathbf{E}[k_0(L)]} = \epsilon$, then

$$\begin{aligned}
\frac{\mathbf{E}[w(L)]}{\mathbf{E}[k_{opt}(L)w_b]} &= \frac{w_b\mathbf{E}[k_0(L)] + \mathbf{E}[w(L) - k_0(L)w_b]}{\mathbf{E}[k_{opt}(L)w_b]} \\
&\leq \frac{w_b\mathbf{E}[k_0(L)] + \mathbf{E}[k_1(L)w_b]}{\mathbf{E}[k_{opt}(L)w_b]} \\
&= \frac{w_b\mathbf{E}[k_0(L)] + \mathbf{E}[k_1(L)w_b]}{w_b\mathbf{E}[k_0(L) + k_1(L)]}.
\end{aligned}$$

The second inequality is an equality only when $k_1(L) = 0$, otherwise

$$\frac{\mathbf{E}[w(L)]}{\mathbf{E}[k_{opt}(L)w_b]} < \frac{w_b\mathbf{E}[k_0(L)] + \mathbf{E}[k_1(L)w_b]}{w_b\mathbf{E}[k_0(L) + k_1(L)]},$$

take the limit of both sides we have that $1 < 1$, getting a contradiction. Hence proved. □

So far we have acquired the asymptotic optimal value of **OFR** which can serve as the lower bound of **OF**. Also we have a benchmark that we can compare our online algorithms against. Now we discuss different ways to construct online algorithms by imitating the same speed property (**Theorem 2.4.1**) of **OFR**.

2.4.2 Fixed Speed Approach

When we have only one revealed workload in system, we have little information to help us make decisions. Using up one tool to process this job is not a good idea since replacement also takes time. We shall see the performance of this policy later in Section 5. We still want to keep some remaining tool level to process jobs in future. In this section we will discuss an online approach which relies on one revealed workload and keeps a reasonable amount of tool life for jobs in future.

As we observe from **OFR**, jobs are allocated to tools in a way which is similar to the bin packing problem. In the bin packing context, we have a policy called Next-Fit ([41]). In one dimensional space, assume pieces each of a specific $length$ come in a sequence, say $i = 1, 2, 3, ...$ We have infinite number of identical bins whose capacity is c and pieces are packed into bins one by one. We define the $level$ of a bin by the sum length of all the pieces in that bin. The Next-Fit tells us that if the length of piece i plus the current bin level is greater than c, we finish our current bin and pack piece i into a new bin. If we index our bin by $j = 1, 2, ...$, the Next-Fit can be described as: If $length_i + level_j < c$, choose bin $j + 1$ for piece i, otherwise choose bin $j + 1$.

We find that it may be a good idea to serve jobs in a Next-Fit way. In our problem, if we fix a processing speed for every job, say s^*, from constraint of $w\frac{D(s^*)}{s^*} = h^*$ we know that each tool can process at most $\frac{h^* s^*}{D(s^*)}$ amount of workload. We let $w_c = \frac{h^* s^*}{D(s^*)}$ be the tool capacity, which is the maximal workload a tool can process. Similarly, we regard each tool as a “bin" which we describe above by defining η_j as the workload tool j has already processed. The remaining tool level then becomes $w_c - \eta_j$ because each tool can process at most w_c amount of workload. Therefore, if we process jobs in a Next-Fit way, when $w_i + \eta_j \leq w_c$ for job i and tool j, we process job i with tool j at speed s^*, otherwise we process job i with tool $j + 1$ at speed s^*.

So the problem is how to find s^*. The speed should not be slow, otherwise the processing may be sluggish hence impacting the objective. On the other hand, from constraint $w_c \frac{D(s^*)}{s^*} = h^*$ and the monotonicity of $\frac{D(s)}{s}$, fast speed always results in a small w_c. If we assume workload of each job is bounded by w_u, then w_c should not be less than w_u, otherwise we would not be able to process jobs with workload w_u at speed s^*. Suppose w_c^* is the optimal tool capacity we fix to determine s^*, we call this the **Fixed Speed Approach** (**FS**). Denote $T_{FS}(L)$ as the processing time for list L under **FS**, and **FS** can be described by **Algorithm 2.1**.

Algorithm 2.1 Fixed Speed Approach

```
Speed $s^*$ is determined by $w_c \frac{D(s^*)}{s^*} = h^*$.
for i=1:$n(L)$ do
    if $h_i - D(s^*)\frac{w_i}{s^*} > 0$ then Keep the current tool and process job $i$ at speed $s^*$;
    else Replace the tool and then process job $i$ at speed $s^*$ using the new tool;
    end if
    if i==$n(L)$ then Replace the tool;
    end if
end for
```

Theorem 2.4.6. *If processing speed is fixed at* $s^* = \left(\dfrac{C}{w_c^*}\right)^{\frac{1}{\nu-1}}$ *and* $w_c^* \geq w_u$ *where* w_c^* *is the optimal tool capacity, then* $\dfrac{\boldsymbol{E}[T_{BC}(L)]}{\boldsymbol{E}[T_{FS}(L)]} \in [\dfrac{(1-\beta)\nu}{(\nu-1)(1-\beta)+1}, 1]$*, where* $1-\beta = \dfrac{\boldsymbol{E}[\eta_i]}{w_c^*}$*, as* $n(L) \to \infty$.

Proof. We first show the result by fixing $w_c = w_b$, i.e. $w_c = \left(\tau(\nu-1)C^{\frac{1}{\nu-1}}\right)^{\frac{\nu-1}{\nu}}$, though we know there should be a better choice for w_c. In this case, the speed is fixed at $s^0 = \left(\dfrac{C}{w_b}\right)^{\frac{1}{\nu-1}}$. Suppose under **FS**, k tools have been used to process jobs in list L and tool $(k+1)$ is just about to be put into use for processing a new job outside of L and this job cannot be processed by tool k under speed s^0. At this point, $n(L)T^0_{FS}(L) = \sum_{i=1}^{k} \dfrac{\eta_i}{s^*} + k\tau$. Workload served by these k tools is given by $w(k) = \sum_{i=1}^{k} \eta_i$. When $w(k)$ workload is done under **BC**, from **Theorem 2.4.4** we know that

$$\frac{\tau^{\frac{1}{\nu}}(\nu-1)^{\frac{1}{\nu}-1}\nu\widehat{w}}{C^{\frac{1}{\nu}}n(L)} \leq T_{BC}(L) \leq \frac{(\frac{\widehat{w}}{w_b}+1)\tau\nu}{n(L)}.$$

From [41] we know that $\{\eta_i\}$ is a continuous Markov Chain, so we know that when $k \to \infty$ we have $\lim_{k\to\infty} \dfrac{\mathbf{E}[w(k)]}{k} = \mathbf{E}[\eta_i] = (1-\beta)w_b$. Suppose w_0 is the first workload served by the last tool, then

$$\begin{aligned}
\lim_{n(L)\to\infty} \frac{\mathbf{E}[T_{BC}(L)]}{\mathbf{E}[T^0_{FS}(L)]} &\geq \lim_{n(L)\to\infty} \frac{\tau^{\frac{1}{\nu}}(\nu-1)^{\frac{1}{\nu}-1}\nu\mathbf{E}[w(k)]}{C^{\frac{1}{\nu}}\mathbf{E}[\sum_{i=1}^{k-1}\frac{\eta_i}{s^*}+\frac{w_b}{s^*}+k\tau]} \\
&\geq \lim_{k\to\infty} \frac{\tau^{\frac{1}{\nu}}(\nu-1)^{\frac{1}{\nu}-1}\nu\mathbf{E}[\sum_{i=1}^{k-1}\eta_i+w_0]}{C^{\frac{1}{\nu}}\mathbf{E}[\sum_{i=1}^{k-1}\frac{\eta_i}{s^*}+\frac{w_b}{s^*}+k\tau]} = \frac{(1-\beta)\nu}{(1-\beta)(\nu-1)+1}.
\end{aligned}$$

Now, to show $\lim_{n(L)\to\infty} \dfrac{\mathbf{E}[T_{BC}(L)]}{\mathbf{E}[T^0_{FS}(L)]} \leq \dfrac{(1-\beta)\nu}{(1-\beta)(\nu-1)+1}$, we have

$$\begin{aligned}
\lim_{n(L)\to\infty} \frac{\mathbf{E}[T_{BC}(L)]}{\mathbf{E}[T^0_{FS}(L)]} &\leq \lim_{n(L)\to\infty} \frac{\mathbf{E}[\frac{w(k)}{w_b}+1]\tau\nu}{\mathbf{E}[\sum_{i=1}^{k-1}\frac{\eta_i}{s^*}+\frac{w_0}{s^*}+k\tau]} \\
&= \lim_{k\to\infty} \frac{(\mathbf{E}[\sum_{i=1}^{k-1}\frac{\eta_i}{w_b}+1)\tau\nu]}{\mathbf{E}[\sum_{i=1}^{k-1}\frac{\eta_i}{s^*}+\frac{w_0}{s^*}+k\tau]} \\
&= \frac{(1-\beta)\nu}{(1-\beta)(\nu-1)+1}.
\end{aligned}$$

We know we can definitely find other better options for w_c, thus proving the theorem. $\square$

Corollary 2.4.7. *If* $w_u = w_b$ *and* $w_c = w_b$*, which means* w_i *is drawn from* $U(0, w_b)$ *and tool capacity is also* w_b*, we have that* $\lim_{n(L)\to\infty} \dfrac{\boldsymbol{E}[T_{BC}(L)]}{\boldsymbol{E}[T_{FS}(L)]} = \dfrac{3\nu}{3\nu+1}$.

Proof. From [41] we know that $\mathbf{E}[\eta] = \frac{3}{4}w_u$, and by **Theorem 2.4.6** we have the result. $\square$

Notice in **FS**, w_c is not necessarily equal to w_b. That is to say, w_b is not the optimal option to fix speed by solving $w_c\frac{D(s)}{s} = h^*$. Here we did not give the actual optimal w_c. Although the cdf of bin level is given by [40], the expectation of bin level is difficult to characterize analytically. Numerical tests are provided in the Section 2.5.

If we do not reveal the workload upon arrival and $w_u = w_b$, the optimal policy is to process each job by exhausting one tool. We call this policy the **Stochastic Optimal Policy (SOP)**. The following corollary provides the performance of this policy.

Corollary 2.4.8. *If* $w_u = w_b$ *and no workload is revealed upon arrival, the optimal speed is determined by* $w_b\dfrac{D(s)}{s} = h^*$*. Then we have* $\lim_{n(L)\to\infty} \dfrac{\boldsymbol{E}[T_{BC}(L)]}{\boldsymbol{E}[T_{SOP}]} = \dfrac{\nu}{\nu+1}$.

Intuitively we can guess that **FS** performs better than **SOP** since **FS** has more information (**FS** reveals one workload and **SOP** reveals none). We will compare the performance of **FS** and **SOP** numerically in Section 2.5. **FS** is simple as it does not require much computation or storage space. It does not rely on workload distribution either, since w_c can be preassigned with only information of w_u (notice that $w_c \geq w_u$). However, as stated before, we have to fix w_c such that it is no smaller than w_u, the upper bound of workload of each job. In the case when w_u is large, w_c must be larger than w_b. Intuitively when w_c is much larger than w_b, **FS** would not be promising. We will show numerical results in Section 2.5. Another drawback of this method is that tools sometimes are discarded though they have not been used up. They are replaced just because the new job has larger workload than the tool can handle under a fixed speed. This implies that **FS** can be definitely improved when there are more than one job in system, so we introduce an algorithm which makes full use of every tool in Section 2.4.3.

2.4.3 Fixed Buffer Approach

In the case where more than one workload is revealed, we may consider another way to better imitate the optimal solution to **OF**. Notice that in **FS** we fix a speed for all jobs, however the solution of **OF** only tells us to fix the speed for the same tool. In other words, once we know how much workload a tool processes, we can then fix the speed accordingly. This is the idea for **Fixed Buffer Approach (FB)** that we will provide in this subsection.

2.4.3.1 Bounded Workload

We first show the performance of **FB** in the case when w_u is bounded. The idea of **FB** in the bounded workload case is very similar to **FS** that we discuss in Section 2.4.2. To see this, we suppose there is a sequence of virtual buffers in front of our machine. Jobs entering the system are not processed instantly one by one, but put into the first buffer. We start processing jobs in this buffer when it is unable to accept more jobs, and put new arrivals into the next buffer, the same as Next-Fit described in Section 2.4.2. Buffers are indexed by $j = 1, 2, ...$ We abuse our notation by defining the capacity of a buffer to be w_c, which is the maximal amount of workload a buffer could hold. Note this is different from the maximal workload that a tool can process that we defined in Section 2.4.1 and 2.4.2. Here we mainly use w_c to denote a threshold because want **FB** to be applied in a more generic case (Section 4.3.2) in which workload may be unbounded. Again, we use η_j to denote the $level$ of buffer j, which means the amount of workload that buffer j already received. So the buffer packing procedure can be described as following: we pack job i into buffer $j+1$ if $w_i + \eta_j > w_c$, and process jobs in buffer j at speed s_j where s_j is the solution to $\eta_j \dfrac{D(s_j)}{s_j} = h^*$. The machine serves the buffers one by one, with one tool serving only one buffer. We call it **Fixed Buffer Approach** (**FB**). As we can see, **FB** can make full use of every tool. No tool would be replaced with non-zero level. **FB** is described in **Algorithm 2.2**.

Notice in Algorithm 2.2, **FB** can also deal with the case when workload is unbounded, we leave the discussion for unbounded workload case to Section 4.3.2. Here we mainly focus on discussion for bounded workload cases.

Algorithm 2.2 Fixed Buffer Approach

```
for i=1:$n(L)$ do
    if $\eta + w_i \leq w_c$ then Put job $i$ into the buffer and $\eta \leftarrow \eta + w_i$;
    else if $\eta + w_i \geq w_c$ && $w_i < w_c$ then Process jobs in buffer with speed $s$ determined by $\eta\frac{D(s)}{s} = h^*$; $\eta \leftarrow w_i$;
    else Process jobs in buffer with speed $s$ determined by $\eta\frac{D(s)}{s} = h^*$; Replace; Process job $w_i$ with speed $s_i$ determined by $w_i\frac{D(s_i)}{s_i} = h^*$; Replace; $\eta \leftarrow 0$;
    end if
    if i==$n(L)$ then Process jobs in buffer with speed $s$ determined by $\eta\frac{D(s)}{s} = h^*$; Replace the tool;
    end if
end for
```

Now the problem is how to fix the buffer's capacity. Notice that if $w_u \leq w_b$, then $w_c = w_b$ may be a good option since perfect packing is achievable in this case. When $w_u > w_b$, we may try fixing $w_c = w_u$. Again, here we assume that workload is uniformly distributed within $[0, w_u]$. We choose $w_c = \max\{w_b, w_u\}$. In this case we denote processing time of job list L under **FB** as $T^0_{FB}(L)$. Notice that this choice of w_c is not optimal. However, by showing the performance of this we will find the lower bound that **FB** can achieve. Notice that the solution of $T_{BC}(L)$ may not be achievable by **OFR** when $w_u > w_b$. Definitely we can use **BC** as a benchmark but it may perform far from **OFR**. To find the optimal value of **OFR**, we provide the following lemma.

Lemma 2.4.9. *Let $D(s) = \frac{s^\nu}{C}h^*$, given a list of jobs L where there is a subset of L in which each workload is less than or equal to w_b, say L_1, if L_1 can be perfectly put into buffers each of capacity w_b, then the optimal solution to **OFR** contains two parts: jobs in L_1 are perfectly packed into buffers thus $T_{OFR}(L_1) = T_{BC}(L_1)$, and jobs in $L_0 = L \setminus L_1$ each is served by one tool thus $T_{OFR}(L_0) = T_{OF}(L_0)$.*

Proof. We prove our theorem by contradiction. Suppose $L^* \in R(L)$ is the optimal order for **OFR**, which means the optimal value of L^* by solving **OF** is less than the optimal value of any other lists in $R(L)$. First we prove that in the optimal solution to the $T_{OF}(L^*)$, it is impossible that one tool serves more than one jobs whose workload is greater than w_b. Suppose that $w_1 \geq w_2 \geq w_b$ and w_1

and w_2 are served by the same tool. Then the processing time plus replacement time is

$$\begin{aligned}\frac{(w_1+w_2)^{\frac{\nu}{\nu-1}}}{C^{\frac{1}{\nu-1}}}+\tau &\geq \frac{w_1^{\frac{\nu}{\nu-1}}+\frac{\nu}{\nu-1}w_1^{\frac{1}{\nu-1}}w_2}{C^{\frac{1}{\nu-1}}}+\tau \\ &\geq \frac{w_1^{\frac{\nu}{\nu-1}}+\frac{\nu}{\nu-1}w_2^{\frac{\nu}{\nu-1}}}{C^{\frac{1}{\nu-1}}}+\tau \\ &\geq \frac{w_1^{\frac{\nu}{\nu-1}}+w_2^{\frac{\nu}{\nu-1}}}{C^{\frac{1}{\nu-1}}}+\frac{1}{\nu-1}\frac{w_b^{\frac{\nu}{\nu-1}}}{C^{\frac{1}{\nu-1}}}+\tau=\frac{w_1^{\frac{\nu}{\nu-1}}+w_2^{\frac{\nu}{\nu-1}}}{C^{\frac{1}{\nu-1}}}+2\tau.\end{aligned}$$

Since the last expression of the above inequalities is the processing time when w_1 and w_2 done by two tools separately, then we cannot have the case where two large jobs are served by one tool.

Now suppose that in list L^*, when getting the optimal solution of **OF**, there are elements in L_0, say w_{0j}, which are served with some of the elements in L_1 by same tools. Since we only consider the workload served by tools, the tools that serve L_0 are serving $\sum_{j=1}^{n(L_0)}(w_{0j}+\gamma_j)$ in total, where γ_j denotes the workload from L_1 which is served with w_{0j} by the same tool. Let $\widehat{w}=w(L)-\sum_{j=1}^{n(L_0)}(w_{0j}+\gamma_j)$. Then the time spent on L^* under **OF** is

$$n(L^*)T_{OF}(L^*) \geq \sum_{j=1}^{n(L_0)}\frac{(w_{0j}+\gamma_j)^{\frac{\nu}{\nu-1}}}{C^{\frac{1}{\nu-1}}}+n(L_0)\tau+\frac{\widehat{w}^{\frac{\nu}{\nu-1}}}{(C\frac{\widehat{w}}{w_b})^{\frac{1}{\nu-1}}}+\tau\frac{\widehat{w}}{w_b}.$$

The inequality holds since given any list L, $T_{OF}(L)\geq T_{BC}(L)$. Let $\gamma=\sum_{i=1}^{n(L_0)}\gamma_i$. Suppose $T_{B1}(L^*)$ is the processing time by using the way described in **Lemma 2.4.9**, i.e.,

$$n(L^*)T_{B1}(L^*)=\frac{\widehat{w}+\gamma}{C^{\frac{1}{\nu}}}\nu(\nu-1)^{\frac{1-\nu}{\nu}}\tau^{\frac{1}{\nu}}+\sum_{j=1}^{n(L_0)}\frac{w_{0j}^{\frac{\nu}{\nu-1}}}{C^{\frac{1}{\nu-1}}}+n(L_0)\tau.$$

From the fact that $\frac{\widehat{w}}{C^{\frac{1}{\nu}}}\nu(\nu-1)^{\frac{1-\nu}{\nu}}\tau^{\frac{1}{\nu}}=\frac{\widehat{w}^{\frac{\nu}{\nu-1}}}{(C\frac{\widehat{w}}{w_b})^{\frac{1}{\nu-1}}}+\tau\frac{\widehat{w}}{w_b}$, we have

$$n(L^*)\left(T_{OF}(L^*)-T_{B1}(L^*)\right)=\sum_{j=1}^{n(L_0)}\frac{(w_{0j}+\gamma_j)^{\frac{\nu}{\nu-1}}}{C^{\frac{1}{\nu-1}}}-\frac{\gamma}{C^{\frac{1}{\nu}}}\nu(\nu-1)^{\frac{1-\nu}{\nu}}\tau^{\frac{1}{\nu}}-\sum_{j=1}^{n(L_0)}\frac{w_{0j}^{\frac{\nu}{\nu-1}}}{C^{\frac{1}{\nu-1}}}.$$

From Taylor's expansion and $w_j > w_b$ we have

$$(w_{0j} + \gamma_j)^{\frac{\nu}{\nu-1}} \quad \geq \quad w_{0j}^{\frac{\nu}{\nu-1}} + \frac{\nu}{\nu-1} w_{0j}^{\frac{1}{\nu-1}} \gamma_j > w_{0j}^{\frac{\nu}{\nu-1}} + \frac{\nu}{\nu-1} \left((\nu-1)\tau C^{\frac{1}{\nu-1}} \right)^{\frac{1}{\nu}} \gamma_j.$$

Thus we show that $\sum_{j=1}^{k} \frac{(w_0 j + \gamma_j)^{\frac{\nu}{\nu-1}}}{C^{\frac{1}{\nu-1}}} - \frac{\gamma}{C^{\frac{1}{\nu}}} \nu(\nu-1)^{\frac{1-\nu}{\nu}} \tau^{\frac{1}{\nu}} - \sum_{j=1}^{k} \frac{w_{0j}^{\frac{\nu}{\nu-1}}}{C^{\frac{1}{\nu-1}}} > 0$. Hence we can see that $T_{OF}(L^*) - T_{B1}(L^*) > 0$, which means that L^* is not the best order. $\square$

Now suppose we have a list L in which workload of each job is uniformly distributed within $[0, w_u]$ where $w_u > w_b$. Extract the elements whose workload are greater than w_b from L to form a new list L_0. From the remaining elements we extract those that can be perfectly packed by tools with capacity w_b and form them as list L_1. The other elements are gathered to be list L_2. Notice we do not consider the order of these lists since we only discuss the rearranging method here.

Given any list L, we have that $n(L)T_{OFR}(L) \leq n(L_1 \cup L_2)T_{OFR}(L_1 \cup L_2) + n(L_0)T_{OFR}(L_0)$ since rearranging always guarantees the optimal order of the entire list L. Also notice that $T_{OFR}(L_1) = T_{BC}(L_1)$, from **Lemma 2.4.9** we have $n(L)T_{OFR}(L) \geq n(L_1 \cup L_0)T_{OFR}(L_1 \cup L_0) = n(L_1)T_{OFR}(L_1) + n(L_0)T_{OFR}(L_0)$. Notice that $L_1 \cup L_2$ contains all the elements whose workload is less than or equal to w_b, and L_0 contains all elements whose workload are greater than w_b. We want to get the expected number of jobs in these sets when given $n(L)$. We have the following lemma, for which we define W as the workload of a job arriving in the future:

Lemma 2.4.10. ***P***$(W \leq a | W > w_b)$ *is the cdf of a uniform distribution in* $[w_b, w_u]$ *if* W *is uniformly distributed in* $[0, w_u]$.

Proof. Consider the conditional cdf when $a > w_b$

$$\mathbf{P}(W \leq a | W > w_b) = \frac{\mathbf{P}(w_b < W < a)}{\mathbf{P}(W > w_b)} = \frac{a - w_b}{w_u - w_b}.$$

If $a \leq w_b$ we have that $\mathbf{P}(W < a | W > w_b) = 0$. $\square$

Similarly, we can show that $\mathbf{P}(W \leq a | W < w_b)$ is also a cdf for a uniform distribution in

$[0, w_b]$. Let $\theta = \frac{w_u}{w_b}$ then $\mathbf{E}[n(L_1 \cup L_2)|n(L)] = \frac{1}{\theta}n(L)$ and $\mathbf{E}[n(L_0)|n(L)] = \frac{\theta-1}{\theta}n(L)$. To characterize $\mathbf{E}[T_{OFR}(L_1 \cup L_2)]$, from **Lemma 2.4.10** we know that the elements in $L_1 \cup L_2$ have uniform distribution with upper bound w_b. From **Theorem 2.4.5** as $n(L_1 \cup L_2) \to \infty$, $\mathbf{E}[T_{OFR}(L_1 \cup L_2)] \to \mathbf{E}[T_{BC}(L_1 \cup L_2)]$. Also from **Lemma 2.4.4** we know that $\lim_{n(L_1 \cup L_2)\to\infty} \mathbf{E}[T_{BC}(L_1 \cup L_2)] \le \lim_{n(L_1 \cup L_2)\to\infty} \dfrac{(\frac{\mathbf{E}[w(L_1 \cup L_2)]}{w_b} + 1)\tau\nu}{n(L_1 \cup L_2)} = \dfrac{\tau\nu}{2}$, and

$$\lim_{n(L_1 \cup L_2)\to\infty} \mathbf{E}[T_{BC}(L_1 \cup L_2)] \ge \lim_{n(L_1 \cup L_2)\to\infty} \frac{\tau^{\frac{1}{\nu}}(\nu-1)^{\frac{1}{\nu}-1}\nu\mathbf{E}[w(L_1 \cup L_2)]}{C^{\frac{1}{\nu}}n(L_1 \cup L_2)} = \frac{\tau\nu}{2}.$$

Thus $\lim_{n(L_1 \cup L_2)\to\infty} \mathbf{E}[T_{BC}(L_1 \cup L_2)] = \dfrac{\tau\nu}{2}$.

Now we wish to show that

$$\lim_{n(L)\to\infty} \frac{\mathbf{E}[n(L_1)T_{BC}(L_1)|n(L)]}{n(L)} = \lim_{n(L)\to\infty} \frac{\mathbf{E}[n(L_1 \cup L_2)T_{BC}(L_1 \cup L_2)|n(L)]}{n(L)}.$$

Since $\dfrac{n(L_1)}{n(L)}T_{BC}(L_1) \le \dfrac{n(L_1 \cup L_2)}{n(L)}T_{BC}(L_1 \cup L_2)$, and its RHS has the property that $\dfrac{n(L_1 \cup L_2)}{n(L)}T_{BC}(L_1 \cup L_2) \le \dfrac{n(L_1)}{n(L)}T_{BC}(L_1) + \dfrac{n(L_2)}{n(L)}T_{BC}(L_2)$, we want to show that

$$\lim_{n(L)\to\infty} \frac{\mathbf{E}[n(L_2)|n(L)]}{n(L)} = 0.$$

Otherwise assume that $\lim_{n(L)\to\infty} \dfrac{\mathbf{E}[n(L_2)|n(L)]}{n(L)} = \delta > 0$ holds, then

$$\lim_{n(L_1 \cup L_2)\to\infty} \frac{\mathbf{E}[w(L_1 \cup L_2)|n(L)]}{\mathbf{E}[k_{opt}(L_1 \cup L_2)|n(L)]} = \lim_{n(L)\to\infty} \frac{\mathbf{E}[w(L_1 \cup L_2)|n(L)]}{\mathbf{E}[k_{opt}(L_1) + k_{opt}(L_2)|n(L)]} < w_b.$$

Thus $\lim_{n(L)\to\infty} \dfrac{\mathbf{E}[n(L_2)T_{BC}(L_2)|n(L)]}{n(L)} = 0$.

Then we have that

$$\begin{aligned}\lim_{n(L)\to\infty}\mathbf{E}[T_{OFR}(L)] &= \lim_{n(L_1\cup L_2)\to\infty}\frac{1}{\theta}\mathbf{E}[T_{BC}(L_1\cup L_2)] + \lim_{n(L_0)\to\infty}\frac{\theta-1}{\theta}\mathbf{E}[T_{OF}(L_0)] \\ &= \frac{\tau\nu}{2\theta} + \lim_{n(L_0)\to\infty}\frac{\theta-1}{\theta}\mathbf{E}[T_{OF}(L_0)].\end{aligned}$$

Next we wish to evaluate $\lim_{n(L_0)\to\infty}\mathbf{E}[T_{OF}(L_0)]$. Let $m = \frac{\nu}{\nu-1}$ and W be uniformly distributed in $[0, w_u]$, then

$$\begin{aligned}\mathbf{E}[W^m|W > w_b] &= \int_{w_b}^{w_u}\frac{1}{w_u - w_b}x^m dx \\ &= \frac{1}{(m+1)(w_u - w_b)}\left(w_u^{m+1} - w_b^{m+1}\right) \\ &= \frac{1}{m+1}\frac{1}{\theta-1}w_b^m(\theta^{m+1}-1).\end{aligned}$$

Then we have $\lim_{n(L_0)\to\infty}\mathbf{E}[T_{OF}(L_0)] = \dfrac{1}{m+1}\dfrac{1}{\theta-1}\dfrac{w_b^m}{C^{\frac{1}{\nu-1}}}(\theta^{m+1}-1)+\tau$. Thus as $n(L)\to\infty$,

$$\mathbf{E}[T_{OFR}(L)] \to \left(\frac{\nu}{2\theta} + \frac{\theta-1}{\theta} + \frac{\nu-1}{\theta(m+1)}(\theta^{m+1}-1)\right)\tau.$$

Lemma 2.4.11. *Let $f_L(x)$ be the probability density function of level of each tool, if capacity of each tool is w_u and the workload of each job is uniformly distributed in $[0, w_u]$, then $f_L(x) = \dfrac{3x^2}{w_u^3}$ where $x \in [0, w_u]$.*

Proof. Proof is given by [41]. □

From **Lemma 2.4.11** we have that

$$\mathbf{E}[\eta^m] = \int_0^{w_u}\frac{3x^{m+2}}{w_u^3}dx = \frac{3}{3+m}w_u^m.$$

Thus $\lim_{n(L)\to\infty}\mathbf{E}[T_{FB}^0(L)] = \dfrac{2}{3}(\dfrac{3}{3+m}\dfrac{\theta^m w_b^m}{C^{\frac{1}{\nu-1}}}+\tau)$.

Then as $n(L) \to \infty$, $\dfrac{\mathbf{E}[T_{OFR}(L)]}{\mathbf{E}[T^0_{FB}(L)]} \to \dfrac{\nu - 2 + 2\theta + \dfrac{2(\nu-1)^2}{2\nu-1}(\theta^{m+1}-1)}{\dfrac{4(\nu-1)^2}{4\nu-3}\theta^{m+1} + \frac{4}{3}\theta}$.

Remark 2.4.12. If workload of each job is uniformly distributed within $[0, w_u]$ and $\theta = \frac{w_u}{w_b} \geq 1$, suppose $w_c^* \geq w_u$ is the optimal tool capacity to be fixed in **FB**, then as $n(L) \to \infty$,

$$\frac{\mathbf{E}[T_{OFR}(L)]}{\mathbf{E}[T_{FB}(L)]} \in [\frac{\nu - 2 + 2\theta + \dfrac{2(\nu-1)^2}{2\nu-1}(\theta^{m+1}-1)}{\dfrac{4(\nu-1)^2}{4\nu-3}\theta^{m+1} + \frac{4}{3}\theta}, 1].$$

Notice we do not use $T_{BC}(L)$ here as frame of reference anymore since it may not be achievable when $w_u > w_b$.

Remark 2.4.13. We use uniform distribution because in Section 2.4.2 we showed that the tool level or the buffer level η is determined by $\frac{w_u}{w_b}$. When tool replacement time is large, w_b is large also. Uniform distribution in this case provides us with an extreme case where w_u can be at least as large as w_b. Results for some other distributions are shown in Section 2.5.

As we can see, if we fix buffer capacity as w_c when $w_u \leq w_c$, both **FB FS** process at most w_c workload. The only difference is that if both **FB** and **FS** process the same amount of workload, say $\widehat{\eta} \leq w_c$, using the same tool, **FB** will process at speed $\widehat{s}$, which is the solution of $\widehat{\eta}\dfrac{D(\widehat{s})}{\widehat{s}} = h^*$, whereas **FS** processes it at speed s^*, where s^* satisfies $w_c\frac{D(s^*)}{s^*} = h^*$. Notice that $\widehat{\eta} \leq w_c$ and the fact that $\dfrac{D(s)}{s}$ is an increasing function of s, we have that $\widehat{s} \geq s^*$. In this case we can see that $T_{FB}(L) \leq T_{FS}(L)$ for any list L. Then we have the following theorem.

Theorem 2.4.14. *For **FS** we fix a speed s^* as the solution of $w_c\frac{D(s^*)}{s^*} = h^*$, for **FB** we fix an buffer capacity also as w_c and use s^* to process jobs in the last buffer, then $T_{FS}(L) \geq T_{FB}(L) \geq T_{OF}(L) \geq T_{OFR}(L) \geq T_{BC}(L)$ for any L.*

Proof. It is easy to observe that $T_{FS}(L) \geq T_{FB}(L)$ and $T_{OF}(L) \geq T_{OFR}(L)$, and $T_{BC}(L)$ would be the smallest as it is the best case. We only need to prove that $T_{FB}(L) \geq T_{OF}(L)$. From **FB** we know its processing speed is determined by $\eta\frac{D(s)}{s} = h^*$, thus remaining tool level cannot be negative. Then it is easy to see that any feasible solutions of **FB** are in the feasible region of **OF**. Thus $T_{FB}(L) \geq T_{OF}(L)$. □

In previous sections we do not consider the waiting process or the queue in the systems since stability does not hamper the performance of our approaches. We now characterize the capacity of the system under different approaches. For **FS** with capacity w_c, the service rate for list L is $\frac{w(L)}{n(L)T_{FS}(L)}$. Notice that if $k+1$ tools are used for processing L and η_i is the workload served by tool i, then the service rate can be given by $\frac{w(L)}{n(L)T_{FS}(L)} = \frac{\sum_{i=1}^{k}\eta_i+\eta_{k+1}}{\sum_{i=1}^{k+1}\frac{\eta_i}{s}+(k+1)\tau} \leq \frac{(k+1)w_c}{\frac{(k+1)w_c}{2s}+(k+1)\tau}$, where $s = \left(\frac{C}{w_c}\right)^{\frac{1}{\nu-1}}$. By dominated convergence theorem and strong law of large numbers, we have that as $n(L) \to \infty$, $\mathbf{E}[\frac{w(L)}{n(L)T_{FS}(L)}] \to \frac{(1-\beta)w_c}{\frac{(1-\beta)w_c^{\frac{\nu}{\nu-1}}}{C^{\frac{1}{\nu-1}}}+\tau} = \frac{(1-\beta)w_c C^{\frac{1}{\nu-1}}}{(1-\beta)w_c^{\frac{\nu}{\nu-1}}+\tau C^{\frac{1}{\nu-1}}}$, where $1-\beta = \mathbf{E}[\eta]$. For the optimal capacity w_c^* for **FS** we have $\mathbf{E}[\frac{w(L)}{n(L)T_{FS}(L)}] \in [\frac{(1-\beta)w_b C^{\frac{1}{\nu-1}}}{(1-\beta)w_b^{\frac{\nu}{\nu-1}}+\tau C^{\frac{1}{\nu-1}}}, \frac{w_b}{\nu\tau}]$ knowing that **BC** provides the maximal service rate and w_c^* provides higher service rate than w_b does for **FS**. Thus an arrival rate no greater than $\frac{(1-\beta)w_c C^{\frac{1}{\nu-1}}}{(1-\beta)w_c^{\frac{\nu}{\nu-1}}+\tau C^{\frac{1}{\nu-1}}}$ is the necessary condition for stability of **FS**. Likewise, for capacity of **FB**, we have as $n(L) \to \infty$, $\mathbf{E}[\frac{w(L)}{n(L)T_{FB}(L)}] \to \frac{\mathbf{E}[\eta]}{\frac{\mathbf{E}[\eta^{\frac{\nu}{\nu-1}}]}{C^{\frac{1}{\nu-1}}}+\tau} = \frac{\mathbf{E}[\eta]C^{\frac{1}{\nu-1}}}{\mathbf{E}[\eta^{\frac{\nu}{\nu-1}}]+\tau C^{\frac{1}{\nu-1}}}$. Specifically, when $W_i \sim U(0, w_u)$, $w_u > w_b$ and based on the optimal capacity, we have the arrival rate up to $\frac{\frac{3}{4}w_u C^{\frac{1}{\nu-1}}}{\frac{3\nu-3}{3\nu-2}w_u^{\frac{\nu}{\nu-1}}+\tau C^{\frac{1}{\nu-1}}}$ as the necessary condition of the stability for **FB**.

2.4.3.2 Unbounded Workload

Here we discuss the case when workload is unbounded. In this case, we may receive jobs with workload greater than w_c so that a single buffer cannot hold it. However, Taylor's Formula in Equation (2.1) tells us to use a relative low speed to process a large job. **FB** can adjust the processing speed accordingly without putting the large job in a "virtual buffer", as stated in Algorithm 2.2. Jobs with small workload are still put into buffers one by one. Once a buffer is full, **FB** processes jobs in this buffer using the optimal speed, and new jobs are put into the next buffer. Once a job with workload $w_i > w_c$ arrives, we know this job cannot be put into either the current buffer or a new buffer whose capacity is w_c. In this case **FB** would firstly process jobs in the current buffer using the optimal speed, then process this large job using one tool by speed s_i, where s_i is the solution to $w_i \frac{D(s_i)}{s_i} = h^*$.

In the unbounded workload case, the optimal value of **OFR** can no longer be approximated by **BC**, thus we can only use **BC** as the benchmark. However **BC** may not be a good benchmark because a real workload greater than w_b can never be packed in a buffer with capacity $w_c = w_b$ (See Section 2.4.1 and Theorem 2.4.5 for detail). We thus provide a modified **BC** to approximate the optimal makespan of $T_{OFR}(L)$. For simplicity in notation, we denote the modified problem **BC***. Similar to Lemma 6, we let L_1 be the subset of L containing jobs whose workload is smaller than w_b and $L_0 = L \backslash L_1$. Let $k^* = \lfloor \frac{w(L_1)}{w_b} \rfloor$, then from Lemma 3 we have $k^* \tau\nu \leq n(L_1) T_{BC}(L_1) < (k^* + 1)\tau\nu$. We let $T_{BC*}(L) = k^*\tau\nu + T_{OFR}(L_0)$, and by Lemma 6 we have $T_{BC^*}(L) \leq T_{OFR}(L) < (k^* + 1)\tau\nu + T_{OFR}(L_0)$. Then $\lim_{n(L)\to\infty} \dfrac{\mathbf{E}[T_{BC*}(L)]}{\mathbf{E}[T_{OFR}(L)]} = 1$ if we assume workload is uniformly distributed in $(0, w_u)$, This implies that we can also use **BC*** to approximate **OFR** for the results in Section 4.3.1. Note that without the uniform distribution assumption, we still have $T_{BC^*}(L) \leq T_{OFR}(L)$. We then use **BC*** as the benchmark to compare **FB** against. The numerical study of this case is provided in Section 2.5.

2.5 Numerical Results

In this section we perform numerical experiments to develop insights in terms of performance for various model parameters as well as compare methods. Since in Section 2.4.2 and 2.4.3 we only give the optimal bound for both **FS** and **FB**, now we will study their performance using simulation. Notice that in **Theorem 2.4.6**, the expression $\lim_{n(L)\to\infty} \dfrac{\mathbf{E}[T_{BC}(L)]}{\mathbf{E}[T_{FS}(L)]}$ depends only on ν and β, where ν is determined by Taylor's Formula (2.1) which is also shown in Equation (2.2) and β is determined by $\frac{w_u}{w_b}$, knowing that w_b is a constant given in **Lemma 2.4.4** and w_u is the upper bound of each workload. Similarly, the expression $\lim_{n(L)\to\infty} \dfrac{\mathbf{E}[T_{BC}(L)]}{\mathbf{E}[T_{FB}(L)]}$ in **Theorem 2.4.9** depends only on ν and θ, where θ is determined by w_u. So we start by testing the performance of **FS** and **FB** for different ν and w_u. For numerical results, we choose $C = 100$ and $\tau = 100$. Once C and τ are fixed, w_b depends only on ν as shown in **Lemma 2.4.4**. All the numerical tests are done in MATLAB and $T_{OF}(L)$ is solved by Bonmin solver (see [42]).

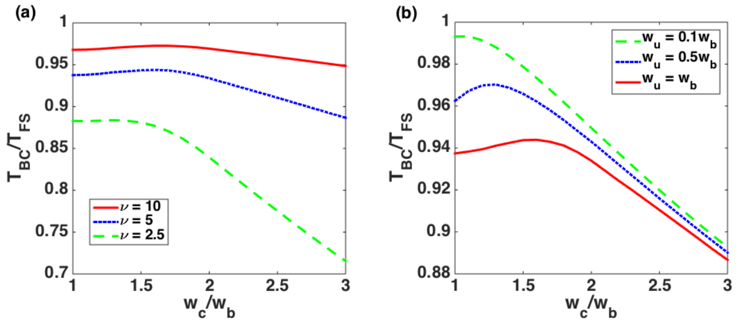


Figure 2.1: Performance of Fixed Speed Approach when (a) $w_u = w_b$, (b) $\nu = 5$

We begin by considering the influence of ν on **FS**. From [36] we can get some typical ν for several common tool materials, which is provided in the Table 2.3. We let $n(L) = 10^5$, a finite but large number which helps show the asymptotic results, selecting w_b as defined in **Lemma 2.4.4**. We then get Figure 2.1(a) for different ν's by letting $w_u = w_b$, and each line is the average result over 100 independent lists with each list having length $n(L) = 10^5$. As shown in Figure 2.1(a), large ν improves the performance of **FS**. When $\nu = 10$, **FS** is close to **BC** no matter how we choose the tool capacity w_c. Next we show the influence of w_u by fixing $n = 0.2$, i.e., $\nu = 5$. Let workload be drawn from $U(0, w_u)$. We test **FS** by letting $n(L) = 10^5$ for each of the 100 lists and we get Figure 2.1(b) for different w_u's. From Figure 2.1(b) we can see small $\frac{w_u}{w_b}$ also improves the performance of **FS**. From a practical standpoint, when replacement time τ is large, then the corresponding w_b is large thus $\frac{w_u}{w_b}$ is small, making **FS** work extremely well. Figure 2.1(b) also shows that w_b is not always the best option for fixing w_c. When $w_u = 0.5w_b$ and $w_u = w_b$, the maximal points of both lines correspond to best w_c's which are larger than w_b.

Now we show the performance of **FB** for different ν's and w_u's. Notice that when $w_u > w_b$ we cannot use **BC** to approximate the optimal value of **OFR**, as stated in **Lemma 2.4.9**. Thus we use **BC*** as the frame of reference (details are in Section 4.3.2). We draw $n(L) = 10^5$ jobs independently for each of 100 job lists to approximate the expected optimal value of **FB** and **OFR**. Similarly, we find from Figure 2.2(a) where we let $w_u = w_b$ that large ν improves the performance of **FB**. When $\nu = 10$, we can see the best performance of **FB** is as good as **OFR**. Next we show

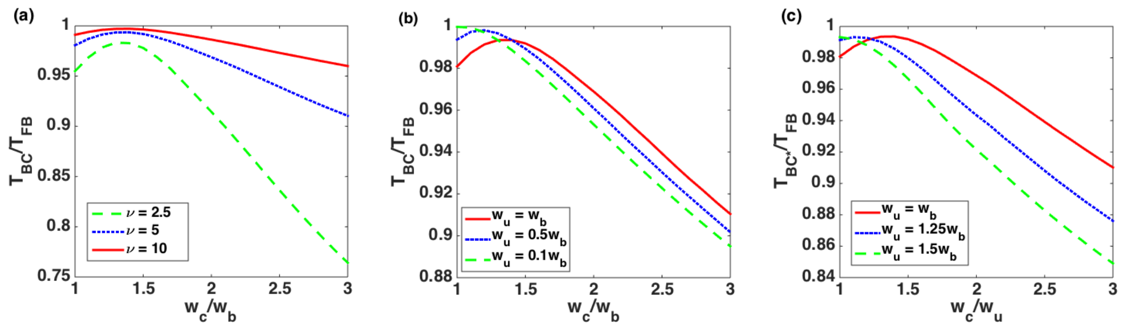


Figure 2.2: Performance of Fixed Buffer Approach when (a) $w_u = w_b$, (b) $w_u \leq w_b$, $\nu = 5$, (c) $w_u \geq w_b$, $\nu = 5$

the performance of **FB** for different w_u's. We fix $\nu = 5$ and let workload be drawn from $U(0, w_u)$. The performance of **FB** is shown in Figure 2.2(b) and Figure 2.2(c) for $w_u \leq w_b$ and $w_u \geq w_b$ respectively. Similarly, we observe that small $\frac{w_u}{w_b}$ improves the performance of **FB**. Notice that when given a large replacement time τ, the corresponding w_b is large, causing $\frac{w_u}{w_b}$ to be small. So the case where $w_u \geq w_b$ is an extreme case, but it turns out that **FB** works well even w_u is large. Besides, as shown in Figure 2.2(d) where we also test **FB** on modified beta distribution with different parameters, the optimal performance of **FB** is not sensitive to the assumption of probability distributions.

In Section 4.3.2 we show **FB** can be applied to the case when workload is unbounded, here we show in Figure 2.3 the performance of **FB** under different workload distributions. For all the cases in Figure 2.3, we assume $\nu = 5$. In Figure 2.3(a) we assume workload is according to modified Beta distributions. **BC** is used as the benchmark in this case since **OFR** cannot be approximated by **BC**. As we can see, the performance of **FB** under Beta distributions is similar to it has for uniform distribution, which shows when workload is bounded, the performance of **FB** is not sensitive to assumptions for workload distribution. Figure 2.3(b) and (c) show the performance for **FB** when workload is unbounded. **BC*** is used as the benchmark. Similarly, we find if we fix tool capacity for **FB** to be around $1.5w_b$, the gap between **BC*** and **FB** is always small ($\frac{T_{BC*}}{T_{FB}} \in (0.98, 1)$), which shows the robustness of **FB**.

We then provide a comparison of methods to show the power of revealing workload using IoT.

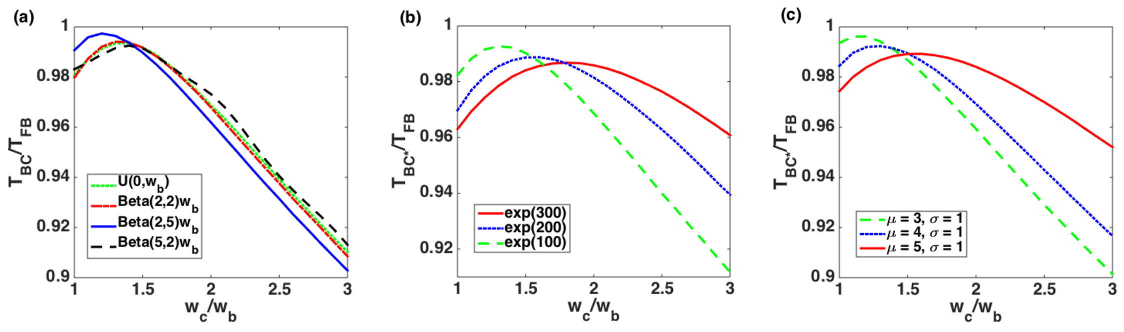


Figure 2.3: Performance of Modified Fixed Buffer Approach when workload is (a) Beta Distributed, $\nu = 5$ (b) Exponential Distributed, $\nu = 5$, (c) Log-Normal Distributed, $\nu = 5$.

We select the parameters so that each method performs its own best. We again let $w_u = w_b$, and fix $w_{c_1} = 1.6w_b$ when $\nu = 5$ and $w_{c_1} = w_b$ when $\nu = 2$ for **FS** since they are numerically near optimal. Similarly we fix $w_{c_2} = 1.4w_b$ when $\nu = 5$ and $w_{c_2} = 1.3w_b$ when $\nu = 2$ for **FB**. We test **FS**, **FB** and **OF** by drawing workload $n(L) = 45$ times independently from $U(0, w_u)$ where we let $w_u = w_b$. We choose $n(L) = 45$ and each tool serves $\frac{3}{2}$ jobs on average as shown in Section 2.4.3, thus every $n(L)$ jobs correspond to nearly 30 tools. We run this experiment for 100 times and the average result over these 100 experiments are shown in Table 2.4.

Knowing the workload also results in a tremendous improvement in the performance of online algorithms. In the case where no workload is revealed and if $w_u = w_b$, then the optimal policy is: the tool processes every job with a fixed speed which is determined by $w_b \dfrac{D(s)}{s} = h^*$. This is also the control policy provided in [10]. The performance comparison is provided by the **Stochastic Optimal Policy (SOP)** in Table 2.4 below. The actual performance of this optimal policy is given by $\lim_{n(L)\to\infty} \dfrac{\mathbf{E}[T_{BC}(L)]}{\mathbf{E}[T_{SOP}(L)]} = \dfrac{\nu}{\nu+1}$. The **Fixed Speed Approach** requires one revealed workload, and we see from Table 2.4 below a big improvement of performance it makes (94.38% of optimal offline) compared to the **Stochastic Optimal Policy** (83.33% of optimal offline). Ability to acquire workload distribution beforehand is also important. It helps improve the performance of online algorithms. If workload is revealed however the distribution of future workload is unknown, the optimal control policy for this case would be myopic as well: we choose the maximal speed (given

Material	n	$\nu = 1/n$
High Speed Steels	0.1-0.17	5.88-10.0
Uncoated WC Tools	0.2-0.25	4.0-5.0
TiN or TiN Coated WC Tools	0.3	3.3
Al_2O_3 Coated WC Tools	0.4	2.5
Solid Ceramics Tools	0.4-0.6	1.67-2.5

Table 2.3: Taylor's Parameter ν for different materials.

$\frac{T_{BC}}{T_{Method}}$ Methods	when $\nu = 5$	when $\nu = 2$
Stochastic Optimal Policy (No revealed workload)	0.8286	0.6600
Simple Approach (No distribution knowledge)	0.8933	0.7347
Fixed Speed (One revealed workload)	0.9402	0.8509
Fixed Buffer (More than one revealed workload)	0.9929	0.9727
Offline (All revealed information)	0.9957	0.9842

Table 2.4: Performance of Different Methods for Different Values of ν

by $w\dfrac{D(s)}{s} = h^*$) to process the current job w. We call it **Simple Approach (SA)** (a simplified data-driven method) which is shown in Table 2.4. When using **Simple Approach** we assume workloads are drawn from a single uniform distribution; however our online algorithms can be modified accordingly if workload distributions are changed.

2.6 Concluding Remarks and Future Work

In this paper we consider a custom-manufacturing setting with IoT. A benefit of using IoT is that it is possible to accurately ascertain workloads and tool degradation before processing a job as well as to know workload distribution beforehand. However until information is available in custom-manufacturing settings there is uncertainty. We use a behavior-imitating method to construct our online algorithms so that algorithms behave similar to offline solutions thus have performance guaranteed. We achieve a middle ground in our algorithm using a bin-packing setting where jobs are “packed" into each tool so that the tool is used to process those jobs and then discarded. We call these algorithms **Fixed Speed Approach** (i.e., **FS**) and **Fixed Buffer Approach** (i.e., **FB**) where information for certain small number of jobs is needed and as a result, they perform exceedingly well. In addition, the setting considers fairly arbitrary arrival processes and hence can easily be extended to multi-station systems.

The contribution of this paper is as follows. We provide a model under IoT manufacturing which considers in a single frame work (1) discrete part manufacturing, (2) machine with degrading tools and degradation is a function of processing speed, (3) jobs where workload is revealed upon “arrival" and (4) objective of minimizing completion time. In addition, we provide the idea of behavior-imitation which can be adapted to other online algorithms as well. There are also several insights that can be gleaned from our problem. We show that job re-sequencing is unnecessary for general settings where the workload is far less than the tool replacement time. This implies when replacement takes a long time, online algorithms perform very well. Also because the loss caused by prohibiting re-sequencing is compensated by the ability of fast processing, the online algorithms that we provide work reasonably well in the case when future jobs' information is unavailable. Moreover, we provide a list of optimal resource (tools) and time needed for processing a job list, which greatly enhance production planning. We show the optimal workload a tool serves depends only on the tool properties. Further, the performance of online algorithm in our setting relies heavily on the tool material. This helps practitioners to choose tool material scientifically.

There are several extensions to this research that can be considered in the future. Single stations with multiple tools per machine and multiple machines can be considered with degrading tools. The key idea would be to extend the policies that have been shown to be powerful in this paper. Subsequently, it would also be possible to extend to multiple workstations with various part flows. It needs to be seen if a decomposed approach would indeed result in efficient operations under such a setting. The effects of uncertainty especially in terms of random failures and random tool wear could also be considered in the future. While these problems do get much more complex than what is considered in this paper, we believe that the theories would nicely extend to more complex scenarios and they can be useful in designing and controlling large scale IoT-based smart-manufacturing systems with customized production.

3. ON COMPETITIVE ANALYSIS FOR POLLING SYSTEMS

3.1 Introduction

This study has been motivated by operations in smart manufacturing systems. As an illustration, consider a 3D printing machine that uses a particular material informally called 'ink' to print. Jobs of the same prototype are printed using the same ink, and when a different prototype (for simplicity, say a different color) is to be printed, a different ink is required and the machine undergoes a setup that takes time to switch inks. The unprocessed jobs of the same prototype can be regarded as a 'queue'. This problem can thus be modeled as a polling system where the server polls a queue, processes jobs, incurs a switch over time, processes another queue, and so on. In practice, besides the ink (material), other factors such as processing temperature, equipment setting and other processing requirements that require by different job prototypes will also incur setup times.

Another interesting feature of 3D printing is that it is possible to reveal the workload of each job (i.e., processing time) upon the job's arrival. This is because before getting printed, the printing requirements such as temperature, nozzle route, printing speed, and so on are specified for the job, and using that we can easily acquire the printing time before processing. Therefore, it is unnecessary to assume that the workload of a job is stochastic at the start of processing, even though many other queueing research works do so. In this paper, we assume that the workload of jobs could be arbitrary, and could be revealed deterministically upon arrival. Furthermore, the 3D printer that prints customized parts usually receives jobs with different processing requirements. Job arrivals thus could be time-varying, non-renewal, in batches, dependent, or even arrivals without a pattern. It motivates us to consider the generic polling system without imposing any stochastic assumptions on future arrivals.

The above is actually an example of the general polling system. In such a system, job arrivals are arbitrary, workload of each job is revealed deterministically upon arrival, and a setup time occurs when the server switches from one queue to another. We call such a polling system 'general' mainly because we do not impose any stochastic assumptions on job arrivals or service times. Polling system was firstly introduced in 1970's and it is one of the most important system in queueing literature due to its wide application [43, 44, 45, 46]. There are practical needs for studying the polling system with general settings as in many cases, job processing requirements and arrivals are unpredictable and could be in any pattern. Besides 3D printing, many other examples of such a general polling system can be found in computer-communication systems, reconfigurable smart manufacturing systems and smart traffic systems [43, 47, 46], where arrivals could be arbitrary and service time gets revealed upon job's arrival. Having a scheduling policy that works well in such general settings could prevent the system from performing erratically when rare events occur. Knowing the worst case performance of a policy also aids in designing reliable systems. There are also needs from theory for studying the polling system with general settings. Most of the previous works model the polling system from a stochastic standpoint by assuming certain distributions for arrival, service and switching processes. As a consequence, these works are limited in describing the average performance of certain polling systems that rely on specific stochastic assumptions. There are very few studies discussing the optimal policies or online scheduling algorithms for the polling system without stochastic assumptions due to the complexity of analysis [48]. It is still unknown if those scheduling policies designed for specific polling systems work well in the general settings. In our paper, we study the polling system from an online optimization perspective and perform worst case analysis for the generic polling system that does not rely on any stochastic assumptions for arrival, service or switching processes. More specifically, we consider a completion time minimization problem in such a polling system and obtain the worst case performance (i.e., competitive ratios) of several widely used scheduling policies with known average performance, such as Gittins Index Policy, Stochastic Largest Queue (SLQ) as well as cyclic exhaustive and gated policies. Our work also bridges the scheduling and queueing communities by showing that some queueing policies that work well under stochastic assumptions also work well in the general scheduling settings. In this work we, for the first time, provide conditions for existence of constant competitive ratios for online policies in polling systems. Moreover, we suggest a mixed strategy in this paper that works well under practical scenarios.

3.1.1 Problem Statement

As mentioned earlier, the unprocessed jobs from the same prototype (or family) can be modeled as a queue. We thus consider a single server system with k parallel queues. Jobs arrive at each queue will wait until being served by the server. Figure 3.1 shows a polling system with $k = 4$ queues. The processing time (workload) p_i of the i^{th} arriving job is revealed instantly upon its arrival time r_i. The server can serve the jobs that are waiting in queues in any order non-preemptively. However, a setup time τ is incurred when the server switches from one queue to another. We assume that once the setup is in progress, it can be interrupted but it is not resumable. Next time when the server switches to this queue, a setup time τ is still needed. Information (r_j, p_j) about a future job j (for all j) remains unknown to the server until job j arrives in the system. The objective is to find service order for jobs and queues to minimize total completion time over all jobs, where the completion time of a job is the time period from time 0 to when the job has been served (exits the system). It is assumed that the number of jobs is an arbitrary large finite quantity. We wish to also state that previous studies of polling systems usually assume that (r_j, p_j) follows certain random distributions (see [49]), but in our paper we study polling systems from an online scheduling perspective and allow polling systems to have arbitrary arrival times and processing times.

3.1.2 Preliminaries

In this subsection we mainly introduce some concepts and terminologies that we use in this paper. Important notations in this paper are provided in Table 3.1.

3.1.2.1 Machine Scheduling Problems

The polling system scheduling problem belongs to the class of one machine online scheduling problems since there is only one server (machine or scheduler) in the system. Using the notation for one machine scheduling problems [50], we write our problem as $1 \mid r_i, \tau| \sum C_i$, where '1'

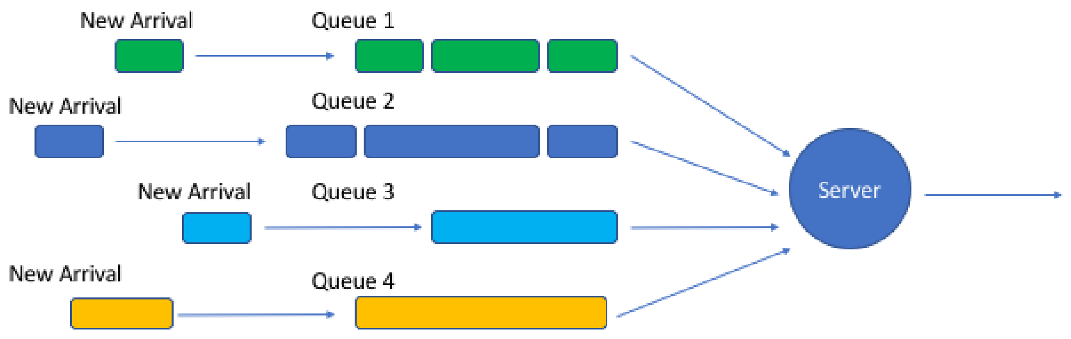


Figure 3.1: Polling System with Four Queues

refers to the single machine in the system, r_i and τ in the middle part of the notation refer to the release date and setup time constraints, and $\sum C_i$ means that the objective of this scheduling problem is to minimize the total completion time. This type of notation is widely used to briefly describe the input, constraints and objective for any scheduling problem. The notation is of the type $P_m \mid constraints \mid objective$. The first column in the notation denotes the number of machines in the system. For example, it could be 1 for one machine or P_m for m parallel machines. The middle column denotes the constraints. In the later part of this paper we would introduce other constraints such as $\tau \leq \theta p_{min}$ and $p_{max} \leq \gamma p_{min}$, where $p_{max} = \sup_i p_i$ is the upper bounded workload and $p_{min} = \inf_i p_i$ is the lower bound workload. We say the workload variation is bounded if $p_{max} \leq \gamma p_{min}$ for constant γ. If no constraint is specified in this column, it means 1) all jobs are available at time zero, 2) no precedence constraints are imposed, 3) preemption is not allowed, and 4) no setup time exists. In this paper, we assume jobs and setup times are non-resumable and all the policies that we discuss are non-preemptive, unless we specifically point out. The last column of the notation represents the objective for the scheduling problem (default to be minimization problems). In this paper it is $\sum C_i$, which means minimizing the total completion time. In other papers it could be C_{max} [51], which means minimizing the maximal completion time (makespan), or $\sum w_i C_i$ which is minimizing the weighted completion time [23], where w_i is the weight for job i.

3.1.2.2 Online and Offline Problems

A job instance I with $n(I)$ number of jobs is defined as a sequence of jobs with certain arrival times and processing times, i.e., $I = \{(r_i, p_i), 1 \leq i \leq n(I)\}$. In this work we mainly focus on an online scheduling problem, where online means r_i and p_i remain unknown to the server until job i arrives. In contrast to the online problem, the offline problem has entire information for the job instance I, i.e., $(r_1, r_2, ..., r_{n(I)})$ and $(p_1, p_2, ..., p_{n(I)})$ from time 0. The offline problem is usually of great complexity. The offline version of the online problem that we want to solve, i.e., $1|r_i, \tau| \sum C_i$, is strongly NP-hard since the offline problem with $\tau = 0$ is strongly NP-hard [52, 23, 53]. However it is important to note that if preemption is allowed, then serving the *shortest remaining processing time* (SRPT) is the optimal policy for $1|r_i, pmtn| \sum C_i$ with $\tau = 0$ (see [54]). SRPT is online and polynomial-time solvable. It can be used as a benchmark for online scheduling policies to compare against.

3.1.2.3 Scheduling Policies

A scheduling policy π specifies when the server should serve which job. In our paper we mainly focus on online policies. Online policies provide feasible solutions for online problems, without knowing the information of future jobs. Online policies are also called non-anticipative policies in some literatures [55, 56]. Online policies can be either *deterministic* or *randomized.* There is only one unique solution if a job instance is given to a deterministic policy. For example, SRPT is a deterministic policy. A randomized policy may toss a coin before making decision, and the decision may depend on the outcome of this coin toss [56]. This means for a random policy with a certain internal random variable, its solution may depend on the outcome of this internal random variable. If the same job instance is given to a randomized policy multiple times, each time the solution may be different from the others due to different outcomes by the internal random variable. Detailed discussions for randomized algorithms can be found in [57, 24]. In this paper, we mainly focus our discussion on deterministic policies.

Notations	Meaning	Notations	Meaning
r_i	Release date, the time when job i arrives in the system	p_i	Workload (processing time) for job i
p_{max}	Maximum workload, $p_{max} = \max_i\{p_i\}$	p_{min}	Minimum workload, $p_{min} = \min_i\{p_i\}$
τ	Setup time is τ for all queues	$I = \{r_i, p_i\}$, for $1 \leq i \leq n(I)$	Job instance, set of jobs with information about release date and processing time for all jobs in it
$C^\pi(I)$	Total completion time for jobs in instance I under policy π	$C^*(I)$	Total completion time for jobs in instance I under the optimal policy of the offline problem
C_i^π	The completion time for job i under policy π	$n(I)$	Number of jobs in job instance I
γ	Workload variation, a constant such that $p_{max} \leq \gamma p_{min}$	θ	A constant such that $\tau \leq \theta p_{min}$
k	Number of queues in the system	$\kappa = \max\{\frac{3}{2}\gamma, k+1\}$	Competitive ratio for cyclic-based and exhaustive-like policies, a constant

Table 3.1: Notations

3.1.2.4 Competitive Ratios

Competitive ratio is a ratio between the solution obtained by an online policy and the *benchmark*. In this paper, the optimal solution to the offline problem is the benchmark that we compare the online policy against. Thus we say the competitive ratio for an online policy is ρ if $\sup_I \frac{C^{\pi}(I)}{C^{*}(I)} \leq \rho$ for any job instance I, where $C^{\pi}(I)$ is the completion time for job instance I by a deterministic scheduling policy π and $C^{*}(I)$ is the optimal completion time of the offline problem. We say a competitive ratio is tight if there exists an instance I such that $\frac{C^{\pi}(I)}{C^{*}(I)} = \rho$.

3.1.3 Related Works

Since in this paper we analyze the polling system from a scheduling perspective, there are many research works in scheduling which are related to our work. The single machine scheduling problems that consider setup times or costs have been widely studied from various perspectives. A detailed review of the literature can be found in [58, 59, 60, 61, 62]. Other works considering machine setup can be found in [63, 64, 65, 66, 67]. However, almost all of these works are focused on solving the offline problem where all the release times and processing times are revealed at time 0.

In this paper we are more interested in the online problem where the current state of the system is known and future is uncertain. Thus our work also falls into the class of the online single machine scheduling problem. Numerous research papers have shed light on the online single machine scheduling problem, without considering setup times. Table 3.2 summarizes the current state of art of competitive ratio analysis over existing online algorithms for single machine scheduling problem without setup times. Besides all the algorithms provided in Table 3.2, a recent work [68] provides a new method to approximate the competitive ratio for general online algorithms. However, since these works mainly focus on solving problem without setup times, the online policies provided in these works are not directly applicable to polling systems.

Only few articles so far have focused on the online single machine scheduling problem with queue setup times, i.e., the polling system. The online makespan minimization problem for the

Problem	Deterministic		Randomized	
	Lower Bounds	Upper Bounds	Lower Bounds	Upper Bounds
$1\|r_i, pmtn\|\sum C_i$	1	1 [54]	1	1 [54]
$1\|r_i, pmtn\|\sum w_iC_i$	1.073 [69]	1.566 [70]	1.038 [69]	$\frac{4}{3}$ [71]
$1\|r_i\|\sum C_i$	2 [72]	2 [72, 73, 74]	$\frac{e}{e-1}$ [57]	$\frac{e}{e-1}$ [24]
$1\|r_i\|\sum w_iC_i$	2 [72]	2 [75]	$\frac{e}{e-1}$ [57]	1.686 [76]
$1\|r_i, \frac{p_{max}}{p_{min}} \leq \gamma\|\sum w_iC_i$	$1+\frac{\sqrt{4\gamma^2+1}-1}{2\gamma}$ [77]	$1+\frac{\sqrt{4\gamma^2+1}-1}{2\gamma}$ [77]	Unknown	Unknown
$1\|r_i, \frac{p_{max}}{p_{min}} \leq \gamma\|\sum C_i$	Numerical [78]	$1+\frac{\gamma-1}{1+\sqrt{1+\gamma(\gamma-1)}}$ [78]	Unknown	Unknown

Table 3.2: Competitive Ratios for Single Machine Scheduling Problem without Setup Times (i.e., $\tau = 0$)

polling system is considered in [79] and an $\boldsymbol{O}(1)$ algorithm is proved to exist. However the competitive ratio provided in [79] is very large. A 3-competitive online algorithm for the polling system with minimizing the completion time is provided in [80], but only for the case where $k = 2$ and jobs are identical. To the best of our knowledge, the online algorithms for general polling systems with setup time consideration have not been well studied.

As we mentioned before, there are articles that study polling systems from a stochastic perspective by assuming job arrivals, service times and setup times are stochastic. Average performance of policies is considered for different types of stochastic assumptions [48]. Exact mean value analysis for cyclic routing policies with exhaustive, gated and limited service disciplines for polling system of $M/G/1$ type queues have been provided in [81, 49, 82, 83, 84, 85]. Service disciplines within queues (such as FCFS, SRPT and others) are discussed in [86], however the routing discipline (choose which queue to serve) and optimal service discipline between queues are not discussed in [86]. Optimal service policy for the symmetric polling system is provided by [87], where 'symmetric' means that jobs arrive evenly into each queue and jobs are stochastically identical. Most of the works from this stochastic perspective are restricted to average performance analysis. The structure of the optimal solution to the general polling system yet remains unknown [47, 48]. Approximating algorithms for the polling system are very few, and none of those popular policies with proven average performance have been showed the adaptivity in general settings.

In summary, so far there is no work which provides optimal or approximating scheduling policies for the polling system with a general setting and the objective of minimizing the total completion times. In addition, the conditions under which the polling system allows constant competitive ratios for online algorithms are also unknown. Besides, the competitive ratios for widely-used policies in polling systems (such as cyclic policies with exhausted and gated service discipline) remain unknown. The contribution of this paper is fourfold: (1) Our work for the first time analyzes polling systems without stochastic assumptions, evaluates policy performance by competitive ratio, provides the conditions for existence of competitive ratio in polling systems, and proves competitive ratios for some well-studied policies such as cyclic exhaustive and gated policy. (2) Our work bridges the queueing and scheduling communities by showing that some widely-used queueing policies also have decent performance in terms of worst case performance in online scheduling problems. (3) We provide a lower bound for the competitive ratio for all the possible online policies, and show that no online algorithm can have a competitive ratio smaller than this lower bound. (4) We also provide new online policies that balance future uncertainty and utilize known information, which may open up a new research direction that would benefit from revealing information and reducing variability. This paper is organized as follows: in Section 3.2 we consider the case of bounded workload variation and obtain competitive ratios for various policies; in Section 3.3 we consider the case with unbounded workload variation; in Section 3.4 we provide a mixed strategy to deal with a practical scenario; we make concluding remarks in Section 3.5.

3.2 Polling System with Bounded Workload Variation

From [54, 88] we know that in the single machine problem (no setup times) with minimizing completion times, i.e., $1|r_i|\sum C_i$, it is always beneficial to schedule jobs preemptively with the smallest remaining processing time (SRPT) first. The reason is that by doing this, small jobs are quickly processed thus the number of waiting jobs is reduced. This idea of scheduling small workload first can be used in polling systems as well. An efficient scheduling policy may avoid the case where a large number of jobs are waiting in the queue while a single large job is in process. It should also avoid the case when a large number of small jobs are waiting in a queue, but the server is busy serving other queues and not able to switch to this queue immediately. Thereby, the main idea of designing a scheduling policy is to avoid either of these extreme cases happening. A good online policy should balance job priority and queue priority so that small jobs are processed in a timely manner and switching does not happen frequently. The priority of a job is usually determined by the job information, such as processing time and release date. In contrast, queue priority is usually determined by the queue information such as the number of waiting jobs, the sum of remaining workload in a queue, and so on. If the variation of processing time is large and setup times are trivial, then perhaps the server should favor the job priority more than the queue priority since a large job in process may cause a much larger delay than switching. In the case where variation of processing times is small, the server may need to favor the queue priority. We leave the discussion for unbounded variation to Section 3.3, and in this section we mainly discuss the case where workload variation is bounded (i.e., $p_{max} \leq \gamma p_{min}$ for some constant γ; recall that $p_{max} = \sup_i p_i$ and $p_{min} = \inf_i p_i$). The case of bounded workload variation, in the 3D printing example that we introduced earlier in this paper, corresponds to the scenario in which jobs have similar printing requirements, i.e., jobs have similar processing times.

In this work, we suppose the decision of when to switch out from a queue and in which order the jobs get served is determined by the *service discipline* of a policy. The decision of which queue to serve next is determined by the *routing discipline.* Notice that once the variation of processing times is small, queue priority should be favored. We shall show later in this section that when workload variation is small, service discipline does not impact the policy performance greatly, as long as it follows an exhaustive-like pattern. Thereby in this section, we mainly focus our discussion on the routing discipline. There are many widely used routing disciplines and here we classify them into two types. One is called *static* routing discipline, which means the server always follows a fixed routing order (also called a general routing table), such as the policy provided in [89]. The other is called *dynamic* routing discipline, which means the server follows a dynamic routing policy which may use any available information as the processing goes on, such as Stochastic Largest Queue (SLQ) provided in [87] or selecting queues randomly such as the random routing discipline provided in [90]. For static routing disciplines, we focus our discussion on the cyclic routing policy (also called periodic), and later in this section we shall show that cyclic routing is the optimal static routing discipline in terms of the worst case performance. We will also show that random routing discipline is generally no better than static ones in terms of the worst case performance, which may be surprising and non-intuitive at beginning but it indeed reflects the underlying difference between average performance with worse case performance. Since in this section we only consider the case where workload variation is bounded, we assume the maximal processing time (across all queues) for an arbitrary job is bounded by a constant ratio of minimal possible processing time across all queues, i.e., $p_{max} \leq \gamma p_{min}$ for some constant γ. And we assume the setup time for all queues is fixed as τ. So our problem is denoted as $1 \mid r_i, \tau, p_{max} \leq \gamma p_{min} \mid \sum C_i$ where $p_{max} \leq \gamma p_{min}$ is a constraint imposed. Unlike in [78] where p_{min} is assumed to be non-zero, here we also allow $p_{min} = p_{max} = 0$, for which we assume $\gamma = 1$.

3.2.1 Cyclic Based and Exhaustive-like Policies

In this subsection we mainly discuss policies that use cyclic routing discipline and serve each queue in an exhaustive manner. A queue is called 'exhausted' if there is no job left in the queue by the time the server switches out. These policies form a policy set Π_r which we will define later. We begin our discussion with a set of policies Π_1 called Cyclic Exhaustive Policies with Skipping Empty Queues whose description is shown in Algorithm 3.1. This set contains all the policies that 1) serves queues in a cyclic way (from queue 1 to k and then again from 1 to k and so on), 2) serves each queue exhaustively, and 3) waits in the queue for a certain amount of time before switching out. Once the server decides to switch out, it selects the next non-empty queue to switch to. If all queues are empty, the server will idle at the last queue that it served. Note that for an arbitrary policy $\pi \in \Pi_1$, during each visit to queue i, the server has to serve all the available jobs in queue i before switching out. After serving all the jobs in queue i, the server is allowed to wait in queue i for some extra time to receive more arrivals. If the server processes n_i^w jobs in its w^{th} visit to queue i, it can stay in queue i for at most $n_i^w p_{max}$ amount of time in this visit. If a new arrival occurs at queue i during the time that the server is waiting, then $n_i^w \leftarrow n_i^w + 1$ and the server can process this job at any time before it switches out, as long as the server does not stay in the queue for time longer than the updated $n_i^w p_{max}$. Note that if during w^{th} visit to queue i, all the jobs have workload p_{max}, then the server will switch out when a queue is exhausted. Also note that we do not specify the service order of jobs for policies in Π_1. As long as a policy satisfies the description of Algorithm 3.1, it belongs to Π_1. The next theorem provides the competitive ratio for policies in Π_1. Since the proofs of this and some subsequent theorems are lengthy, they are provided in the appendices.

Theorem 3.2.1. *Any policy $e \in \Pi_1$ has competitive ratio of $\kappa = \max\{\frac{3}{2}\gamma, k+1\}$ for the polling system $1 \mid r_i, \tau, p_{max} \leq \gamma p_{min} \mid \sum C_i$. When $\frac{3}{2}\gamma \leq k+1$, for arbitrary $\epsilon > 0$, there is an instance I such that $\frac{C^e(I)}{C^*(I)} > \kappa - \epsilon$.*

Proof. The main idea of the proof is to consider each 'batch' under the online policy, which are the jobs served by the server during each visit to a queue. The detail of the proof is in Appendix A.1.

□

Algorithm 3.1 Cyclic Exhaustive Policies with Skipping Empty Queues Π_1

```
Require: Instance $I$
1: $server = 1; w = 1$
2: while $I$ has not been fully processed do
3:     Process all jobs that are present in $queue_{server}$ during server's $w^{th}$ visit to $queue_{server}$ (the
       total number of jobs is denoted as $n^w_{server}$), where the length of each visit period is at most
       $n^w_{server} p_{max}$
4:     if queue $server < j \leq k$ is not empty then
5:         $server = server + min\{j\}$
6:     else if queue $1 \leq j \leq server - 1$ is not empty then
7:         $server = min\{j\}; w = w + 1$
8:     else
9:         $server = l$ if next arrival occurs at queue $l$; $w = 1$
10:    end if
11: end while
12: return Total completion time $C^e(I)$
```

Notice that if $p_i = 1$ for all jobs i and $k = 2$, then $\kappa = 3$, which is the result shown in [80].

However Theorem 3.2.1 implies a stronger result since we allow k to be general ($k \geq 2$) and p_i's to be different. Next we show a set of policies Π_2 which keep switching even when the system is empty, called Cyclic Exhaustive Policies without Skipping Empty Queues and shown as Algorithm 3.2. The only difference between Π_1 and Π_2 is under any policy from Π_2, the server sets up a queue no matter if it is empty or not. Similar to policies in Π_1, policies from Π_2 allow the server to stay in the queue i for no more than $n_i^w p_{max}$ amount of time for its w^{th} visit, so waiting is also allowed. The policy without waiting (just exhaustively serving) belongs to Π_2 and it has provable average performance for $M/G/1$ type queues [81, 49, 82, 83]. Here we show the competitive ratio for the policies in Π_2 is also κ.

Algorithm 3.2 Cyclic Exhaustive policy without Skipping Empty Queues Π_2

```
Require: Instance $I$
1: $server = 1; w = 1$
2: while $I$ has not been fully processed do
3:     Process all jobs that are present in $queue_{server}$ during server's $w^{th}$ visit to $queue_{server}$ (the
   total number of jobs is denoted as $n^w_{server}$), where the length of each visit period is at most
   $n^w_{server} p_{max}$
4:     $server = Rem(server, k) + 1$
5:     if $server == 1$ then
6:         $w = w + 1$
7:     end if
8: end while
9: return Total completion time $C^e(I)$
```

Theorem 3.2.2. *Any policy $e \in \Pi_2$ has competitive ratio of $\kappa = \max\{\frac{3}{2}\gamma, k+1\}$ for the polling system $1 \mid r_i, \tau, p_{max} \leq \gamma p_{min} \mid \sum C_i$. When $\frac{3}{2}\gamma \leq k+1$, for arbitrary $\epsilon > 0$, there is an instance I such that $\frac{C^e(I)}{C^*(I)} > \kappa - \epsilon$.*

Proof. The proof of Theorem 2 is similar to the one for Theorem 1. A detailed proof is provided in Appendix A.2. □

Notice that the policies in Π_1 and Π_2 are different only in the way that the server deals with empty queues. All the policies in Π_1 and Π_2 use an exhaustive service discipline. Besides the exhaustive discipline, the gated discipline is often discussed because its average performance for $M/G/1$ type queues is also provable under cyclic routing discipline without skipping empty queues [49, 83]. Under the gated discipline, the server only serves the jobs that are in the queue before the server sets up the queue, and jobs that arrive after that time would be left to the next cycle of service. We do not specify the service order for the gated discipline either. We let Π_3 be the set of policies that follow cyclic routing discipline without skipping the empty queues and serve each queue with gated discipline. We also allow server to wait after clearing a queue under Π_3. Once the server has set up a queue, the number of jobs that are served during this visit is determined. This is different from policies from Π_1 or Π_2. Similarly, we can have a policy set Π_4 in which policies are cyclic and gated, but skipping empty queues when switching. We do not provide the detailed description of Π_4 here since it is similar to policy set Π_3. Policies in Π_3 and Π_4 also have competitive ratio κ, as shown in the following theorem.

Theorem 3.2.3. *Any policy $e \in \Pi_3 \cup \Pi_4$ has competitive ratio of $\kappa = \max\{\frac{3}{2}\gamma, k+1\}$ for the polling system $1 \mid r_i, \tau, p_{max} \leq \gamma p_{min} \mid \sum C_i$. When $\frac{3}{2}\gamma \leq k+1$, for arbitrary $\epsilon > 0$, there is an instance I such that $\frac{C^e(I)}{C^*(I)} > \kappa - \epsilon$.*

Proof. The proof of Theorem 3 is also similar to the one for Theorem 1. A detailed proof is provided in Appendix A.3. □

So far we have shown the competitive ratio for policies based on exhaustive and gated disciplines, with and without skipping empty queues. Notice that all the policies in $\Pi_i, i = 1, ..., 4$ use the cyclic routing discipline and have the same competitive ratio. We let these policies form a policy set Π_r, i.e., $\Pi_r = \cup_{i=1}^{4} \Pi_i$. Again, it is important to note that the service order of jobs during the server's visit to a queue is not specified by Π_r. The service order could be First Come First Serve (FCFS), Shortest Processing Time First (SPT) or any other non-preemptive processing order, but all of them result in the same competitive ratio and the competitive ratio is approximately tight when $\frac{3}{2}\gamma \leq k+1$. It indicates that although revealing processing times provides additional information for job scheduling, when the workload variation is small, the revealed information does not improve the worst case performance of an online policy. It is not a coincidence that all policies in Π_r have the same competitive ratio. All the policies in Π_r follow an exhaustive-like manner, even for the gated discipline. An arbitrary gated policies from Π_3 would exhaust all the jobs that arrive before the queue is set up, and if a large number of jobs arrive after the queue is set up, they will anyway be exhaustively processed in the next round. So we can see the gated discipline also has some 'exhaustive' characteristics. In fact, Theorem 3.2.4 in the following shows that exhaustive discipline is the optimal service discipline when $p_{max} = p_{min}$.

Theorem 3.2.4. *For the polling system* $1|r_i, \tau, p_{min} = p_{max}| \sum C_i$*, there always exists an exhaustive policy which outperforms a non-exhaustive policy in terms of total completion time.*

Proof. The case where $p_i = 0$ is trivial. Now we let $p_i = 1$ with appropriate units. Since preemption is not allowed and all the jobs are identical, the server only needs to decide when to switch out and which queue to switch to. If there are jobs in the queue that the server is currently serving, there are two options for the server: to continue serving the next job in this queue, or to switch to a nonempty queue and later come back to this queue again. If at time 0 the server is at queue 1 and there is an unfinished job in queue 1, then the server has to come back after it switches out. Suppose under a non-exhaustive policy $\pi^{'}$, the server chooses to switch to some queue(s) and come back to queue 1 at time T. Say the server serves instance $I^{'}$ during this period. Suppose there is an adversary policy which has the same instance at time 0, and this policy chooses to serve one more job in queue 1, and then follows all decisions that policy $\pi^{'}$ has made (including waiting). Note that every decision policy $\pi^{'}$ made is available to the adversary policy because the adversary policy serves one more job before leaving queue 1. The total completion time under $\pi^{'}$ is $C^{\pi^{'}} = 1 + T + C(I^{'})$, and the completion time achieved by the adversary is $C^{ad} = 1 + C(I^{'}) + n(I^{'})$, where $C(I^{'})$ is the completion time of instance I served by policy $\pi^{'}$ during $(0, T]$. Thus $C^{ad} - C^{\pi^{'}} \leq n(I^{'}) - T \leq 0$. Notice that the makespan of these two schemes are the same (including the final setup time of queue 1 for the adversary). The theorem is proved. □

Recall that all the policies in Π_r use a static and cyclic routing discipline. Next we show in Theorem 3.2.5, the cyclic routing discipline results in the smallest competitive ratio among all the static routing disciplines.

Theorem 3.2.5. *No online policy with a static or random routing discipline can guarantee a competitive ratio smaller than k for $1|r_i, \tau, p_{max} \leq \gamma p_{min}| \sum C_i$. Further, cyclic is the optimal static routing discipline for such a system. No online policy has a competitive ratio smaller than 2 for $1|r_i, \tau, p_{max} \leq \gamma p_{min}| \sum C_i$.*

Proof. Since $1|r_i, \tau, p_i = 1| \sum C_i$ is a special case of $1|r_i, \tau, p_{max} \leq \gamma p_{min}| \sum C_i$, we only need to show the result holds for $1|r_i, \tau, p_i = 1| \sum C_i$. We first show the case for static routing policies. For an arbitrary policy that follows a static routing discipline, we suppose the server starts from queue 1, and queue k is the last one visited. Before the server visits queue k, queue i $(1 \leq i \leq k-1)$ has been visited v_i times (suppose $v_{(1)} \leq v_{(2)} \leq ... \leq v_{(k-1)}$ is the ascending order for v_i's). We construct a special job instance I by assuming that there is one job arriving at each queue i every time the server visits queue i for $i = 1, ..., k-1$. Also we suppose there are n_k jobs at queue k at time 0 for a large n_k, and say they form a batch b_k. If we let $g(b_k) = \frac{n_k(n_k+1)}{2}$, then we have

$$C^{\pi}(I \cup b_k) \geq C^{\pi}(I) + n_k\tau(\sum_{i=1}^{k-1} v_i + 1) + g(b_k),$$

$$C^{*}(I \cup b_k) \leq C^{\pi}(I) + n_k\tau + g(b_k) + n(I)(n_k + \tau),$$

and $\frac{C^{\pi}(I \cup b_k)}{C^{*}(I \cup b_k)} \geq \sum_{i=1}^{k-1} v_i + 1$ if we let $\tau = (n_k)^2$ and $n_k \to \infty$. Since for any static routing discipline we can construct a job instance like this, to achieve the smallest ratio we need $v_i = 1$ for $i = 1, ..., k-1$. Thus cyclic is the optimal static routing discipline. Notice a random routing policy always generates a routing table randomly. Thus the result holds for random routing policies as well.

The lower bound competitive ratio for $1|r_i, \tau, p_i = 1| \sum C_i$ is 2 as given in [80]. Since problem $1|r_i, \tau, p_i = 1| \sum C_i$ is a special case for $1 \mid r_i, \tau, p_{max} \leq \gamma p_{min} \mid \sum C_i$ and $1|r_i, \tau| \sum C_i$, we know that no online algorithm can have a competitive ratio smaller than 2 for $1 \mid r_i, \tau, p_{max} \leq \gamma p_{min} \mid \sum C_i$ or $1|r_i, \tau| \sum C_i$. $\square$

So far we have shown that for problem $1|r_i, \tau, p_{max} \leq \gamma p_{min}| \sum C_i$, the policies in Π_r all have competitive ratio $\kappa = \max\{\frac{3}{2}\gamma, k+1\}$. From Theorem 3.2.5 we know that the smallest competitive ratio based on cyclic routing discipline is at least k. The problem that whether there exists either a lower bound competitive ratio greater than k or an online policy whose competitive ratio is smaller than κ remains open. If we let $\gamma \to \infty$, then $\kappa \to \infty$. This means that the competitive ratio we provide goes to infinity in this case, but it does not mean that policies in Π_r all have infinite competitive ratios, since the competitive ratio is approximately tight only when $\frac{3}{2}\gamma \leq k + 1$ as shown in Theorems 3.2.1, 3.2.2 and 3.2.3. However, Theorem 3.2.6 that we introduce next shows that policies in Π_r do not have constant competitive ratio if γ is infinite.

Theorem 3.2.6. *Policies in Π_r do not guarantee constant competitive ratios for $1|r_i, \tau| \sum C_i$.*

Proof. We prove the theorem by giving a special job instance I. We assume $p_{min} = 0$ and $p_{max} = p$ so that $\gamma = \infty$. Suppose at time 0 each of queue $i = 2, ..., k$ has one job of processing time p and queue 1 has no job. At time $\tau + \epsilon$ there are n jobs arriving at queue 1, with each of these n jobs has processing time 0. For any policy π in Π_r, the server would either setup queue 1 at time 0 then switch to queue 2 at time τ, or setup queue 2 at time 0. In either of the case the server will be back to queue 1 when queue k is served in the first cycle. Then we have

$$C^\pi(I) \geq \frac{k(k-1)}{2}p + n(k-1)p + \tau\left((n+k-1) + (n+k-2) + ... + n\right),$$

and

$$C^*(I) \quad = \quad \frac{k(k-1)}{2}p + \tau\left((n+k-1)+(k-1)+...+1\right)+\epsilon(n+k-1).$$

Letting $p = (n)^2$ and $n \to \infty$, we have $\frac{C^\pi(I)}{C^*(I)} \to \infty$. $\square$

3.2.2 Other Queue-length Based Policies

Note that not all the policies with cyclic routing discipline have competitive ratio κ. Policies in Π_r have a constant competitive ratio because they all serve as many available jobs as possible in each visit to a queue, which reduces the frequency of switching. Some other cyclic policies may not have constant competitive ratios for $1|r_i, \tau, p_{max} \leq \gamma p_{min}| \sum C_i$. We first consider a policy called *l-limited* policy. This policy is also based on the cyclic routing discipline. However, the server under l-limited policy only serves at most l jobs during each visit to a queue, then switches to the next queue. A detailed description for this policy and its average performance for $M/G/1$ type queues can be found in [49, 85, 84]. Interestingly, as we shall show in Corollary 3.2.7, no constant competitive ratio is guaranteed by l-limited policy, no matter the server sets up empty queues or not.

Corollary 3.2.7. *The l-limited policy ($l < \infty$, with or without skipping empty queues) does not have a constant competitive ratio for* $1 \mid r_i, \tau, p_{max} \leq \gamma p_{min} \mid \sum C_i$.

Proof. We prove this result by giving a special instance I. Suppose there are $(l * n)$ number of jobs ($l, n \in \mathbf{Z}_+$) at every queue at time 0, and each job has processing time $p = 1$. The server would 1) sets up the first queue, 2) serve l number of jobs in the first queue, and 3) make a tour from queue 2 to queue k by setting up each queue and serving l jobs at each queue. After returning to queue 1, the server will again set up queue 1 and serve l jobs. This process will be repeated for n times before the entire instance I is processed. Let $C^l(I)$ be the total completion time for the l-limited policy (the policies with or without skipping empty queues have the same completion time for this job instance), we have

$$\frac{C^l(I)}{C^*(I)} \;=\; \frac{\frac{knl(knl+1)}{2} + \tau l \frac{kn(kn+1)}{2}}{\frac{knl(knl+1)}{2} + \tau nl \frac{k(k+1)}{2}}.$$

If we let $\tau = (n)^2$ and $n \to \infty$, then $\frac{C^l(I)}{C^*(I)} \to \infty$. $\square$

Although from Theorem 3.2.6 we show policies in Π_r do not have constant competitive ratios for $1 \mid r_i, \tau \mid \sum C_i$, they do have a constant competitive ratio for $1 \mid r_i, \tau, p_{max} \leq \gamma p_{min} \mid \sum C_i$. Corollary 3.2.7 shows that l-limited policy does not belong to Π_r, and it does not have a constant competitive ratio for either $1 \mid r_i, \tau \mid \sum C_i$ or $1 \mid r_i, \tau, p_{max} \leq \gamma p_{min} \mid \sum C_i$.

Next we discuss the worst case performance of a policy that selects queues based on queue length information. As provided in [87], to serve the *stochastic largest queue (SLQ)* is the optimal policy when the system is stochastically symmetric. Here 'symmetric' means that 1)if an arrival occurs, it will join one of the k queues randomly but equally likely, and 2) service time for jobs in different queues are identically distributed. Note that 'symmetric' does not mean the inter-arrival time of jobs are fixed or job workload is revealed to be identical. It only means that the queues are equivalent from a stochastic perspective. The following proposition shows that SLQ is the optimal policy for a symmetric and stochastic polling system.

Proposition 3.2.8. *For a symmetric polling system, the optimal policy is given by SLQ which is as follows: 1) The server serves jobs in a queue non-preemptively; 2)The server should neither idle nor switch when it is at a non-empty queue (exhaustive); 3)The server stays idling in the last queue it visits when the system is empty; 4) When a queue is finished, the server switches to the next queue with the largest number of jobs in queue.*

Proof. See [87]. $\square$

For a symmetric stochastic polling system, although we know the stochastic assumptions for each queue is identical, once the arrival and service times (random variables) are revealed deterministically, they may not be the same from a deterministic standpoint. The following corollary says SLQ for the symmetric system is no longer optimal when information is revealed deterministically. Further, we show that SLQ does not even have a constant competitive ratio.

Corollary 3.2.9. *The SLQ policy does not have a constant competitive ratio for* $1|r_i, \tau| \sum C_i$*, thus it is not the optimal policy for* $1|r_i, \tau| \sum C_i$*.*

Proof. We prove the result by giving a special instance I. Suppose there are only 2 queues in the system. The first queue has 1 job with processing time p at time 0, while the second queue has no job at time 0 but has n jobs with processing time 0 arriving at time τ. SLQ will serve from queue 1 to queue 2, so that

$$C^{SLQ}(I) \quad = \quad \tau + 2n\tau + p + np,$$

and

$$C^{*}(I) \quad = \quad n\tau + 2\tau + p.$$

Let $p = n$ and $n \to \infty$ we have $\frac{C^{SLQ}(I)}{C^{*}(I)} \to \infty$. □

We conclude this section by noting that all of the policies we have discussed are mainly focused on routing disciplines: policies in Π_r use cyclic routing discipline, and SLQ are purely queue-length based. Some of these policies have constant competitive ratios when the workload variation is bounded, but when workload variation is unbounded, these policies no longer have constant competitive ratios. In the Section 3.3 we will introduce some job-priority based policies under the condition where workload variation is unbounded.

3.2.3 Simulation-based Policies

Now we consider policies that are based on simulation results. There are many simulation-based online algorithms in the literature such as the *One Machine* policy in [74] and the α-*scheduling* provided in [24]. These policies make decisions based on the result of simulations on a virtual system. For instance, One Machine policy simulates a virtual system that has the same arrivals as the real system, however in the virtual system preemption is allowed and SRPT is the optimal policy. So One Machine policy schedules jobs in the order that they are scheduled in the virtual system. Simulation-based policies usually need to simulate a virtual online benchmark in parallel and use the simulation result. For the polling system, we may want to simulate policies on virtual instances. Here we introduce two instances $\underline{I}$ and $\bar{I}$ which we call workload reduced and augmented instance respectively, such that $\underline{I}$ and $\bar{I}$ are of the same arrivals as I but $\underline{I}$ is of processing time p_{min} and $\bar{I}$ is of processing time p_{max} for all jobs. We can construct policies for I by simulating the policies on $\underline{I}$ or $\bar{I}$ and following the simulation results. Now we show that a policy which follows the decision that $\pi \in \Pi_r$ makes on either $\underline{I}$ or $\bar{I}$ has competitive ratio $\gamma(k+1)$.

Corollary 3.2.10. *A policy π that works on I but follows the decision that a policy $\bar{\pi} \in \Pi_r$ makes on either $\underline{I}$ or $\bar{I}$ is of competitive ratio $\gamma(k+1)$ for $1|r_i, p_{max} \leq \gamma p_{min}, \tau| \sum C_i$.*

Proof. Since there is only one type of jobs in either $\underline{I}$ or $\bar{I}$, preemption is not needed. We suppose $\bar{\pi} \in \Pi_r$ is a policy that works on $\underline{I}$, then $C^{\bar{\pi}}(\underline{I}) \leq (k+1)C^*(\underline{I})$ where $C^*(\underline{I})$ is the optimal completion time for instance $\underline{I}$. Because $\bar{\pi}$ is not preemptive, when $\bar{\pi}$ starts processing a job in $\underline{I}$ at time t, we know the process will be done at time $t + p_{min}$. Thus we construct the policy π by letting π follow the same service order that $\bar{\pi}$ has. Easily we have $C^{\pi}(I) \leq \gamma C^{\bar{\pi}}(\underline{I})$ because $p_i \leq \gamma p_{min}$ for every job i in instance I. Since $\underline{I}$ is a reduced instance, we have $C^*(\underline{I}) \leq C^*(I)$. Thus $C^{\pi}(I) \leq \gamma C^{\bar{\pi}}(\underline{I}) \leq \gamma(k+1)C^*(\underline{I}) \leq \gamma(k+1)C^*(I)$.

If $\bar{\pi}$ works on $\bar{I}$, where instance $\bar{I}$ is the augmented instance for instance I, with workload for all jobs in $\bar{I}$ is p_{max}. Then, we have $C^{\pi}(I) \leq C^{\bar{\pi}}(\bar{I}) \leq (k+1)C^*(\bar{I})$. Let $C^{\bar{*}}(\bar{I})$ be the completion time for a new policy which works on $\bar{I}$ but always makes the same decisions as the optimal policy for I, then $C^{\bar{*}}(\bar{I}) \leq \gamma C^*(I)$. Using $C^*(\bar{I}) \leq C^{\bar{*}}(\bar{I})$ we have

$$C^{\pi}(I) \quad \leq \quad C^{\bar{\pi}}(\bar{I}) \leq (k+1)C^*(\bar{I}) \leq (k+1)C^{\bar{*}}(\bar{I}) \leq \gamma(k+1)C^*(I).$$

□

Corollary 3.2.10 says we can simulate policies from Π_r on both workload augmented and reduced instances and follow the decisions that simulated policies make. Interestingly we find that choosing either $\underline{I}$ or $\bar{I}$ to simulate policies on results in the same competitive ratios. However, the competitive ratio for this simulation-based policy is larger than κ defined for policies in Π_r, since $\gamma(k+1) - (k+1) \geq 0$ and $\gamma(k+1) - \frac{3}{2}\gamma > 0$.

We conclude this section by noting that all of the policies we have discussed are mainly focused on queue priority, where policies in Π_r use cyclic routing discipline, and SLQ are purely queue-length based. These policies may have constant competitive ratios under the bounded condition for the workload. In the case when workload variation is unbounded, it is costly to process a large job and let small jobs wait. In the next section we will introduce some job-priority based policies under the condition where workload variation is unbounded.

3.3 Polling System with Bounded Setup Times

In this section we mainly discuss policies for the polling system with bounded setup times. Here we allow the workload variation to be unbounded. This setting in the 3D printing example corresponds to the scenario when jobs of a different color need to be printed, inks need to be switched and the switching time is bounded. When the setup times are bounded and workload variation is unbounded, an online policy may want to favor job priority instead of queue priority. For convenience of analysis, we assume switching time τ is bounded by a ratio of the minimal workload, that is $\tau \leq \theta p_{min}$. If $\tau = p_{min} = 0$, we let $\theta = 1$. Using the standard notation for scheduling problems from [50], we denote this polling problem as $1|r_i, \tau \leq \theta p_{min}| \sum C_i$. Notice that when setup time is small, i.e., θ is small, switching may not be the major contributor to the completion time delay. We shall show how to define a 'small' θ in Section 3.4. In this section, we mainly show that several policies which are designed for solving the problem without setup times, also work well in the polling system when setup times are small. Since job processing time is revealed upon arrival, we may want online policies to use job size information by selecting small jobs to process first.

3.3.1 One Machine Policy and Gittins Index Policy for Polling system

In this subsection we introduce two policies that favor jobs with short processing times. We first introduce a benchmark for deriving the competitive ratio of those policies. Usually the competitive ratio ρ is defined by $\sup_I \frac{C^{\pi}(I)}{C^{*}(I)} \leq \rho$ where $C^{*}(I)$ is the completion time for I in the offline optimal solution. However the offline problem is strongly NP-hard. To non-rigorously show the NP-hardness, we know if no preemption is allowed, even the easier problem $1|r_i|\sum C_i$ (without switching time) is strongly NP-hard [52, 23, 53]. Instead of using offline optimal solution to serve as the benchmark for online policies, in this section we mainly use a lower bound of the optimal solution as the benchmark. To get a lower bound for the optimal solution in the case with unbounded processing times and bounded setup times, we introduce the idea of setup time reduced instance. Suppose instance I is an arbitrary instance, the setup time reduced instance of I, say $\underset{\sim}{I}$, is an instance that has the same arrivals and workload as I but has no setup times. The optimal scheduling policy for $\underset{\sim}{I}$ to minimize total completion times is SRPT [54]. This schedule policy is also online, which is handy for online policies to emulate. The completion time of instance $\underset{\sim}{I}$ under SRPT is denoted as $C^{p}(\underset{\sim}{I})$. Note that setup time does not exist in $\underset{\sim}{I}$, thus $C^{p}(\underset{\sim}{I}) \leq C^{*}(I)$. In our problem, we only consider policies without preemption, but using SRPT as the benchmark. When preemption is not allowed, *One Machine (OM)* policy is proved to be the online scheduling policy with smallest competitive ratio for $\underset{\sim}{I}$ [74]. When setup time is small, we can adopt OM directly into I, regardless of the setup times. It is important to note that OM is a simulation based policy. Under OM, SRPT is simulated in parallel and decisions for OM is based on the job sequences under SRPT. The description of OM is provided in Algorithm 3.3, and the competitive ratio of OM is provided in Theorem 3.3.1.

Algorithm 3.3 One Machine Scheduling (OM)

```
1. Simulate SRPT policy on the setup time reduced instance $\underset{\sim}{I}$.
2. Schedule the jobs non-preemptively in the order of completion time of jobs by SRPT on $\underset{\sim}{I}$.
```

Theorem 3.3.1. *OM is a $(2+\theta)$-competitive online algorithm for the polling system $1|r_i, \tau \leq \theta p_{min}| \sum C_i$.The competitive ratio is tight when using SRPT on the reduced instance as the benchmark.*

Proof. Let the completion time of the j^{th} job under OM scheduling be C_j^o, and the completion time of the j^{th} job completed under SRPT as C_j^p. Since job j is also the j^{th} job that completes service under SRPT, we have $\sum_{i=1}^{j} p_i \leq C_j^p$. Then we have $C_j^o \leq C_j^p + \sum_{i:C_i^p \leq C_j^p} p_i + j\tau \leq (1+\theta)C_j^p + \sum_{i=1}^{j} p_i \leq (2+\theta)C_j^p$. Since $C^p(\underset{\sim}{I}) \leq C^*(I)$, we get $\sum C_i^o \leq (2+\theta)\sum C_i^p \leq (2+\theta)\sum C_i^*$. The competitive ratio is tight when there is only one job in the instance I which is available at time 0. Suppose this job has processing time 1. Then $C^p(I) = 1$, and $C^o(I) = 1 + (1+\theta) = 2+\theta$. □

OM algorithm is intuitive, easy to apply and polynomial-time solvable. Despite its simplicity, we may find it inefficient since setup times are ignored. Although each of the unnecessary switch only brings a small amount of delay, we may still want to avoid switching too often. Thus we provide another policy which is based on Gittins Index. Gittins Index policy is a well-studied method for solving problems such as the multi-armed bandit problem [91, 88]. Gittins Index policy is also the optimal policy for the $M/G/1$ multi-class queue scheduling problem to minimize the mean average sojourn times [92]. Here we modify the Gittins Index policy and use it on the polling system with setup time by assigning indices to jobs and choosing the best index. We call it the Gittins Index policy for polling system (GIPP), which is shown in Algorithm 3.4. In GIPP, we first simulate SRPT, and then regard the departure time of each job under SRPT as the new 'arrival' time in GIPP. We next assign Gittins index for these newly 'arrived' jobs, where the Gittins index for a job with processing time p is given by $\frac{1}{p}$ if this job and the server are at the same queue; if not, the Gittins index of this job is given by $\frac{1}{p+\tau}$. Among all the jobs that are waiting in the queue, we select the one with the largest Gittins index. By doing this, jobs from the queue which the server is serving may have larger indices than jobs from other queues. The server will prefer the jobs from the queue that it is currently serving, thus avoid switching frequently. We denote the completion time of job i in I by GIPP as C_i^g. The performance of GIPP is provided in Theorem 3.3.2.

Algorithm 3.4 Gittins Index Policy for Polling System (GIPP)

```
Require: Instance $I$
 1: Denote the queue that the server is serving as $queue_{server}$
 2: while $I$ has not been fully processed do
 3:     Simulate SRPT. Regard the departure time of the $i^{th}$ job in SRPT as the $i^{th}$ arrival time in
        GIPP. Denote the Gittins index of job $i$ as $index_i$
 4:     if $i^{th}$ arrival is at $queue_{server}$ then
 5:         $index_i = \frac{1}{p_i}$
 6:     else
 7:         $index_i = \frac{1}{p_i+\tau}$
 8:     end if
 9:     Schedule the available jobs by the largest index first
10: end while
11: return Total completion time $C^g(I)$
```

Theorem 3.3.2. *GIPP is a $(2+\theta)$-competitive online algorithm for $1|r_i, \tau \leq \theta p_{min}| \sum C_i$.The competitive ratio is tight when using SRPT on the reduced instance as the benchmark.*

Proof. Note both GIPP and OM simulate SRPT on $\underset{\sim}{I}$ and schedule job i only after job i has been processed in SRPT. So we can regard OM as FCFS in a job instance whose arrival times are $\{C_i^p, i = 1, 2, ...\}$, while GIPP serves the job with the largest Gittins index first in this instance of arrival times $\{C_i^p, i = 1, 2, ...\}$. We have $C_j^g \leq C_j^p + \sum_{i=1}^{j} \tilde{p}_i$, where $\tilde{p}_i$ is given by inverse of the Gittins index of job i. Since GIPP schedules the available jobs in the descending order of $\tilde{p}_i$, we have $C_j^g \leq C_j^p + \sum_{i=1}^{j} \tilde{p}_i \leq C_j^p + \sum_{i:C_i^p \leq C_j^p} (p_i + \tau) \leq (2+\theta)C_j^p$. We give the same example as the one in Theorem 3.3.1 to show the tightness of competitive ratio: Suppose there is only one job with $p = 1$ in instance I, available at time 0. Then $C^p(I) = 1$, and $C^g(I) = 1 + (1+\theta) = 2+\theta$. Hence proved. □

Intuitively, GIPP might perform better than OM since GIPP always schedules jobs of large indices and leaves jobs with small indices later. A job from a queue different from the server may have a small Gittins Index due to the setup time τ, thus GIPP may avoid switching frequently. Surprisingly however, GIPP does not have a competitive ratio smaller than OM. Both OM and GIPP have the same competitive ratio (see Theorems 3.3.1 and 3.3.2), when using SRPT on reduced instance as the benchmark.

Next we show the lower bound competitive ratio of the problem $1|r_i, \tau = \theta p_{min}| \sum C_i$. Notice this is a special case for the problem $1|r_i, \tau \leq \theta p_{min}| \sum C_i$ if τ and p_{min} are both revealed deterministically. There is no online algorithm that has a competitive ratio smaller than the lower bound for $1|r_i, \tau = \theta p_{min}| \sum C_i$.

Theorem 3.3.3. *If $\tau = \theta p_{min}$ and $\theta \geq 0$, then there is no online algorithm whose competitive ratio is smaller than $\theta + 1$, using SRPT on the reduced instance as the benchmark.*

Proof. If there is one job of processing time p_{min} in the system we have $\frac{C^{\pi}(I)}{C^{p}(I)} \geq \frac{(1+\theta)p_{min}}{p_{min}} = 1+\theta$. If $\tau = p_{min} = 0$, then the lower bounded ratio is $\theta + 1 = 2$ as provided in [72] since we assume $\theta = 1$ in this case. □

A natural question is whether this lower bound is the best lower bound that one can have. The answer remains open. There could be either an online policy whose competitive ratio is exactly equal to this lower bound, or a larger lower bound which is closer to the ratio $(2 + \theta)$.

3.3.2 Simulation-based Policies

In Section 3.2 we introduced the idea of simulation-based policies. A simulation-based policy usually simulates an online policy in parallel, and schedules jobs based on results of the simulated policy. In the case when setup times are bounded, we can also construct simulation-based policies, knowing that SRPT is the optimal policy for the problem without setup times. Given any instance I for the polling system, there is a reduced instance $\underset{\sim}{I}$ in which arrival times are the same as I but setup time is 0. There is also a setup time augmented instance $\tilde{I}$, where setup time is 0 but each workload is augmented with τ, i.e., $p_i \in I$ corresponds $p_i + \tau$ in $\tilde{I}$. Any online algorithm can simulate policies on $\tilde{I}$ and $\underset{\sim}{I}$ in parallel, and make decisions based on the simulation results. We have the following results via Corollaries 3.3.4 and 3.3.5.

Corollary 3.3.4. *By simulating any ρ-competitive online algorithm that we call σ on $\underset{\sim}{I}$ and scheduling jobs by the order of completion times under σ, we obtain an online algorithm on I that is $\rho(2 + \theta)$-competitive for $1|r_i, \tau \leq \theta p_{min}| \sum C_i$.*

Proof. Denote the new online algorithm on I as π. $C_j^\pi \leq C_j^\sigma + \sum_{i=1}^j p_i + j\tau \leq (2+\theta)C_j^\sigma$. Thus $\sum C_j^\pi \leq \sum (2+\theta)C_j^\sigma \leq \sum \rho(2+\theta)C_j^p$.

SRPT is the optimal policy on $\underset{\sim}{I}$ and it is online, thus it is $1-$competitive for $1 \mid r_i \mid \sum C_i$. By following the decision that SRPT makes we have a policy with competitive ratio $(2+\theta)$, which is the same as the result of Theorem 3.3.1. We can also simulate online policies on augmented instance $\tilde{I}$ and follow their decisions, by which we have the following corollary. □

Corollary 3.3.5. *By simulating a ρ-competitive online algorithm that we call σ on $\tilde{I}$ and scheduling jobs by the order of their completion times under σ, we obtain an online algorithm on I that is $2\rho(1+\theta)$-competitive for $1|r_i, \tau \leq \theta p_{min}| \sum C_i$.*

Proof. Denote the online algorithm on I as π. We first show that $C^p(\tilde{I}) \leq (1+\theta)C^p(\underset{\sim}{I})$. For an arbitrary job $\tilde{p}_i \in \tilde{I}$, it satisfies $\tilde{p}_i = p_i + \tau \leq (1+\theta)\underset{\sim}{p_i}$. Let $\tilde{\sigma}$ be a policy that works on $\tilde{I}$ but schedules jobs in the same order as SRPT on $\underset{\sim}{I}$, and serves each job the same portion as SRPT on $\underset{\sim}{I}$. Thus $C^p(\tilde{I}) \leq C^{\tilde{\sigma}}(\tilde{I}) \leq (1+\theta)C^p(\underset{\sim}{I})$. By $C_j^\pi \leq \tilde{C}_j^\sigma + \sum_{i=1}^j p_i + j\tau \leq 2\tilde{C}_j^\sigma$ we have

$$C^\pi(I) \leq 2C^\sigma(\tilde{I}) \leq 2\rho C^p(\tilde{I}) \leq 2\rho(1+\theta)C^p(\underset{\sim}{I}) \leq 2\rho(1+\theta)C^p(I).$$

Hence proved. □

Since the optimal policy for $\tilde{I}$ is SRPT, ρ in Corollary 3.3.4 is at least 1. By simulating SRPT on $\tilde{I}$ and following its decisions we have $C^\pi(I) \leq 2(1+\theta)C^p(I)$, which is greater than ratio $(2+\theta)$ that we get by simulating SRPT on $\underset{\sim}{I}$. Furthermore, if we simulate the same online policy based on either $\underset{\sim}{I}$ or $\tilde{I}$, we have $\rho(2+\theta) \leq 2\rho(1+\theta)$ for any $\rho \geq 1$, which implies that it is always better to simulate it on the reduced instance $\underset{\sim}{I}$.

3.3.3 Other Results

So far we have discussed the cases of bounded workload variation in Section 3.2 and bounded setup times in this section. We next discuss the case when both setup time and workload variation are bounded. Obviously both OM and GIPP work under this scenario and have constant competitive ratios. The algorithms in Π_r which we provide in Section 3.2 also have a constant competitive ratio in this case when the number of queues is fixed. However, in this special case we may have more policies with constant competitive ratios. Next we show that all the work-conserving policies (in which server never idles when the system is not empty) have a constant competitive ratio when setup time and workload variation are bounded.

Theorem 3.3.6. *Any non-preemptive work-conserving (WC) policy on the polling system with $p_{max} \leq \gamma p_{min}$ and $\tau \leq \theta p_{min}$ is at least $(\gamma + \theta)$-competitive with respect to the optimal solution to the offline problem.*

Proof. Let $\hat{I}$ be the workload and setup time augmented instance that all jobs are of workload $\hat{p} = p_{max} + \tau \leq (\gamma + \theta)p_{min}$, setup time is 0 in $\hat{I}$ and arrivals are the same as I. Note that any non-preemptive work-conserving policy on $\hat{I}$ is optimal since processing times for jobs in $\hat{I}$ are identical. Let σ be a non-preemptive work-conserving policy on I, and $\hat{\sigma}$ is a policy that works on $\hat{I}$ and serves jobs in the same order as σ does in I. Then $\hat{\sigma}$ is work-conserving since σ never idles when there are unfinished jobs in system, and therefore $C^{\hat{\sigma}}(\hat{I}) = C^*(\hat{I})$. Now we show $C^*(\hat{I}) \leq (\gamma + \theta)C^*(I)$. Let $\hat{S}_i^*$ be the starting time of job i in $\hat{I}$ under the optimal solution and S_i^* be the starting time of job i in I under the optimal solution. Let δ be a policy that works on $\hat{I}$ and finishes each job i at $(\gamma + \theta)C_i^*$. Notice that $(\gamma + \theta)C_i^* = (\gamma + \theta)(S_i^* + p_i) \geq (\gamma + \theta)S_i^* + \hat{p}$. Let W_i^* be the non-service times of the optimal policy on I from time 0 to S_i^*, i.e., sum of setup times and idling times till S_i^*. Then $(\gamma + \theta)S_i^* = (\gamma + \theta)(W_i^* + \sum_{j=1}^{i-1} p_j) \geq (\gamma + \theta)W_i^* + (i-1)\hat{p} \geq \hat{S}_i^*$. The last inequality holds because $\hat{S}_i^* - (i-1)\hat{p}$ is the non-service time of the optimal policy on $\hat{I}$. Since the the optimal policy on $\hat{I}$ is work-conserving, its non-service time is shorter than the non-service time of policy δ which is $(\gamma + \theta)W_i^*$. Thus we have shown that δ is a feasible schedule on $\hat{I}$. In summary, we have

$$C^\sigma(I) \leq C^{\hat{\sigma}}(\hat{I}) = C^*(\hat{I}) \leq C^\delta(\hat{I}) = (\gamma + \theta)C^*(I).$$

Assumption	p_{max}	τ	Competitive Ratio
Bounded Workload and Setup Time	$p_{max} \leq \gamma p_{min}$	$\tau \leq \theta p_{min}$	$\min\{2+\theta, \gamma+\theta, \max\{\frac{3}{2}\gamma, k+1)\}\}$
Bounded Setup Time	N/A	$\tau \leq \theta p_{min}$	$2+\theta$
Bounded Workload	$p_{max} \leq \gamma p_{min}$	N/A	$\max\{\frac{3}{2}\gamma, k+1\}$
Unbounded Workload and Setup Time	N/A	N/A	≥ 2

Table 3.3: Competitive Ratios for Different Cases

Hence proved. □

So far we have shown the existence of constant competitive ratios under different assumptions for polling systems. Table 3.3 summarizes the competitive ratios that we prove in this paper. We find from Table 3.3 that when either $p_{max} \leq \gamma p_{min}$ or $\tau \leq \theta p_{min}$ holds we can have constant competitive ratio for polling systems. In fact in many practical scenarios either the workload (processing time) is bounded or the setup time is bounded or both, where constant competitive ratio algorithms exist. In the following theorem we show that certain type of policies cannot achieve competitive ratios smaller than k.

Theorem 3.3.7. *There is no online policy with competitive ratio smaller than k for* $1 \mid r_i, \tau \mid \sum C_i$ *if its routing discipline is static or random, or is purely queue-length based, or purely job-priority based.*

Proof. Theorem 3.2.5 shows that no policy with static or random routing discipline has competitive ratio smaller than k. To show that it holds for any queue-length based routing policy π_1, we give an instance I where there is one job at each queue at time 0, with the job in queue 1 having workload p and jobs in other queues having workload 0. Since π_1 is purely queue-length based, it treats all queues equally since the queue lengths are equal. Suppose the server serves from queue 1 to queue k, we then have

$$C^{\pi_1}(I) \;\; \geq \;\; p + (k-1)p + \tau\frac{k(k+1)}{2},$$

and

$$C^*(I) \;\; = \;\; p + \tau\frac{k(k+1)}{2}.$$

Letting $p \to \infty$, we have $\frac{C^{\pi_1}(I)}{C^*(I)} \geq k$.

To show that the result holds for any purely job-priority based policy π_2, we give an instance $I^{'}$ where there are n jobs at queue 1 and one job at queue $i = 2, ..., k$ at time 0, with all jobs having workload 0. Since π_2 is purely job priority based, it does not consider any queue information. Notice both OM and GIPP are purely job priority based. Suppose π_2 serves $I^{'}$ from queue k to queue 1, thus

$$C^{\pi_2}(I^{'}) \;\; \geq \;\; \tau\left((n+k-1) + (n+k-2) + ... + n\right),$$

and

$$C^*(I^{'}) \;\; = \;\; \tau\left((n+k-1) + (k-1) + ... + 1\right).$$

Letting $n \to \infty$, we have $\frac{C^{\pi_2}(I^{'})}{C^*(I^{'})} \geq k$. Hence proved. □

Theorem 3.2.5 says that the lower bound competitive ratio is given by 2, however Theorem 3.3.7 says that if an online policy has purely static or random routing policy, or purely queue-length based routing policy or purely job-priority based routing policy, the best achievable competitive ratio is $k \geq 2$. This implies that if an online policy wants to achieve a competitive ratio smaller than k, a novel routing discipline which combines queue priority and job priority is needed. So far we have introduced policies that are either static (such as Π_r) or job priority based (such as OM and GIPP) in terms of routing discipline, and they all have competitive ratios greater than k when $\theta \geq k-2$. To achieve a smaller competitive ratio, we may need a strategy that balance both queue and job priorities. We use this idea and introduce a mixed strategy in Section 3.4 to show that a better performance can be achieved when we balance queue and job priorities.

3.3.4 Clearing Problem

In Subsection 3.3.1 we show both OM and GIPP are $(2+\theta)$ competitive, however the lower bound competitive ratio is $(1+\theta)$ as shown in Theorem 3.3.3. This means either there exists a better algorithm with competitive ratio smaller than $(2+\theta)$ or the actual lower bound is greater than $(1+\theta)$. However, in this subsection we show the lower bound $(1+\theta)$ is not trivial for the clearing problem. Note that the clearing problem is a special case of the online problem, with all arrivals happen at time 0. We show in this subsection that for the clearing problem, both of OM and GIPP have a tight competitive ratio $(1+\theta)$. We denote the clearing problem as $1|r_i = 0, \tau \leq \theta p_{min}| \sum C_i$. Moreover, since there is no arrival in the future, both OM and GIPP do not need to simulate SRPT in parallel. We thus modify OM and GIPP, by truncating the simulation part, then OM schedules jobs by smallest workload first and GIPP schedules jobs by the largest Gittins index first.

Theorem 3.3.8. *OM and GIPP both have competitive ratio $\theta+1$ for $1|r_i = 0, \tau \leq \theta p_{min}| \sum C_i$.*

Proof. There is no idling time in schedule C^o. Suppose jobs are indexed in ascending workload order so that $p^p_1 \leq p^p_2 \leq ... \leq p^p_n$, then $C^o_j \leq \sum_{i=1}^{j} p^p_i + \tau j$. Since $C^p_j = \sum_{i=1}^{j} p^p_i$ and $\tau \leq \theta p_{min}$, we have $C^o_j \leq (1+\theta)C^p_j$, thus $C^o(I) \leq (1+\theta)C^p(I)$.

We now follow jobs scheduled by GIPP. Let p^g_i be the inverse of Gittins index for i^{th} job scheduled by GIPP. If p^p_i is available for GIPP to schedule, then we have $p^g_i \leq p^p_i + \tau$. Now we consider the case in which p^p_i is not available to GIPP as GIPP serves p^p_i before p^g_i. Let $\underset{\sim}{S}(i)$ be the set of jobs served by SRPT but not by GIPP after the i^{th} schedule. At time C^g_{i-1}, if $\underset{\sim}{S}(i-1) = \emptyset$, then p_i^p is available for GIPP to schedule. Thus we assume $\underset{\sim}{S}(i-1)$ is non-empty. Choose an arbitrary job p^* from $\underset{\sim}{S}(i-1)$ we have $p_i^g \le p^* + \tau \le p_i^p + \tau$, so that $p_i^g \le p_i^p + \tau$. Therefore $C_j^g = \sum_{i=1}^{j} p_i^g \le \sum_{i=1}^{j} p_i^p + j\tau$ and $C^g(I) \le (1+\theta)C^p(I)$. □

We know the clearing problem with fixed number of queues is polynomial time solvable [93, 94]. The problem with dynamic number of queues is solvable using a 2-approximating algorithm provided in [95]. However here we say that if $p_{min} > 0$ and τ is bounded, simple algorithms exist for the clearing problem and their competitive ratios are proved to be tight and optimal. The results look worse than competitive ratio 2, however here we use SRPT as the benchmark, which means the actual competitive ratio could be smaller than $\theta + 1$. The work [95] does not show the tightness of algorithms. Before presents a mixed strategy in Section 3.4, we next present algorithms for the offline algorithm.

3.3.5 On Approximating Algorithms for the Generalized Offline Problem

In this subsection we discuss the case in which job information is available to the server at time 0 but arrival times in the future are known, unlike the clearing problem where all jobs are available at time 0. Because the focus of this paper is mainly on solving the online scheduling problem, we do not wish to discuss the offline problem in detail. The main purpose of this subsection is to model the offline problem and provide an approximation method as well. To make the formulation more comprehensive and general, we add another constraint known as *precedence*, through the use of priorities. If job j has higher priority than job i, we note $j \succ i$ and job i cannot be served before j. We suppose each job i has a weight w_i, and we want to minimize the total weighted completion times. This problem is NP-hard [23]. For the convenience of notation, we denote this problem as $1|r_i, \tau, prec| \sum w_i C_i$ and formulate it in the following manner, where ξ_i is the queue where job i arrives:

$$\begin{array}{llll} \min & \sum_{i=1}^{n} w_i C_i & & \\ s.t. & C_i \geq r_i + p_i & \text{for } i = 1, ..., n, & (3.1) \\ & C_j \geq C_i + p_i & \text{for all } j \succ i \text{ and } i = 1, ..., n & (3.2) \\ & C_i \geq C_j + p_j + \tau 1_{\xi_i \neq \xi_j} \; or \; C_j \geq C_i + p_i + \tau 1_{\xi_i \neq \xi_j} & \text{for } each\ pair\ (i,j). & (3.3) \end{array}$$

Notice the constraint (3.3) is non-linear, we relax it by using the method provided in [23]. We thus have

$$\sum_{i=1}^{n} p_i C_i \geq \sum_{j=1}^{n} p_j (\sum_{i=1}^{j} p_i) = \frac{1}{2}\left(p(I)^2 + p^2(I)\right), \tag{3.4}$$

where $p(I) = \sum_{i=1}^{n} p_i$ and $p^2(I) = \sum_{i=1}^{n} p_i^2$.

We extend Inequality (3.4) to every subset of instance I, so that for arbitrary $S \subseteq I$ we have

$$\sum_{i \in S} p_i C_i \;\; \geq \;\; \frac{1}{2}\left(p(S)^2 + p^2(S)\right). \tag{3.5}$$

Notice the linear problem which minimizes $\sum w_i C_i$ with only constraints (3.1), (3.2) and (3.5) is solvable in polynomial time via the ellipsoid algorithm [96, 23], we obtain the optimal solution to this linear problem as $\bar{C}_1, ..., \bar{C}_n$, and then schedule the jobs in order of non-decreasing $\bar{C}_i$. We note this schedule as $Schedule\ by\ \bar{C}_i$.

Theorem 3.3.9. *The* $Schedule\ by\ \bar{C}_i$ *is a* $(3 + 2\theta)-$*competitive algorithm for* $1|r_i, \tau \leq \theta p_{min}, prec|\sum w_i C_i$.

Proof. We let $\tilde{C}_i$ be the actual completion time for job i by following $schedule\ by\ \bar{C}_i$. We assume $\bar{C}_1 \leq ... \leq \bar{C}_n$ are completion times on solving $\min \sum w_i C_i$ under constraints (3.1), (3.2) and

(3.5), then for any $S = \{1, 2, ..., j\}$ we have

$$\bar{C}_j \sum_{i=1}^{j} p_i \geq \sum_{i=1}^{j} \bar{C}_i p_i \geq \frac{1}{2}\left(p(S)^2 + p^2(S)\right).$$

Since $\tilde{C}_j \leq \max_{i=1}^{j} r_i + \sum_{i=1}^{j} p_i + j\tau$ and $\max_{i=1}^{j} r_i \leq \bar{C}_j$, therefore $\tilde{C}_j \leq (3+2\theta)\bar{C}_j$. Suppose C_i^* is the optimal completion time for job i in the original problem with constraints (3.1) and (3.3), because constraint (3.5) is a relaxation of (3.3), then $\sum w_i \bar{C}_i \leq \sum w_i C_i^*$. Thus $\sum w_i \tilde{C}_i \leq (3 + 2\theta) \sum w_i C_i^*$. □

This competitive ratio looks worse than the ratio $2 + \theta$ achieved by online policies, however we should notice in this offline problem we have precedence constraints. This also points to the problem with precedence is harder than the one without. The discussion for the offline precedence and release time constrained scheduling problem can be found in [97, 74, 23], but none of them consider a setup time constraint.

3.4 A Mixed Strategy

In Section 3.2 we provided a policy set Π_r. Policies in Π_r are based on cyclic routing discipline, and they have competitive ratio κ when workload variation is bounded. When workload variation is unbounded, we may want to use GIPP, with competitive ratio $\theta+2$. In reality, it is common to see cases where workload variation is unbounded, and setup times are large but bounded. For example in the reconfigurable manufacturing, robust components are used to process customized jobs and also designed to reduce setup times to make the manufacturing system more efficient. It is very rare to see unbounded setup times [98]. In the 3D printing example, jobs are usually highly customized and heterogeneous, and the workload variation could be large in this case. Besides, when jobs from a very different prototype is received, many setup steps need to be performed so that setting up the 3D printer would take a large amount of time (usually bounded). Thus in this section we discuss the problem where setup time is large but bounded, and workload variation is unbounded. From Section 3.3 we know that this online problem can be solved by OM or GIPP, however, if a large workload is rare, we may want to adopt a policy π^e from Π_r most of time when it gives a competitive ratio smaller than $\theta + 2$. This motivates us to construct a mixed strategy such that if there is no workload greater than a threshold, say ηp_{min}, then a policy from Π_r is applied, resulting in a competitive ratio $\max\{\frac{3}{2}\eta, k+1\}$; if there is a new arrival with workload greater than ηp_{min}, then GIPP is applied so competitive ratio is $\theta + 2$. If $\kappa(\eta) = \max\{\frac{3}{2}\eta, k+1\} \leq \theta + 2$, this mixed strategy has a better expected performance than simply using GIPP for a finite job instance, as we will see later. Specifically, we assume that the exhaustive policy π^e serves continuously without waiting and skips empty queues when switching. Within a queue we let π^e serve following *Shortest Processing Time First* (SPT). It helps reduce the queue length within each queue, though this does not reflect on the competitive ratio. A formal statement of this mixed strategy π^m is provided in Algorithm 3.5.

Algorithm 3.5 Deterministic Mixed Strategy π^m

```
1: while The system is not empty do
2:     Use $\pi^e$: When a service is done, serve the next job in the queue with shortest processing
   time first; switch to the next non-empty queue when the current queue has been exhausted
3:     if There is an arrival whose workload is greater than $\eta p_{min}$ then
4:         if The server is serving then
5:             Finish the current job
6:         else if The server is setting up then
7:             Halt setting up and the server stays in the current queue
8:         end if
9:         Use GIPP
10:    end if
11: end while
12: return Total completion time $C^{\pi_m}(I)$
```

Theorem 3.4.1. *If* $p_{min} > 0$, $\eta \leq \theta$, $\kappa(\eta) \leq \theta+2$ *and workload is drawn from a known distribution, then*

$$\mathbf{E}[\frac{C^{\pi^m}(I)}{C^*(I)}] \leq \nu(\eta),$$

where $\nu(\eta) = \kappa(\eta)\mu(\eta)^{n(I)} + (\theta+2)(1-\mu(\eta)^{n(I)})$ *and* $\mu(\eta) = \mathbf{P}(p_i \leq \eta p_{min})$*, prior to revealing.*

Proof. Let $p_{min} = 1$. If all the jobs are of workload smaller than η, then throughout the busy period, $\pi^m = \pi^e$ and $\frac{C^{\pi^m}(I)}{C^*(I)} \leq \kappa(\eta)$. Now we show that if there is a workload in the busy period with workload greater than η, then $\frac{C^{\pi^m}(I)}{C^*(I)} \leq \theta+2$. Suppose $I = I_1 \cup I_2$, where I_1 is the job instance that is served by π^e in I, and I_2 is the rest of I which is served by GIPP. Let S_2 be the time when GIPP is triggered. Then $C^{\pi^m}(I) = C^{\pi^e}(I_1) + S_2 n(I_2) + C^g(I_2)$ where $C^g(I_2)$ is the completion time for GIPP triggered at time S_2. For the optimal solution, we have $C^*(I) = C^*(I_1 \cup I_2) \geq C^*(I_1) + R_2 n(I_2) + C^p(I_2)$, where R_2 is the time when a job with workload greater than η arrives and $C^p(I_2)$ is the completion time for I_2 under SRPT. Since $R_2 \leq S_2 \leq R_2 + \max\{\theta, \eta\} = R_2 + \theta$ by $\eta \leq \theta$, we have,

$$\frac{C^{\pi^m}(I)}{C^*(I)} \leq \frac{C^{\pi^e}(I_1) + S_2 n(I_2) + C^g(I_2)}{C^*(I_1) + R_2 n(I_2) + C^p(I_2)} \leq \frac{C^{\pi^e}(I_1) + (R_2+\theta)n(I_2) + C^g(I_2)}{C^*(I_1) + R_2 n(I_2) + C^p(I_2)}.$$

If $R_2 \geq \theta$, then we have $\frac{C^{\pi^m}(I)}{C^*(I)} \leq \max\{\kappa(\eta), 2, \theta+2\} = \theta+2$. If $R_2 < \theta$, then the server is setting up when the new arrival occurs. Setup is aborted immediately and GIPP is started, thus $S_2 = R_2$ and

$$\frac{C^{\pi^m}(I)}{C^*(I)} \leq \frac{C^g(I_2)}{C^p(I_2)} \leq \theta + 2.$$

Therefore

$$\begin{aligned}\mathbf{E}[\frac{C^{\pi^m}(I)}{C^*(I)}] &= \mathbf{E}[\frac{C^{\pi^m}(I)}{C^*(I)}|I_2 = \emptyset]\mathbf{P}(I_2 = \emptyset) + \mathbf{E}[\frac{C^{\pi^m}(I)}{C^*(I)}|I_2 \neq \emptyset]\mathbf{P}(I_2 \neq \emptyset) \\ &\leq \kappa(\eta)\mu(\eta)^{n(I)} + (\theta+2)(1-\mu(\eta)^{n(I)}).\end{aligned}$$

Hence proved. □

If only a policy from Π_r is used when $I_2 \neq \emptyset$, then the expected performance is smaller than

$\kappa = \max\{\frac{3}{2}\gamma, k+1\}$, which may be greater than $\theta + 2$ when γ is large. Also we know that if only GIPP is applied, the expected performance is bounded by $\theta + 2$. Since $\nu(\eta) = \kappa(\eta)\mu(\eta)^{n(I)} + (\theta + 2)(1 - \mu(\eta)^{n(I)})$, the Deterministic Mixed Strategy has an expected performance bound smaller than $\kappa(\eta)$ or $(\theta + 2)$. Note that $\kappa(\eta) = \max\{\frac{3}{2}\eta, k+1\}$. If $\frac{3}{2}\eta \leq k+1$, then $\kappa(\eta) = k+1$. The optimal η^* for minimizing $\nu(\eta)$ in this case is given by $\frac{2}{3}(k+1)$ since $\mu(\eta)$ is an increasing function of η. If $\frac{3}{2}\eta \geq k+1$, then $\nu(\eta) = (\theta+2) + \mu(\eta)^{n(I)}(\frac{3}{2}\eta - (\theta+2))$, and the optimal value η^* can be obtained by solving

$$\begin{aligned} \min \quad & (\theta+2) + \mu(\eta)^{n(I)}(\tfrac{3}{2}\eta - (\theta+2)) \\ s.t. \quad & \tfrac{2}{3}(k+1) \leq \eta \leq \min\{\tfrac{2}{3}(\theta+2), \theta\}. \end{aligned} \tag{3.6}$$

Notice $\eta = \frac{2}{3}(k+1)$ is a feasible solution to System (3.6), thus by solving System (3.6) we can obtain the optimal solution to $\nu(\eta)$, which is η^*. Note that $\nu(\eta^*)$ is smaller than $\kappa(\eta^*)$ and $\theta + 2$.

In the online problem where the information of future jobs is unknown to the server, the server cannot actually optimize the System (3.6). A practical way is letting $\eta = \frac{2}{3}(k+1)$ for the Deterministic Mixed Strategy, which eventually results in an expected performance bound smaller than $\theta + 2$ for a finite $n(I)$. It is important to note that Deterministic Mix Strategy has expected performance bound smaller than $\max\{\kappa, \theta+2\}$, however this only happens when p_i values are drawn from a single distribution and the number of jobs $n(I)$ is finite. It is also important to point out that $\nu(\eta)$ is an expected value. The real competitive ratio of this strategy is $\theta + 2$. However, this strategy shows that by balancing queue and job priorities, one could design policies with better performance. This strategy also gives us insights of revealing future information, and shows that if one can reveal or estimate the number of jobs in a busy period as well as the workload distribution, then the System (3.6) is solvable and a smaller expected competitive ratio may be obtained. In a future research study we will discuss how to estimate future information and use it for better system performance.

3.5 Concluding Remarks and Future Works

In this paper we consider scheduling policies in the polling system without stochastic assumptions. Our analysis provides a novel way to classify scheduling policies for polling systems by considering their worst case performance, i.e., competitive ratio. It allows one to describe the performance of some policies even when their average performance is intractable. Conditions for the existence of constant competitive ratio are discussed and the worst case performance for several well-studied polling system scheduling policies are provided. We show that both cyclic exhaustive policy and cyclic gated policy have a constant competitive ratio κ for problem $1 \mid r_i, \tau, p_{max} \leq \gamma p_{min} \mid \sum C_i$, but they do not have a constant competitive ratio for the problem $1 \mid r_i, \tau \mid \sum C_i$. Interestingly, we find cyclic is the optimal static routing discipline, and when cyclic routing discipline is adopted, revealing the processing times for jobs is not helpful in reducing the competitive ratio if $\frac{3}{2}\gamma \leq k+1$. We also find some policies with provable average performance do not have constant competitive ratios for $1 \mid r_i, \tau, p_{max} \leq \gamma p_{min} \mid \sum C_i$ such as l-limited policy and SLQ policy. We provide online policies that balance well the future uncertainty and current information availability, such as GIPP. Besides, we show that if the routing discipline for an online policy relies only on a routing table (static or random), queue-length, or job processing times, then the competitive ratio of this policy cannot be smaller than k. We then provide a mixed strategy which performs better that Π_r and GIPP when number of jobs in finite. Our analysis suggests a policy with competitive ratio smaller than k may need to incorporate more information besides job processing times or number of jobs in queue. However, the question that whether there exists an online policy with constant competitive ratio for the problem $1 \mid r_i, \tau \mid \sum C_i$ without any bound conditions for workload and setup times remains open. Also, it is unclear if there exists a better lower bound competitive ratio for all the online policies. A future problem to consider will be searching for online policies with smaller competitive ratios and deriving a better competitive ratio lower bound for all the online policies.

4. PEAK AGE OF INFORMATION IN PRIORITY QUEUEING SYSTEMS

4.1 Introduction

In the recent years the notion of Age of Information (AoI) has garnered attention from several researchers. The main applications that have been cited include sensor networks, wireless networks and autonomous vehicle systems [99], as in all those cases it is important to know the freshness of information. Our research has been motivated by an application in smart manufacturing of the future where edge devices, sensors in particular, with limited processing capabilities, would monitor the health of various tools, condition of components and quality of work pieces in machines. This sensed information would be used to make real-time decisions such as tool changes, re-calibration and rework, thereby improving overall cost and quality of the manufactured products. Hence it is crucial to consider the freshness of information to make these decisions, for some type of which AoI is an ideal choice.

AoI is a metric defined and used by researchers such as Kaul et al [99] to describe the freshness of data. We consider a system where a data source (sensor or resource) from time to time sends updates or files (in this paper we call each update or file a "packet") to the processor (also called server). The time when the data source generates a packet can also be regarded as the arrival time (also called release time) of the packet into the system. The server processes packets in a non-preemptive way. Unprocessed packets are queued due to the limited processing capacity of the server. AoI at an arbitrary time point t is defined as the length of period between time t and the most recent release time among all the packets that have been processed. Mathematically, we define the AoI at time t as $\triangle(t) = t - \max\{r_l : C_l \leq t\}$, where r_l is the release time of the l^{th} packet that is generated and C_l is the time when it is processed by the server (also called completion time). While the time-average AoI could be a metric to measure data freshness, many researchers consider Peak Age of Information (PAoI) as a more tractable metric [100, 101]. We let the n^{th} peak value of $\triangle(t)$ be A_n, which is a random variable and it is shown in Figure 4.1. The

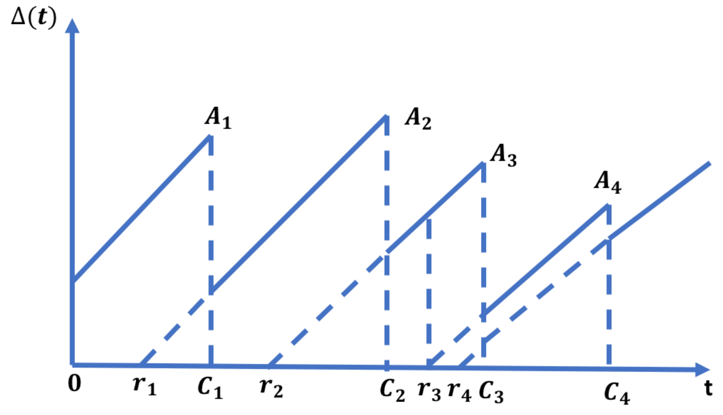


Figure 4.1: Age of Information for a Single Queue

expectation of this peak value, i.e., $\boldsymbol{E}[A_n]$, is then defined as PAoI for this data source. Next we extend this notion to multiple sources and formulate our model.

It has been well documented and accepted that monitoring and sensing according to a Poisson process is effective [102]. In that light we consider multiple data sources (sensors) that monitor according to a Poisson process with potentially different rates due to the difficulty in sensing (recall our motivation example of a manufacturing setting). Also, not all streams of packets have the same priorities, and we consider a setting where there are k data sources prioritized from 1 (highest) to k (lowest). There is a single processor (server) that "serves" the k packet streams based on a static priority mechanism. We study the static queue priorities mainly because in many cases, there are some data sources whose packets contain age sensitive information or emergency information such as high temperature, high pressure (see [103]). These data packets need to be transmitted as soon as possible, thus high static priorities for these data sources are needed. Another example is given by Maatouk et al [104], which says that in the vehicle network, the safety related data should be allocated higher priorities over the other non-safety related data to improve the traveling experience. All these real applications motivate us to consider a multi-queue system with static priorities.

The system model is provided in Figure 4.2. We consider two settings in this paper: one in which there is a buffer for each data source that can hold only one packet at a time; another where each buffer can hold infinitely many packets. It is still unknown which setting has a smaller PAoI for each queue. For the first setting we discuss the system M/G/1/1+$\sum 1^*$ and M/G/1/1+$\sum 1$ for general service times. The notation $1 + \sum 1^*$ means besides the processing area at the server, each data source has a buffer with size one. The asterisk means that the packet waiting in the buffer is replaced by the newest arrival, the same as the notation used in [105]. If there is no asterisk, i.e., M/G/1/1+$\sum 1$, then it means that packet that enters the buffer will not be replaced by new arrivals. For the setting of infinite buffer size, we discuss the M/G/1 type queues with First Come First Serve (FCFS) and Last Come First Serve (LCFS) service disciplines respectively. Our objective is to obtain the PAoI for each class of sensed information under each setting assuming a general distribution of service time for packets, and then discuss the advantage of each setting.

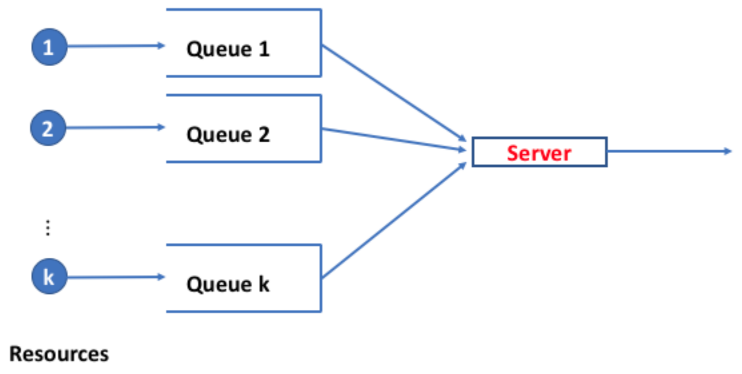


Figure 4.2: System Model

This system is modeled as a multi-class multiple parallel queueing system with static priorities. The performance of PAoI in this system has not been studied before. The main contributions of our paper are listed as follows:

1. We first provide a novel modeling method to evaluate the PAoI of multi-queue systems by

focusing on the busy period of server and buffer status. Using this method we provide the exact PAoI for single buffer prioritized system M/M/1/1 $+\sum 1^*$ and system M/G/1/1 $+\sum 1$ with small number of queues k. And we further provide the bounds (which are also excellent approximations) for PAoI of the system M/G/1/1 $+\sum 1^*$.

2. We provide the exact PAoI for the infinite buffer prioritized system M/G/1 with FCFS and LCFS for general number of queues k. We also provide a mixed strategy which allows some queues to apply FCFS and the others to apply LCFS, and the PAoI of each queue under this mixed strategy can be obtained by our exact analysis.

3. By providing the exact PAoI of systems above, we show a surprising result that LCFS is not the optimal service discipline for minimizing PAoI among all the non-preemptive work-conserving disciplines in the system where buffer size of each queue is infinite. We also show a counter-intuitive finding that having a buffer with size one at each queue does not always provide lower PAoI than the having a buffer with infinite size.

4. We reveal the fact that PAoI of queues with low priorities are sensitive to the traffic intensity of queues with high priorities, so queues that contain important or time-sensitive information should be given high priorities. Also, if the PAoI averaged across queues is to be minimized, we show that it is beneficial to assign low priorities to high traffic queues.

The rest of this paper is organized as follows. A summary of the literature is provided in Section 4.2. Then, in Section 4.3 we provide the PAoI analysis for M/G/1/1+$\sum 1^*$ and M/G/1/1+ $\sum 1$ type queues, where arrivals are Poisson and service times for packets are iid and generally distributed. In Section 4.4 we provide the PAoI analysis for queues with infinite buffer size, under both FCFS and LCFS disciplines within each queue. We perform numerical studies in Section 4.5, and make concluding remarks as well as discuss the future work in Section 4.6.

4.2 Related Work

The idea of data age, freshness and timeliness for data warehouses are introduced and discussed in [106, 107]. In recent years, data freshness has drawn much more attention because of the development of Internet of Things, fog computing and edge data storage [108, 109]. Kaul et al [99] firstly provided average AoI for M/M/1, M/D/1 and D/M/1 type queues. Costa et al [105] then obtained analytical results of average AoI and PAoI under FCFS for M/M/1/1, M/M/1/2 (which allows drop of new arrivals) as well as M/M/1/2* (which allows update for the waiting packet) queues. The performance of LCFS policy for the single queue case where service times are gamma distributed was provided by Najm and Nasser [110]. Soysal and Ulukus [111] considered G/G/1/1 type queues and provided bounds of AoI for different arrival and service processes. Zou et al [112] discussed PAoI and AoI under the waiting procedure in M/G/1/1 and M/G/1/2* cases. Kosta et al [113] discussed the performance of AoI and PAoI for the single-queue slotted-time system with and without packets management. Inoue et al [101] discussed the relationship between PAoI and AoI for the single queue case, and they provided the AoI/PAoI analysis for different single queue models including M/G/1 and G/M/1 systems with FCFS, preemptive LCFS, and non-preemptive LCFS. Some recent works have considered AoI for single server with multiple queues. Huang and Modiano [100] provided the PAoI for multi-class M/G/1 and M/G/1/1 queues where all packets flow into a combined queue. Najm and Telatar [114] considered the M/G/1/1 system with multiple sources updating while allowing preemption. Kosta et al [115] considered a slotted time system and discussed the performance of round-robin, working-conserving and random policy. Jiang et al [116] considered an AoI minimization problem with Bernoulli arrivals in slotted time system and modeled it as a MDP problem to determine which data source to serve next. They showed that Whittle's index policy is a near optimal policy and they also provided a decentralized policy which achieves nearly identical performance as the Whittle's index policy. The optimality of Whittle's index policy was further discussed by Maatouk et al in [117]. Kadota et al [118] considered an AoI minimization problem in a slotted time system with throughput constraint considerations. Talak et al [119] considered weighted AoI and PAoI minimization problem in a discrete timed system with channel errors. It is also pointed out by [119] that the PAoI/AoI for the discrete time queues may differ significantly from their continuous time counterpart. Other AoI/PAoI minimization problems for discrete time systems can be found in [120, 121, 116, 122, 123, 124]. The multi-class queues with FCFS and LCFS across queues are discussed in Yates and Kaul [125]. A detailed review for the current literature for AoI is also provided in [125].

However, we notice that if FCFS or LCFS across queues is adopted in the multi-queue system, queues with high arrival rates will be served more frequently. It is not always the case that the queues with higher arrival rates are more important. Queues with low traffic intensities may also be important, and their packets may need to be processed as soon as they enter the queues. Besides, spending too much time processing a certain data source is a waste of service resource. We thus want to consider a service policy which gives certain queues higher priorities. Such a multiple-queue system with queue priorities has been studied for a long time, however most of previous works focused on different metrics such as queue lengths and waiting time distributions [126, 127]. AoI and PAoI are metrics introduced in recent years, and their performance under queue priorities are not well understood. Najm et al [103] considered a system with two streams of different priorities and discussed different service disciplines for the low priority stream. Recently, Kaul and Yates [128] modeled the AoI of M/M/1 priority queues as a hybrid system by assuming the waiting room (buffer size) for the system is either null or one. However, their model is restrictive since there is at most one buffer for all queues. Maatouk et al [104] discussed the model where each queue has an individual buffer, and provided a closed-form expression for AoI using a hybrid system analysis. However, it is assumed in [104] that arrival and service rates for all queues are exponential and identical. It is still unknown if having finite buffer can help reduce the PAoI for each queue in the multi-queue scenario, especially when arrival and service rates differ from one queue to another. In our work, we for the first time provide the exact PAoI for the system where queues are prioritized and each queue has its independent waiting room (buffer), arrival rate and service rate. In this paper we only consider static queue priorities because in many applications, some data streams have more important information which need to be transmitted as soon as possible. Moreover, the queue performance is more tractable when queue priorities are fixed, however the behavior of PAoI in this case has not been fully understood. In this paper, we provide a new modeling approach of calculating age-related metrics by focusing on the buffer state. We derive the exact

PAoI for M/M/1 $+ \sum 1^*$ system and M/G/1 $+ \sum 1$ system, as well as bounds for M/G/1 $+ \sum 1^*$ queues with priorities. Also, for the infinite buffer size case, we derive the exact PAoI for M/G/1 system with FCFS and LCFS service disciplines within each queue. We seek to find a priority order and service discipline that would result in low average PAoI across queues. We also seek to understand the effect of arrival rates and service times on the PAoI for systems under different settings for buffer size and service disciplines.

4.3 Queues with Buffer Size One

In this section we first discuss the system M/G/1/1+$\sum 1^*$ in which the buffer size for each queue is one, and the arrival process for each queue i is a Poisson process with rate λ_i. The service time (processing time) P_i for packets from queue i is iid with cdf $F_i(x)$ and mean $\frac{1}{\mu_i}$. A new arrival will replace the packet waiting in the queue (if there is one) since the newest packet contains the most recent information of the source. Note that this model is different from the M/G/1/1 model introduced in [100, 114]. In their model, there is no buffer for each queue, so whenever a packet arrives and sees the server being busy, the packet is either rejected or preempts the packet in service. Further, when the server becomes available, it has to wait until the next packet arrives. In our model, the buffer allows the server to serve packets whenever the server becomes available, which is potentially more efficient by not waiting for the next packet. Moreover, only keeping the most recent packet in the buffer can possibly reduce the server's load, and also guarantee that the most recent packet can be processed once the server becomes available.

The difficulty in analyzing such a system with waiting room for one packet in each queue is that packets entering the system are only a subset of packets generated by the data source, due to some getting rejected. Focusing on how each packet goes through the system often makes modeling more complicated [128]. Instead, in our model we introduce a new way of modeling such systems, which is to incorporate the buffer state. Note that we can also use this idea to derive PAoI for other systems with buffer size more than one, as we will see in Section 4.4. In this section we only consider the model with buffer size of one for each queue, we now show how this buffer size of one helps us characterize PAoI. We depict a sample path of the buffer state for queue i in Figure 4.3

with notations described subsequently. From Figure 4.3 we can see that buffer state of queue i is either 0 or 1. When the buffer state is 1 (the buffer is full), we say the buffer is busy. We use r_{ij}, S_{ij} and C_{ij} to denote the release time, starting time of processing and completion time of j^{th} packet that arrives at queue i (note that S_{ij} and C_{ij} only exist if the packet is processed by the server). Suppose at time r_{i1} packet 1 arrives at queue i. It waits until time S_{i1} when the server becomes available to serve it by removing the packet from the buffer and placing it in the processing area. Right after time S_{i1} buffer i becomes empty until packet 2 arrives at time r_{i2}. Packet 2 stays in the buffer for a while, then gets replaced by packet 3 at time r_{i3}. Packet 3 is then replaced by packet 4 at time r_{i4}. At time S_{i4} the server becomes available and starts serving packet 4, and the buffer becomes empty again. The service of packet 4 is completed at time C_{i4}, and the peak age of information upon the completion of packet 4 is given as $C_{i4} - r_{i1}$, which is equal to

$$C_{i4} - r_{i1} \quad = \quad (C_{i4} - S_{i4}) + (S_{i4} - r_{i2}) + (r_{i2} - S_{i1}) + (S_{i1} - r_{i1}). \qquad (4.1)$$

The term $(C_{i4} - S_{i4})$ of Equation (4.1) is the processing time of packet 4, and $(S_{i4} - r_{i2})$ is the time period during which the buffer has one packet. The third term $(r_{i2} - S_{i1})$ is the time period during which the buffer stays empty, and the last term $(S_{i1} - r_{i1})$ is the waiting time of packet 1. Recall that the processing times of packets from the same source are iid, so the expected value of $(C_{i4} - S_{i4})$ is $\boldsymbol{E}[P_i] = \frac{1}{\mu_i}$. The buffer is empty during time $(r_{i2} - S_{i1})$, and we know that there is no arrival in $(r_{i1}, S_{i1}]$. Using the memoryless property of exponential inter-arrival times, the expected time of buffer staying empty is the expected inter-arrival time, $\boldsymbol{E}[I_i] = \frac{1}{\lambda_i}$. Therefore we can write the PAoI for source i as

$$\boldsymbol{E}[A_i] \quad = \quad \boldsymbol{E}[P_i] + \boldsymbol{E}[W_i] + \boldsymbol{E}[I_i] + \boldsymbol{E}[G_i], \qquad (4.2)$$

where $\boldsymbol{E}[G_i]$ is the expected waiting time (in buffer) of the packet that is eventually processed by the server, and $\boldsymbol{E}[W_i]$ is the expected length of time period when the buffer is continuously occupied (busy). Note that Equation (4.2) holds true for every queue i. For M/G/1 type queues,

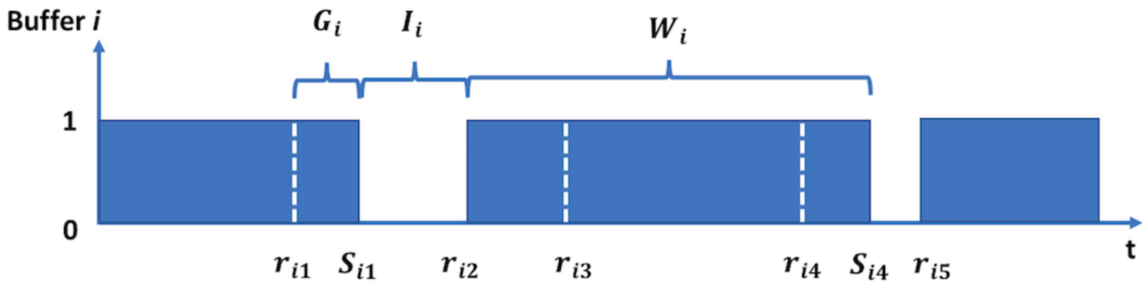


Figure 4.3: Buffer State for Queue i

we have already stated that $\boldsymbol{E}[P_i] = \frac{1}{\mu_i}$ and $\boldsymbol{E}[I_i] = \frac{1}{\lambda_i}$. The difficult part remains in calculating $\boldsymbol{E}[W_i]$ and $\boldsymbol{E}[G_i]$. From Figure 4.3 we observe that if we reject new arrivals (instead of new arrivals replacing ones in the buffer) when the buffer is full, W_i is not changed. If we reject the most recent arrival (instead of the system that we are analyzing) when the buffer is full, then W_i is the waiting time for the packet that enters the buffer, which equals to W_i if we keep the most recent arrival. Using this property, if we let p_i be the probability that buffer i is full, then from Little's Law we know the average queue length is $p_i = \lambda_i(1-p_i)\boldsymbol{E}[W_i]$. So we have

$$\boldsymbol{E}[W_i] \quad = \quad \frac{p_i}{\lambda_i(1-p_i)}. \tag{4.3}$$

From Equation (4.3), $\boldsymbol{E}[W_i]$ can be obtained once we know p_i. We shall discuss how to find p_i later in this section. Now we continue with the system where new arrivals replace the existing ones in queue. We first characterize G_i, which depends on W_i, as we will see in Lemma 4.3.1.

Lemma 4.3.1. $\boldsymbol{E}[G_i|W_i = t] = \frac{1}{\lambda_i}(1-e^{-\lambda_i t})$.

Proof. Suppose there are $N(t) = m$ packets arriving during W_i, then G_i is the time gap from the release time of the m^{th} packet R_m to time t. From Campbell's Theorem (P173, Theorem 5.14 in [129]) we have

$$
\begin{aligned}
& \boldsymbol{P}(G_i < x | N(t) = m, W_i = t) \\
= \;& \boldsymbol{P}(R_m > t - x | N(t) = m, W_i = t) \\
= \;& \int_{t-x}^{t} \frac{m}{t} (\frac{u}{t})^{m-1} du \\
= \;& 1 - (\frac{t-x}{t})^m.
\end{aligned}
$$

Thus by integrating $\boldsymbol{P}(G_i > x | N(t) = m, W_i = t)$ for x from 0 to t, we have

$$\boldsymbol{E}[G_i | N(t) = m, W_i = t] = \frac{t}{m+1}.$$

Then, unconditioning using $\boldsymbol{P}(N(t) = m) = e^{-\lambda_i t} \frac{(\lambda_i t)^m}{m!}$, we get

$$
\begin{aligned}
\boldsymbol{E}[G_i | W_i = t] &= \sum_{m=0}^{\infty} \frac{t}{m+1} e^{-\lambda_i t} \frac{(\lambda_i t)^m}{m!} \\
&= \sum_{m=0}^{\infty} e^{-\lambda_i t} \frac{(\lambda_i t)^{m+1}}{(m+1)!} \frac{1}{\lambda_i} \\
&= \frac{e^{-\lambda_i t}}{\lambda_i} (e^{\lambda_i t} - 1).
\end{aligned}
$$

□

Lemma 4.3.1 shows that one needs to know the distribution of W_i or its Laplace–Stieltjes transform (LST) to get $\boldsymbol{E}[G_i]$. The exact LST of W_i can be obtained when service times are exponentially distributed, as we will see in Section 4.3.1. If service times are generally distributed, we provide the bounds for PAoI based on result of Lemma 4.3.1, which we will see in Section 4.3.2.

4.3.1 Exact Analysis for M/M/1/1+$\sum 1^*$ Type Queues

In this subsection we consider a special case where the processing time P_i is $exp(\mu_i)$ for all i and discuss how to calculate $\boldsymbol{E}[W_i]$ and $\boldsymbol{E}[G_i]$. Knowing the LST of W_i can help us obtain both

$\boldsymbol{E}[W_i]$ and $\boldsymbol{E}[G_i]$, so in this subsection we focus on calculating LST of W_i. Since W_i is not affected by which packet we reject when the buffer is full, in this subsection, we assume that we reject the most recent arrivals. We adopt the method used to characterize the busy period in [130] to derive the LST of W_i, i.e., $\boldsymbol{E}[e^{-sW_i}]$. Let $B_i(t)$ be the number of priority i packets in buffer i at time t, $B_i(t) \in \{0, 1\}$. Let $J(t) \in \{0, 1, ..., k\}$ be the packet that is in service at time t, where $J(t) = 0$ means the server is idling. The vector $S(t) = (J(t), B_1(t), ..., B_k(t))$ thus indicates the state of the system at time t. Obtaining the stationary state seen by packets that enter the system (which are not all the arrivals) is crucial for our analysis, so in the following we introduce an approach to find its stationary probability. From PASTA [129] we know that the time average performance of the system is the same as that seen by Poisson arrivals. If a packet from class i sees $B_i(t) = 0$, it then enters the buffer if the server is busy, or enters the server directly if the server is idling. Thus the state that an entering packet from source i observes is always $B_i(t) = 0$. We let $\psi_j(s) = \frac{\mu_j}{\mu_j+s}$ be the LST of service time for packets from queue j. Because the service time is exponential, $\psi_j(s)$ is also the LST of remaining service time of the packet observed by an entering packet, if the packet in service is from queue j. Let U_i be the remaining service time observed by a packet entering queue i. If we assume that the system starts from time 0, then we have for queue 1 that

$$\begin{aligned}\boldsymbol{E}[e^{-sW_1}|B_1(0) = 0] &= \boldsymbol{E}[e^{-sU_1}|B_1(0) = 0]\\ &= \boldsymbol{P}(J(0) = 0|B_1(0) = 0) + \sum_{j=1}^{k} \psi_j(s)\boldsymbol{P}(J(0) = j|B_1(0) = 0).\end{aligned}$$

Before characterizing $\boldsymbol{E}[e^{-sW_2}]$ for buffer 2, we first introduce the busy period of the server. Let T_1 be the time period that the server is continuously busy processing packets from buffer 1, and $\eta_1(s) = \boldsymbol{E}[e^{-sT_1}]$. The busy period T_1 always starts from processing a packet from buffer 1. Suppose the processing time of this packet is of length $P_1 = l$ and if there is more than one priority 1 packet arriving during $[0, l]$, then another busy period will start from time l and the new busy period is identically distributed as T_1. Thus we have $\boldsymbol{E}[e^{-s(l+T_1)}|P_1 = l, B_1(l) = 1] = e^{-sl}\eta_1(s)$.

If there is no arrival then the busy period would be l only, then by unconditioning on $B_1(l)$ we have

$\boldsymbol{E}[e^{-(l+T_1)}|P_1 = l] = e^{-sl}\eta_1(s)(1 - e^{-\lambda_1 l}) + e^{-sl}e^{-\lambda_1 l}$.

Unconditioning on $P_1 = l$ we have

$$\eta_1(s) \;=\; \eta_1(s)[\psi_1(s) - \psi_1(s+\lambda_1)] + \psi_1(s+\lambda_1).$$

Thus the LST of T_1 is given by $\eta_1(s) = \frac{\psi_1(s+\lambda_1)}{1-\psi_1(s)+\psi_1(s+\lambda_1)} = \frac{\mu_1(s+\mu_1)}{s^2+2\mu_1 s+s\lambda_1+\mu_1^2}$ and the derivative of $\eta_1(s)$ at $s = 0$ is given by $\eta_1^{'}(s)|_{s=0} = \frac{-\lambda_1-\mu_1}{\mu_1^2}$.

Now we characterize the LST of W_2 by the fact that $\boldsymbol{E}[e^{-sW_2}] = \boldsymbol{E}[e^{-s(U_2+T_1)}]$ and conditioning on different scenarios observed by the packets that enter buffer 2. If the server is idling when a packet from source 2 enters, then

$$\boldsymbol{E}[e^{-s(U_2+T_1)}|B_1(0) = 0, J(0) = 0, B_2(0) = 0] \;=\; 1.$$

If the server is busy processing a packet from buffer j for $j \in \{1, ..., k\}$, and buffer 1 is not empty, then we have

$$\boldsymbol{E}[e^{-s(U_2+T_1)}|B_1(0) = 1, J(0) = j, B_2(0) = 0] \;=\; \boldsymbol{E}[e^{-sU_2}|J(0) = j]\boldsymbol{E}[e^{-sT_1}] = \psi_j(s)\eta_1(s).$$

If the server is busy processing a packet from buffer j for $j \in \{1, ..., k\}$, and buffer 1 is empty, then we have

$$\boldsymbol{E}[e^{-s(U_2+T_1)}|B_1(0) = 0, J(0) = j, U_2 = u, B_1(u) = 1, B_2(0) = 0] \;=\; e^{-su}\eta_1(s),$$

and

$$\boldsymbol{E}[e^{-s(U_2+T_1)}|B_1(0)=0, J(0)=j, U_2=u, B_1(u)=0, B_2(0)=0] \;=\; e^{-su}.$$

By unconditioning on $B_1(u)$ we have

$$\boldsymbol{E}[e^{-s(U_2+T_1)}|B_1(0)=0, J(0)=j, U_2=u, B_2(0)=0] \;=\; e^{-su}\eta_1(s)(1-e^{-\lambda_1 u})+e^{-su}e^{-\lambda_1 u}.$$

By unconditioning on $U_2=u$ we have

$$\boldsymbol{E}[e^{-s(U_2+T_1)}|B_1(0)=0, J(0)=j, B_2(0)=0] \;=\; \psi_j(s)\eta_1(s)-\psi_j(s+\lambda_1)\eta_1(s)+\psi_j(s+\lambda_1).$$

So far we have characterized the LST of W_2 conditioning on different scenarios. We only need the probabilities of $\boldsymbol{P}(B_1(0)=\{0,1\}, J(0)=j|B_2(0)=0)$ to obtain $\boldsymbol{E}[e^{-sW_2}]$, which we will discuss at the end of this subsection. Before doing that, we now consider how to obtain LST of W_3 by conditioning on different scenarios. For simplicity of analysis we here assume $\lambda_1=\lambda_2$ and $\mu_1=\mu_2$. The argument for distinct λ_1 and λ_2 or μ_1 and μ_2 are similar, however notationally cumbersome. We let T_{12} be the busy time during which the server continuously serves packets from queue 1 and queue 2 and let $B_{12}(t)=B_1(t)+B_2(t)$. We now characterize the LST of T_{12} by letting $\eta_{12,0}(s)=\boldsymbol{E}[e^{-sT_{12}} \mid B_{12}(0)=0]$ and $\eta_{12,1}(s)=\boldsymbol{E}[e^{-sT_{12}} \mid B_{12}(0)=1]$.

Since the busy time T_{12} always starts with processing either a packet from source 1 or 2, we suppose that the busy period starts with processing a packet with processing time $P_1=l$. We then have

$$\boldsymbol{E}[e^{-s(l+T_{12})}|B_{12}(0)=0, P_1=l, B_{12}(l)=0, B_3(0)=0] \;=\; e^{-sl},$$

$$\boldsymbol{E}[e^{-s(l+T_{12})}|B_{12}(0)=0, P_1=l, B_{12}(l)=1, B_3(0)=0] \;=\; e^{-sl}\eta_{12,0}(s),$$

$$\boldsymbol{E}[e^{-s(l+T_{12})}|B_{12}(0)=0, P_1=l, B_{12}(l)=2, B_3(0)=0] \;=\; e^{-sl}\eta_{12,1}(s),$$

$$\boldsymbol{E}[e^{-s(l+T_{12})}|B_{12}(0)=1, P_1=l, B_{12}(l)=1, B_3(0)=0] \;=\; e^{-sl}\eta_{12,0}(s),$$

$$\boldsymbol{E}[e^{-s(l+T_{12})}|B_{12}(0)=1, P_1=l, B_{12}(l)=2, B_3(0)=0] \;=\; e^{-sl}\eta_{12,1}(s).$$

Note that $B_{12}(0)=2$ has probability 0 since the busy period T_{12} always starts with processing a packet from either buffer 1 or 2. Unconditioning on $B_{12}(l)$, we have

$$\begin{aligned}\boldsymbol{E}[e^{-s(l+T_{12})}|B_{12}(0)=0, P_1=l, B_3(0)=0] \;&=\; e^{-sl}e^{-2\lambda_1 l} + 2(1-e^{-\lambda_1 l})e^{-\lambda_1 l}e^{-sl}\eta_{12,0}(s) \\ &\quad +e^{-sl}(1-e^{-\lambda_1 l})^2\eta_{12,1}(s),\end{aligned}$$

and

$$\boldsymbol{E}[e^{-s(l+T_{12})}|B_{12}(0)=1, P_1=l, B_3(0)=0] \;=\; e^{-sl}e^{-\lambda_1 l}\eta_{12,0}(s) + e^{-sl}(1-e^{-\lambda_1 l})\eta_{12,1}(s).$$

Unconditioning on $P_1 = l$, we have

$$\begin{aligned}\eta_{12,0}(s) &= \psi_1(s+2\lambda_1) + 2[\psi_1(s+\lambda_1) - \psi_1(s+2\lambda_1)]\eta_{12,0}(s) \\ &\quad +[\psi_1(s) - 2\psi_1(s+\lambda_1) + \psi_1(s+2\lambda_1)]\eta_{12,1}(s),\end{aligned}$$

and

$$\eta_{12,1}(s) = \psi_1(s+\lambda_1)\eta_{12,0}(s) + [\psi_1(s) - \psi_1(s+\lambda_1)]\eta_{12,1}(s).$$

By solving the two equations above for $\eta_{12,0}(s)$ and $\eta_{12,1}(s)$, we have

$$\begin{aligned}&\eta_{12,0}(s) \\ = &\frac{\psi_1(s+2\lambda_1)}{1 - 2[\psi_1(s+\lambda_1) - \psi_1(s+2\lambda_1)] - \frac{\psi_1(s+\lambda_1)}{1-\psi_1(s)+\psi_1(s+\lambda_1)}[\psi_1(s) - 2\psi_1(s+\lambda_1) + \psi_1(s+2\lambda_1)]},\end{aligned}$$

and

$$\eta_{12,1}(s) = \frac{\eta_{12,0}(s)\psi_1(s+\lambda_1)}{1-\psi_1(s)+\psi_1(s+\lambda_1)}.$$

Recall that U_3 is the remaining service time observed by a packet that enters buffer 3, we then have the LST of busy period of buffer 3 as conditioned on various scenarios:

$$\boldsymbol{E}[e^{-s(U_3+T_{12})}|B_{12}(0)=0, J(0)=0, B_3(0)=0] = 1,$$

$$\begin{aligned} &\boldsymbol{E}[e^{-s(U_3+T_{12})}|B_{12}(0)=0,J(0)=j,B_3(0)=0] \\ = \;& \psi_j(s+2\lambda_1)+2[\psi_j(s+\lambda_1)-\psi_j(s+2\lambda_1)]\eta_{12,0}(s) \\ &+[\psi_j(s)-2\psi_j(s+\lambda_1)+\psi_j(s+2\lambda_1)]\eta_{12,1}(s), \end{aligned}$$

$$\begin{aligned} \boldsymbol{E}[e^{-s(U_3+T_{12})}|B_{12}(0)=1,J(0)=j,B_3(0)=0] \;=\;& \psi_j(s+\lambda_1)\eta_{12,0}(s) \\ &+[\psi_j(s)-\psi_j(s+\lambda_1)]\eta_{12,1}(s), \end{aligned}$$

and

$$\boldsymbol{E}[e^{-s(U_3+T_{12})}|B_{12}(0)=2,J(0)=j,B_3(0)=0]=\psi_j(s)\eta_{12,1}(s).$$

Thus we can characterize the LST of W_3 once we know the stationary probability of each scenario. For queues with lower priorities, the analysis requires more argument, but they are all similar (albeit cumbersome notationally). To get the stationary probability of each scenario, we model $S(t)=(J(t),B_1(t),B_2(t),...,B_k(t))$ as a CTMC and obtain the stationary probabilities. Here we only show the example for the case of $k=2$, for $k>2$ the analysis is similar. The rate matrix Q of the two-queue case is:

$$Q = \begin{array}{c|ccccccccc} & (0,0,0) & (1,0,0) & (2,0,0) & (1,1,0) & (1,0,1) & (2,1,0) & (2,0,1) & (1,1,1) & (2,1,1) \\ \hline (0,0,0) & -\lambda_1-\lambda_2 & \lambda_1 & \lambda_2 & 0 & 0 & 0 & 0 & 0 & 0 \\ (1,0,0) & \mu_1 & -\lambda_1-\lambda_2-\mu_1 & 0 & \lambda_1 & \lambda_2 & 0 & 0 & 0 & 0 \\ (2,0,0) & \mu_2 & 0 & -\lambda_1-\lambda_2-\mu_2 & 0 & 0 & \lambda_1 & \lambda_2 & 0 & 0 \\ (1,1,0) & 0 & \mu_1 & 0 & -\lambda_2-\mu_1 & 0 & 0 & 0 & \lambda_2 & 0 \\ (1,0,1) & 0 & 0 & \mu_1 & 0 & -\lambda_1-\mu_1 & 0 & 0 & \lambda_1 & 0 \\ (2,1,0) & 0 & \mu_2 & 0 & 0 & 0 & \lambda_2-\mu_2 & 0 & 0 & \lambda_2 \\ (2,0,1) & 0 & 0 & \mu_2 & 0 & 0 & 0 & -\lambda_1-\mu_2 & 0 & \lambda_1 \\ (1,1,1) & 0 & 0 & 0 & 0 & \mu_1 & 0 & 0 & -\mu_1 & 0 \\ (2,1,1) & 0 & 0 & 0 & 0 & \mu_2 & 0 & 0 & 0 & -\mu_2 \end{array}.$$

The stationary distribution $\hat{\pi}$ (which is a vector) is given by solving $\hat{\pi}Q = 0$ and $\hat{\pi}\mathbf{1} = 1$, and we have

$$p_1 = \hat{\pi}(1,1,0) + \hat{\pi}(2,1,0) + \hat{\pi}(1,1,1) + \hat{\pi}(2,1,1),$$

$$p_2 = \hat{\pi}(1,0,1) + \hat{\pi}(2,0,1) + \hat{\pi}(1,1,1) + \hat{\pi}(2,1,1),$$

$$\boldsymbol{P}(J(0) = 1 | B_1(0) = 0) = \frac{\hat{\pi}(1,0,0) + \hat{\pi}(1,0,1)}{1 - p_1},$$

and

$$\boldsymbol{P}(B_1(0) = 0, J(0) = 1 | B_2(0) = 0) = \frac{\hat{\pi}(1,0,0)}{1 - p_2}.$$

The other conditional probabilities can be calculated similarly.

In summary, in order to obtain the exact PAoI for queue i in M/M/1/1+$\sum \mathbf{1}^*$ type queues, one needs to first have the LST of W_i conditioning on each event of $(B_1(0) = \{0,1\}, ..., B_{i-1}(0) = \{0,1\}, B_i(0) = 0, J(0) = \{0,1,...,k\})$, then apply the CTMC analysis to obtain the steady state probability of each event of $(B_1(0) = \{0,1\}, ..., B_{i-1}(0) = \{0,1\}, B_i(0) = 0, J(0) =$

$\{0, 1, ..., k\})$. By further unconditioning on each event one can eventually get the LST of W_i. This approach becomes cumbersome when the number of queues becomes large. However, this modeling method by focusing on the busy period of the server could useful in many cases as we will see in Section 4.4.

4.3.2 Bounds and Approximation for M/G/1/1+$\sum 1^*$ Type Queues

Here we generalize the analysis in Subsection 4.3.1 so that the service times are generally distributed, hence the system is an M/G/1/1+ $\sum 1^*$ system. Without the assumption that the service time is exponentially distributed, the remaining service time observed at an arbitrary time is no longer what is observed by an entering packet, thus the analysis in Subsection 4.3.1 does not hold for M/G/1 type queues. However, since arrivals still follow Poisson processes, Lemma 4.3.1 holds. We can write the PAoI of queue i as

$$\begin{aligned}\boldsymbol{E}[A_i] &= \boldsymbol{E}[P_i] + \boldsymbol{E}[W_i] + \boldsymbol{E}[I_i] + \boldsymbol{E}[G_i] \\ &= \frac{1}{\mu_i} + \frac{p_i}{\lambda_i(1-p_i)} + \frac{2}{\lambda_i} - \frac{1}{\lambda_i}\boldsymbol{E}[e^{-\lambda_i W_i}] \\ &\leq \frac{1}{\mu_i} + \frac{p_i}{\lambda_i(1-p_i)} + \frac{2}{\lambda_i} - \frac{1}{\lambda_i}e^{-\frac{p_i}{1-p_i}}. \end{aligned} \tag{4.4}$$

The last inequality of (4.4) follows from the Jensen's inequality by knowing that $e^{-\lambda_i x}$ is a convex function. Notice that Equation (4.4) gives an upper bound of PAoI in terms of probability p_i (which is the steady state probability that buffer i is full). Takenaka [131] considered a multi-queue M/G/1 system with each queue having a unique buffer size. Our system thus becomes a special case of the model in Takenaka [131] since in our model each queue has a buffer with size one. Takenaka [131] introduces the relationship of p_i with the stationary state that is seen by departures, for the system in which service times for packets from different queues are identically distributed with $F_i(x) = F(x)$ and $\mu_i = \mu$ for all i. Thus one can get the stationary distribution of states by solving an embedded Markov chain. It is important to note that the result in [131] only works for identically distributed service times. For heterogenous service times with $k > 2$, the results are difficult to obtain [132, 131]. So till the end of this subsection, we assume that service times for packets across queues are identically distributed. To use the result in [131] to get p_i's, we first introduce some notations here. Let $\psi(s)$ be the LST of service time. Let $\mathcal{S}_k$ be our original system which has k queues. Say $\mathcal{S}_l$ is the subsystem of $\mathcal{S}_k$ which contains only queue 1 to queue l, and packets from queue $l+1$ to k do not arrive in system $\mathcal{S}_l$. Let $\pi_l(B_1, B_2, ..., B_l)$ be the stationary distribution in which the system $\mathcal{S}_l$ has $B_i \in \{0, 1\}$ number of packets in queue i immediately after the departure of a packet. Now we re-write a theorem from [131] for our model.

Theorem 4.3.2. *(Theorem 3 of [131]) The steady state probability of the buffer with size one at queue i being full is given by* $p_i = 1 - \frac{\pi_{i-1}(0,...0)-\pi_i(0,...,0)}{\frac{\lambda_i}{\mu}+\frac{\lambda_i}{\sum_{j=1}^{k}\lambda_j}\pi_k(0,...,0)} - \frac{\pi_k(0,...,0)}{\frac{\sum_{j=1}^{k}\lambda_j}{\mu}+\pi_k(0,...,0)}$ *for all* $i \in \{1, 2, ..., k\}$, *where* $\pi_0(0, ..., 0) = 1$.

To obtain the probability p_i, we only need to find the stationary distribution that is seen by departures. For that, we model the system state seen by departures as an embedded Markov chain. We only introduce the case for $k = 2$ here. For $k > 2$ the analysis is similar but not presented here for notational and space restrictions. Since the departure can see at most one packet waiting at each buffer, the transition matrix for $k = 2$ is given as follows:

$$\tilde{P}_2 = \begin{array}{c} \\ (0,0) \\ (0,1) \\ (1,0) \\ (1,1) \end{array} \begin{array}{c} \begin{array}{cccc} (0,0) & (0,1) & (1,0) & (1,1) \end{array} \\ \left[\begin{array}{cccc} a_0 & a_1 & a_2 & a_3 \\ a_0 & a_1 & a_2 & a_3 \\ a_0 & a_1 & a_2 & a_3 \\ 0 & b_0 & 0 & 1-b_0 \end{array}\right] \end{array},$$

where $a_0 = \int_0^\infty e^{-(\lambda_1+\lambda_2)x}dF(x)$, $a_1 = \int_0^\infty e^{-\lambda_1 x}(1 - e^{-\lambda_2 x})dF(x)$, $a_2 = \int_0^\infty(1 - e^{-\lambda_1 x})e^{\lambda_2 x}dF(x)$, $a_3 = \int_0^\infty(1-e^{-\lambda_1 x})(1-e^{-\lambda_2 x})dF(x)$ and $b_0 = \int_0^\infty e^{-\lambda_1 x}dF(x)$. The stationary distribution $\pi_2(0, 0)$ can thus be obtained by solving the linear system $\pi_2\tilde{P}_2 = \pi_2$ with $\pi_2\mathbf{1} = 1$, where $\pi_2 = (\pi_2(0, 0), \pi_2(0, 1), \pi_2(1, 0), \pi_2(1, 1))$. Notice from Theorem 4.3.2 that we also need $\pi_1(0)$ to get p_i's. To obtain $\pi_1(0)$ we solve the subsystem $\mathcal{S}_1$ with $\pi_1\tilde{P}_1 = \pi_1$ and $\pi_1\mathbf{1} = 1$, where the transition matrix $\tilde{P}_1$ of the embedded Markov chain is given by

$$\tilde{P}_1 = \begin{array}{c} \\ (0) \\ (1) \end{array} \begin{array}{c} \begin{array}{cc} (0) & (1) \end{array} \\ \begin{bmatrix} b_0 & 1-b_0 \\ b_0 & 1-b_0 \end{bmatrix} \end{array}.$$

By solving the embedded Markov chains, we have $\pi_1(0) = \psi(\lambda_1)$ and $\pi_2(0,0) = \frac{\psi(\lambda_1+\lambda_2)\psi(\lambda_1)}{1-\psi(\lambda_2)+\psi(\lambda_1+\lambda_2)}$. Then using Theorem 4.3.2 we have $p_1 = 1 - \frac{1-\psi(\lambda_1)}{\frac{\lambda_1}{\mu}+\frac{\lambda_1}{\lambda_1+\lambda_2}\pi_2(0,0)} - \frac{\pi_2(0,0)}{\frac{\lambda_1+\lambda_2}{\mu}+\pi_2(0,0)}$ and $p_2 = 1 - \frac{\psi(\lambda_1)-\pi_2(0,0)}{\frac{\lambda_2}{\mu}+\frac{\lambda_2}{\lambda_1+\lambda_2}\pi_2(0,0)} - \frac{\pi_2(0,0)}{\frac{\lambda_1+\lambda_2}{\mu}+\pi_2(0,0)}$. In summary, to obtain the probability p_i of a system with k queues, one needs to compute the stationary probability $\pi_j(0,...,0)$ for $j = 1,...,i-1$ by solving the embedded Markov chain and then apply Theorem 4.3.2. For systems with large k, solving all the embedded Markov chains could be tedious. Fast approximations for p_i's are provided in [133].

Corollary 4.3.3. *The PAoI for a single M/G/1/2* queue is upper bounded by* $\frac{1}{\mu} + \frac{\frac{\lambda_1}{\mu}+\psi(\lambda_1)-1}{\lambda_1} + \frac{2}{\lambda_1} - \frac{1}{\lambda_1}e^{-\frac{\lambda_1}{\mu}-\psi(\lambda_1)+1}$*, where* $\psi(s)$ *is the LST of service time.*

Proof. It follows directly from Theorem 4.3.2 that when $k = 1$, the probability of buffer being full is $1 - \frac{1}{\frac{\lambda_1}{\mu}+\psi(\lambda_1)}$. □

So far we characterized the probability p_i for Equation (4.4), which we can use to obtain the bounds of PAoI for each queue. In fact, the upper bounds that we provide in Equation (4.4) are decent approximations of PAoI for queues. We will show it numerically in Section 4.5.

It is found by Costa et al [105] that for M/M/1/1, M/M/1/2 and M/M/1/2* queues, increasing the arrival rate can reduce the PAoI continuously. However, it is not the case in our model with multiple queues. We find that by increasing the arrival rate of a certain queue, its PAoI will be decreased, however PAoI for queues with lower priorities will be increased drastically. We will show the detail numerically in Section 4.5. Besides, we have the following theorem discussing the scenario when the arrival rate of a certain queue becomes large. We still keep the assumption that the service times are homogeneous with cdf $F(x)$.

Theorem 4.3.4. *For $1 \leq i \leq k$, if $\lambda_i \to \infty$, then $\boldsymbol{E}[A_j] \to \infty$ for $j > i$, and $\boldsymbol{E}[A_j]$ will be bounded for $j \leq i$.*

Proof. We first show that as $\lambda_i \to \infty$, then $\pi_j(0, ..., 0) \to 0$ for $j \geq i$. To show this, we know that in the subsystem $\mathcal{S}_j$, the first element of the transition matrix for the embedded Markov chain is given by

$$a_0 \;=\; \int_0^\infty e^{-(\sum_{l=1}^{j}\lambda_l)x} dF(x) = \int_0^\infty (\sum_{l=1}^{j}\lambda_l) e^{-(\sum_{l=1}^{j}\lambda_l)x} F(x)dx.$$

Since $F(x) \leq 1$ for any $x \in [0, \infty)$, by dominated convergence theorem, we have

$$\lim_{\lambda_i \to \infty} \int_0^\infty e^{-(\sum_{l=1}^{j}\lambda_l)x} dF(x)dx \;=\; 0.$$

Thus $a_0 \to 0$ and by result from [131] that $\pi_j(0, ..., 0) = \sum_{l=1}^{j} a_0 \pi_j(0, ..., \overset{l}{1}, ..., 0) + a_0 \pi_j(0, ..., 0)$, we have $\pi_j(0, ..., 0) \to 0$ for $j \geq i$. From Theorem 4.3.2 we have $p_j = 1 - \frac{\pi_{j-1}(0,...0) - \pi_j(0,...,0)}{\frac{\lambda_j}{\mu} + \frac{\lambda_j}{\sum_{l=1}^{k}\lambda_l}\pi_k(0,...,0)} - \frac{\pi_k(0,...,0)}{\frac{\sum_{l=1}^{k}\lambda_l}{\mu} + \pi_k(0,...,0)} \to 1$ for $j \geq i$. We then have $\boldsymbol{E}[A_j] \to \infty$ for $j > i$.

For $j < i$, we have $p_j \leq 1 - \frac{\pi_{j-1}(0,...0) - \pi_j(0,...,0)}{\frac{\lambda_j}{\mu} + \frac{\lambda_j}{\sum_{l=1}^{k}\lambda_l}}$. From the fact that $\pi_j(0, ..., 0)$ for $j < i$ will not be affected by λ_i and $\pi_j(0, ..., 0) < \pi_{j-1}(0, ..., 0)$ if $\lambda_j \neq 0$ (see Theorem 1 of [131]), we then have $p_j \leq 1 - \frac{\pi_{j-1}(0,...0) - \pi_j(0,...,0)}{\frac{\lambda_j}{\mu}} < 1$ as $\lambda_i \to \infty$. Thus $\boldsymbol{E}[A_j]$ is bounded by Equation (4.4).

For $j = i$, we have

$$\begin{aligned}
\frac{p_i}{\lambda_i(1-p_i)} &= \frac{1 - \frac{\pi_{i-1}(0,...0) - \pi_i(0,...,0)}{\frac{\lambda_i}{\mu} + \frac{\lambda_i}{\sum_{j=1}^{k}\lambda_j}\pi_k(0,...,0)} - \frac{\pi_k(0,...,0)}{\frac{\sum_{j=1}^{k}\lambda_j}{\mu} + \pi_k(0,...,0)}}{\lambda_i\left(\frac{\pi_{i-1}(0,...0) - \pi_i(0,...,0)}{\frac{\lambda_i}{\mu} + \frac{\lambda_i}{\sum_{j=1}^{k}\lambda_j}\pi_k(0,...,0)} + \frac{\pi_k(0,...,0)}{\frac{\sum_{j=1}^{k}\lambda_j}{\mu} + \pi_k(0,...,0)}\right)} \\
&= \frac{1 - \frac{\pi_{i-1}(0,...0) - \pi_i(0,...,0)}{\frac{\lambda_i}{\mu} + \frac{\lambda_i}{\sum_{j=1}^{k}\lambda_j}\pi_k(0,...,0)} - \frac{\pi_k(0,...,0)}{\frac{\sum_{j=1}^{k}\lambda_j}{\mu} + \pi_k(0,...,0)}}{\frac{\pi_{i-1}(0,...0) - \pi_i(0,...,0)}{\frac{1}{\mu} + \frac{1}{\sum_{j=1}^{k}\lambda_j}\pi_k(0,...,0)} - \frac{\lambda_i \pi_k(0,...,0)}{\frac{\sum_{j=1}^{k}\lambda_j}{\mu} + \pi_k(0,...,0)}} \\
&\leq \frac{1}{\frac{\pi_{i-1}(0,...0) - \pi_i(0,...,0)}{\frac{1}{\mu} + \frac{1}{\sum_{j=1}^{k}\lambda_j}\pi_k(0,...,0)} - \frac{\pi_k(0,...,0)}{\frac{\sum_{j=1}^{k}\lambda_j}{\mu\lambda_i} + \frac{\pi_k(0,...,0)}{\lambda_i}}}.
\end{aligned}$$

As $\lambda_i \to \infty$, $\frac{p_i}{\lambda_i(1-p_i)} \leq \frac{1}{\mu\pi_{i-1}(0,\ldots,0)}$. By Equation (4.4) we prove the theorem. $\square$

Theorem 4.3.4 shows that if we increase the arrival rate for a queue, the PAoI of queues with lower priorities will be greatly increased, while PAoI of queues with higher priorities will be bounded. Like we discussed in Section 4.1, some data sources may have information more important or time-sensitive than other data sources. Theorem 4.3.4 implies that if higher priorities are given to data sources which are more important, the PAoI of these data sources with high priorities can always be guaranteed at a bounded level. It also implies that if we have queues with traffic intensity significantly greater than the others, it is better to give high priorities to those queues with low traffic intensities to guarantee that all queues have a relatively low PAoI.

4.3.3 Exact Analysis for M/G/1/1+$\sum \mathbf{1}$ Type Queues

Notice that in system M/G/1/1+$\sum \mathbf{1}^*$, we keep the most recent arrival in the buffer. If we instead, keep the first arrival in the buffer and reject the future arrival before the buffer becomes empty, the system becomes M/G/1/1+$\sum$1. In the single queue case, the system is denoted as M/G/1/2 which was analyzed in [105]. From Equation (4.2), the PAoI of source i in M/G/1/1+$\sum \mathbf{1}$ is given by

$$\boldsymbol{E}[A_i] \quad = \quad \boldsymbol{E}[P_i] + \mathbf{2}\boldsymbol{E}[W_i] + \boldsymbol{E}[I_i]. \tag{4.5}$$

From Subsection 4.3.2, we know the method of calculating the probability p_i. From the fact that $\boldsymbol{E}[W_i] = \frac{p_i}{\lambda_i(1-p_i)}$ from Equation (4.3) we find that the exact PAoI of queues in M/G/1/1+$\sum \mathbf{1}$ system can also be obtained. However, the following corollary states that the PAoI of each source in M/G/1/1+$\sum \mathbf{1}$ system is always larger than or equal to that of an M/G/1/1+$\sum \mathbf{1}^*$ system.

Corollary 4.3.5. *PAoI of each queue in M/G/1/1+$\sum \mathbf{1}$ is always greater than or equal to that of an M/G/1/1+$\sum \mathbf{1}^*$ system, if both systems have the same parameters.*

Proof. From Equation (4.2) and (4.5) we only need to show that $\boldsymbol{E}[G_i] \leq \boldsymbol{E}[W_i]$. Since $\boldsymbol{E}[G_i] = \frac{1}{\lambda_i}(1-\boldsymbol{E}[e^{-\lambda_i W_i}])$ and from the fact that $e^{-\lambda_i W_i} \geq 1-\lambda_i W_i$, we have $\boldsymbol{E}[G] \leq \frac{1}{\lambda_i}(1-(1-\lambda_i W_i)) =$

$\boldsymbol{E}[W_i]$. Hence proved. $\square$

Corollary 4.3.5 reveals the fact that if service times are iid for each source, then for the system with buffer size one at each queue, it is always beneficial to keep the most recent arrival in the buffer for reducing PAoI.

4.4 Infinite Buffer Size

Although dropping redundant packets such as in system M/G/1+$\sum 1^*$ can potentially reduce the system traffic, it is not clear if keeping all the packets can result in a smaller PAoI. More importantly, for some applications, dropping packets is not an option when the entire data stream must been obtained for performing offline diagnostics (also see [113]). In such a scenario, processing all the generated packets is necessary and for that, buffer size of each queue needs to be large enough. In this section we discuss a model in which buffer size of each queue is infinite. This model has been discussed in [100, 125], however they do not consider queues with priorities. In this section, since there could be multiple packets waiting in each queue, it is necessary to ascertain the order of service within a queue. We consider FCFS and LCFS service discipline separately when the server serves packets from the same queue. Still, the server starts serving packets from high priority queues when the server becomes available. Throughout this section, we assume that $\sum_{j=1}^{k} \frac{\lambda_j}{\mu_j} < 1$ so that the system is stable.

4.4.1 Exact Analysis for M/G/1 Type Queues with FCFS

We first discuss the model in which each queue is served according to FCFS discipline. From the definition of PAoI we know that when processing is complete for the j^{th} arrival from queue i, the random variable corresponding to PAoI is equal to $A_{ij} = C_{ij} - r_{i(j-1)} = (C_{ij} - r_{ij}) + (r_{ij} - r_{i(j-1)})$. Since $C_{ij} - r_{ij}$ is the sojourn time of packet j and $r_{ij} - r_{i(j-1)}$ is the inter-arrival time between packet $j-1$ and j, the PAoI for queue i can be written as $\boldsymbol{E}[A_i] = \boldsymbol{E}[P_i] + \boldsymbol{E}[W_i] + \boldsymbol{E}[I_i]$, where $\boldsymbol{E}[W_i]$ is the expected waiting time in queue and $\boldsymbol{E}[I_i] = \frac{1}{\lambda_i}$ is the expected inter-arrival time. From [85] we have the exact expression of $\boldsymbol{E}[W_i]$ for M/G/1 type queues with priority, thus the PAoI of queue i is given by:

$$\begin{aligned}\boldsymbol{E}[A_i] &= \boldsymbol{E}[W_i] + \boldsymbol{E}[I_i] + \boldsymbol{E}[P_i] \\ &= \frac{\frac{1}{2}\sum_{j=1}^{k} \lambda_j \boldsymbol{E}[P_j^2]}{(1-\sum_{j=1}^{i}\frac{\lambda_j}{\mu_j})(1-\sum_{j=1}^{i-1}\frac{\lambda_j}{\mu_j})} + \frac{1}{\lambda_i} + \frac{1}{\mu_i}. \end{aligned} \tag{4.6}$$

Interestingly, from the expression of $\boldsymbol{E}[W_i]$, we find that the packets from higher priority queues always have shorter expected waiting times in queue compared with those from low priority queues. However, Equation (4.6) shows that higher priority queues do not always have shorter PAoI because $\frac{1}{\lambda_i}$ and $\frac{1}{\mu_i}$ also contribute to PAoI. Another interesting point from Equation (4.6) is that by increasing arrival rate λ_i we can reduce the PAoI for queue i but greatly enlarge the PAoI for queues with priority lower than i. We will also show this result numerically in Section 4.5.

Bedewy et al [134] considered the scheduling policy to minimize the average PAoI across queues, i.e., $\frac{1}{k}\sum_{i=1}^{k}\boldsymbol{E}[A_i]$. If we also consider the same objective and ask the design question of how to minimize the average PAoI across queues by assigning queue priorities, the answer is assigning high priorities to queues with low $\rho_i = \frac{\lambda_i}{\mu_i}$, as we see in Theorem 4.4.1.

Theorem 4.4.1. *If the queue priorities satisfy $\rho_1 \leq \rho_2 \leq ... \leq \rho_k$, then the average PAoI across queues given by this priority order is the smallest among all the priority orders.*

Proof. Since

$$\frac{1}{k}\sum_{i=1}^{k}\boldsymbol{E}[A_i] = \frac{1}{k}\sum_{i=1}^{k}\left[\frac{\frac{1}{2}\sum_{j=1}^{k}\lambda_j \boldsymbol{E}[P_j^2]}{(1-\sum_{j=1}^{i}\rho_j)(1-\sum_{j=1}^{i-1}\rho_j)} + \frac{1}{\lambda_i} + \frac{1}{\mu_i}\right], \tag{4.7}$$

changing priority orders only affects the denominator of the first term in Equation (4.7). So minimizing the average PAoI across queues is equivalent to minimizing $\sum_{i=1}^{k}\frac{1}{(1-\sum_{j=1}^{i}\rho_j)(1-\sum_{j=1}^{i-1}\rho_j)}$. If $(\rho_1, \rho_2, ..., \rho_k)$ is the optimal priority order with $\rho_i \geq \rho_{i+m}$, by switching the order of ρ_i and ρ_{i+m} we have a new priority order $(\rho_1^*, \rho_2^*, ..., \rho_k^*)$ with $\rho_i^* = \rho_{i+m}$, $\rho_{i+m}^* = \rho_i$ and $\rho_j^* = \rho_j$ for $j \in \{1, ..., k\}\backslash\{i, i+m\}$. Then we have $\sum_{l=1}^{j}\rho_l = \sum_{l=1}^{j}\rho_l^*$ for $j < i$, $\sum_{l=1}^{j}\rho_l \geq \sum_{l=1}^{j}\rho_l^*$ for

$i \leq j < i+m$ and $\sum_{l=1}^{j} \rho_l = \sum_{l=1}^{j} \rho_l^*$ for $j \geq i+m$. Thus we have

$$\begin{aligned} & \sum_{l=1}^{k} \frac{1}{(1-\sum_{j=1}^{l}\rho_j)(1-\sum_{j=1}^{l-1}\rho_j)} - \sum_{l=1}^{k} \frac{1}{(1-\sum_{j=1}^{l}\rho_j^*)(1-\sum_{j=1}^{l-1}\rho_j^*)} \\ = & \sum_{l=i}^{i+m} \left[\frac{1}{(1-\sum_{j=1}^{l}\rho_j)(1-\sum_{j=1}^{l-1}\rho_j)} - \frac{1}{(1-\sum_{j=1}^{l}\rho_j^*)(1-\sum_{j=1}^{l-1}\rho_j^*)} \right] \\ \geq & \ 0, \end{aligned}$$

which contradicts to the assumption that $(\rho_1, \rho_2, ..., \rho_k)$ is the optimal priority order. Therefore we prove the theorem. $\square$

From Theorem 4.4.1 we see that for M/G/1 type queues with FCFS discipline, it is always better to give higher queue priorities (if we have the option) to queues with smaller traffic intensities when the objective is to minimize the average PAoI across all queues. In fact, this observation is also true for M/G/1/1+$\sum 1^*$ queues that we discussed in Section 4.3. The intuitive reason for this is if we do the opposite, i.e., allowing high traffic queues to have high priority, the server would be busy serving high traffic intensity queues and barely have chance to serve low priority queues. Packets from low priority queues therefore would suffer a large waiting time. We will show this numerically in Section 4.5.

4.4.2 Exact Analysis for M/G/1 Type Queues with LCFS

In this subsection, we derive the PAoI for priority queues with LCFS within each queue. The server still chooses the highest priority queue when it becomes available, and from each queue it serves the last arrived packet first. There is no preemption during service. To derive the exact expression of PAoI, we use the method in Section 4.3 and focus on buffer state. Different from the model discussed in Section 4.3, here in each queue the buffer size is infinite. We now introduce a new service scheme here which has the same PAoI as LCFS. We first divide each queue into two virtual parts: initial buffer and main queue. The initial buffer can hold only one packet. Whenever a new arrival occurs, we send this new arrival into the initial buffer if it is empty. If there is a packet waiting in the initial buffer when a new arrival occurs, we replace it with the newly arrived packet and transfer the old one to the main queue. When the server starts serving a queue, it serves the packet from initial buffer first if it is not empty, then serves packets from main queue in an arbitrary order with the understanding that service times are iid. However, if an arrival occurs when the server is busy, this arrival goes to the initial buffer and waits for the next available service. A demonstrative graph of the idea of initial buffer and main queue is shown in Figure 4.4. As we see from Figure 4.5, the initial buffer is to hold the most recent arrivals. For queue 1, the initial buffer is empty since its most recent arrival has been processed. The initial buffer of queue 2 is full. When the server switches to queue 2, the packet in initial buffer 2 will be processed first.

This service scheme has the same PAoI as LCFS since the staled packets would not incur age peaks and the most recent arrival is always stored in the initial buffer. The benefit of modeling the LCFS system in this way is that we can characterize the PAoI of each queue by focusing on the initial buffer. The state of the initial buffer is either 0 or 1, and each period length of state 0 (when the buffer is empty), is equal to the inter-arrival time I_i between packets. We abuse our notation by letting the time period of state 1 (when the initial buffer is full) be W_i, which we call the busy period of the initial buffer. Using the analysis in Section 4.3.1, the PAoI for queue i is given as $\boldsymbol{E}[A_i] = \boldsymbol{E}[P_i] + \boldsymbol{E}[W_i] + \boldsymbol{E}[I_i] + \boldsymbol{E}[G_i]$, where $\boldsymbol{E}[P_i] = \frac{1}{\mu_i}$ is the expected service time, $\boldsymbol{E}[W_i]$ is the expected length of period when initial buffer is full, $\boldsymbol{E}[I_i] = \frac{1}{\lambda_i}$ is the expected inter-arrival time, and $\boldsymbol{E}[G_i] = \boldsymbol{E}[\frac{1}{\lambda_i}(1 - e^{-\lambda_i W_i})]$ is the expected waiting time of the most recently arrived packet before the buffer becomes empty, which is given in Lemma 4.3.1.

Similar to what we did in Section 4.3.2, since in the system of M/G/1 type queues with LCFS, there is one initial buffer in each queue, we can thus give the PAoI for queue i as:

$$\begin{aligned} \boldsymbol{E}[A_i] &= \boldsymbol{E}[P_i] + \boldsymbol{E}[W_i] + \boldsymbol{E}[I_i] + \boldsymbol{E}[G_i] \\ &= \frac{1}{\mu_i} + \frac{p_i}{\lambda_i(1-p_i)} + \frac{2}{\lambda_i} - \frac{1}{\lambda_i}\boldsymbol{E}[e^{-\lambda_i W_i}], \end{aligned} \tag{4.8}$$

for all $i \in \{1, ..., k\}$, where we abuse our notation here by letting p_i be the steady state prob-

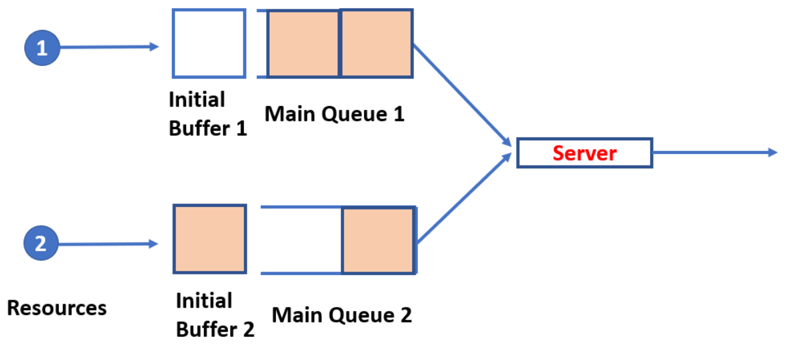


Figure 4.4: Initial Buffer and Main Queue for LCFS

ability that the initial buffer is full (notice that in Section 4.3 we used it as the probability that the buffer is full). Now we introduce the method of finding p_i's by providing Lemma 4.4.2 for the case of $k = 1$ first.

Lemma 4.4.2. *For the M/G/1 queue with LCFS of $k = 1$, the probability that the initial buffer is full is given by $p_1 = \frac{\lambda_1}{\mu_1} - 1 + \psi_1(\lambda_1)$, where $\psi_1(u)$ is the LST of the service time.*

Proof. From Figure 4.5 we find that the busy period of the initial buffer always occurs when the server is serving (busy), and ends when the service is complete. From Figure 4.5 we see that the period during which the initial buffer being full, i.e., $\hat{W}$, is the waiting time of the first packet that arrives during the processing time P_1 of a certain packet. From the property of Poisson arrivals and Campbell's Theorem we have

$$\begin{aligned}\boldsymbol{E}[\hat{W}|P_1 = u] &= \sum_{m=1}^{\infty} \frac{m}{m+1} u e^{-\lambda_1 u} \frac{(\lambda_1 u)^m}{m!} \\ &= u - \frac{1}{\lambda_1} + \frac{1}{\lambda_1} e^{-\lambda_1 u}.\end{aligned}$$

By unconditioning on $P_1 = u$ we have $\boldsymbol{E}[\hat{W}] = \int_0^\infty (u - \frac{1}{\lambda_1} + \frac{1}{\lambda_1} e^{-\lambda_1 u}) dF_1(u) = \frac{1}{\mu_1} - \frac{1}{\lambda_1} + \frac{\psi_1(\lambda_1)}{\lambda_1}$. However, it is important to note that this $\boldsymbol{E}[\hat{W}]$ is the expected busy time for initial buffer during the processing time of a packet. To obtain p_i, we need the following argument. Suppose

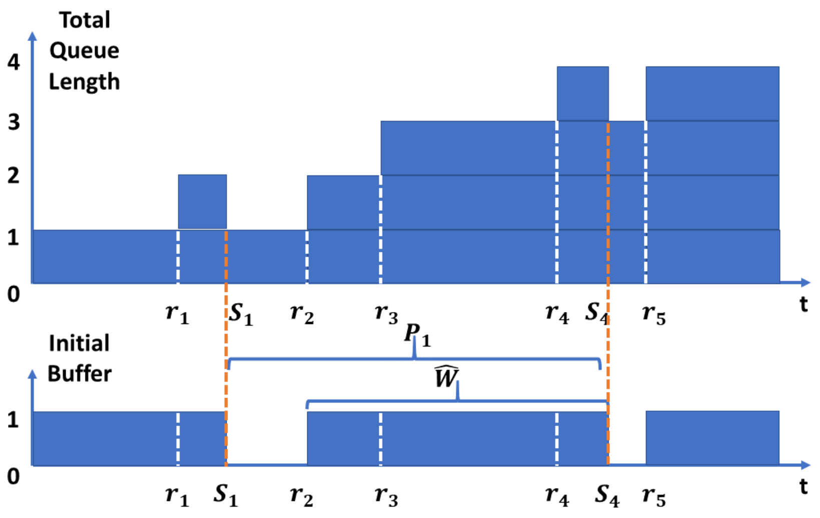


Figure 4.5: Initial Buffer and Total Queue Length (Including Initial Buffer and Main Queue) for LCFS

$n(t)$ packets have been served during $(0, t]$. Thus the amount of time that the initial buffer being full during $(0, t]$ is $n(t)\boldsymbol{E}[\hat{W}]$. If the queue is stable, we have $n(t)$ converging to $\lambda_1 t$ as $t \to \infty$. Therefore

$$\lim_{t\to\infty} \frac{n(t)\boldsymbol{E}[\hat{W}]}{t} = \lambda_1 \boldsymbol{E}[\hat{W}],$$

which is the stationary probability that the initial buffer is full (i.e., p_1). Note that $\frac{\lambda_1}{\mu_1} - 1 + \psi_1(\lambda_1)$ is a legitimate probability as it always lies within $[0, 1]$. To show this, from the fact that $\psi_1(\lambda_1) = \int_0^\infty e^{-\lambda x} dF_1(x) \leq 1$, we have $\frac{\lambda_1}{\mu_1} - 1 + \psi_1(\lambda_1) \leq \frac{\lambda_1}{\mu_1} < 1$ from stability assumption. Since $\psi_1(\lambda_1) = \int_0^\infty e^{-\lambda x} dF_1(x) \geq \int_0^\infty (1 - \lambda_1 x) dF_1(x) = 1 - \frac{\lambda_1}{\mu_1}$, we have $\frac{\lambda_1}{\mu_1} - 1 + \psi_1(\lambda_1) \geq 0$. Thus $\frac{\lambda_1}{\mu_1} - 1 + \psi_1(\lambda_1)$ is a legitimate probability. □

Now we discuss the case when $k \geq 2$. Notice that for each packet that is in service, if there is a new arrival from queue 1 occurring during this service time, then the busy period for initial buffer 1 is from the arrival time of this new packet to the completion time of the packet being processed. From Lemma 4.4.2, the busy period for initial buffer 1 if a type i packet is being processed when the busy period starts, is given by $\frac{1}{\mu_i}-\frac{1}{\lambda_1}+\frac{1}{\lambda_1}\psi_i(\lambda_1)$, and we have $p_1 = \sum_{i=1}^{k}\lambda_i(\frac{1}{\mu_i}-\frac{1}{\lambda_1}+\frac{1}{\lambda_1}\psi_i(\lambda_1))$.

To get the probability p_i for queue $i \geq 2$, we use the idea introduced by Kella and Yechiali [135]. We merge the queues with priority higher than i as one class and the other queues as another class by letting $\lambda_{ai} = \sum_{j=1}^{i-1}\lambda_j$, $\lambda_{bi} = \sum_{j=i}^{k}\lambda_j$, $\rho_{ai} = \sum_{j=1}^{i-1}\rho_j$ and $\rho_{bi} = \sum_{j=i}^{k}\rho_j$. We also let $F_{ai}(x) = \sum_{j=1}^{i-1}\frac{\lambda_j}{\lambda_{ai}}F_j(x)$ be the service time distribution for packets from queue $j < i$, with mean $\boldsymbol{E}[P_{ai}]$ and $F_{bi}(x) = \sum_{j=i}^{k}\frac{\lambda_j}{\lambda_{bi}}F_j(x)$ be the service time distribution for packets from queue $j \geq i$, with mean $\boldsymbol{E}[P_{bi}]$. Notice that the busy period of initial buffer i only ends when there is no packet from queue $j < i$. We now classify the busy periods of server (the time period during which the server is continuously serving packets) into two types, and by doing so we can further characterize the busy period of initial buffer i. One type of busy period V_{ai} of the server starts with processing a packet with priority higher than i, and ends when there is no packet of priority higher than i left in the system. The other type of busy period V_{bi} start with processing a packet with priority equal to or lower than i, and also ends when there is no packet of priority higher than i left in the system. Notice that if there is no higher priority packet arriving during processing the first packet in V_{bi}, the length of V_{bi} is the processing time of a packet of priority i or lower. If there is one packet of higher priority arriving during processing the first packet in V_{bi}, then after the current processing, a busy period V_{ai} is followed. Similar to the analysis in [135, 130] and what we did in Section 4.3.1, by conditioning on service time of the first packet in a busy period, the LST of V_{ai} and V_{bi}, denoted as $\tilde{V}_{ai}(s)$ and $\tilde{V}_{bi}(s)$, are given as

$$\tilde{V}_{ai}(s) \quad = \quad \psi_{ai}(s+\lambda_{ai}-\lambda_{ai}\tilde{V}_{ai}(s)) \tag{4.9}$$

and

$$\tilde{V}_{bi}(s) \quad = \quad \psi_{bi}(s+\lambda_{ai}-\lambda_{ai}\tilde{V}_{ai}(s)), \tag{4.10}$$

where $\psi_{ai}(s)$ is the LST of $F_{ai}(x)$ and $\psi_{bi}(s)$ is the LST of $F_{bi}(x)$. By taking the derivative of

$\tilde{V}_{ai}(s)$ and $\tilde{V}_{bi}(s)$ at $s = 0$, the expected length of server's busy periods can be given as

$$\boldsymbol{E}[V_{ai}] = \frac{\boldsymbol{E}[P_{ai}]}{1-\rho_{ai}},$$

and

$$\boldsymbol{E}[V_{bi}] = \frac{\boldsymbol{E}[P_{bi}]}{1-\rho_{ai}}.$$

Note that busy period V_{bi} always starts with one packet from queue $j \geq i$, thus we know

$$\boldsymbol{P}(system\ in\ V_{bi}) = \lambda_{bi}\frac{\boldsymbol{E}[P_{bi}]}{1-\rho_{ai}}.$$

Since when the server is busy, it is either in busy period V_{ai} or V_{bi} , we have

$$\boldsymbol{P}(system\ in\ V_{ai}) = \sum_{j=1}^{k}\rho_j - \lambda_{bi}\frac{\boldsymbol{E}[P_{bi}]}{1-\rho_{ai}} = \hat{\lambda}_{ai}\frac{\boldsymbol{E}[P_{ai}]}{1-\rho_{ai}},$$

where

$$\hat{\lambda}_{ai} = \frac{\sum_{j=1}^{k}\rho_j - \lambda_{bi}\frac{\boldsymbol{E}[P_{bi}]}{1-\rho_{ai}}}{\frac{\boldsymbol{E}[P_{ai}]}{1-\rho_{ai}}} \tag{4.11}$$

is the "arrival rate" of busy period V_{ai}. We now use $F_{ai}^{V}(x)$ and $F_{bi}^{V}(x)$ to denote the CDF of V_{ai} and V_{bi}. From Lemma 4.4.2 we know that during busy period V_{ai}, the time period of initial buffer being busy is given as

$$\begin{aligned}\boldsymbol{E}[\hat{W}_{ai}] &= \int_0^\infty (u - \frac{1}{\lambda_i} + \frac{1}{\lambda_i}e^{-\lambda_i u})dF_{ai}^{V}(u) \\ &= \frac{\boldsymbol{E}[P_{ai}]}{1-\rho_{ai}} - \frac{1}{\lambda_i} + \frac{1}{\lambda_i}\tilde{V}_{ai}(\lambda_i). \end{aligned} \tag{4.12}$$

Similarly, we have $\boldsymbol{E}[\hat{W}_{bi}] = \frac{\boldsymbol{E}[P_{bi}]}{1-\rho_{ai}} - \frac{1}{\lambda_i} + \frac{1}{\lambda_i}\tilde{V}_{bi}(\lambda_i)$. In many cases where $\tilde{V}_{ai}(\lambda_i)$ and $\tilde{V}_{bi}(\lambda)$

cannot be solved analytically, numerical methods such as bisection method or Newton's method (see [136]) can be applied to find the root numerically. From the same argument in Lemma 4.4.2, we have

$$p_i = \hat{\lambda}_{ai}\boldsymbol{E}[\hat{W}_{ai}] + \lambda_{bi}\boldsymbol{E}[\hat{W}_{bi}]. \tag{4.13}$$

Next we introduce the process of obtaining $\boldsymbol{E}[e^{-\lambda_i W_i}]$ for Equation (4.8). Notice that $\hat{W}_{ai}$ is the length of initial buffer being full during period V_{ai}. Similar to the argument in Lemma 4.4.2, we have

$$\begin{aligned}
\boldsymbol{E}[e^{-s\hat{W}_{ai}}|V_{ai} = t] &= \int_0^t e^{-sx}\sum_{m=1}^{\infty}\frac{mx^{m-1}}{t^m}e^{-\lambda_i t}\frac{(\lambda_i t)^m}{m!}dx + e^{-\lambda_i t} \\
&= \int_0^t e^{-sx}e^{-\lambda_i t}\sum_{m=1}^{\infty}x^{m-1}\frac{(\lambda_i)^m}{(m-1)!}dx + e^{-\lambda_i t} \\
&= e^{-\lambda_i t}\frac{\lambda_i}{\lambda_i - s}(e^{(\lambda_i - s)t} - 1) + e^{-\lambda_i t} \\
&= \frac{\lambda_i}{\lambda_i - s}e^{-st} - \frac{s}{\lambda_i - s}e^{-\lambda_i t}.
\end{aligned}$$

By unconditioning on $V_{ai} = t$ we have

$$\boldsymbol{E}[e^{-s\hat{W}_{ai}}] = \frac{\lambda_i}{\lambda_i - s}\tilde{V}_{ai}(s) - \frac{s}{\lambda_i - s}\tilde{V}_{ai}(\lambda_i),$$

and

$$\boldsymbol{E}[e^{-s\hat{W}_{bi}}] = \frac{\lambda_i}{\lambda_i - s}\tilde{V}_{bi}(s) - \frac{s}{\lambda_i - s}\tilde{V}_{bi}(\lambda_i).$$

Using L'Hospital rule taking the limit $s \to \lambda_i$, we have

$$\boldsymbol{E}[e^{-\lambda_i\hat{W}_{ai}}] = -\lambda_i\tilde{V}'_{ai}(\lambda_i) + \tilde{V}_{ai}(\lambda_i), \tag{4.14}$$

and

$$\boldsymbol{E}[e^{-\lambda_i \hat{W}_{bi}}] \;=\; -\lambda_i \tilde{V}_{bi}^{'}(\lambda_i) + \tilde{V}_{bi}(\lambda_i). \tag{4.15}$$

From the formula of $\tilde{V}_{ai}(s)$ and $\tilde{V}_{bi}(s)$ given above we have

$$\tilde{V}_{ai}^{'}(\lambda_i) \;=\; \frac{\psi_{ai}^{'}(\lambda_i + \lambda_{ai} - \lambda_{ai}\tilde{V}_{ai}(\lambda_i))}{1 + \lambda_{ai}\psi_{ai}^{'}(\lambda_i + \lambda_{ai} - \lambda_{ai}\tilde{V}_{ai}(\lambda_i))}, \tag{4.16}$$

and

$$\tilde{V}_{bi}^{'}(\lambda_i) \;=\; \psi_{bi}^{'}(\lambda_i + \lambda_{ai} - \lambda_{ai}\tilde{V}_{ai}(\lambda_i))(1 - \lambda_{ai}\tilde{V}_{ai}^{'}(\lambda_i)). \tag{4.17}$$

Notice that W_i is the busy period of the initial buffer during each age peak A_i, and only $(1-p_i)$ portion of arrivals in queue i incur peak ages, so the "arrival rate" for W_i is $\lambda_i(1-p_i)$. Since $\hat{W}_i$ is the busy period of initial buffer during each V_{ai} and V_{bi} with arrival rate $\hat{\lambda}_{ai}$ and λ_{bi} respectively, from the fact that $\lambda_{bi} = \sum_{j=i}^{k} \lambda_j \geq \lambda_i(1-p_i)$, we have the following relationship

$$\hat{\lambda}_{ai}\boldsymbol{E}[e^{-\lambda_i \hat{W}_{ai}}] + \lambda_{bi}\boldsymbol{E}[e^{-\lambda_i \hat{W}_{bi}}] \;=\; \lambda_i(1-p_i)\boldsymbol{E}[e^{-\lambda_i W_i}] + (\hat{\lambda}_{ai} + \lambda_{bi} - \lambda_i(1-p_i)).$$

Therefore,

$$\boldsymbol{E}[e^{-\lambda_i W_i}] \;=\; \frac{\hat{\lambda}_{ai}\boldsymbol{E}[e^{-\lambda_i \hat{W}_{ai}}] + \lambda_{bi}\boldsymbol{E}[e^{-\lambda_i \hat{W}_{bi}}] - (\hat{\lambda}_{ai} + \lambda_{bi} - \lambda_i(1-p_i))}{\lambda_i(1-p_i)}. \tag{4.18}$$

A closed-form formula of PAoI in M/G/1 type queues with LCFS is then given in the following theorem.

Theorem 4.4.3. *The PAoI of queue i in M/G/1 system with LCFS is given by* $\boldsymbol{E}[A_i] = \frac{1}{\mu_i} + \frac{1}{\lambda_i(1-p_i)} - \frac{1}{\lambda_i^2(1-p_i)}\Big[\hat{\lambda}_{ai}(-\lambda_i\tilde{V}_{ai}^{'}(\lambda_i) + \tilde{V}_{ai}(\lambda_i) - 1) + \lambda_{bi}(-\lambda_i\tilde{V}_{bi}^{'}(\lambda_i) + \tilde{V}_{bi}(\lambda_i) - 1)\Big]$, *where* $\tilde{V}_{ai}(s) = \psi_{ai}(s + \lambda_{ai} - \lambda_{ai}\tilde{V}_{ai}(s))$, $\tilde{V}_{bi}(s) = \psi_{bi}(s + \lambda_{ai} - \lambda_{ai}\tilde{V}_{ai}(s))$, $p_i =$

$\hat{\lambda}_{ai}\left[\frac{\boldsymbol{E}[P_{ai}]}{1-\rho_{ai}} - \frac{1}{\lambda_i} + \frac{1}{\lambda_i}\tilde{V}_{ai}(\lambda_i)\right] + \lambda_{bi}\left[\frac{\boldsymbol{E}[P_{bi}]}{1-\rho_{ai}} - \frac{1}{\lambda_i} + \frac{1}{\lambda_i}\tilde{V}_{bi}(\lambda_i)\right]$, *and* $\hat{\lambda}_{ai} = \frac{\sum_{j=1}^{k}\rho_j - \lambda_{bi}\frac{\boldsymbol{E}[P_{bi}]}{1-\rho_{ai}}}{\frac{\boldsymbol{E}[P_{ai}]}{1-\rho_{ai}}}$.

Proof. To obtain the exact PAoI for queue i of M/G/1 system with LCFS, we first solve $\tilde{V}_{ai}(\lambda_i)$ and $\tilde{V}_{bi}(\lambda_i)$ using Equation (4.9) and (4.10). And all the required components for computing $\boldsymbol{E}[e^{-\lambda_i W_i}]$ in Equation (4.18) can be then obtained from Equations (4.11) - (4.17). After that the PAoI of queue i can be obtained using Equation (4.8). $\square$

Note here that this approach of calculating PAoI of queue i in M/G/1 type system LCFS can still be applied even when the number of queues k is large. In such a case, queues with priority lower than i can be merged as a single queue and queues with priority equal to or greater than i can also be merged into another queue. Then the analysis provided above can be applied easily to obtain PAoI. The numerical test of this approach will be provided in Section 4.5.

4.4.3 Discussion of the Single Queue Case

The LST of PAoI for single queue with LCFS was provided in [101], however its expression is quite involved (see Equation (99) in [101]). Here we use our approach introduced in Subsection 4.4.2 to provide a concise expression for PAoI of M/G/1/LCFS queue in the following corollaries. Since we only have one queue here, for simplicity of the notation, in this subsection we remove the subscript of each variable.

Corollary 4.4.4. *The PAoI of M/G/1/LCFS is given by* $\boldsymbol{E}[A] = \frac{1}{\mu} + \frac{2}{\lambda} + \frac{2\frac{\lambda}{\mu} - 2 + \psi(\lambda) + \lambda\psi'(\lambda)}{\lambda(2-\frac{\lambda}{\mu}-\psi(\lambda))}$, *where* $\psi(\lambda)$ *is the LST of service time.*

Proof. Since we know $\boldsymbol{E}[e^{-\lambda\hat{W}}] = -\lambda\psi'(\lambda) + \psi(\lambda)$ from Equation (4.15) and $\boldsymbol{E}[e^{-\lambda W}] = \frac{\boldsymbol{E}[e^{-\lambda\hat{W}}]-p}{1-p}$ from Equation (4.18), also from Lemma 4.4.2 we know $p = \frac{\lambda}{\mu} - 1 + \psi(\lambda)$, we have our corollary proved. $\square$

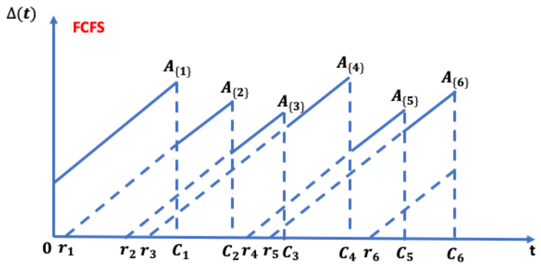
Corollary 4.4.5. *The PAoI of M/M/1/LCFS is given by* $\boldsymbol{E}[A] = \frac{1}{\lambda} + \frac{1}{\mu} + \frac{2\lambda\mu+\lambda^2}{(\lambda+\mu)(\mu^2+\lambda\mu-\lambda^2)}$.

It was shown in [137] that LCFS can reset the generation time of the freshest packet that has arrived (i.e., $\max\{r_l : C_l \leq t\}$) into a lower level than the other non-preemptive service disciplines do. However, here we show that LCFS is actually not the optimal service discipline for minimizing PAoI among all the non-preemptive service disciplines. To do this, we simply consider the exponential service case. Then the PAoI of FCFS is given by $\boldsymbol{E}[A^{FCFS}] = \frac{1}{\lambda} + \frac{1}{\mu} + \frac{\lambda}{\mu(\mu-\lambda)}$. We thus have

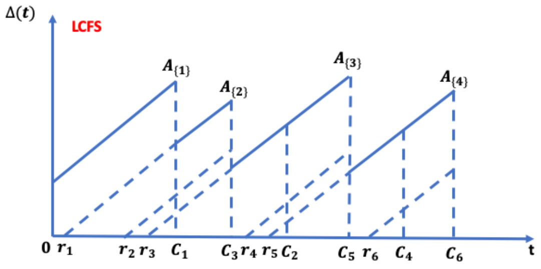
$$\begin{aligned}
& \boldsymbol{E}[A^{FCFS}] - \boldsymbol{E}[A^{LCFS}] \\
= \ & \frac{\lambda}{\mu(\mu-\lambda)} - \frac{2\lambda\mu+\lambda^2}{(\lambda+\mu)(\mu^2+\lambda\mu-\lambda^2)} \\
= \ & \frac{-\lambda^4+\lambda^3\mu+3\lambda^2\mu^2-\lambda\mu^3}{\mu(\mu-\lambda)(\lambda+\mu)(\mu^2+\lambda\mu-\lambda^2)}.
\end{aligned}$$

If we let $\mu = 1$ in the formula above, we have $\boldsymbol{E}[A^{FCFS}] - \boldsymbol{E}[A^{LCFS}] = \frac{-\lambda^4+\lambda^3+3\lambda^2-\lambda}{(1-\lambda)(\lambda+1)(1+\lambda-\lambda^2)}$. By numerically solving it we know that $\boldsymbol{E}[A^{FCFS}] \leq \boldsymbol{E}[A^{LCFS}]$ when $0 \leq \lambda < \lambda^* = 0.3111$, and $\boldsymbol{E}[A^{FCFS}] > \boldsymbol{E}[A^{LCFS}]$ when $1 > \lambda > \lambda^*$, which is also shown in Figure 4.7(a). Similarly, in Section 4.5, we will show that LCFS is not the optimal service discipline for the multi-queue case either, when the objective is to minimize PAoI of each queue.

This result is counter-intuitive considering the optimality of LCFS was correctly proven in [137] where each time when a packet is processed, LCFS would reset the age to the lowest level. The reason why LCFS cannot always have the lowest PAoI is when we calculate PAoI under LCFS, we average on the number of packets that enter the initial buffer (not all the arrivals). And when we calculate PAoI under FCFS, we take average over the number of all arrivals. Demonstrative graphs for this are shown in Figure 4.6. As we see from Figure 4.6, upon the processing of each packet, the age level under LCFS (Figure 4.6(b)) is always smaller than that under FCFS (Figure 4.6(a)). However, the average PAoI in Figure 4.6(a) is smaller than PAoI in Figure 4.6(b), simply because in LCFS the PAoI is only averaged on the number of packets that cause age decrease (instead of the number of all packets that are served). The advantage of FCFS will become more obvious when the traffic intensity is small, since when traffic intensity is large, the waiting time of each packet under FCFS becomes large, which overshadows the effect caused by the number of packets that PAoI is averaged over.



(a) PAoI under FCFS. The first 6 packets result in 6 age peaks



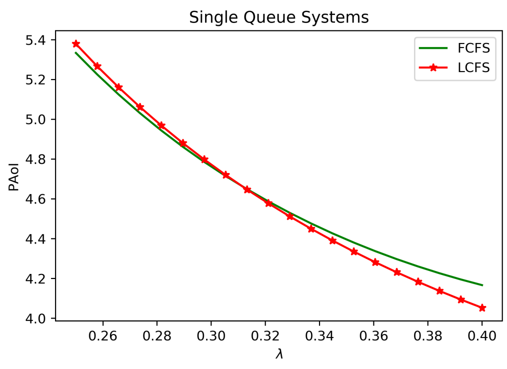
(b) PAoI under LCFS. The first 6 packets result in 4 age peaks

Figure 4.6: PAoI under FCFS and LCFS

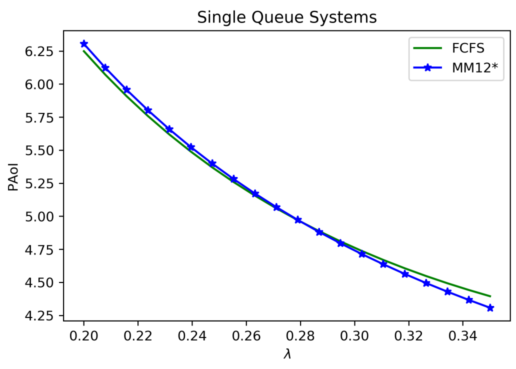
It is shown in [105] that M/M/1/2* system has smaller PAoI than M/M/1/2 and M/M/1/1 systems. Interestingly, we also find that M/M/1/2* system has smaller PAoI than M/M/1/LCFS, since

$$\boldsymbol{E}[A^{M/M/1/LCFS}] - \boldsymbol{E}[A^{M/M/1/2^*}] \quad = \quad \frac{\lambda^3(2\mu+\lambda)}{(\lambda+\mu)^2(\mu^2+\lambda\mu-\lambda^2)\mu} \geq 0.$$

However, FCFS can still sometimes have smaller PAoI than M/M/1/2* system, as shown in Figure 4.7(b). This result indicates that although having buffer with size one can potentially reduce the server's load, it does not always minimize the PAoI. In Section 4.5 we will show that for the multi-queue case, having buffer with size one can sometimes result a larger PAoI than FCFS.



(a) FCFS versus LCFS



(b) FCFS versus $M/M/1/2^*$

Figure 4.7: FCFS for M/M/1 Queue with $\mu = 1$

4.4.4 A Mixed Strategy

Notice that when we introduce the method of obtaining the exact PAoI of M/G/1 type queues with FCFS in Subsection 4.4.1, the PAoI of queue i does not depend on the service sequence of the other queues. That is, if FCFS is only applied in queue i while other work-conserving service disciplines are applied in other queues, we can still use Equation (4.6) to obtain the exact PAoI for queue i. Similarly, in Subsection 4.4.2 we find that the PAoI of queue i does not depend on the service sequence of the other queues when LCFS is applied to queue i. Also, from Subsection 4.4.3 we find that FCFS sometimes has a smaller PAoI than LCFS. It thus motivates us to consider a mixed strategy in which some queues are served following FCFS and the others are served by LCFS. We can still use the methods introduced in Subsection 4.4.1 and 4.4.2 to obtain the exact PAoI of each queue. When system parameters such as k, λ and μ are given, one can first calculate the PAoI of each queue under FCFS and LCFS using the exact analysis that we introduced earlier, and then choose the service discipline with smaller PAoI. We next introduce an example to show how the mixed strategy is constructed.

Example 4.4.6. Suppose we have a system with $k = 8$. The service time of packets from all queues are exponentially distributed with the same parameter $\mu = 1$. The parameters of λ_i and exact PAoI under each service discipline are shown in Table 4.1.

Queue	1	2	3	4	5	6	7	8
λ_i	$\frac{1}{20}$	$\frac{1}{3}$	$\frac{1}{20}$	$\frac{1}{20}$	$\frac{1}{3}$	$\frac{1}{20}$	$\frac{1}{20}$	$\frac{1}{20}$
PAoI FCFS	**22.0175**	**5.6501**	**23.7663**	**24.3017**	14.2053	60.5455	108	369
PAoI LCFS	22.8842	5.8018	25.4927	26.1755	**7.8664**	**44.7605**	**54.2891**	**73.7247**
Service Discipline under Mixed Strategy	FCFS	FCFS	FCFS	FCFS	LCFS	LCFS	LCFS	LCFS
PAoI Mixed Strategy	22.0175	5.6501	23.7663	24.3017	7.8664	44.7605	54.2891	73.7247

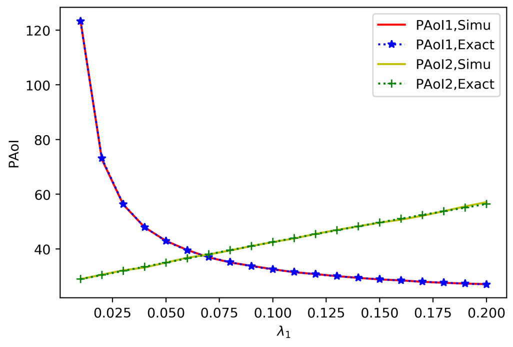
Table 4.1: An Example of the Mixed Strategy

From Table 4.1 we see that the mixed strategy always selects the service discipline with smaller PAoI. The PAoI of each queue under the mixed strategy is thus smaller than simply using FCFS or LCFS over all queues. The PAoI of each queue under the mixed strategy can be obtained using the method that we introduced in Subsection 4.4.1 and 4.4.2.

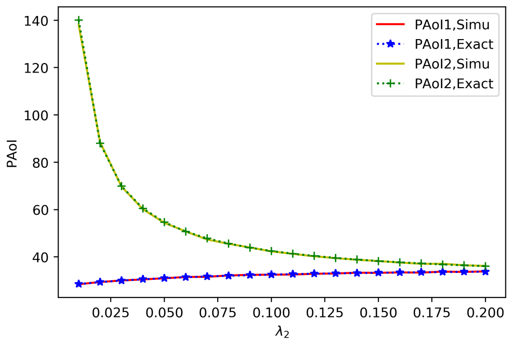
4.5 Numerical Study

In this section we will firstly use a numerical study to verify the exact solutions for M/M/1+$\sum 1^*$ that we provided in Section 4.3.1, and then test the bounds of M/G/1+$\sum 1^*$ which we provided in Section 4.3.2. We will then verify the exact solution for M/G/1 type queues with LCFS. Besides, we will compare the performance of different service disciplines, and develop our insights based on the numerical studies.

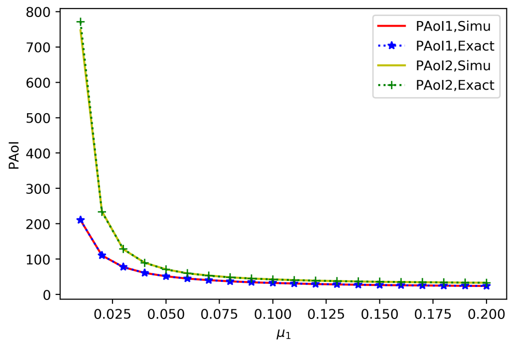
We begin our discussion by comparing simulation results with exact solutions for M/M/1+$\sum 1^*$ system with $k = 2$. The comparison is done by changing one parameter from $\lambda_1,\ \lambda_2,\ \mu_1$ and μ_2 while keeping the others fixed. The results are shown in Figure 4.8. From plots in Figure 4.8 we can see that the simulation results match the exact solutions that we provide in Section 4.3.1, thus verifying our results. Figure 4.8(a) shows that when we increase the arrival rate for the priority 1 queue, its PAoI is drastically decreased, while the PAoI for queue 2 increasing linearly. Figure 4.8(b) shows that if we increase the arrival rate of queue 2, its PAoI will decrease dramatically, while PAoI of queue 1 increases slowly. Figure 4.8(c) and (d) show that when service rate increases, PAoI for both queues are decreased. Interestingly, we find that when queue 1 has a low service rate, PAoI for both queues will be large, while PAoI of queue 1 is not significantly affected by the service rate change of queue 2. It also implies that the average PAoI across all queues, i.e., $\frac{1}{k}\sum_{i=1}^{k}\boldsymbol{E}[A_i]$, is more sensitive to the arrival rate and service rate of high priority queues. We then test how the average PAoI across queues is affected by parameters, which we show in Figure 4.9. From Figure 4.9(a) we see that by increasing the service rate of either queues, the average PAoI across queues will be reduced, and increasing the service rate of queue 1 makes this reduction more significant. Figure 4.9(b) shows that by increasing the arrival rate of queue 2, the average PAoI across queues is decreased. This is because the PAoI for queue 1 is not sensitive to the arrival rate of queue 2, as we also show in Theorem 4.3.4. However, when we increase the arrival rate of queue 1, the average PAoI will decrease drastically at the beginning, and increase afterwards. This is because the PAoI of queue 2 increases constantly when we increase λ_1, which we also see from Figure 4.8(a). Note that although we only discuss the optimization problem of minimizing average PAoI across queues here, since we have the exact solution for PAoI, we could also formulate and solve optimization problems such as minimizing average weighted PAoI (similar to [119]) and minimizing the maximum PAoI (similar to [100]).



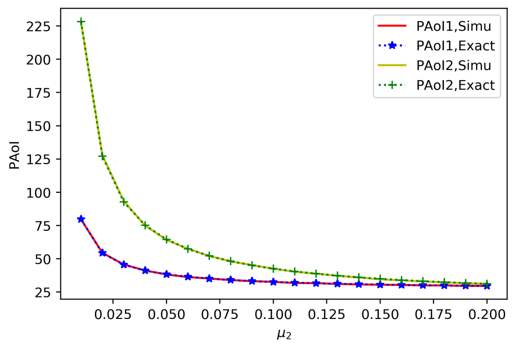
(a) $\mu_1 = \frac{1}{10}, \lambda_2 = \frac{1}{10}, \mu_2 = \frac{1}{10}$



(b) $\lambda_1 = \frac{1}{10}, \mu_1 = \frac{1}{10}, \mu_2 = \frac{1}{10}$

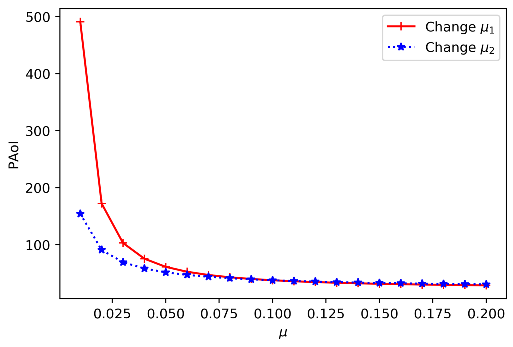


(c) $\lambda_1 = \frac{1}{10}, \lambda_2 = \frac{1}{10}, \mu_2 = \frac{1}{10}$

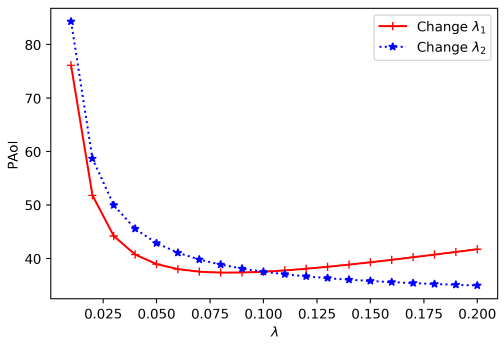


(d) $\lambda_1 = \frac{1}{10}, \mu_1 = \frac{1}{10}, \lambda_2 = \frac{1}{10}$

Figure 4.8: M/M/1+$\sum 1^*$ Type Queues with Buffer Size One

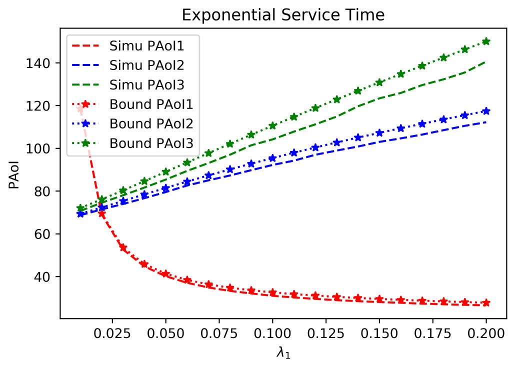


(a) $\lambda_1 = \frac{1}{10}, \lambda_2 = \frac{1}{10}$

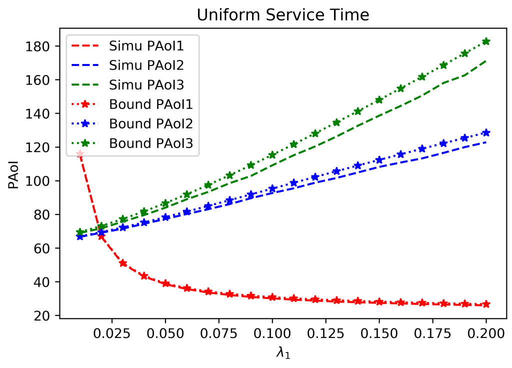


(b) $\mu_1 = \frac{1}{10}, \mu_2 = \frac{1}{10}$

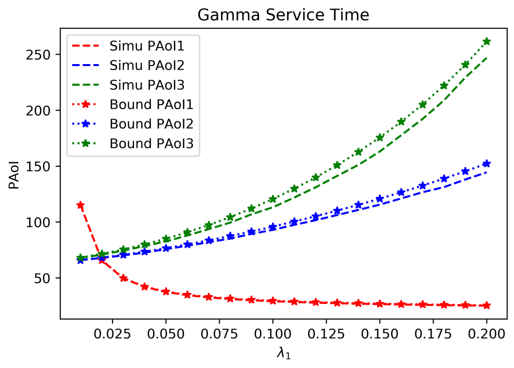
Figure 4.9: Average PAoI of M/M/1+$\sum 1^*$ Queues with Buffer Size One



(a) $\lambda_2 = \lambda_3 = \frac{1}{30}$, $P_1, P_2, P_3 \sim exp(\frac{1}{10})$



(b) $\lambda_2 = \lambda_3 = \frac{1}{30}$, $P_1, P_2, P_3 \sim Unif(0, 20)$



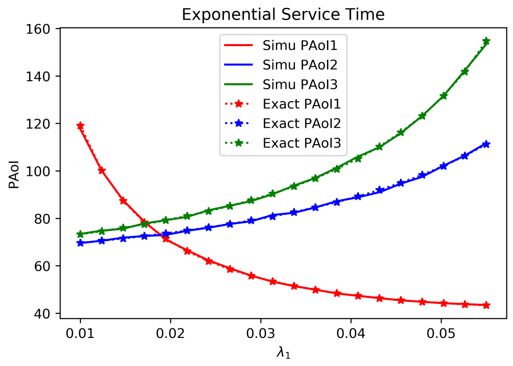
(c) $\lambda_2 = \lambda_3 = \frac{1}{30}$, $P_1, P_2, P_3 \sim Gamma(10, 1)$

Figure 4.10: Bounds for M/G/1+$\sum 1^*$ Type Queues with Buffer Size One

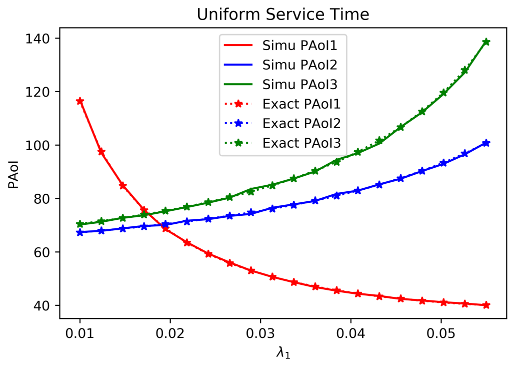
Next we consider queues with general service times. The bounds for M/G/1+$\sum 1^*$ type queues with buffer size one and $k = 3$ are shown in Figure 4.10, where the bounds are provided by Equation (4.4). We test the bounds by letting service time follow exponential, uniform and gamma distributions. Note that in Figure 4.10 we provide the approximations for exponential service too, although we have the exact solution for PAoI when service times are exponential. We find from Figure 4.10 that Equation (4.4) serves as a decent approximation for the actual PAoI since the bounds and simulation curves for all queues are close. The three service distributions in Figure 4.10 have the same mean but the LST of these distributions vary from each other. From our discussion in Section 4.3 we find that the probability p_i is related to the LST of service time. Therefore, different service time distributions result in different probability p_i, and further result in different PAoI.

Then we consider queues with infinite buffer size. The exact PAoI and simulation results for M/G/1 type queues with LCFS are shown in Figure 4.11. We also test the cases for exponential, uniform and gamma distributed service times. In Figure 4.11 we see that the exact PAoI that we provide in Subsection 4.4.2 match the simulation results. We also find that in M/G/1 type queues with LCFS, by increasing the arrival rate of queue 1, PAoI of queue 1 is significantly reduced, and PAoI for lower priority queues is increased at the same time. We do not present the numerical test for M/G/1 queues with FCFS here, as its analysis is exact and also straightforward.

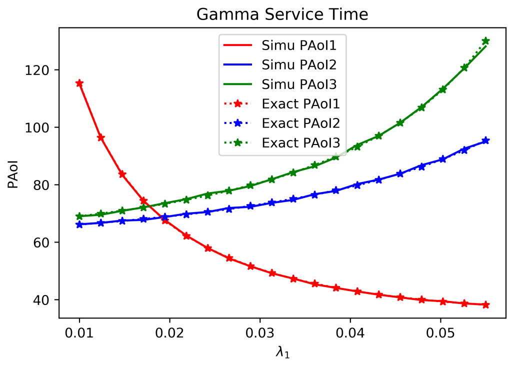
Next we address PAoI by comparing the single buffer size case against infinite buffer size cases under FCFS and LCFS. In fact, since in the M/G/1 system with infinite buffer size, if we keep replacing the packets in buffers with new arrivals, then there is at most one packet waiting in each queue, therefore the system will act exactly the same as M/G/1+$\sum 1^*$ system. So here we consider the PAoI under M/G/1+$\sum 1^*$and M/G/1 with FCFS and LCFS altogether. In Figure 4.12 we plot the PAoI for each queue in the case of $k = 2$, and in Figure 4.13 we plot the average PAoI across all queues ($\frac{1}{k}\sum_{i=1}^{k} \boldsymbol{E}[A_i]$). In both Figure 4.12 and 4.13 we use exact results for exponential service case with FCFS, LCFS and M/M/1+$\sum 1^*$ model with $k = 2$. From Figure 4.12 we see that under FCFS, LCFS and M/M/1+$\sum 1^*$, PAoI of queue 2 is sensitive to the change of λ_1, however



(a) $\lambda_2 = \lambda_3 = \frac{1}{50}$, $P_1, P_2, P_3 \sim exp(\frac{1}{10})$



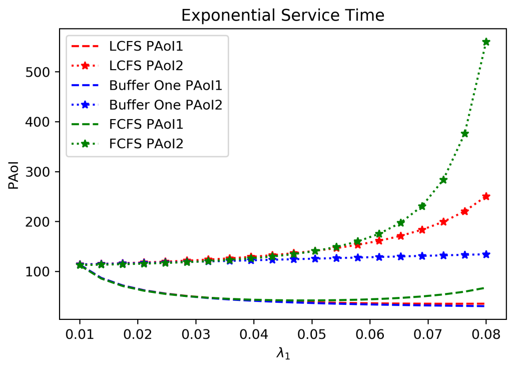
(b) $\lambda_2 = \lambda_3 = \frac{1}{50}$,$P_1, P_2, P_3 \sim Unif(0, 20)$



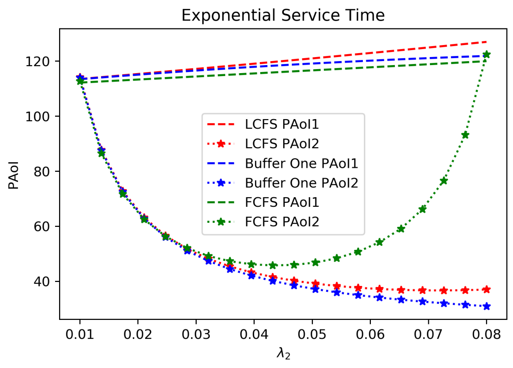
(c) $\lambda_2 = \lambda_3 = \frac{1}{50}$,$P_1, P_2, P_3 \sim Gamma(10, 1)$

Figure 4.11: M/G/1 Type Queues with LCFS

PAoI of queue 1 is less sensitive to λ_2. This is because the PAoI for queue 2 highly depends on the busy time of queue 1. For FCFS, the PAoI increases greatly when arrival rate becomes large. This is because under FCFS, every packet that arrives the system needs to be processed, and increasing arrival rate enlarges the average queue size, causing packets to wait a longer time. From the average PAoI across queues shown in Figure 4.13, we can see that increasing the arrival rate for the high priority queue enlarges the PAoI much faster than increasing λ_2. It indicates that when designing the priority for queues to minimize average PAoI across queues, the one with the lowest traffic intensity should be allocated with the highest priority. We also proved this result in Section 4.4 for M/G/1 queues with FCFS. Note that in Subsection 4.4.4 we introduced a mixed strategy for smaller PAoI. However, the PAoI of each queue under the mixed strategy is the smaller value

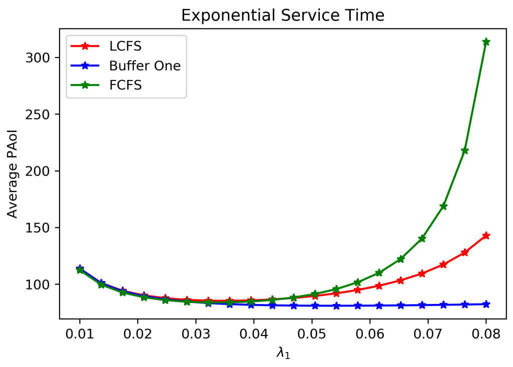


(a) $\lambda_2 = \frac{1}{100}, \mu_1 = \mu_2 = \frac{1}{10}$

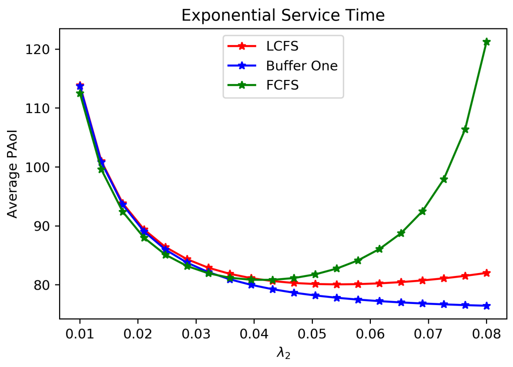


(b) $\lambda_1 = \frac{1}{100}, \mu_1 = \mu_2 = \frac{1}{10}$

Figure 4.12: PAoI under Different Service Disciplines



(a) $\lambda_2 = \frac{1}{100}, \mu_1 = \mu_2 = \frac{1}{10}$



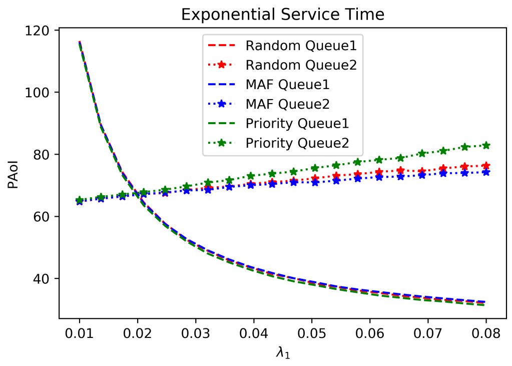
(b) $\lambda_1 = \frac{1}{100}, \mu_1 = \mu_2 = \frac{1}{10}$

Figure 4.13: Average PAoI Across Queues under Different Service Disciplines

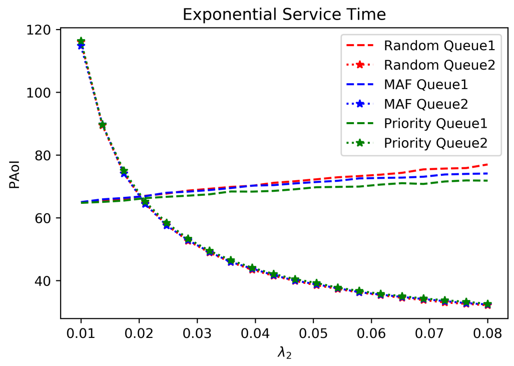
of the PAoI under FCFS and LCFS. For conciseness, here we do not plot the PAoI for the mixed strategy.

Also, it is interesting to observe that having a single-sized buffer at each queue is not always the optimal strategy to minimize PAoI, as we observe from Figure 4.12 and Figure 4.13. This fact can be seen more clearly in Figure 4.12(b) and Figure 4.13(b) when the traffic intensity of queue 2 is higher. In Figure 4.12(b), the PAoI of queue 1 under FCFS is lower than that under the other two policies. In Figure 4.13(b), FCFS results in lower average PAoI than the other two policies when the traffic intensity is low. This result also indicates that LCFS is not the optimal policy among all work-conserving non-preemptive policies when minimizing PAoI. The advantage of FCFS is more obvious when the traffic intensity is small since when traffic intensity becomes large, the drastic increase in waiting time $\boldsymbol{E}[W]$ under FCFS overshadows the effect caused by the number of packets that PAoI is averaged on.

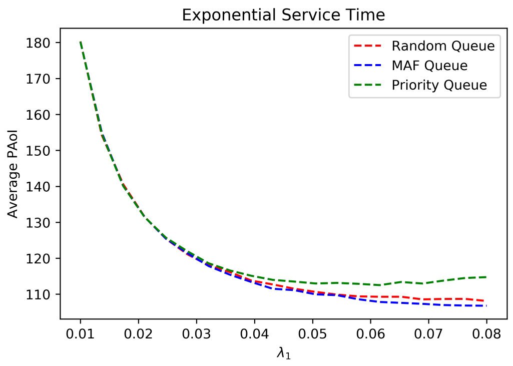
We now compare the performance of priority queue policy (which we introduce in this paper), Maximal Age First (MAF) (see [134]) and random policy (see [115]) in M/M/1/1 $+ \sum 1^*$ system. Under the MAF policy, the server would choose the queue with the largest age as the candidate to serve next. Under the random policy, the server would pick a queue randomly and equally likely from the queues with full buffers. The simulation results for these three policies in a system with $k = 2$ are provided in Figure 4.14. As we can see from figure 4.14(a), when the traffic intensity of the first queue is large, priority queue policy has similar performance with the other two policies on queue 1, however priority queue policy causes a higher PAoI for queue 2. This is because queue 1 has higher priority under the priority queue policy and the server is busy processing packets from queue 1 most of times. When traffic intensity of queue 2 is large, priority queue policy results a smaller PAoI for queue 1 than the other policies, as we can see from Figure 4.14(b). It is because priority queue policy guarantees packets with high priorities to be transmitted once the server becomes available. Like we mentioned in Section 4.1, there could be data sources which have information more important or time-sensitive than the other data sources. Priority queue policy actually helps reduce the PAoI of these data sources by assigning them with higher priorities. Even when the traffic of those unimportant data sources is large, priority queue policy can still guarantee that the PAoI of these prioritized sources remains at a low level, which we can see from Figure 4.14(b). When we consider the average PAoI across queues, as we see from Figure 4.14(c) and (d), priority queue policy can result a smaller average PAoI than the other two policies if higher priority is assigned to queues with low traffic intensity. This also implies that in a system where traffic intensities of queues are not even, priority queue policy is recommended to be applied to reduce the average PAoI over queues.



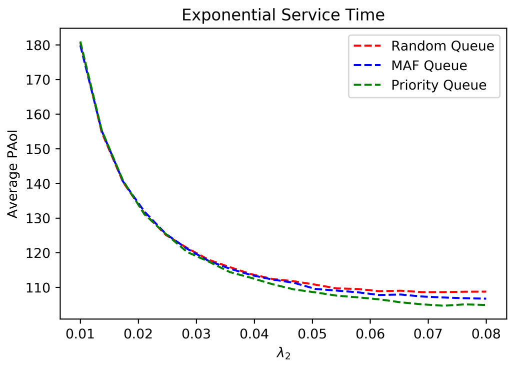
(a) $\lambda_2 = \frac{1}{50}, \mu_1 = \mu_2 = \frac{1}{10}$



(b) $\lambda_1 = \frac{1}{50}, \mu_1 = \mu_2 = \frac{1}{10}$



(c) $\lambda_2 = \frac{1}{50}, \mu_1 = \mu_2 = \frac{1}{10}$



(d) $\lambda_1 = \frac{1}{50}, \mu_1 = \mu_2 = \frac{1}{10}$

Figure 4.14: Policy Comparison

4.6 Concluding Remarks and Future Work

In this research we considered a multi-class multi-buffer queueing system where each class of data source generates packets according to a Poisson process and a single processor uses a static priority scheme to serve the packets. We characterized the PAoI for such a system under two situations: (i) when the buffer size for each queue is one; (ii) when the buffer size for each queue is infinite and service disciplines within each queue can be FCFS or LCFS. We obtained exact expressions for PAoI in case (i) when the service times are exponential, and bounds (which serve as excellent approximations) for case (i) when service times are generally distributed. The method of obtaining PAoI for case (i) becomes cumbersome when the number of queues is large. For case (ii) with general service times, we provide the exact methods for calculating PAoI, and this method can be applied when the number of queues is large.

Using PAoI results we make a few observations that are useful in determining priorities, service disciplines and sampling rates for arrivals. We first find that LCFS is not the optimal service discipline in minimizing PAoI, and we also find that systems with buffer size one at each queue does not always provide smaller PAoI than the systems with infinite buffer size. From both analytical and numerical results, we show that for minimizing the average PAoI across queues, it is beneficial to give higher priorities to queues with lower traffic intensities. Besides, we find that the PAoI of queues with low priorities are more sensitive to the packet arrival rate of high priority queues, and increasing the arrival rate for one queue, while reducing the PAoI for this certain data source, would significantly increase the PAoI of queues with lower priorities. We also provide a mixed strategy which allows the server to use different service disciplines (FCFS or LCFS) across queues for a smaller PAoI.

Since in this paper we mainly focus on static queue priorities, in our future work we will consider a system with dynamic priorities. Besides, in smart manufacturing systems where the status of machines changes over time, sampling with a time-varying rate is also possible and it is interesting to consider the PAoI with time-varying arrival rates. Moreover, the variance of PAoI is also useful in measuring the data freshness in real-time systems, and the distribution of PAoI is also of interest. Thus, there are numerous opportunities for research in the area of PAoI for multi-priority queues.

5. AGE OF INFORMATION FOR SINGLE BUFFER SYSTEMS WITH VACATION SERVER

5.1 Introduction

Age of Information (AoI) has drawn wide attention of researchers recently due to its applications in sensor networks, communication networks and autonomous vehicle systems [99, 105]. Different from the long-established queueing metrics such as delay or waiting time, AoI measures the time elapsed since the generation time of a packet that is most recently delivered. AoI is deemed as a new but useful metric to describe the freshness of data and timeliness of information [138, 100, 137]. An example of application of AoI can be found in smart manufacturing where data sensed by sensors at machines are further processed by edge devices and processors with limited processing capabilities. Processed information would be used for estimating the Remaining Useful Life (RUL) [139], detecting defects of manufactured products [140], or making real-time process controls [141]. However, the long processing time of data packets and inter-sampling time would prevent the decision maker from knowing the real-time status of the ongoing manufacturing processes. In this scenario, AoI can be used as a metric to characterize the information freshness.

AoI was first defined by [99] to describe the freshness of data and it is usually analyzed by queueing models (see [105, 100, 101, 104]). For the purpose of introducing the idea of AoI, we only discuss the system with one date source, which we call the primary data source. Since it is widely accepted and documented that sampling following Poisson processes is effective [102], we assume that the time period between generating two data packets (although these chunks of information could be in several packets, we here regard each chunk itself as a packet) in this primary data source, follows an exponential distribution with parameter λ. Packets generated by the data source are sent to the server for further processing. There is a buffer at the server which can hold at most one packet at a time. We assume that only the most recent arrived packet is kept in the buffer as it contains the freshest information about the data source. If a new packet is generated and there is an old packet waiting in the buffer, the old packet will be discarded when the new packet enters the buffer. There is a single server in the system which processes (serves) packets from the buffer once it becomes available. The processing time of each packet is i.i.d. The *age* at time t is thus defined as $\Delta(t) = t - \max\{r_{\{l\}} : C_{\{l\}} \leq t\}$, where $r_{\{l\}}$ is the generation time of the l^{th} packet that is processed by the server, and $C_{\{l\}}$ is the time when this packet has been processed by the server. Note that the packets that are not processed by the server are not included in the age calculation. The *time-average age* is then defined as $\bar{\Delta} = \lim_{T\to\infty} \frac{1}{T}\int_0^T \Delta(t)dt$. By assuming the system being ergodic, we have $\boldsymbol{E}[\Delta] = \lim_{t\to\infty} \boldsymbol{E}[\Delta(t)] = \bar{\Delta}$, and in this paper we use the term "AoI" to refer $\boldsymbol{E}[\Delta]$. While AoI is a useful metric to measure data freshness, many researchers also analyzed a metric called *Peak Age of Information* (PAoI) for its tractability [105, 100, 101]. We let the l^{th} peak of $\Delta(t)$ be $A_{\{l\}}$, and define the expectation of this peak value, i.e., $\boldsymbol{E}[A_{\{l\}}]$, as PAoI.

Specifically, in this paper we consider AoI in scenarios where the server/processor in the communication network takes "vacations" over time. We assume that the server takes a vacation once a packet has been processed. If the server finds no packet waiting in the buffer upon its returning from a vacation, it then takes another vacation. This specific model is motivated by the application in smart manufacturing, where keeping the energy-consuming server/processor idling when there is no data packet, is not efficient in terms of energy saving. To save system energy as well as guarantee other system performance, a strategy is letting the server go for a low-energy-cost sleeping period when there is no packet waiting in the buffer, and wake up if a packet is observed in buffer when a sleeping period is over. The advantage of this strategy in energy saving has been discussed in [142, 143], however its performance in AoI related metrics have not been studied. Another motivation of our model comes from the underwater sensor networks in the petroleum industry or underwater environment monitoring, where acoustic transmissions are not energy efficient and would result in the batteries needing frequent replacements (see [144, 145]). In such a case, an efficient way for data transmission is to store the sensed data in an underwater node, and use a rechargeable autonomous underwater vehicle that is sent from surface to upload or collect data from the node in a periodic way (see [146, 147]). So we abstract the underwater vehicle arrival as a completion of a vacation, collection of data as service and leaving back to the surface as the start of the next vacation. A third application of our model can be found in remote health monitoring, where the health data is acquired by a wearable device from a patient and transmitted to the healthcare provider over time (see [148, 149]). Our model corresponds to the case where a doctor at the healthcare center checks the patient's updated information from time to time, where the time for the doctor to analyze the data can be modeled as the service time and the time between the doctor checks the updated data for a specific patient can be regarded as the vacation time.

In addition, this vacation server model has a wide range of other applications in smart manufacturing sensor networks and computer-communication systems where the server has additional tasks aside from processing the primary data source of interest [47, 4, 135]. Whenever the server schedules these "non-primary" tasks during the idling period of the primary data source, we can regard this server as "taking vacations". Many queueing network systems such as the priority queue system [4] and the polling system [150], can also be regarded as vacation server systems. Systems with server maintenance (see [151]) or server turning on/off (see [152]) can be modeled as vacation server systems as well. Other vacation server models have been discussed in the queueing literature such as [153, 154, 155, 156], however age related metrics in vacation server models have not been fully studied yet.

In this paper we consider three following variations of the vacation server model with single buffer system:

- *Conventional Buffer System (CBS)* (see [157, 158, 159, 160]): In this system, the buffer becomes empty only when the server finishes serving the packet. New arrivals during processing will be rejected.

- *Buffer Relaxation System (BRS)* (see [158, 159]): In this system, the buffer becomes available once the server starts serving. The vacation will start once a service is done.

- *Conventional Buffer System with Preemption in Service (CBS-P)*: In this system, new arrival during processing will preempt the packet in service. The preempted packet will be discarded.

Note that in BRS, there could be at most two packets in the system at the same time, with one in processing and one waiting in the buffer. We also notice that in BRS, the server will anyway take a vacation after processing a packet, and the packet arriving during processing will be processed only when the vacation is over. This service discipline is also called *gated* in some literatures about vacation servers (see [150, 49, 85]). In these three systems above, a packet arriving during vacation will always preempt the packet waiting in the buffer.

The major contributions of this paper are summarized as follows:

- We provide a decomposition approach which decomposes the peak age of single buffer system into independent components, so that the Laplace-Stieltjes Transform (LST) of peak age in CBS, BRS and CBS-P can be obtained when the vacation time is i.i.d. We then provide closed-form expressions for AoI, PAoI and variance of peak age for CBS, BRS and CBS-P.

- We prove that when vacation time is i.i.d., BRS always has smaller PAoI than CBS, regardless of vacation or service time distribution. We also provide the condition under which CBS-P always has smaller PAoI than CBS.

- Unlike PAoI, for AoI we show that when vacation time is i.i.d., BRS does not always have smaller AoI than CBS, and CBS-P does not always have smaller AoI than CBS.

- We extend our discussion to systems with non-i.i.d vacation times, and provide an approach to calculate PAoI for polling systems with Markovian polling schemes. We show that in polling systems, BRS no longer has advantage over CBS in terms of small PAoI, and CBS-P has smaller PAoI than CBS when the service time is exponential.

The rest of this paper is organized as follows: A summary of the literature is provided in Section 5.2. In Section 5.3 we consider the cases where server takes i.i.d vacations. In Section 5.4 we consider the case with non-i.i.d vacations, and discuss the polling system as an example of non-i.i.d vacation model. We perform numerical studies and develop insights in Section 5.5, and provide concluding remarks and ideas of future work in Section 5.6.

5.2 Related Work

The system with vacation server has been studied by many researchers due to its wide applications [153, 154, 155, 156, 135, 150]. However, most of the previous papers focused on metrics such as average waiting time, queue length, throughput and rejection rate. The AoI related metrics of the system with vacation server has not been fully studied. In the last few years, the data freshness has drawn much attention due to the need of timely information processing and sharing. AoI and PAoI as metrics that measure data freshness, have been studied mostly from a queueing perspective. Kaul et al [99] first introduced the idea of AoI, and provided the average AoI for M/M/1, M/D/1 and D/M/1 queues. Costa et al [105] provided the average AoI and PAoI for M/M/1/1, M/M/1/2 and M/M/1/2* queues (the asterisk means keeping the most recent packet in the buffer), and pointed out that retaining the most recent packet is more efficient than keeping all packets that the data source generates. Najm and Telatar [114] considered M/G/1/1 queue with preemption. Najm and Nasser [110] considered Last Come First Serve (LCFS) scheme with and without preemption in a single queue system with single buffer and gamma service time. M/G/1/1 queue systems with hybrid ARQ (HARQ) protocols are discussed in Najm et al[161]. Soysal and Ulukus [111] considered G/G/1/1 type queues and provided bounds of AoI for different arrival or service processes. Zou et al [112] discussed the waiting procedure in M/G/1/1 and M/G/1/2* systems. Inoue et al [101] discussed the relationship between AoI and PAoI for single queue systems, and derived the LST of AoI and PAoI for different variations of single queue systems. Some recent papers have considered the system with single server but multiple queues. Huang and Modiano [100] considered PAoI of multi-class M/G/1 and M/G/1/1 queues, and they assumed there is one single buffer for all queues in the M/G/1/1 case. Kaul and Yates [128] also considered a model with priority queues with and without waiting rooms for preempted packets. Moltafet et al [162] derived the closed-form of AoI for multi-class M/G/1 queues with First Come First Serve (FCFS) scheme. The system with packet deadlines is considered in Kam et al[163]. Many research articles considered AoI/PAoI in slotted time systems, such as [115, 123, 120, 121, 119]. However, as pointed out by Talak et al [119], PAoI/AoI for the discrete time systems may differ significantly from their continuous time counterpart. For continuous time systems, in many cases the average AoI or PAoI is difficult to obtain, and advanced modeling and mathematical methodologies are thus needed, such as the Stochastic Hybrid System (SHS) used in [125, 104, 128]. Among all the AoI related literature, to the best of our knowledge, there are very few papers discussing systems with vacation server. Maatouk et al [164] considered a special case where server vacations occur in a random manner. Najm et al [103] considered a system with two streams with different priorities and discussed several service disciplines for the low priority stream. Xu and Gautam [4] discussed the M/G/1/2* and M/G/1 priority queues and allowed each queue to have an individual buffer. Closed-form expressions for PAoI under different service disciplines are derived in [4] by modeling the system as vacation server system. However, the general system with single buffer and vacation server has not been fully studied. It is still unclear which variation of the single buffer system, which we introduced in Section 5.1, has smallest AoI or PAoI. And it is unknown how vacation times will influence the AoI/PAoI performance of each system. In this paper, we extend our discussion of vacation server in [4], and introduce a simple but useful decomposition approach to derive closed-form expressions of AoI and PAoI for different variation of the single buffer system. Using the decomposition approach we can also obtain the variance of peak age for general queueing systems with single buffer. We further discuss the advantage of each system under certain conditions. We then extend our discussion to polling system with a single buffer at each queue, and show that the decomposition approach can be easily applied to multi-queue systems to derive closed-form expression of PAoI.

5.3 Age of Information for Systems with Independent Vacations

In this section we consider the system in which the vacations that the server takes are i.i.d. If we regard a vacation as the period when server is sleeping for energy saving, then this setting is the same as the *multi-sleep* scheme that was discussed in [142]. Later in Section 5.4 we will discuss the case where vacations are non-i.i.d. Throughout this section, we assume that each vacation V

that the server takes is a random variable with Laplace–Stieltjes transform (LST) $V^*(s)$. The server will take a vacation once the service is over. When the server comes back from a vacation, it will process the packet if the buffer is non-empty; otherwise a new vacation is taken. In this section we consider three variations of the system by varying the assumption of buffer availability: CBS, BRS and CBS-P, which we defined in Section 5.1.

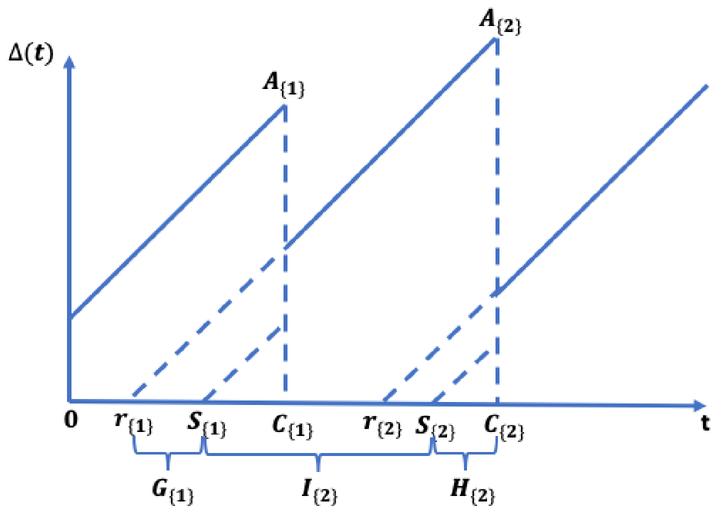


Figure 5.1: Age of Information Decomposition For Non-preemptive Service Systems. Variables $r_{\{l\}}$, $S_{\{l\}}$ and $C_{\{l\}}$ are the generation time, time to start service and completion time for l^{th} packet that is served by the server. The second age peak $A_{\{2\}}$ is decomposed into three components $A_{\{2\}} = G_{\{1\}} + I_{\{2\}} + H_{\{2\}}$. The first component $G_{\{1\}}$ is the waiting time of the first served packet. The second component $I_{\{2\}}$ is the time between the server starts serving two packets. The third component $H_{\{2\}}$ is the service time of the second served packet.

In our previous work [4], we decomposed PAoI into four components where each component could be derived easily. However, such a decomposition cannot be used to derive AoI as the decomposed components are not mutually independent. Therefore, here we introduce a new decomposition method for computing AoI and PAoI in non-preemptive service systems, i.e., CBS and BRS. Since the decomposition approach for CBS-P differs from the decomposition approach for CBS and BRS, we leave our discussion for CBS-P in Appendix B.2. From Figure B.1, we find that the peak age of CBS or BRS is always the time span from the completion time of the recently processed packet, to the generation time (arrival time) of the previously processed packet. This time span can then be divided into three components: waiting time G (in queue) of the previous packet, inter-service-starting time I between the recent and previous processed packets, and service time H of the recent packet. These three components are mutually independent. This is because how long the packet has waited in the buffer has no influence on its processing time or the vacation that the server will take next, thus G is independent of I and H. Also because the service time does not depend on how long the previous vacation lasted, then G, I and H are mutually independent. Thus the PAoI of this system can be given as

$$\boldsymbol{E}[A] \quad = \quad \boldsymbol{E}[G] + \boldsymbol{E}[I] + \boldsymbol{E}[H], \tag{5.1}$$

and the AoI can be given as

$$\boldsymbol{E}[\Delta] \quad = \quad \frac{\boldsymbol{E}[(G+I+H)^2] - \boldsymbol{E}[(G+H)^2]}{2\boldsymbol{E}[I]} = \frac{\boldsymbol{E}[I^2]}{2\boldsymbol{E}[I]} + \boldsymbol{E}[G] + \boldsymbol{E}[H]. \tag{5.2}$$

Note that Equation (5.1) and (5.2) are for non-preemptive service systems, i.e., CBS and BRS. The discussion for CBS-P is left in Appendix B.1. It is important to point out that Equations (5.1) and (5.2) still hold true even when the vacations are non-i.i.d., as the independence of the three decomposed component does not rely on the assumption about vacations.

Assume the LST of G, I and H exist and are given by $G^*(s), I^*(s)$ and $H^*(s)$. Since those three components are mutually independent, we have the LST of A as

$$A^*(s) \quad = \quad G^*(s)I^*(s)H^*(s). \tag{5.3}$$

PAoI can be easily obtained by calculating the first moment of $A^*(s)$. The variance of peak age can be used as a metric to measure the age violations, and the variance of peak age can be given as

$$\begin{aligned} Var(A) &= G^{*(2)}(0) + I^{*(2)}(0) + H^{*(2)}(0) - \{G^{*(1)}(0)\}^2 - \{I^{*(1)}(0)\}^2 - \{H^{*(1)}(0)\}^2 \quad (5.4) \\ &= Var(G) + Var(I) + Var(H). \quad (5.5) \end{aligned}$$

It is shown in [4] that

$$\boldsymbol{E}[G] = \frac{1}{\lambda}(1 - \boldsymbol{E}[e^{-\lambda W}]), \quad (5.6)$$

where W is the time period that the buffer is occupied. Note that if we consider a different system where we only keep the first packet that arrives in the buffer and reject the others, W is also the waiting time of the packet that enters the buffer. If $W^*(s)$ is the LST of W, then $\boldsymbol{E}[G] = \frac{1}{\lambda}(1 - W^*(\lambda))$. Therefore, once we have $W^*(s)$ and $I^*(s)$, we are able to obtain $\boldsymbol{E}[\Delta]$ and $\boldsymbol{E}[A]$. In order to obtain the variance of A, one also needs to know the LST of G. The LST of G can also be written as a function of $W^*(s)$, as shown in the following lemma.

Lemma 5.3.1. $G^*(s) = \frac{\lambda}{\lambda+s} + \frac{s}{\lambda+s} W^*(\lambda + s)$ *for the system with single buffer.*

Proof. It is shown in [4] that $\boldsymbol{P}(G \leq x | m(t) = m, W = t) = 1 - (\frac{t-x}{t})^m$ if there are $m(t) = m$ packets arriving during time W. From the fact that $\boldsymbol{E}[e^{-sG} | m(t) = 0, W = t] = e^{-st}$ we have

$$\begin{aligned} \boldsymbol{E}[e^{-sG} | W = t] &= \int_{x=0}^{t} e^{-sx} \sum_{m=1}^{\infty} \frac{m(t-x)^{m-1}}{t^m} e^{-\lambda t} \frac{(\lambda t)^m}{m!} dx + e^{-st} e^{-\lambda t} \\ &= \frac{\lambda}{\lambda + s} + \frac{s}{\lambda + s} e^{-(\lambda+s)t}. \end{aligned}$$

By unconditioning on $W = t$ we can prove the lemma. □

In the remaining part of this section we introduce the way to derive $I^*(s)$ and $W^*(s)$ for CBS, BRS and CBS-P.

5.3.1 Conventional Buffer System

In this subsection we mainly derive the $\boldsymbol{E}[\Delta]$, $\boldsymbol{E}[A]$ and $Var(A)$ of the CBS where arrivals are rejected when the server is serving. Recall that in CBS, the buffer will not be available until the processing is done, and the server will start a vacation once the buffer becomes empty. We provide the results for CBS in the following theorem.

Theorem 5.3.2. *The AoI of CBS is given as* $\boldsymbol{E}[\Delta_{CBS}] = -\frac{H^{*(2)}(0)+2H^{*(1)}(0)\frac{V^{*(1)}(0)}{1-V^{*}(\lambda)}+\frac{V^{*(2)}(0)}{1-V^{*}(\lambda)}+2\frac{V^{*(1)}(0)V^{*(1)}(\lambda)}{(1-V^{*}(\lambda))^2}}{2\left(H^{*(1)}(0)+\frac{V^{*(1)}(0)}{1-V^{*}(\lambda)}\right)} + \frac{1}{\lambda} + \frac{V^{*(1)}(\lambda)}{1-V^{*}(\lambda)} - H^{*(1)}(0)$*, the PAoI of CBS is given as* $\boldsymbol{E}[A_{CBS}] = \frac{1}{\lambda} + \frac{V^{*(1)}(\lambda)-V^{*(1)}(0)}{1-V^{*}(\lambda)} - 2H^{*(1)}(0)$*, and the variance of peak age of CBS is given by* $Var(A_{CBS}) = \frac{1}{\lambda^2} + \frac{V^{*(2)}(0)-V^{*(2)}(\lambda)}{1-V^{*}(\lambda)} - \left(\frac{V^{*(1)}(\lambda)-V^{*(1)}(0)}{1-V^{*}(\lambda)}\right)^2 + 2H^{*(2)}(0) - 2\left(H^{*(1)}(0)\right)^2.$

Proof. We first show that $I^*(s) = H^*(s)\frac{V^*(s)-V^*(s+\lambda)}{1-V^*(s+\lambda)}$ for CBS. Notice that the period I starts once the server starts serving, and ends when the server comes back from a vacation and observes a packet waiting in the buffer. Therefore we have $I^*(s) = \boldsymbol{E}[e^{-s(H+B)}]$, where B is the time period during which the server is continuously in vacation. Let $B^*(s)$ be the LST of B. By conditioning on the length of the first vacation V_1 that the server takes after serving a packet, and also conditioning on the number of arrivals (including the rejected ones) during V_1, i.e., $m(V_1)$, we have $\boldsymbol{E}[e^{-sB}|V_1 = v_1, m(V_1) \geq 1] = e^{-sv_1}$ and $\boldsymbol{E}[e^{-sB}|V_1 = v_1, m(V_1) = 0] = e^{-sv_1}B^*(s)$. Unconditioning on $m(v_1)$ we have $\boldsymbol{E}[e^{-sB}|V_1 = v_1] = e^{-sv_1}(1 - e^{-\lambda v_1}) + e^{-sv_1}B^*(s)e^{-\lambda v_1}$. Then by unconditioning on V_1, we have $B^*(s) = V^*(s) - V^*(s+\lambda) + B^*(s)V^*(s+\lambda)$ and $B^*(s) = \frac{V^*(s)-V^*(s+\lambda)}{1-V^*(s+\lambda)}$. Therefore we have $I^*(s) = H^*(s)\frac{V^*(s)-V^*(s+\lambda)}{1-V^*(s+\lambda)}$.

We next derive the expression for $\boldsymbol{E}[G]$. From Equation (5.6) we know that $\boldsymbol{E}[G]$ can be given using the formula of the LST of W, where W is the time period when the buffer is occupied. So we now derive the LST of W. Since when the buffer becomes occupied, the server must be on a vacation. From Campbell's Theorem (P173, Theorem 5.14 in [129]) we have

$$\boldsymbol{E}[e^{-sW}|m(t) = m, V_1 = t] \;=\; \int_0^t e^{-sx}\frac{mx^{m-1}}{t^m}dx.$$

Unconditioning on $m(t) = m$ and using the fact that $\boldsymbol{P}(m(t) = m|m(t) \geq 1) = \frac{(\lambda t)^m}{m!}\frac{e^{-\lambda t}}{1-e^{-\lambda t}}$,we have

$$\begin{aligned}
\boldsymbol{E}[e^{-sW}|V_1 = t, m(V_1) \geq 1] &= \sum_{m=1}^{\infty}\int_{x=0}^{t} e^{-sx}\frac{mx^{m-1}}{t^m}\frac{e^{-\lambda t}}{1-e^{-\lambda t}}\frac{(\lambda t)^m}{m!}dx \\
&= \int_{x=0}^{t} e^{-sx}\frac{e^{-\lambda t}}{1-e^{-\lambda t}}\sum_{m=1}^{\infty}\frac{(\lambda x)^{m-1}}{(m-1)!}\lambda dx \\
&= \frac{e^{-\lambda t}-e^{-st}}{(s-\lambda)(1-e^{-\lambda t})}\lambda.
\end{aligned}$$

Now we need to find $\boldsymbol{P}(t < V_1 \leq t + dt|m(V_1) \geq 1)$. From

$$\begin{aligned}
\boldsymbol{P}(V_1 \leq x|m(V_1) \geq 1) &= \frac{\boldsymbol{P}(V_1 \leq x, m(V_1) \geq 1)}{\boldsymbol{P}(m(V_1) \geq 1)} = \frac{\int_0^x dV(u)(1-e^{-\lambda u})}{\int_0^\infty dV(u)(1-e^{-\lambda u})} \\
&= \frac{\int_0^x dV(u)(1-e^{-\lambda u})}{1-V^*(\lambda)},
\end{aligned}$$

we have $\boldsymbol{P}(t < V_1 \leq t + dt|m(V_1) \geq 1) = \frac{dV(t)(1-e^{-\lambda t})}{1-V^*(\lambda)}$. Therefore

$$\begin{aligned}
\boldsymbol{E}[e^{-sW}|m(V_1) \geq 1] &= \int_0^\infty \boldsymbol{E}[e^{-sW}|V_1 = t, m(V_1) \geq 1]\frac{dV(t)(1-e^{-\lambda t})}{1-V^*(\lambda)} \\
&= \int_0^\infty \frac{e^{-\lambda t}-e^{-st}}{(s-\lambda)(1-e^{-\lambda t})}\lambda\frac{dV(t)(1-e^{-\lambda t})}{1-V^*(\lambda)} \\
&= \frac{V^*(\lambda)-V^*(s)}{(s-\lambda)(1-V^*(\lambda))}\lambda.
\end{aligned}$$

Since W is the period that the buffer is occupied, and the buffer is only occupied when $m(V_1) \geq 1$, so that $\boldsymbol{E}[e^{-sW}|m(V_1) \geq 1] = \boldsymbol{E}[e^{-sW}]$. We thus have $W^*(s) = \frac{V^*(\lambda)-V^*(s)}{(s-\lambda)(1-V^*(\lambda))}\lambda$. Using L'Hospital rule at $s = \lambda$ we have $\boldsymbol{E}[e^{-\lambda W}] = \frac{-\lambda V^{*(1)}(\lambda)}{1-V^*(\lambda)}$. From the fact that $\boldsymbol{E}[G] = \frac{1}{\lambda}(1 - W^*(\lambda))$, we have $\boldsymbol{E}[G] = \frac{1}{\lambda} + \frac{V^{*(1)}(\lambda)}{1-V^*(\lambda)}$. Then from Equation (5.1) and (5.2) we can obtain the closed-form expressions for AoI and PAoI using the expression for $\boldsymbol{E}[G]$ and $I^*(s)$. By Lemma 5.3.1 we have $G^*(s) = \frac{\lambda}{\lambda+s}\frac{1-V^*(s+\lambda)}{1-V^*(\lambda)}$. By taking the second derivative of $G^*(s)$ we have

$$G^{*(2)}(0) = \frac{2}{\lambda^2} + \frac{2}{\lambda}\frac{V^{*(1)}(\lambda)}{1-V^*(\lambda)} - \frac{V^{*(2)}(\lambda)}{1-V^*(\lambda)}.$$

Using Equation (5.4), the variance A is given by

$$\begin{aligned} Var(A_{CBS}) &= \frac{1}{\lambda^2} + \frac{V^{*(2)}(0) - V^{*(2)}(\lambda)}{1 - V^*(\lambda)} \\ &\quad - \left(\frac{V^{*(1)}(\lambda) - V^{*(1)}(0)}{1 - V^*(\lambda)}\right)^2 + 2H^{*(2)}(0) - 2\left(H^{*(1)}(0)\right)^2. \end{aligned}$$

□

In the following corollary we provide the $\boldsymbol{E}[\Delta]$, $\boldsymbol{E}[A]$ and $Var(A)$ for the case where service times and vacation times are both exponential.

Corollary 5.3.3. *If the vacation time is exponentially distributed with parameter* v*, and service time is exponentially distributed with parameter* μ*, then we have* $\boldsymbol{E}[\Delta_{CBS}] = \frac{1}{\lambda} + \frac{1}{v} - \frac{\lambda+v+\mu}{v\lambda+\mu\lambda+\mu v} + \frac{1}{v+\lambda} + \frac{2}{\mu}$*,* $\boldsymbol{E}[A_{CBS}] = \frac{1}{\lambda} + \frac{1}{v} + \frac{1}{v+\lambda} + \frac{2}{\mu}$ *and* $Var(A_{CBS}) = \frac{1}{(\lambda+v)^2} + \frac{1}{\lambda^2} + \frac{1}{v^2} + \frac{2}{\mu^2}$*.*

Proof. When the vacation time is exponentially distributed, we have $I^*(s) = \frac{\mu v\lambda}{(\mu+s)(v+s)(\lambda+s)}$. So from $\boldsymbol{E}[I] = \frac{v\lambda+\mu\lambda+\mu v}{\mu v\lambda}$ and $\boldsymbol{E}[I^2] = 2\frac{(v\lambda+\mu\lambda+\mu v)^2}{\mu^2 v^2\lambda^2} - 2\frac{\lambda+v+\mu}{\mu v\lambda}$, we have $Var(I) = \frac{1}{\mu^2} + \frac{1}{v^2} + \frac{1}{\lambda^2}$. Also we know $\boldsymbol{E}[G] = \frac{1}{v+\lambda}$ and $\boldsymbol{E}[G^2] = \frac{2}{(v+\lambda)^2}$. So that we have the results from Equation (5.1) and (5.2). □

5.3.2 Buffer Relaxation System

In this subsection we derive the AoI and PAoI for the BRS, where the buffer becomes available as soon as the service starts. Recall that the server will go on a vacation after serving one packet. The arrival during processing a packet will be processed after the vacation is over. A potential benefit by applying this “gated” policy is that it prevents the server from serving the buffer continuously without taking vacations, when the arrival rate is large. As discussed in Section 5.1, many systems can be modeled as queueing systems with vacation server, and the vacation is also important for some systems such as the priority queue systems [4], in which “vacation” actually corresponds to “serving the non-primary queues”. Therefore, it is sometime crucial for the server to take vacations. Also, as we will see in this subsection, BRS may have advantage over CBS in some cases.

Next we derive the AoI and PAoI for BRS in the following theorem.

Theorem 5.3.4. *The AoI of BRS is given by* $\boldsymbol{E}[\Delta_{BRS}] = \frac{-I^{*(2)}(0)}{2I^{*(1)}(0)} + \frac{1}{\lambda} + V^{*(1)}(\lambda)H^*(\lambda) + V^*(\lambda)H^{*(1)}(\lambda) + \frac{V^{*(1)}(\lambda)}{1-V^*(\lambda)}V^*(\lambda)H^*(\lambda) - H^{*(1)}(0)$, *and PAoI of BRS is given by* $\boldsymbol{E}[A_{BRS}] = -I^{*(1)}(0) + \frac{1}{\lambda} + V^{*(1)}(\lambda)H^*(\lambda) + V^*(\lambda)H^{*(1)}(\lambda) + \frac{V^{*(1)}(\lambda)}{1-V^*(\lambda)}V^*(\lambda)H^*(\lambda) - H^{*(1)}(0)$, *where* $I^*(s) = H^*(s)V^*(s) + H^*(\lambda+s)\frac{V^*(s+\lambda)(V^*(s)-1)}{1-V^*(s+\lambda)}$.

Proof. We first show that in BRS, $I^*(s) = H^*(s)V^*(s) + H^*(\lambda+s)\frac{V^*(s+\lambda)(V^*(s)-1)}{1-V^*(s+\lambda)}$. Since each I starts with processing a packet with processing time H, if there is more than one arrival during the processing time H, then the server only takes one vacation after processing the current packet. If there is no arrival during this processing time, the server takes vacations until a packet is observed in buffer when a vacation is over. By conditioning on scenarios during H, we have $\boldsymbol{E}[e^{-sI}|H = h, m(H) \geq 1] = e^{-sh}V^*(s)$, and $\boldsymbol{E}[e^{-sI}|H = h, m(H) = 0] = e^{-sh}B^*(s)$. We thus have $\boldsymbol{E}[e^{-sI}|H = h] = e^{-sh}V^*(s)(1-e^{-\lambda h}) + e^{-sh}B^*(s)e^{-\lambda h}$. Therefore $\boldsymbol{E}[e^{-sI}] = H^*(s)V^*(s) - H^*(\lambda+s)V^*(s) + H^*(\lambda+s)B^*(s)$, where $B^*(s) = \frac{V^*(s)-V^*(s+\lambda)}{1-V^*(s+\lambda)}$.

We next derive $\boldsymbol{E}[G]$ for BRS. From Equation (5.6) we know that $\boldsymbol{E}[G]$ can be written as a formula of the LST of W, which is the time period when the buffer is occupied. So in the following we first derive the LST of W. If there is more than one arrival before the server returns from the first vacation, then

$$\boldsymbol{E}[e^{-sW}|m(V_1+H) \geq 1] = \frac{V^*(\lambda)H^*(\lambda) - V^*(s)H^*(s)}{(s-\lambda)(1-V^*(\lambda)H^*(\lambda))}\lambda.$$

If there is no arrival before the server returns from the first vacation, we have

$$\boldsymbol{E}[e^{-sW}|m(V_1+H) = 0] = \frac{V^*(\lambda) - V^*(s)}{(s-\lambda)(1-V^*(\lambda))}\lambda.$$

We thus have

$$\begin{aligned}\boldsymbol{E}[e^{-sW}] &= \frac{V^*(\lambda)H^*(\lambda)-V^*(s)H^*(s)}{(s-\lambda)(1-V^*(\lambda)H^*(\lambda))}\lambda\left\{1-V^*(\lambda)H^*(\lambda)\right\} \\ &\quad +\frac{V^*(\lambda)-V^*(s)}{(s-\lambda)(1-V^*(\lambda))}\lambda V^*(\lambda)H^*(\lambda).\end{aligned}$$

Using L'Hospital rule at $s=\lambda$, we have

$$\boldsymbol{E}[e^{-\lambda W}] = -\lambda V^{*(1)}(\lambda)H^*(\lambda)-\lambda V^*(\lambda)H^{*(1)}(\lambda)-\frac{V^{*(1)}(\lambda)}{1-V^*(\lambda)}\lambda V^*(\lambda)H^*(\lambda).$$

Therefore $\boldsymbol{E}[G]=-V^{*(1)}(\lambda)H^*(\lambda)-V^*(\lambda)H^{*(1)}(\lambda)-\frac{V^{*(1)}(\lambda)}{1-V^*(\lambda)}\lambda V^*(\lambda)H^*(\lambda)$. Using Equation (5.1) and (5.2) we can then obtain the PAoI and AoI of BRS. □

We can also obtain the variance of peak age for BRS, although its closed-form expression is complex. To obtain the variance of peak age, we need the LST of G, I and H as we show in Equation (5.4). The LST of I has been given in Theorem 5.3.4, which is $I^*(s)=H^*(s)V^*(s)+H^*(\lambda+s)\frac{V^*(s+\lambda)(V^*(s)-1)}{1-V^*(s+\lambda)}$. We also have

$$G^*(s) = \frac{\lambda}{\lambda+s}\left\{1+\frac{V^*(\lambda)H^*(\lambda)}{1-V^*(\lambda)}(1-V^*(\lambda+s))-V^*(\lambda+s)H^*(\lambda+s)\right\}$$

from the fact that $G^*(s)=\frac{\lambda}{\lambda+s}+\frac{s}{\lambda+s}W^*(\lambda+s)$ where $W^*(s)$ is given in Theorem 5.3.4. We will show the numerical results for the variance of peak age for BRS in Section 5.5. In the next corollary we show the results for the system with exponential service time and exponential vacation time.

Corollary 5.3.5. *For exponential vacation time with parameter v and exponential service time with parameter μ, we have $\boldsymbol{E}[\Delta_{BRS}]=\frac{\frac{1}{v^2}+\frac{1}{v\mu}+\frac{1}{\mu^2}+\frac{\mu}{\lambda v(\lambda+\mu)}+\frac{\mu}{\lambda^2(\lambda+\mu)}+\frac{\mu}{\lambda(\lambda+\mu)^2}}{\frac{1}{v}+\frac{1}{\mu}+\frac{1}{\lambda}-\frac{1}{\lambda+\mu}}+\frac{1}{\lambda+v}+\frac{\lambda v}{(\lambda+\mu)^2(\lambda+v)}+\frac{1}{\mu}$ and $\boldsymbol{E}[A_{BRS}]=\frac{\mu^2-\mu v+\lambda\mu}{(\lambda+\mu)^2(\lambda+v)}+\frac{1}{v}+\frac{2}{\mu}+\frac{1}{\lambda}$.*

Proof. The results follow from Theorem 5.3.4 with $V^*(s)=\frac{v}{v+s}$ and $H^*(s)=\frac{\mu}{\mu+s}$. □

A question is whether BRS always has smaller AoI or PAoI than CBS. In the next theorem we show that BRS always has smaller PAoI than CBS, for all arbitrary service and vacation distributions.

Theorem 5.3.6. *PAoI in BRS is always smaller than PAoI in CBS, if the arrival process is Poisson, and service times as well as vacation times are i.i.d.*

Proof. From Theorem 5.3.2 we have

$$\boldsymbol{E}[A_{CBS}] = -2H^{*(1)}(0) + \frac{1}{\lambda} + \frac{V^{*(1)}(\lambda) - V^{*(1)}(0)}{1 - V^{*}(\lambda)},$$

and from Theorem 5.3.4 we have

$$\begin{aligned}\boldsymbol{E}[A_{BRS}] &= -2H^{*(1)}(0) - V^{*(1)}(0) + \frac{1}{\lambda} + V^{*(1)}(\lambda)H^{*}(\lambda) + V^{*}(\lambda)H^{*(1)}(\lambda) \\ &\quad + \frac{H^{*}(\lambda)V^{*}(\lambda)}{1 - V^{*}(\lambda)}(V^{*(1)}(\lambda) - V^{*(1)}(0)).\end{aligned}$$

We then have

$$\begin{aligned}&\boldsymbol{E}[A_{CBS}] - \boldsymbol{E}[A_{BRS}] \\ =\ & \frac{V^{*(1)}(\lambda) - V^{*(1)}(0)}{1 - V^{*}(\lambda)}(1 - H^{*}(\lambda)V^{*}(\lambda)) + V^{*(1)}(0) - V^{*(1)}(\lambda)H^{*}(\lambda) - V^{*}(\lambda)H^{*(1)}(\lambda) \\ =\ & \frac{\left[V^{*(1)}(\lambda) - V^{*(1)}(0)V^{*}(\lambda)\right](1 - H^{*}(\lambda)) + V^{*}(\lambda)H^{*(1)}(\lambda)(V^{*}(\lambda) - 1)}{1 - V^{*}(\lambda)}.\end{aligned}$$

Notice that $H^{*(1)}(\lambda) \leq 0$ and $0 \leq V^{*}(\lambda) \leq 1$, we have $V^{*}(\lambda)H^{*(1)}(\lambda)(V^{*}(\lambda) - 1) \geq 0$. Since $0 \leq H^{*}(\lambda) \leq 1$, to show that $\boldsymbol{E}[A_{CBS}] - \boldsymbol{E}[A_{BRS}] \geq 0$, we only need to show $V^{*(1)}(\lambda) - V^{*(1)}(0)V^{*}(\lambda) \geq 0$. Since $V^{*(1)}(\lambda) - V^{*(1)}(0)V^{*}(\lambda) = -\boldsymbol{E}[Ve^{-\lambda V}] + \boldsymbol{E}[V]\boldsymbol{E}[e^{-\lambda V}]$, we let $X = V$, $Y = e^{-\lambda V}$ with CDF $F_X(x)$, $F_Y(x)$ and joint CDF $F(x, y)$. We now show that $\boldsymbol{P}(X \leq$

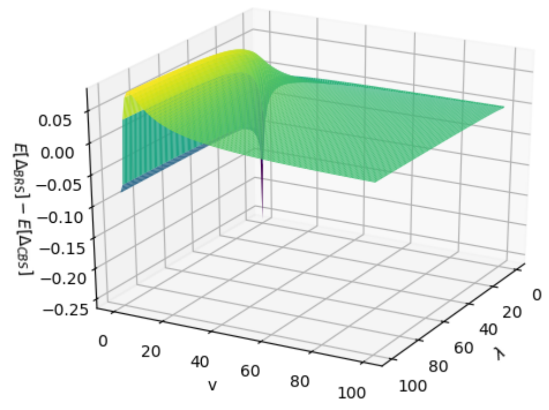


Figure 5.2: AoI in CBS vs AoI in BRS. Service and vacation times are exponential.

$x, Y \leq y) \leq \boldsymbol{P}(X \leq x)\boldsymbol{P}(Y \leq y)$. Notice that

$$\begin{aligned} F(x,y) = \boldsymbol{P}(X \leq x, Y \leq y) &= \boldsymbol{P}(V \leq x, e^{-\lambda V} \leq y) = \boldsymbol{P}(-\frac{\ln y}{\lambda} \leq V \leq x) \\ &= \boldsymbol{P}(V \leq x) - \boldsymbol{P}(V \leq -\frac{\ln y}{\lambda}) \\ &\leq \boldsymbol{P}(V \leq x) - \boldsymbol{P}(V \leq -\frac{\ln y}{\lambda})\boldsymbol{P}(V \leq x) \\ &= F_X(x)F_Y(y). \end{aligned}$$

From [165] we know $\boldsymbol{E}[XY] - \boldsymbol{E}[X]\boldsymbol{E}[Y] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} [F(x,y) - F_X(x)F_Y(y)]\,dxdy$. Therefore $V^{*(1)}(\lambda) - V^{*(1)}(0)V^*(\lambda) = \boldsymbol{E}[X]\boldsymbol{E}[Y] - \boldsymbol{E}[XY] \geq 0$ and $\boldsymbol{E}[A_{CBS}] - \boldsymbol{E}[A_{BRS}] \geq 0$. $\square$

However, BRS does not always have smaller AoI than CBS. A graph for comparison is provided in Figure 5.2 where service time and vacation time are both exponential. As we see from Figure 5.2, when vacation time is large (i.e., small v), $\boldsymbol{E}[\Delta_{BRS}]$ is smaller than $\boldsymbol{E}[\Delta_{CBS}]$. However when vacation time is small, the CBS has smaller AoI than BRS.

5.3.3 Conventional Buffer System with Preemption in Service

In this subsection we consider the system when preemption is allowed in service, i.e., CBS-P. Note that when allowing preemption in service, both CBS and BRS will reduce to CBS-P. Different from the non-preemptive service case, in CBS-P, the age peak cannot be decomposed as shown in Equation (5.1) simply because the packet that result in age peak may not have waiting time G (as it may be a preemptive packet). A detailed decomposition approach for CBS-P is given in Appendix B.1. The AoI and PAoI in CBS-P is given in the following theorem.

Theorem 5.3.7. *The PAoI for CBS-P is given by* $\boldsymbol{E}[A_{CBS-P}] = \frac{1-H^*(\lambda)-\lambda H^{*(1)}(\lambda)+H^*(\lambda)^2}{\lambda H^*(\lambda)} + \frac{H^*(\lambda)V^{*(1)}(\lambda)-V^{*(1)}(0)}{1-V^*(\lambda)}$*, and the AoI for this system is given by*

$$\boldsymbol{E}[\Delta_{CBS-P}] = \frac{\frac{V^{*(2)}(0)}{1-V^*(\lambda)} + 2\frac{V^{*(1)}(0)V^{*(1)}(\lambda)}{(1-V^*(\lambda))^2} - 2\frac{V^{*(1)}(0)}{1-V^*(\lambda)}\frac{1-H^*(\lambda)}{\lambda H^*(\lambda)} + \frac{2}{\lambda H^*(\lambda)^2}\left[\frac{1}{\lambda} - \frac{H^*(\lambda)}{\lambda} + H^{*(1)}(\lambda)\right]}{2(-\frac{V^{*(1)}(0)}{1-V^*(\lambda)} + \frac{1-H^*(\lambda)}{\lambda H^*(\lambda)})}$$
$$-\frac{H^{*(1)}(\lambda)}{H^*(\lambda)} + H^*(\lambda)\left(\frac{1}{\lambda} + \frac{V^{*(1)}(\lambda)}{1-V^*(\lambda)}\right).$$

Proof. Detailed proof is shown in Appendix B.1. □

Corollary 5.3.8. *For exponential vacation time with parameter v and exponential service time with parameter μ, we have* $\boldsymbol{E}[A_{CBS-P}] = \frac{1}{\lambda} + \frac{1}{\mu} + \frac{1}{v} + \frac{\lambda+\mu+p}{(\lambda+\mu)(\lambda+v)}$ *and* $\boldsymbol{E}[\Delta_{CBS-P}] = \frac{\frac{v+\lambda}{v^2\lambda} + \frac{1}{\lambda^2} + \frac{v+\lambda}{\lambda v\mu} + \frac{1}{\mu^2}}{\frac{1}{\lambda} + \frac{1}{\mu} + \frac{1}{v}} + \frac{1}{\mu+\lambda} + \frac{\mu}{(\mu+\lambda)(v+\lambda)}$.

Theorem 5.3.9. *If the service time is exponentially distributed, then the system CBS-P has both AoI and PAoI smaller than CBS.*

Proof. Detailed proof is shown in Appendix B.2. □

Notice that Theorem 5.3.9 holds true for exponential service times only. A question is whether the inequalities still holds between CBS and CBS-P if the service is not exponential. In the next theorem, we provide a sufficient condition under which CBS-P will always have smaller PAoI than CBS.

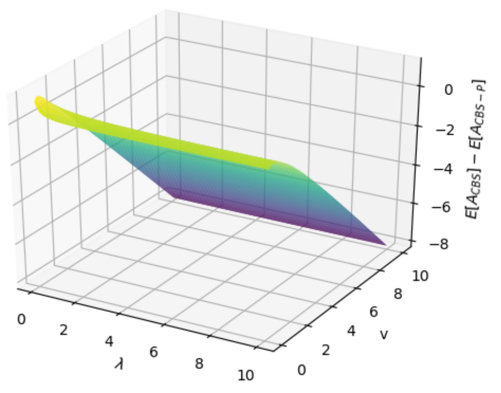
Theorem 5.3.10. *If the service time H satisfies $\boldsymbol{E}[H] \geq \frac{1-H^*(s)}{sH^*(s)}$ for all $s > 0$, then CBS-P always has smaller PAoI than CBS.*

Proof. Detailed proof is shown in Appendix B.3. □

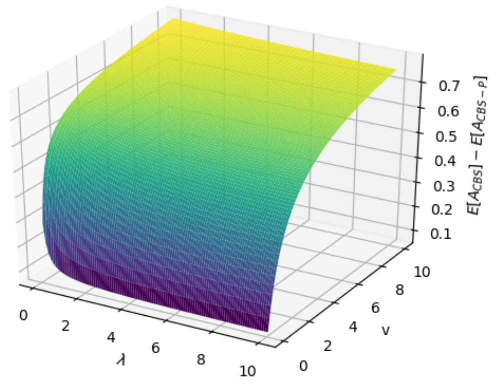
Theorem 5.3.10 provides a useful sufficient condition for checking whether CBS-P has smaller PAoI than CBS, and this sufficient condition does not rely on the vacation time distribution. Necessary condition can be obtained by directly comparing the closed-form expression of PAoI for CBS-P and CBS, however it is quite involved. We can also provide sufficient and necessary conditions for CBS-P to have smaller AoI or variance of peak age than CBS or BRS, by simply comparing their closed-form expressions. However those conditions are specific and complicated. In Section 5.5 we will compare $\boldsymbol{E}[A]$, .$\boldsymbol{E}[\Delta]$ and $Var(A)$ numerically. We now provide some examples on how Theorem 5.3.10 can be applied. When the service time is exponential with parameter μ, we have $\frac{1-H^*(s)}{sH^*(s)} = \frac{1-\frac{\mu}{\mu+s}}{s\frac{\mu}{\mu+s}} = \frac{1}{\mu} = \boldsymbol{E}[H]$. Then by Theorem 5.3.10 we can conclude that CBS-P has smaller PAoI than CBS, which is the same as our conclusion in Theorem 5.3.9. We next give an example where the service time is Gamma distributed with parameters α and β. Since the LST of Gamma distribution is given by $H^*(s) = (1+\beta s)^{-\alpha}$, we have $\frac{1-H^*(s)}{sH^*(s)} = \frac{(1+\beta s)^{\alpha}-1}{s}$. By Bernoulli's inequality we have that $(1+\beta s)^{\alpha} \geq 1+\alpha\beta s$ when $\alpha \geq 1$, and $(1+\beta s)^{\alpha} < 1+\alpha\beta s$ when $\alpha < 1$. From the fact that $\boldsymbol{E}[H] = \alpha\beta$, we have $\frac{1-H^*(s)}{sH^*(s)} \geq \boldsymbol{E}[H]$ when $\alpha \geq 1$ and $\frac{1-H^*(s)}{sH^*(s)} \leq \boldsymbol{E}[H]$ when $\alpha < 1$. A numerical study of this example is given in Figure 5.3, where service time is Gamma distributed and vacation time is exponential. In Figure 5.3(a), we find that when $\alpha = \frac{1}{2}$, CBS-P does not always have smaller PAoI than CBS. However, in Figure 5.3(b) where $\alpha = 2$, the service distribution satisfies $\boldsymbol{E}[H] \geq \frac{1-H^*(s)}{sH^*(s)}$, we find that CBS-P has smaller PAoI than CBS, for all the positive values of λ and v.

5.3.4 Discussions for Systems without Server Vacation

We realize that when the server takes no vacations or takes vacation infinitely fast, then CBS is equal to the M/G/1/1 non-preemptive system, BRS is equal to the M/G/1/2* system (the asterisk means that only the most recent packet is kept in the buffer as defined in [105, 112]), and CBS-P



(a) $H \sim Gamma(\frac{1}{2}, 1)$



(b) $H \sim Gamma(2, 1)$

Figure 5.3: PAoI in CBS vs PAoI in CBS-P. Service time is Gamma distributed. Vacation time is exponentially distributed.

becomes M/G/1/1/preemptive system. Different variations of these systems has been discussed in [114, 100, 112, 105, 163, 101], however the variance of peak age in these single buffer systems has not been studied. We here provide the variance of peak age for the systems without server vacations, as an extension of our discussion about vacation server systems. With the decomposition approach that we introduced earlier, we are able to provide the variance of peak age for M/G/1/1, M/G/1/2* and M/G/1/1/preemptive systems, as shown in the following theorem.

Theorem 5.3.11. *For M/G/1/1 system we have* $\boldsymbol{E}[A_{M/G/1/1}] = \frac{1}{\lambda} - 2H^{*(1)}(0)$, $\boldsymbol{E}[\Delta_{M/G/1/1}] = \frac{\frac{2}{\lambda^2} - \frac{2}{\lambda}H^{*(1)}(0) + H^{*(2)}(0)}{\frac{1}{\lambda} - H^{*(1)}(0)} - H^{*(1)}(0)$ *and* $Var(A_{M/G/1/1}) = \frac{1}{\lambda^2} + 2H^{*(2)}(0) - 2\{H^{*(1)}(0)\}^2$.

For M/G/1/1/preemptive system we have $\boldsymbol{E}[A_{M/G/1/1/preemptive}] = \frac{-H^{*(1)}(\lambda)}{H^*(\lambda)} + \frac{1}{\lambda H^*(\lambda)}$, $\boldsymbol{E}[\Delta_{M/G/1/1/preemptive}] = \frac{1}{\lambda H^*(\lambda)}$, *and* $Var(A_{M/G/1/1/preemptive}) = \frac{H^{*(2)}(\lambda)}{H^*(\lambda)} - \frac{\{H^{*(1)}(\lambda)\}^2}{H^*(\lambda)^2} + \frac{1}{\lambda^2 H^*(\lambda)^2} + \frac{2H^{*(1)}(\lambda)}{\lambda H^*(\lambda)^2}$.

For M/G/1/2 system we have* $\boldsymbol{E}[A_{M/G/1/2^*}] = -2H^{*(1)}(0) + \frac{1}{\lambda} + H^{*(1)}(\lambda)$, $\boldsymbol{E}[\Delta_{M/G/1/2^*}] = \frac{\frac{1}{2}H^{*(2)}(0) + \frac{1}{\lambda^2}H^*(\lambda) - \frac{1}{\lambda}H^{*(1)}(\lambda)}{-H^{*(1)}(0) + \frac{H^*(\lambda)}{\lambda}} + \frac{1}{\lambda} - \frac{1}{\lambda}H^*(\lambda) + H^{*(1)}(\lambda) - H^{*(1)}(0)$, *and* $Var(A_{M/G/1/2^*}) = 2H^{*(2)}(0) - 2H^{*(1)}(0) + \frac{2H^*(\lambda)[H^{*(1)}(0) + H^{*(1)}(\lambda)]}{\lambda} + \frac{2H^*(\lambda)(1 - H^*(\lambda))}{\lambda^2} + \frac{1}{\lambda^2} - H^{*(2)}(\lambda) - \frac{2}{\lambda}H^{*(1)}(\lambda) - H^{*(1)}(\lambda)^2$.

Proof. The detailed proof is shown in Appendix B.4. □

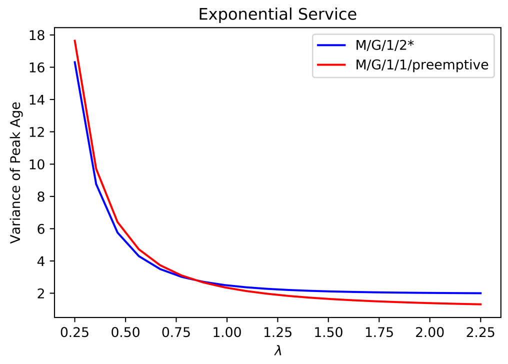


Figure 5.4: Variance of Peak Age for M/M/1/1/preemptive system and M/M/1/2* system with $\mu = 1$.

Corollary 5.3.12. *The variance of peak age in M/M/1/1 is* $Var(A_{M/M/1/1}) = \frac{1}{\lambda^2} + \frac{2}{\mu^2}$*, the variance of peak age in M/M/1/1/preemptive is* $Var(A_{M/M/1/1/preemtive}) = \frac{1}{(\lambda+\mu)^2} + \frac{1}{\lambda^2} + \frac{1}{\mu^2}$*, and the variance of peak age in M/M/1/2* is* $Var(A_{M/M/1/2^*}) = \frac{1}{\lambda^2} + \frac{2}{\mu^2} - \frac{2\lambda^2+4\lambda\mu+3\mu^2}{(\lambda+\mu)^4}$.

From Corollary 5.3.12 we find that the variance of PAoI in M/M/1/2* system is smaller than PAoI in M/M/1/1 system. Also we find that the variance of peak age in M/M/1/1/preemptive system is smaller than it in M/M/1/1 system. Now we compare the variance of peak age for M/M/1/1/preemptive system with M/M/1/2* system. First we have

$$Var(A_{M/M/1/1/preemptive}) - Var(A_{M/M/1/2^*}) = \frac{-\lambda^4 - 4\lambda^3\mu - 3\lambda^2\mu^2 + 2\lambda\mu^3 + 3\mu^4}{(\lambda+\mu)^4\mu^2}.$$

We find that only when λ is large can we have $Var(A_{M/M/1/1/preemptive}) - Var(A_{M/M/1/2^*}) \leq 0$. A demonstrative graph of the case where $\mu = 1$ is shown in Figure 5.4. We can get from numerical study and Figure 5.4 that when $\lambda \leq 0.8168$, M/M/1/2* system has smaller variance of peak age than M/M/1/1/preemptive system.

5.4 Peak Age of Information for Systems with Dependent Vacations

In this section we extend our discussion in Section 5.3 to a more general case by allowing the vacations to be non-i.i.d. In this case, obtaining LST or the second moment of I is difficult, and the closed-form expression for AoI may become intractable. However, the PAoI is still solvable in this case. In this section, we will discuss the approach for deriving the exact solution for PAoI.

From our discussion in Section 5.3 we have $\boldsymbol{E}[G] = \frac{1}{\lambda}(1 - W^*(\lambda))$, where W is the time period when the buffer is occupied. Also from Equation (5.1) and our discussion in Appendix B.5, we have

$$\boldsymbol{E}[A] = \begin{cases} -\frac{1}{\lambda}W^*(\lambda) + \frac{2}{\lambda} + \boldsymbol{E}[W] + 2\boldsymbol{E}[H] & \text{for CBS,} \\ -\frac{1}{\lambda}W^*(\lambda) + \frac{2}{\lambda} + \boldsymbol{E}[W] + \boldsymbol{E}[H] & \text{for BRS, and} \\ -\frac{H^{*(1)}(\lambda)}{H^*(\lambda)} + H^*(\lambda)\frac{1}{\lambda}(1 - W^*(\lambda)) + \boldsymbol{E}[W] + \frac{1}{\lambda H^*(\lambda)} & \text{for CBS-P.} \end{cases}$$

When $W^*(s)$ is available, then the closed-form of PAoI can be obtained. In the remaining part of this section, we will focus our discussion on the polling system as it is a system where the server takes non-i.i.d vacations (see [166]). We will next show the approach to calculate exact PAoI for polling system by obtaining $W^*(s)$.

A polling system is a queueing system that contains a single server and k classes of packets. Each packet class would have its own queue, so there are k queues in the system. The server serves packets by switching between queues, and a switchover time is incurred when the server switches from one queue to another. A demonstrative graph of polling systems is provided in Figure 5.5. Polling systems have a wide application in communication networks and other networks (see [86, 47, 5]), while the PAoI in polling systems has not been fully studied. Specifically, if there are multiple data nodes in the underwater sensor network example which we discussed in Section 5.1 (also see [147, 146]), we can then model the underwater system as a polling system, where each data node can be modeled as a queue/buffer and the autonomous vehicle can be regarded as the server that collects/processes data from each node in a periodic manner.

In this paper we are interested in single buffer systems, so we assume that each queue has a single buffer that can hold only one packet at a time. Similar to our discussion in Section 5.3, we assume that only the most recently arrived packet is kept in the buffer, and we consider three variations of the polling system by making different assumptions about the buffer and service preemption. To distinguish from the names in Section 5.4, we call these three polling systems *Conventional Buffer Polling System (CBPS)*, *Buffer Relaxation Polling System (BRPS)*, and *Conventional Buffer Polling System with Preemption in Service (CBPS-P)* respectively. In CBPS, the buffer is not available until the current packet completes its service. When the server is busy processing, newly arrived packets in this queue will be rejected. So that each queue can only have at most one packet at any time. In BRPS, the buffer becomes available once the service has started, however the new arrival during the service time will be served in the next polling instant. In CBPS-P, the new arrival will preempt the packet in service, and the preempted packet will be discarded. The server will switch to next queue when the service of a packet is complete. In all these three systems, the server will start another switching process if the it observes an empty queue. We assume that the arrival process of packets in each queue i follows a Poisson process with rate λ_i, and the service time H_i for packets at each queue is i.i.d. with mean h_i and LST $H_i^*(s)$. The switchover time from queue i to queue j has mean u_{ij}, CDF $U_{ij}(x)$ and LST $U_{ij}^*(s)$. In the remaining part of this section we use the subscript i to denote the parameter for queue i in the polling system.

There are multiple widely used routing schemes that determine which queue to switch to next for the server. Routing schemes include cyclic [160, 158, 167, 81], Markovian polling [159, 168] and random polling [157]. For most of those polling systems with single buffer in each queue, $W^*(s)$ can be derived (see [158, 159, 157]). In this paper we mainly discuss the Markovian polling scheme, since random polling and cyclic polling schemes are both special cases of the Markovian polling scheme, as we will see later. In the Markovian polling scheme, after serving queue i, the probability of serving queue j next is given by p_{ij}. Considering all the possible states for the current queue and next queue, the switching process can be characterized by a discrete Markov chain with transition matrix $P = [p_{ij}]$. In this paper we assume that P is irreducible positive recurrent. For the cyclic polling scheme, the transition matrix is given by for $i, j \in \{1, 2, ..., k\}$,

$$p_{ij} = \begin{cases} 1 & \text{if } j = i + 1, \\ 0 & \text{otherwise.} \end{cases}$$

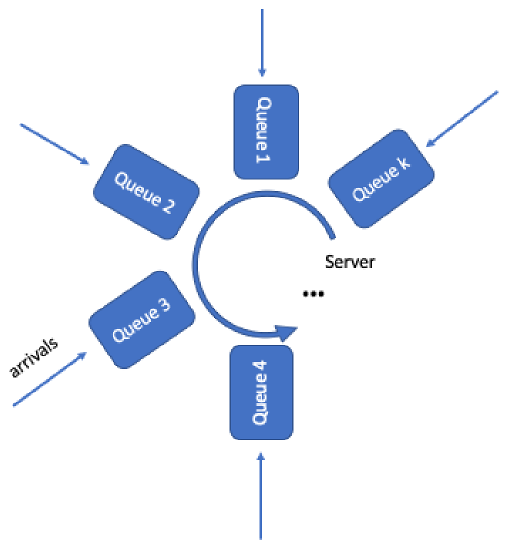


Figure 5.5: A k-queue Polling System with Cyclic Polling Scheme

Two special polling schemes were discussed in [159]. One is called *load-oriented-policy* (LOP), which is defined by the transition matrix with $p_{ij} = \frac{\lambda_j}{\sum_{l=1}^{k} \lambda_l}$ for all i and j. The other polling scheme is called *symmetric* random polling, in which $p_{ij} = \frac{1}{k}$ for all i and j. We will show the performance of these schemes numerically in Section 5.5.

The service process for each individual queue in polling systems can be modeled as a single server with multiple vacations: when the server polls the queue, it serves the packet if the queue is not empty, and takes a vacation (switches out and serves other queues) once the service completes; if the queue is empty when polled, the server takes another vacation. It is important to note that as pointed out by Kofman in [166], even when cyclic polling scheme is applied, the vacations that the server takes in a polling system are not i.i.d. Suppose W_i is the time period that the buffer i is occupied, with LST $W_i^*(s)$. Then our methods for deriving $W^*(s)$ for i.i.d vacations in Section 5.3 cannot be applied here for deriving $W_i^*(s)$ in general polling systems.

We now summarize how $W_i^*(s)$ is obtained by Chung et al[159] and use it to derive the PAoI for queue i (i.e., $\boldsymbol{E}[A_i]$). The main idea in [159] of deriving $W_i^*(s)$ is to solve the following linear system:

$$F_i(z_1,...,z_k) = \sum_{j=1}^{k} \frac{\pi_j}{\pi_i} p_{ji} \tilde{U_{ij}^*} \left\{(1-\tilde{H_j^*})F_j(z_1,...,z_k)_{z_j=0} + \tilde{H_j^*} F_j(z_1,...,z_k)_{z_j=1}\right\} \quad \text{for } i=1,...,k, \tag{5.7}$$

where $F_i(z_1,...,z_k)$ is a probability generating function with $F_i(1,...,1) = 1$, $(\pi_1,...,\pi_k)$ is the stationary distribution of the transition matrix P, $\tilde{U_{ij}^*} = U_{ij}^*(\sum_{l=1}^{k} \lambda_l(1-z_l))$, and

$$\tilde{H_j^*} = \begin{cases} H_j^*(\sum_{l=1,l\neq j}^{k} \lambda_l(1-z_l)) & \text{for CBPS,} \\ H_j^*(\sum_{l=1}^{k} \lambda_l(1-z_l)) & \text{for BRPS, and} \\ \dfrac{H_j^*(\sum_{l=1,l\neq j}^{k} \lambda_l(1-z_l)+\lambda_j)}{\frac{\sum_{l=1,l\neq j}^{k} \lambda_l(1-z_l)}{\sum_{l=1,l\neq j}^{k} \lambda_l(1-z_l)+\lambda_j} + \frac{\lambda_j}{\sum_{l=1,l\neq j}^{k} \lambda_l(1-z_l)+\lambda_j} H_j^*(\sum_{l=1,l\neq j}^{k} \lambda_l(1-z_l)+\lambda_j)} & \text{for CBPS-P.} \end{cases} \tag{5.8}$$

In [159] only $\tilde{H_j^*}$ in CBPS and BRPS are discussed. In both CBPS and BRPS, the server would switch out from queue j after serving a packet from queue j. Here we also discuss the case of CBPS-P. Notice that in CBPS-P, the server switches out from queue j only when one packet has been completely served. If we consider the time period when the server is continuously serving in CBPS-P as the service time for "one packet", then we can also regard CBPS-P as CBPS. The only difference is that in CBPS, each service period is H_j for queue j. While in CBPS-P, the service period is L_j with LST

$$L_j^*(s) = \frac{H_j^*(s+\lambda_j)}{\frac{s}{s+\lambda_j} + \frac{\lambda_j}{s+\lambda_j} H_j^*(s+\lambda_j)}. \tag{5.9}$$

A detailed derivation of Equation (5.9) can be found in Appendix B.1. Then, the formula of $\tilde{H_j^*}$ for CBPS-P in Equation (5.8) is obtained by simply using Equation (5.9) and the formula $\tilde{H_j^*}$ for CBPS.

To solve the system (5.7) analytically is quite involved as shown in [159], however the expected value of W_i can be obtained by solving the system (5.7) with $z_j = 0$ or 1 for $j = 1,...,k$, where only $k(2^k - 1)$ linear equations need to be solved. The expected time W_i is then given as

$$\boldsymbol{E}[W_i] = \frac{\gamma_i}{\lambda_i \alpha_i} - \frac{1}{\lambda_i},$$

where $\alpha_i = 1 - F_i(1, ..., \overset{i}{0}, ..., 1)$ (the notation $F_i(1, ..., \overset{i}{0}, ..., 1)$ means that $z_i = 0$ and $z_{l \neq i} = 1$ in $F_i(z_1, ..., z_k)$) and

$$\gamma_i = \begin{cases} \frac{\lambda_i}{\pi_i} \sum_{j=1}^{k} \pi_j (\alpha_j h_j + \sum_{l=1}^{k} p_{jl} u_{jl}) - \lambda_i \alpha_i h_i & \text{for CBPS,} \\ \frac{\lambda_i}{\pi_i} \sum_{j=1}^{k} \pi_j (\alpha_j h_j + \sum_{l=1}^{k} p_{jl} u_{jl}) & \text{for BRPS, and} \\ \frac{\lambda_i}{\pi_i} \sum_{j=1}^{k} \pi_j (\alpha_j \frac{1 - H_j^*(\lambda_j)}{\lambda H_j^*(\lambda_j)} + \sum_{l=1}^{k} p_{jl} u_{jl}) - \lambda_i \alpha_i \frac{1 - H_i^*(\lambda_i)}{\lambda_i H^*(\lambda_i)} & \text{for CBPS-P.} \end{cases} \quad (5.10)$$

To obtain $\boldsymbol{E}[G_i]$, we need to get $W_i^*(\lambda_i)$. From [159, 158] we have

$$W_i^*(s) = \frac{1}{\alpha_i} \frac{\lambda_i}{s - \lambda_i} \left\{ 1 - \alpha_i - f_i(1 - \frac{s}{\lambda_i}) \right\},$$

where $f_i(z) = F_i(1, ..., \overset{i}{z}, ..., 1)$. Using L'Hospital rule we have

$$W_i^*(\lambda_i) = \frac{f_i^{(1)}(0)}{\alpha_i} = \frac{1}{\alpha_i} \frac{\partial F_i(1, ..., z, ..., 1)}{\partial z}|_{z=0},$$

in which the derivative of $F_i(1, ..., z, ..., 1)$ is needed. Therefore we need to compute the partial derivative of Equation (5.7) with respect to z_l for $l = 1, ..., k$, which is to solve the following linear system:

$$\frac{\partial F_i(z_1, ..., z_k)}{\partial z_l} = \frac{\partial}{\partial z_l} \left\{ \sum_{j=1}^{k} \frac{\pi_j}{\pi_i} p_{ji} \tilde{U_{ij}^*} \left((1 - \tilde{H_j^*}) F_j(z_1, ..., z_k)_{z_j=0} + \tilde{H_j^*} F_j(z_1, ..., z_k)_{z_j=1} \right) \right\}$$
$$\text{for } i = 1, ..., k \text{ and } l = 1, ..., k. \quad (5.11)$$

Note here we only need to solve system (5.11) for $z_j = 0$ or 1 for $j = 1, ..., k$ to obtain $W_i^*(\lambda_i)$, so that $k^2 2^k$ number of equations need to be solved. After solving system (5.7) and (5.11), the closed-form of PAoI can be obtained from the following equations:

$$\boldsymbol{E}[A_i] = \begin{cases} -\frac{1}{\lambda_i}W_i^*(\lambda_i) + \frac{2}{\lambda_i} + \boldsymbol{E}[W_i] + 2\boldsymbol{E}[H_i] & \text{for CBPS,} \\ -\frac{1}{\lambda_i}W_i^*(\lambda_i) + \frac{2}{\lambda_i} + \boldsymbol{E}[W_i] + \boldsymbol{E}[H_i] & \text{for BRPS, and} \\ -\frac{H_i^{*(1)}(\lambda_i)}{H_i^*(\lambda_i)} + H_i^*(\lambda_i)\frac{1}{\lambda_i}(1 - W_i^*(\lambda_i)) + \boldsymbol{E}[W_i] + \frac{1}{\lambda_i H_i^*(\lambda_i)} & \text{for CBPS-P.} \end{cases}$$

Similar to our discussion in Section 5.3, in the next theorem we show that for polling system with exponential service time, CBPS-P always has smaller PAoI than CBPS.

Theorem 5.4.1. *If the service time for each queue is exponentially distributed, then CBPS-P will always have smaller PAoI than CBPS.*

Proof. A detailed proof is shown in Appendix B.6. □

However, when the service time is not exponential, CBPS-P will not always have smaller PAoI than CBPS. We will show more computational results in Section 5.5.

5.5 Numerical Study: Verification, Findings and Explanations

In this section, we first perform a set of numerical experiments for systems with i.i.d. vacations that we introduced in Section 5.3, and then provide the numerical results to verify the exact solution of PAoI for polling systems. We then describe the results for polling system under different Markovian polling schemes.

5.5.1 CBS, BRS and CBS-P

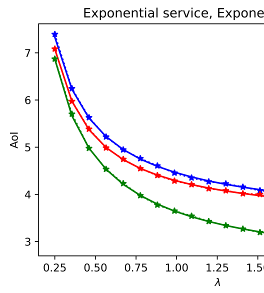
We begin our discussion by comparing the AoI, PAoI and variance of peak age for CBS, BRS, and CBS-P, as shown in Figure 5.6. In each subfigure of Figure 5.6 we plot simulation and exact results. As we observe from each subfigure of Figure 5.6, the simulation result matches the exact result for each system.

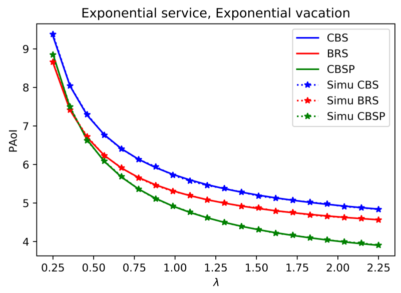
In Figure 5.6(a) and Figure 5.6(d) we compare the AoI for these three systems under different service and vacation time assumptions. As we see from Figure 5.6(a), CBS-P has advantage over the other two systems in terms of smaller AoI, when service time is exponentially distributed. However, in Figure 5.6(d) when service time is deterministic, CBS-P does not have smaller AoI than the other two systems when the arrival rate is large. This is because in CBS-P, the server would start a new packet when an arrival preempts the service. The server will continuously serve only until an inter-arrival time is smaller than the constant service time. If arrival rate is large (which means the expected inter-arrival time is small), then the probability of the inter-arrival time being smaller than the constant service time is small. Thus the AoI of CBS-P becomes large when arrival rate is large, for deterministic service time cases. In Section 5.5.2 we will see that this observation is mainly due to the constant service time and not the change of vacation distributions, since in the cases when the server does not take vacations, we observe similar phenomenons.

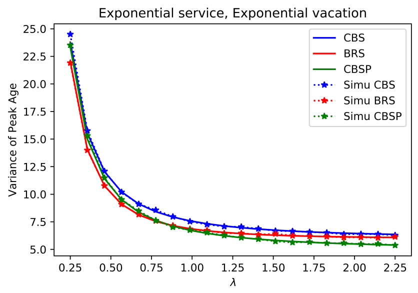
In Figure 5.6(b) and Figure 5.6(e) we compare the PAoI of these three systems. We find that CBS always has larger PAoI than BRS for both exponential and deterministic service cases, as we proved in Theorem 5.3.6. Also we find that when service time is deterministic, the PAoI of CBS-P will increase drastically as arrival rate increases. From Figure 5.6(a) and Figure 5.6(b) we also find that CBS-P has smaller AoI and PAoI than CBS for the exponential service cases, as we proved in Theorem 5.3.9. In Figure 5.6(c) and Figure 5.6(f) we compare the variance of peak age for these three systems. We find that when service time is exponential, the CBS as relatively larger variance than the other two systems, and when service time is deterministic, CBS-P will have large variance as arrival rate becomes large. From all the subfigures in Figure 5.6, we find that for both CBS and BRS, increasing the arrival rate would reduce the AoI, PAoI and variance of peak age.

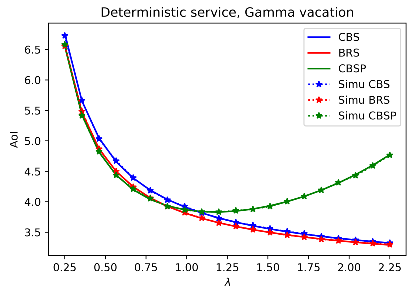
5.5.2 Systems with No Vacations

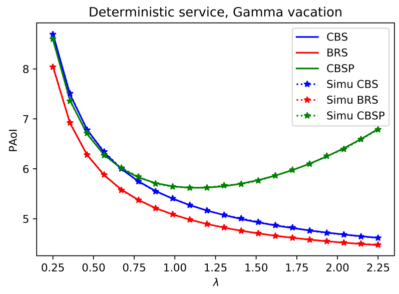
We next compare the AoI, PAoI and variance of peak age for M/G/1/1, M/G/1/2* and M/G/1/1/preemptive systems, under exponential and deterministic service cases. As we discussed in Section 5.3.4, when the vacation time becomes zero, CBS would become M/G/1/1 system, BRS would be M/G/1/2* system, and CBS-P now becomes M/G/1/1/preemptive system. The comparison results are shown in Figure 5.7. In each subfigure of Figure 5.7, we plot the exact result and simulation result for each system, and we find that the simulation result matches the exact result


(a) AoI Comparison, $H \sim exp(1)$,$V \sim exp(\frac{1}{2})$


(b) PAoI Comparison, $H \sim exp(1)$,$V \sim exp(\frac{1}{2})$


(c) Variance of Peak Age Comparison, $H = exp(1)$, $V \sim exp(\frac{1}{2})$


(d) AoI Comparison, $H = 1$, $V \sim gamma(2,1)$


(e) PAoI Comparison, $H = 1$, $V \sim gamma(2,1)$

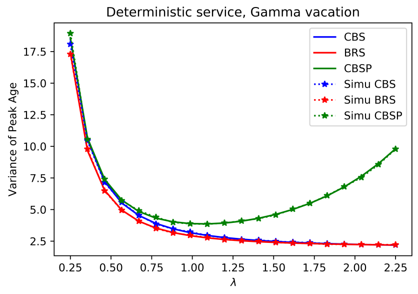
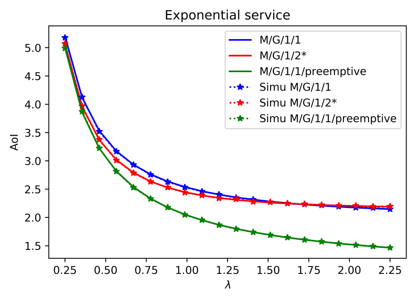
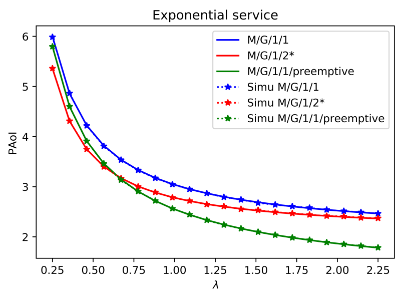

(f) Variance of Peak Age Comparison, $H = 1$, $V \sim gamma(2,1)$

Figure 5.6: Vacation Server Systems with $\boldsymbol{E}[H] = 1$ and $\boldsymbol{E}[V] = 2$

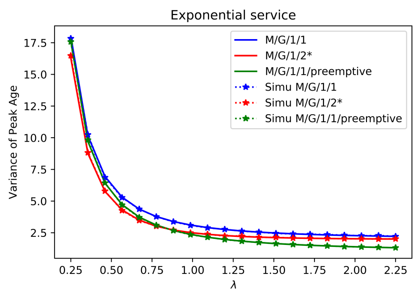
that we provided in Section 5.3.4 for each system. Similar to the vacation server cases, we find that when there is no vacation for the server, AoI, PAoI and variance of peak age will still decrease in M/G/1/1 and M/G/1/2* as arrival rate increases. For M/G/1/1/preemptive system, the AoI, PAoI and variance of peak age will increase dramatically when arrival rate becomes large, when the service time is deterministic. We also find that when service time is exponential, the variance of peak age in M/M/1/2* system is smaller than the variance in M/M/1/1 system, and the variance of peak age in M/M/1/1/preemptive system is smaller than the variance in M/M/1/1 system, as we showed in Section 5.3.4. When the service time is deterministic, M/D/1/2* system has lower variance of peak age than the other two systems.



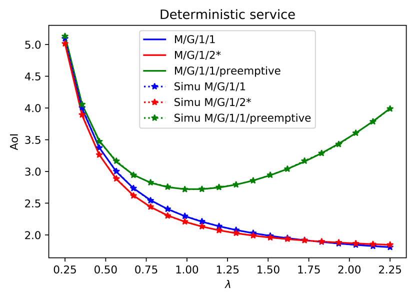
(a) AoI Comparison with $H \sim exp(1)$



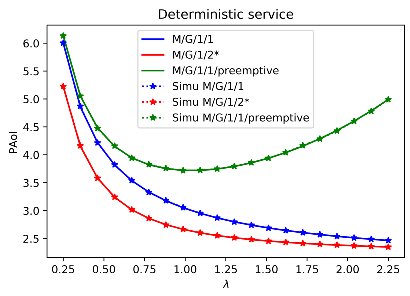
(b) PAoI Comparison with $H \sim exp(1)$



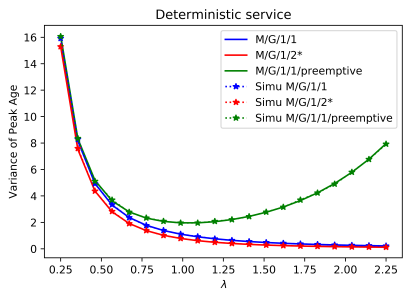
(c) Variance of Peak Age Comparison with $H \sim exp(1)$



(d) AoI Comparison with $H = 1$



(e) PAoI Comparison with $H = 1$

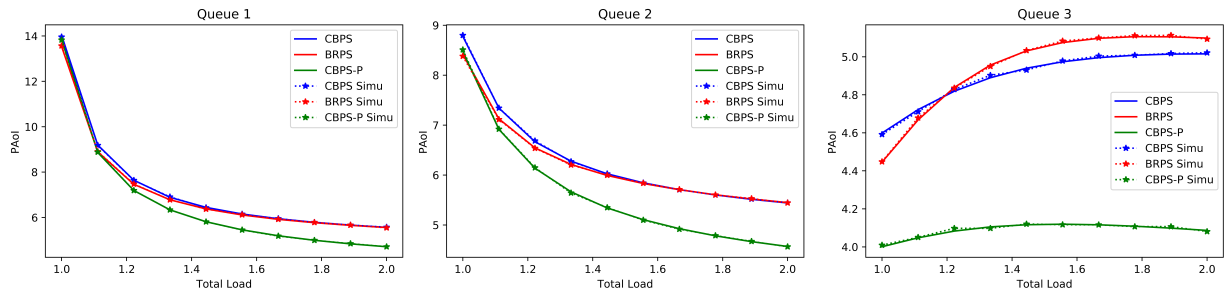


(f) Variance of Peak Age Comparison with $H = 1$

Figure 5.7: Single Queue System with $\boldsymbol{E}[H] = 1$

5.5.3 Polling Systems

We now perform numerical studies for different polling systems. In Figure 5.8 we compare the exact solutions of PAoI that we provided in Section 5.4 with the simulation results for the polling system with $k = 3$ and cyclic polling scheme. We find that the exact results match the simulation results from Figure 5.8. Interestingly, we find that increasing the traffic load will not always reduce the PAoI for CBPS, BRPS and CBPS-P. As we observed from Figure 5.8(c), when the traffic load increases, the PAoI of queue 3 in all three systems will increase. This is mainly because the numerical test of Figure 5.8 is based on the cyclic polling scheme, and increasing the traffic load for all queues will reduce the chance that the server observes an empty buffer when the queue is polled. For queue 3, the vacation time actually increases since the other queues are more likely to be served during the server's vacation. Although we know that increasing the traffic load will reduce the waiting time of the packet that is eventually processed (i.e., the server is more likely to find a fresh packet when vacation is over), the vacation time for queue 3 will also be affected as the server becomes more likely to serve queue 1 and queue 2 during vacation. Therefore, the increase in vacation time for queue 3 overshadows the reduction in G, so that we see in Figure 5.8 that the PAoI is increasing for queue 3 as total traffic load increases.



(a) PAoI of Queue 1 (b) PAoI of Queue 2 (c) PAoI of Queue 3

Figure 5.8: PAoI of Polling Systems with Cyclic Scheme, $\boldsymbol{\lambda} = (0.1, 0.2, 0.7) * Total\ Load$, $H_i = H \sim exp(1), U_i = U = 0.2$

The numerical study for a polling system with $k = 8$ and cyclic scheme is provided in Table 5.1. We choose the same system parameters as the numerical study in [158] by letting two queues be heavily loaded (each takes 45% of the total load). As we proved in Theorem 5.3.6, BRS always has smaller PAoI than CBS when the server's vacations are i.i.d. However, we observe in polling system that PAoI of BRPS is not always smaller than PAoI of CBPS. This is because in polling systems the vacations that the server takes are not i.i.d., as pointed out in [166]. Moreover, the vacation that the server takes also depends on the service time of the previous packet. If the service time of a packet is long, the other queues will become more likely to receive packets during this service time, resulting in a longer vacation time in the next cycle. The complexity of the vacation in polling system thus prevents Theorem 5.3.6 from holding true. However, from Table 5.1 we can see that it still holds true that PAoI in CBPS is larger than PAoI in CBPS-P when the service time is exponential, as we proved in Theorem 5.4.1.

Now we consider the PAoI of the polling system under different polling schemes that are described in Section 5.4. We keep the same set of parameters for service and switching time for Table 5.2 and 5.3, and provide the computational results for cyclic, LOP and symmetric random polling schemes with different total traffic loads. From both Table 5.2 and 5.3, we find that cyclic scheme and symmetric random scheme perform similarly when total traffic load is low. When traffic load is high, symmetric scheme provides lower PAoI for those queues with high arrival rates than cyclic scheme, and provides higher PAoI for other queues than cyclic scheme. From both Table 5.2 and 5.3 we find that LOP has lower PAoI than the other two polling schemes for queues with high arrival rates, especially when total traffic load is high. However LOP causes very large PAoI for queues with low arrival rates. This is because the server under LOP would serve queues with high arrival rates more frequently. Notice that in Theorem 5.4.1, we do not specify the polling scheme for CBPS or CBPS-P. So when service time is exponential, CBPS-P will always have smaller PAoI than CBPS, regardless of the polling scheme. This can also be observed from Table 5.2 and 5.3.

Next we consider the average PAoI across queues (i.e., $\sum_{i=1}^{k} \boldsymbol{E}[A_i]$) under those three different Markovian polling schemes, as shown in Figure 5.9. The average PAoI across queues was also

Queue	CBPS		BRPS		CBPS-P	
	PAoI	Simu	PAoI	Simu	PAoI	Simu
1	5.4396	5.4235	5.0996	5.1078	5.0688	5.0567
2	74.2941	75.7875	73.6306	73.9982	74.2684	74.1001
3	74.2984	74.6491	73.6372	74.9442	74.2726	72.9671
4	5.4386	5.4292	5.0985	5.1076	5.0677	5.0804
5	74.2897	73.3433	73.6236	75.2181	74.2639	74.6225
6	74.2938	73.2033	73.6300	74.3852	74.2680	75.7437
7	74.2980	75.8521	73.6366	74.2756	74.2723	75.8249
8	74.3024	75.7529	73.6433	73.2163	74.2766	73.6263

(a) Total load = 0.85

Queue	CBPS		BRPS		CBPS-P	
	PAoI	Simu	PAoI	Simu	PAoI	Simu
1	8.7298	8.7368	8.8934	8.8892	7.7298	7.7360
2	10.9433	10.9366	10.9663	10.9606	10.0502	10.0833
3	10.9513	10.9366	10.9697	10.9589	10.0584	10.0699
4	8.7296	8.7433	8.8935	8.8942	7.72963	7.7357
5	10.9352	10.9290	10.9630	10.9432	10.0419	10.0419
6	10.9426	10.9026	10.9662	10.9835	10.0494	10.0817
7	10.9509	10.9874	10.9698	10.9990	10.0578	10.0799
8	10.9601	10.9653	10.9735	10.9509	10.0672	10.0768

(b) Total load = 30

Table 5.1: Exact PAoI for the system with $k = 8$ and cyclic scheme. Queue 1 and 4 are heavily loaded: each with 45% total load. $H_i = H \sim exp(1), U_i = U = \frac{1}{80}$.

considered in [134, 4]. In Figure 5.9 we find that cyclic scheme achieves the lowest average PAoI under different traffic loads for both CBPS, BRPS and CBPS-P. LOP has the highest average PAoI among these three polling schemes. This is because under LOP, the server would serve the queues with high arrival rates more likely, and queues with low arrival rates would be polled infrequently. Since PAoI is more sensitive to the arrival rate change when arrival rate is small (which we can observe from Figure 5.6 and 5.7), the PAoI reduction in queues with high arrival rates would be easily overshadowed by the PAoI increase caused by queues with low arrival rates, when LOP is applied. This observation implies that if one wants to reduce the average PAoI for the entire system, a good strategy is to avoid polling certain queues too frequently. Therefore, policies with even polling frequency for queues such as symmetric or cyclic scheme are recommended for small average PAoI.

Queue	CBPS			BRPS			CBPS-P		
	Cyclic	LOP	Symmetric	Cyclic	LOP	Symmetric	Cyclic	LOP	Symmetric
1	7.0216	6.9340	7.1243	6.4694	6.3262	6.5428	6.7901	6.7137	6.8840
2	123.1109	125.6743	123.2638	122.2918	126.2261	122.6218	123.0980	125.5646	123.2504
3	123.1121	125.6743	123.2638	122.2935	126.2261	122.6218	123.0992	125.5646	123.2504
4	7.0212	6.9340	7.1243	6.4690	6.3262	6.5428	6.7897	6.7137	6.8840
5	123.1097	125.6743	123.2638	122.2900	126.2261	122.6218	123.0969	125.5646	123.2504
6	123.1108	125.6743	123.2638	122.2917	126.2261	122.6218	123.0980	125.5646	123.2504
7	123.1120	125.6743	123.2638	122.2933	126.2261	122.6218	123.0991	125.5646	123.2504
8	123.1131	125.6743	123.2638	122.2951	126.2261	122.6218	123.1003	125.5646	123.2504

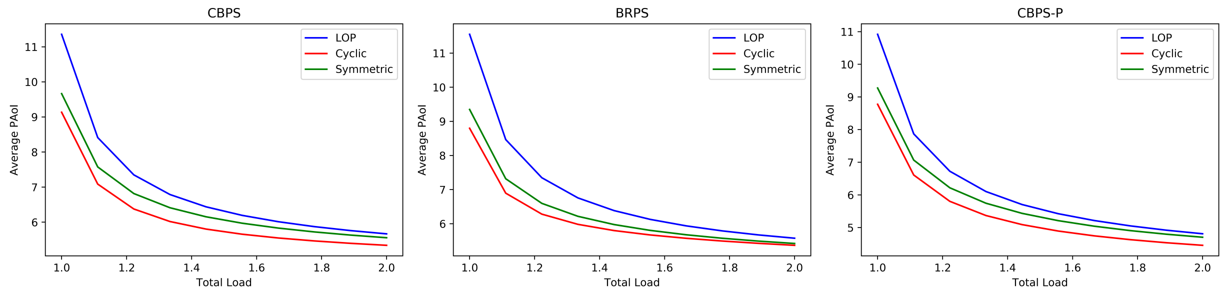
Table 5.2: Exact PAoI for the system with $k = 8$ and different polling schemes. Queue 1 and 4 are heavily loaded: each with 45% total load. Total load = 0.5. $H_i = H \sim exp(1), U_i = U = \frac{1}{80}$.

Queue	CBPS			BRPS			CBPS-P		
	Cyclic	LOP	Symmetric	Cyclic	LOP	Symmetric	Cyclic	LOP	Symmetric
1	8.0632	3.5189	6.9849	8.3780	3.3630	7.0477	7.0635	2.5353	5.9902
2	11.6450	42.6585	12.2968	11.6605	63.3207	12.3081	10.8810	41.7180	11.5688
3	11.6663	42.6585	12.2968	11.6715	63.3207	12.3081	10.9019	41.7180	11.5688
4	8.0620	3.5189	6.9849	8.3778	3.3630	7.0477	7.0622	2.5353	5.9902
5	11.6232	42.6585	12.2968	11.6493	63.3207	12.3081	10.8596	41.7180	11.5688
6	11.6413	42.6585	12.2968	11.6590	63.3207	12.3081	10.8773	41.7180	11.5688
7	11.6624	42.6585	12.2968	11.6700	63.3207	12.3081	10.8980	41.7180	11.5688
8	11.6870	42.6585	12.2968	11.6825	63.3207	12.3081	10.9221	41.7180	11.5688

Table 5.3: Exact PAoI for the system with $k = 8$ and different polling schemes. Queue 1 and 4 are heavily loaded: each with 45% total load. Total load = 20. $H_i = H \sim exp(1), U_i = U = \frac{1}{80}$.

5.6 Concluding Remarks

In this paper we considered AoI related metrics such as AoI, PAoI as well as variance of peak age in queueing systems with server vacations. We discussed cases with both i.i.d vacations and non-i.i.d vacations, and for non-i.i.d vacation systems we considered polling system specifically. We provided a general decomposition approach that decomposes the system age into independent components, which can be used to derive AoI, PAoI as well as the variance of peak age for i.i.d vacation systems, and PAoI for non-i.i.d vacation systems. In these systems with vacation servers, we discussed three system variations, i.e., CBS, BRS and CBS-P, which differs in assumption about buffer availability and service preemptions. We proved that when vacation time is i.i.d, PAoI



(a) CBPS (b) BRPS (c) CBPS-P

Figure 5.9: Average PAoI Across Queues, $\boldsymbol{\lambda} = (0.1, 0.2, 0.7) * Total\ Load$, $H_i = H \sim exp(1), U_i = U = 0.2$

in BRS is always smaller than PAoI in CBS. However, when vacation time is nom-i.i.d., this result is no longer true. We derived the conditions under which CBS-P system has smaller PAoI than CBS. We further provided numerical study to justify our findings, and discuss the advantage of each system in terms of AoI, PAoI or variance of peak age. In our future work, we will consider the closed-form of AoI for system with non-i.i.d vacations such as polling systems. We will also consider the optimal switching scheme and scheduling scheme for polling systems in the future.

6. CONCLUSION

6.1 Summary of Research

This dissertation research is motivated by smart manufacturing applications which need to solve optimization problems. Equipped with sensor networks, communication networks and computing entitles, smart manufacturing also brings research questions such a how to utilize available information in decision making, how to make decisions without future information, and how to guarantee information freshness. In this dissertation work, we addressed those research questions by considering several optimization problems that are or envisioned to be prevalent in smart manufacturing systems. The research objectives of this dissertation research include providing online scheduling strategies and system designs, and developing mathematical frameworks to evaluate the performance of online scheduling strategies and system designs.

In Chapter 2 we considered a joint production and maintenance decision making problem in a custom manufacturing system with degrading server. We solved this problem by providing several online scheduling policies that do not require the future information. To evaluate the performance of those online policies, we derived a benchmark using some special properties of the offline optimal solution, and then compared our online policies against the benchmark. We showed both analytically and numerically that our online policies perform closely to the optimal solution. We found that job re-sequencing is unnecessary when the replacement time is large compared with the workload. Besides, our analysis showed that revealing the workload upon a job's arrival is crucial for the efficiency of an online policy. We also used our analytic model to show that the more job information revealed, the better decisions the online policies can make.

In Chapter 3 we considered an online scheduling problem in polling systems. Different from other studies that analyzed polling system from a stochastic perspective, our analysis in this chapter does not rely on any assumptions about arrival processes or service processes. We compared the online policies with the offline optimal policy by considering their worst-case performance.

We then provided the competitive ratios for several widely-used policies in polling systems, and we showed that some policies that have a decent long-run average performance may also have a bounded competitive ratio. We also provided the conditions for existence of constant competitive ratios in polling systems, and we derived the competitive ratio lower bound for general scheduling policies in polling systems. In addition, several online policies that utilize the available information but also balance future uncertainties were provided in this research, and these policies can be applied to general polling systems without any stochastic assumptions.

It is important to point out that the problems studied in both Chapter 2 and Chapter 3 are mainly from the physical layer of smart manufacturing systems. When studying those problems, we assumed that each job reveals its workload or processing time upon its arrival into the system. In both chapters, we analyzed the problems from an online scheduling or a combinatorial optimization perspective. We evaluated the performance of online policies by comparing them with the offline optimal policy. The research work in these two chapters contributes to the methodologies in online optimization, combinatorial optimization, and scheduling theory.

We then studied the age performance of a system with single server and multiple data sources in Chapter 4. Motivated by the applications where some data sources are more time-sensitive thus requires higher priorities, we modeled the system as a priority queue system. In this research, we provided a novel modeling approach to derive closed-form expressions of PAoI for systems with different buffer capacity assumptions. Moreover, we evaluated the PAoI performance for service disciplines such as FCFS and LCFS. Although LCFS was believed to have advantage over other scheduling policies in minimizing PAoI, our analysis showed that LCFS is not the optimal service discipline, and FCFS would result in a smaller PAoI than LCFS when arrival rate is small. Also, it was believed that keeping the most recent packet in the buffer was effective in minimizing PAoI, however our analysis showed that this strategy does not always perform better than FCFS.

In Chapter 5 we consider the AoI and PAoI in a system where the server takes a vacation after serving a packet. This research is motivated by applications in smart manufacturing networks and other communication networks where the server becomes unavailable from time to time. In this research, we considered packet management strategies in buffers and provide closed-form expressions for AoI and PAoI for three systems with different packet management strategies: CBS, BRS and CBS-P. We also extended our work to systems with non-i.i.d vacations such as polling systems, and we used our analysis to exactly compute the PAoI for polling system. Our analysis explored the condition under which one packet management strategy can perform better than the other two in terms of either AoI or PAoI.

The problems studied in both Chapter 4 and Chapter 5 exist mainly in the cyber layer of smart manufacturing systems. We analyzed the problems in both chapters from a queueing or stochastic perspective, and evaluated the performance of each policy by its long-run average AoI or PAoI, which is completely different from our analysis in Chapter 2 and Chapter 3. The methodologies contributions in Chapter 4 and Chapter 5 are in probability theory, renewal processes and queueing theory.

The models and methodologies that are used in this dissertation research are summarized in Table 6.1. We also want to mention that polling systems are studied in both Chapter 3 and Chapter 5. However, we studied different metrics of polling systems in those two chapters. In Chapter 3 we focused on the total completion time of an arbitrary job instance that occurs in a general polling system, while in Chapter 5, we focused on the PAoI in polling systems. Moreover, the methodologies used in those two chapters are distinct, like we mentioned earlier. In Chapter 3 we considered the worst-case performance of online scheduling policies, and the analysis does not rely on stochastic assumptions. In Chapter 5, the analysis relies on the Poisson assumption about arrivals and IID assumptions about processing times. Interestingly, in both chapters, we found that some routing disciplines, such as cyclic routing discipline, being efficient in many scenarios on minimizing both completion time and PAoI. Both Chapter 3 and Chapter 5 analyzed polling systems from novel perspectives which were not found in the literature. This dissertation research therefore furthers knowledge and understanding about polling systems.

6.2 Contributions

We summarize the contributions of this dissertation research as follows.

Chapter	Type	Model	Methodologies
Chapter 2	Physical Layer	Degrading Server	Online optimization, combinatorial optimization, and scheduling theory
Chapter 3		Polling Systems	
Chapter 4	Cyber Layer	Priority Queues	Probability theory, renewal processes, and queueing theory.
Chapter 5		Vacation Server	

Table 6.1: Summary of Research

- We discussed the merits of smart manufacturing in a quantitative manner by modeling and analyzing specific scheduling problems that exist in both physical layer and cyber layer of smart manufacturing systems, and integrated elements and objectives including productivity, maintenance, service efficiency, and information availability in a united and collaborative framework.

- We provided online policies with decent performance that can be used in general manufacturing systems with degrading server. More importantly, we showed mathematically that the performance of online policies can be greatly improved by information availability and forecasting ability, which indicates the benefit of bringing in smartness and value of information in manufacturing systems. (Chapter 2)

- Our research took a non-traditional approach especially in the queueing and scheduling research communities by evaluating the worst-case performance for several widely-used scheduling policies in polling systems. We also provided a bound for competitive ratio for general online policies in polling systems. Several online scheduling policies that can be used in general polling settings were also proposed by our research and proved to have decent performance guarantee. (Chapter 3)

- We proposed a novel modeling approach to derive closed-form expressions of PAoI for queueing systems, and we used that to provide closed-form expressions of PAoI for several service disciplines in priority queue systems. Our analysis revealed that LCFS is not the optimal service discipline to minimize PAoI for each queue, and having at most one packet stored in the buffer is not always better than keeping all the packets, in terms of minimizing PAoI. Our work also showed that the PAoI for low priority queues are sensitive to the arrivals with high priority queues, thus it is important to let age-sensitive queues have high priorities. (Chapter 4)

- We proposed a decomposition approach to derive closed-form expressions for AoI, PAoI and variance of peak age for systems with server vacations. We showed that having the buffer available when service starts is not always optimal in terms of minimizing AoI, PAoI or variance of peak age. We also showed that preemption in service is not always better than non-preemption in terms in minimizing AoI, PAoI or variance of peak age. (Chapter 5)

6.3 Future Work

This dissertation work brings up many new research directions in smart manufacturing domain. In addition to the future work discussion at the end of each chapter of this dissertation work, here we provide some of other future work that is motivated by the entire dissertation work in the following:

- In most of current online scheduling literature, job information is assumed to be completely unknown until its arrival into a system. However, with the advent of smart manufacturing technologies, forecasts for job information become available and may contribute in improvement of online scheduling optimality. The quantitative relation between forecasting accuracy and online scheduling optimality remains unknown.

- AoI and PAoI are metrics to measure information freshness. It is believed that information freshness will affect the online decision quality, however the model to quantify the relation between information freshness and online scheduling optimality has not been studied.

REFERENCES

[1] L. Monostori, B. Kádár, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart, O. Sauer, G. Schuh, W. Sihn, and K. Ueda, "Cyber-physical systems in manufacturing," *CIRP Annals*, vol. 65, no. 2, pp. 621–641, 2016.

[2] J. Xu, H. M. Tran, N. Gautam, and S. T. Bukkapatnam, "Joint production and maintenance operations in smart custom-manufacturing systems," *IISE Transactions*, vol. 51, no. 4, pp. 406–421, 2019.

[3] J. Xu, I.-H. Hou, and N. Gautam, "Age of information for single buffer systems with vacation server," *arXiv preprint arXiv:2004.11847*, 2020.

[4] J. Xu and N. Gautam, "Peak age of information in priority queueing systems," *arXiv preprint arXiv:1906.12278*, 2019.

[5] J. Xu and N. Gautam, "On Competitive Analysis for Polling Systems," *arXiv preprint arXiv: 2001.02530*, 2020.

[6] M. K. Thompson, G. Moroni, T. Vaneker, G. Fadel, R. I. Campbell, I. Gibson, A. Bernard, J. Schulz, P. Graf, B. Ahuja, *et al.*, "Design for additive manufacturing: Trends, opportunities, considerations, and constraints," *CIRP annals*, vol. 65, no. 2, pp. 737–760, 2016.

[7] R. Gao, L. Wang, R. Teti, D. Dornfeld, S. Kumara, M. Mori, and M. Helu, "Cloud-enabled prognosis for manufacturing," *CIRP annals*, vol. 64, no. 2, pp. 749–772, 2015.

[8] T. Takenaka, Y. Yamamoto, K. Fukuda, A. Kimura, and K. Ueda, "Enhancing products and services using smart appliance networks," *CIRP Annals*, vol. 65, no. 1, pp. 397–400, 2016.

[9] E. Uhlmann, B. Mullany, D. Biermann, K. Rajurkar, T. Hausotte, and E. Brinksmeier, "Process chains for high-precision components with micro-scale features," *CIRP Annals - Manufacturing Technology*, vol. 65, no. 2, pp. 549–572, 2016.

[10] C. Conrad and N. McClamroch, “The drilling problem: A stochastic modeling and control example in manufacturing,” *IEEE Transactions on automatic control*, vol. 32, no. 11, pp. 947–958, 1987.

[11] G. Niu, B.-S. Yang, and M. Pecht, “Development of an optimized condition-based maintenance system by data fusion and reliability-centered maintenance,” *Reliability Engineering and System Safety*, vol. 95, no. 7, pp. 786–796, 2010.

[12] K. Liu and J. Shi, “Internet of things (iot)-enabled system informatics for service decision making: Achievements, trends, challenges, and opportunities,” *IEEE Intelligent Systems*, vol. 30, no. 6, pp. 18–21, 2015.

[13] F. Martinelli, “Optimality of a two-threshold feedback control for a manufacturing system with a production dependent failure rate,” *IEEE Transactions on Automatic Control*, vol. 52, no. 10, pp. 1937–1942, 2007.

[14] J.-Q. Hu, P. Vakili, and G.-X. Yu, “Optimality of hedging point policies in the production control of failure prone manufacturing systems,” *IEEE Transactions on Automatic Control*, vol. 39, no. 9, pp. 1875–1880, 1994.

[15] N. Srivatsan and Y. Dallery, “Partial characterization of optimal hedging point policies in unreliable two-part-type manufacturing systems,” *Operations Research*, vol. 46, no. 1, pp. 36–45, 1998.

[16] R. Akella and P. Kumar, “Optimal control of production rate in a failure prone manufacturing system,” *IEEE Transactions on Automatic control*, vol. 31, no. 2, pp. 116–126, 1986.

[17] A. Sharifnia, “Production control of a manufacturing system with multiple machine states,” *IEEE Transactions on Automatic Control*, vol. 33, no. 7, pp. 620–625, 1988.

[18] D.-P. Song, “Optimal production and backordering policy in failure-prone manufacturing systems,” *IEEE transactions on automatic control*, vol. 51, no. 5, pp. 906–911, 2006.

[19] T. Cheng, C. Gao, and H. Shen, "Production and inventory rationing in a make-to-stock system with a failure-prone machine and lost sales," *IEEE transactions on automatic control*, vol. 56, no. 5, pp. 1176–1180, 2011.

[20] Z. Pang, "Optimal control of a single-product assemble-to-order system with multiple demand classes and backordering," *IEEE Transactions on Automatic Control*, vol. 60, no. 2, pp. 480–484, 2015.

[21] Y. Feng and B. Xiao, "Optimal threshold control in discrete failure-prone manufacturing systems," *IEEE Transactions on Automatic Control*, vol. 47, no. 7, pp. 1167–1174, 2002.

[22] E.-K. Boukas, Q. Zhang, and G. Yin, "Robust production and maintenance planning in stochastic manufacturing systems," *IEEE Transactions on Automatic Control*, vol. 40, no. 6, pp. 1098–1102, 1995.

[23] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein, "Scheduling to minimize average completion time: Off-line and on-line approximation algorithms," *Mathematics of operations research*, vol. 22, no. 3, pp. 513–544, 1997.

[24] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein, "Approximation techniques for average completion time scheduling," *SIAM Journal on Computing*, vol. 31, no. 1, pp. 146–166, 2001.

[25] N. Gans and G. Van Ryzin, "Optimal control of a multiclass, flexible queueing system," *Operations Research*, vol. 45, no. 5, pp. 677–693, 1997.

[26] S. V. Sevastianov and G. J. Woeginger, "Makespan minimization in open shops: A polynomial time approximation scheme," *Mathematical Programming*, vol. 82, no. 1, pp. 191–198, 1998.

[27] L. A. Hall, "Approximability of flow shop scheduling," *Mathematical Programming*, vol. 82, no. 1-2, pp. 175–190, 1998.

[28] G. Rabadi, R. J. Moraga, and A. Al-Salem, “Heuristics for the unrelated parallel machine scheduling problem with setup times,” *Journal of Intelligent Manufacturing*, vol. 17, no. 1, pp. 85–97, 2006.

[29] C.-Y. Lee and C.-S. Lin, “Single-machine scheduling with maintenance and repair rate-modifying activities,” *European Journal of Operational Research*, vol. 135, no. 3, pp. 493–513, 2001.

[30] C.-Y. Lee and V. J. Leon, “Machine scheduling with a rate-modifying activity,” *European Journal of Operational Research*, vol. 128, no. 1, pp. 119–128, 2001.

[31] X. Cai, X. Wu, and X. Zhou, “Stochastic scheduling subject to preemptive-repeat breakdowns with incomplete information,” *Operations Research*, vol. 57, no. 5, pp. 1236–1249, 2009.

[32] Y. Feng and H. Yan, “Optimal production control in a discrete manufacturing system with unreliable machines and random demands,” *IEEE Transactions on Automatic Control*, vol. 45, no. 12, pp. 2280–2296, 2000.

[33] S. Sana, S. K. Goyal, and K. Chaudhuri, “A production–inventory model for a deteriorating item with trended demand and shortages,” *European Journal of Operational Research*, vol. 157, no. 2, pp. 357–371, 2004.

[34] P. J. Schweitzer and A. Seidmann, “Optimizing processing rates for flexible manufacturing systems,” *Management Science*, vol. 37, no. 4, pp. 454–466, 1991.

[35] R. Rishel, “Controlled wear process: Modeling optimal control,” *IEEE Transactions on Automatic Control*, vol. 36, no. 9, pp. 1100–1102, 1991.

[36] D. A. Stephenson and J. S. Agapiou, *Metal Cutting Theory and Practice, Third Edition*. CRC Press, April 2016.

[37] W. Xu and L. Cao, “Optimal tool replacement with product quality deterioration and random tool failure,” *International Journal of Production Research*, vol. 53, no. 6, 2015.

[38] B. Korte, J. Vygen, B. Korte, and J. Vygen, *Combinatorial optimization*, vol. 2. Springer, 2012.

[39] I. Griva, S. G. Nash, and A. Sofer, *Linear and Nonlinear Programming: Second Edition*. Society for Industrial and Applied Mathematics, January 2009.

[40] N. Karmarkar, "Probabilistic analysis of some bin-packing problems," in *23rd Annual Symposium on Foundations of Computer Science, IEEE SFCS'08.*, pp. 107–111, 1982.

[41] E. Coffman, K. So, M. Hofri, and A. Yao., "A stochastic model of bin-packing," *Information and Control*, vol. 44, pp. 105–115, February 1980.

[42] P. Bonami, "Bonmin users' manual," 2007.

[43] H. Levy and M. Sidi, "Polling systems: applications, modeling, and optimization," *IEEE Transactions on Communications*, vol. 38, no. 10, pp. 1750–1760, 1990.

[44] R. D. van der Mei and A. Roubos, "Polling models with multi-phase gated service," *Annals of Operations Research*, vol. 198, no. 1, pp. 25–56, 2012.

[45] M. A. Boon, I. J. Adan, E. M. Winands, and D. Down, "Delays at signalized intersections with exhaustive traffic control," *Probability in the Engineering and Informational Sciences*, vol. 26, no. 3, pp. 337–373, 2012.

[46] D. Miculescu and S. Karaman, "Polling-systems-based autonomous vehicle coordination in traffic intersections with no traffic signals," *IEEE Transactions on Automatic Control*, 2019.

[47] M. A. Boon, R. Van der Mei, and E. M. Winands, "Applications of polling systems," *Surveys in Operations Research and Management Science*, vol. 16, no. 2, pp. 67–82, 2011.

[48] V. Vishnevskii and O. Semenova, "Mathematical methods to study the polling systems," *Automation and Remote Control*, vol. 67, no. 2, pp. 173–220, 2006.

[49] H. Takagi, "Queuing analysis of polling models," *ACM Computing Surveys (CSUR)*, vol. 20, no. 1, pp. 5–28, 1988.

[50] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan, “Optimization and approximation in deterministic sequencing and scheduling: a survey,” in *Annals of discrete mathematics*, vol. 5, pp. 287–326, Elsevier, 1979.

[51] J. K. Lenstra, D. B. Shmoys, and E. Tardos, “Approximation algorithms for scheduling unrelated parallel machines,” *Mathematical programming*, vol. 46, no. 1-3, pp. 259–271, 1990.

[52] E. L. Lawler, J. K. Lenstra, A. H. R. Kan, and D. B. Shmoys, “Sequencing and scheduling: Algorithms and complexity,” *Handbooks in operations research and management science*, vol. 4, pp. 445–522, 1993.

[53] A. R. Kan, *Machine scheduling problems: classification, complexity and computations*. Springer Science & Business Media, 2012.

[54] D. R. Smith, “A new proof of the optimality of the shortest remaining processing time discipline,” *Operations Research*, vol. 26, no. 1, pp. 197–199, 1978.

[55] A. Wierman and B. Zwart, “Is tail-optimal scheduling possible?,” *Operations research*, vol. 60, no. 5, pp. 1249–1257, 2012.

[56] N. Bansal, B. Kamphorst, and B. Zwart, “Achievable performance of blind policies in heavy traffic,” *Mathematics of Operations Research*, 2018.

[57] L. Stougie and A. P. Vestjens, “Randomized algorithms for on-line scheduling problems: how low can’t you go?,” *Operations Research Letters*, vol. 30, no. 2, pp. 89–96, 2002.

[58] E. Altman, A. Khamisy, and U. Yechiali, “On elevator polling with globally gated regime,” *Queueing Systems*, vol. 11, no. 1, pp. 85–90, 1992.

[59] A. Allahverdi, C. Ng, T. E. Cheng, and M. Y. Kovalyov, “A survey of scheduling problems with setup times or costs,” *European journal of operational research*, vol. 187, no. 3, pp. 985–1032, 2008.

[60] A. Allahverdi, “The third comprehensive survey on scheduling problems with setup times/costs,” *European Journal of Operational Research*, vol. 246, no. 2, pp. 345–378, 2015.

[61] A. Allahverdi and H. Soroush, “The significance of reducing setup times/setup costs,” *European Journal of Operational Research*, vol. 187, no. 3, pp. 978–984, 2008.

[62] R. Ruiz and J. A. Vázquez-Rodríguez, “The hybrid flow shop scheduling problem,” *European Journal of Operational Research*, vol. 205, no. 1, pp. 1–18, 2010.

[63] E. Vallada and R. Ruiz, “A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times,” *European Journal of Operational Research*, vol. 211, no. 3, pp. 612–622, 2011.

[64] O. Hinder and A. J. Mason, “A novel integer programing formulation for scheduling with family setup times on a single machine to minimize maximum lateness,” *European Journal of Operational Research*, vol. 262, no. 2, pp. 411–423, 2017.

[65] C. Ng, T. E. Cheng, J. Yuan, and Z. Liu, “On the single machine serial batching scheduling problem to minimize total completion time with precedence constraints, release dates and identical processing times,” *Operations Research Letters*, vol. 31, no. 4, pp. 323–326, 2003.

[66] G. Mosheiov, D. Oron, and Y. Ritov, “Minimizing flow-time on a single machine with integer batch sizes,” *Operations Research Letters*, vol. 33, no. 5, pp. 497–501, 2005.

[67] L. Shen, S. Dauzère-Pérès, and J. S. Neufeld, “Solving the flexible job shop scheduling problem with sequence-dependent setup times,” *European Journal of Operational Research*, vol. 265, no. 2, pp. 503–516, 2018.

[68] E. Lübbecke, O. Maurer, N. Megow, and A. Wiese, “A new approach to online scheduling: Approximating the optimal competitive ratio,” *ACM Transactions on Algorithms (TALG)*, vol. 13, no. 1, p. 15, 2016.

[69] L. Epstein and R. van Stee, “Lower bounds for on-line single-machine scheduling,” *Theoretical Computer Science*, vol. 299, no. 1, pp. 439–450, 2003.

[70] R. Sitters, “Competitive analysis of preemptive single-machine scheduling,” *Operations Research Letters*, vol. 38, no. 6, pp. 585–588, 2010.

[71] A. S. Schulz and M. Skutella, “The power of α-points in preemptive single machine scheduling,” *Journal of Scheduling*, vol. 5, no. 2, pp. 121–133, 2002.

[72] J. A. Hoogeveen and A. P. Vestjens, “Optimal on-line algorithms for single-machine scheduling,” in *International Conference on Integer Programming and Combinatorial Optimization*, pp. 404–414, Springer, 1996.

[73] X. Lu, R. Sitters, and L. Stougie, “A class of on-line scheduling algorithms to minimize total completion time,” *Operations Research Letters*, vol. 31, no. 3, pp. 232–236, 2003.

[74] C. Phillips, C. Stein, and J. Wein, “Minimizing average completion time in the presence of release dates,” *Mathematical Programming*, vol. 82, no. 1-2, pp. 199–223, 1998.

[75] E. J. Anderson and C. N. Potts, “Online scheduling of a single machine to minimize total weighted completion time,” *Mathematics of Operations Research*, vol. 29, no. 3, pp. 686–697, 2004.

[76] M. X. Goemans, M. Queyranne, A. S. Schulz, M. Skutella, and Y. Wang, “Single machine scheduling with release dates,” *SIAM Journal on Discrete Mathematics*, vol. 15, no. 2, pp. 165–192, 2002.

[77] J. Tao, Z. Chao, Y. Xi, and Y. Tao, “An optimal semi-online algorithm for a single machine scheduling problem with bounded processing time,” *Information Processing Letters*, vol. 110, no. 8-9, pp. 325–330, 2010.

[78] J. Tao, Z. Chao, and Y. Xi, “A semi-online algorithm and its competitive analysis for a single machine scheduling problem with bounded processing times,” *Journal of Industrial and Management Optimization*, vol. 6, no. 2, pp. 269–282, 2010.

[79] S. Divakaran and M. Saks, “An online algorithm for a problem in scheduling with set-ups and release times,” *Algorithmica*, vol. 60, no. 2, pp. 301–315, 2011.

[80] L. Zhang and A. Wirth, "Online machine scheduling with family setups," *Asia-Pacific Journal of Operational Research*, vol. 33, no. 04, p. 1650027, 2016.

[81] M. Ferguson and Y. Aminetzah, "Exact results for nonsymmetric token ring systems," *IEEE Transactions on Communications*, vol. 33, no. 3, pp. 223–231, 1985.

[82] D. Sarkar and W. Zangwill, "Expected waiting time for nonsymmetric cyclic queueing systems exact results and applications," *Management Science*, vol. 35, no. 12, pp. 1463–1474, 1989.

[83] E. M. Winands, I. J.-B. F. Adan, and G.-J. van Houtum, "Mean value analysis for polling systems," *Queueing Systems*, vol. 54, no. 1, pp. 35–44, 2006.

[84] M. Van Vuuren and E. M. Winands, "Iterative approximation of k-limited polling systems," *Queueing Systems*, vol. 55, no. 3, pp. 161–178, 2007.

[85] N. Gautam, *Analysis of queues: methods and applications*. CRC Press, 2012.

[86] A. Wierman, E. M. Winands, and O. J. Boxma, "Scheduling in polling systems," *Performance Evaluation*, vol. 64, no. 9, pp. 1009–1028, 2007.

[87] Z. Liu, P. Nain, and D. Towsley, "On optimal polling policies," *Queueing Systems*, vol. 11, no. 1-2, pp. 59–83, 1992.

[88] J. Gittins, K. Glazebrook, and R. Weber, *Multi-armed bandit allocation indices*. John Wiley & Sons, 2011.

[89] J. Baker and I. Rubin, "Polling with a general-service order table," *IEEE Transactions on Communications*, vol. 35, no. 3, pp. 283–288, 1987.

[90] L. Kleinrock and H. Levy, "The analysis of random polling systems," *Operations Research*, vol. 36, no. 5, pp. 716–732, 1988.

[91] J. N. Tsitsiklis, "A short proof of the Gittins index theorem," *The Annals of Applied Probability*, vol. 4, no. 1, pp. 194–199, 1994.

[92] S. Aalto, U. Ayesta, and R. Righter, “On the gittins index in the M/G/1 queue,” *Queueing Systems*, vol. 63, no. 1-4, p. 437, 2009.

[93] C. L. Monma and C. N. Potts, “On the complexity of scheduling with batch setup times,” *Operations research*, vol. 37, no. 5, pp. 798–804, 1989.

[94] J. B. Ghosh, “Batch scheduling to minimize total completion time,” *Operations Research Letters*, vol. 16, no. 5, pp. 271–275, 1994.

[95] S. Divakaran and M. Saks, “Approximation algorithms for problems in scheduling with set-ups,” *Discrete Applied Mathematics*, vol. 156, no. 5, pp. 719–729, 2008.

[96] M. Queyranne, “Structure of a simple scheduling polyhedron,” *Mathematical Programming*, vol. 58, no. 1, pp. 263–285, 1993.

[97] J. Du, J. Y.-T. Leung, and G. H. Young, “Minimizing mean flow time with release time constraint,” *Theoretical Computer Science*, vol. 75, no. 3, pp. 347–355, 1990.

[98] M. G. Mehrabi, A. G. Ulsoy, and Y. Koren, “Reconfigurable manufacturing systems: Key to future manufacturing,” *Journal of Intelligent manufacturing*, vol. 11, no. 4, pp. 403–419, 2000.

[99] S. Kaul, R. Yates, and M. Gruteser, “Real-time status: How often should one update?,” in *INFOCOM, 2012 Proceedings IEEE*, pp. 2731–2735, IEEE, 2012.

[100] L. Huang and E. Modiano, “Optimizing age-of-information in a multi-class queueing system,” in *2015 IEEE International Symposium on Information Theory (ISIT)*, pp. 1681–1685, IEEE, 2015.

[101] Y. Inoue, H. Masuyama, T. Takine, and T. Tanaka, “A general formula for the stationary distribution of the age of information and its application to single-server queues,” *IEEE Transactions on Information Theory*, vol. 65, no. 12, pp. 8305–8324, 2019.

[102] E. Masry, “Poisson sampling and spectral estimation of continuous-time processes,” *IEEE Transactions on Information Theory*, vol. 24, no. 2, pp. 173–183, 1978.

[103] E. Najm, R. Nasser, and E. Telatar, “Content based status updates,” *IEEE Transactions on Information Theory*, 2019.

[104] A. Maatouk, M. Assaad, and A. Ephremides, “Age of information with prioritized streams: When to buffer preempted packets?,” in *2019 IEEE International Symposium on Information Theory (ISIT)*, pp. 325–329, IEEE, 2019.

[105] M. Costa, M. Codreanu, and A. Ephremides, “On the age of information in status update systems with packet management,” *IEEE Transactions on Information Theory*, vol. 62, no. 4, pp. 1897–1910, 2016.

[106] D. Theodoratos and M. Bouzeghoub, “Data currency quality factors in data warehouse design.,” in *DMDW*, p. 15, 1999.

[107] M. Bouzeghoub, “A framework for analysis of data freshness,” in *Proceedings of the 2004 international workshop on Information quality in information systems*, pp. 59–67, ACM, 2004.

[108] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, “Hedera: dynamic flow scheduling for data center networks.,” in *Nsdi*, vol. 10, pp. 89–92, 2010.

[109] L. M. Vaquero and L. Rodero-Merino, “Finding your way in the fog: Towards a comprehensive definition of fog computing,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 27–32, 2014.

[110] E. Najm and R. Nasser, “Age of information: The gamma awakening,” in *2016 IEEE International Symposium on Information Theory (ISIT)*, pp. 2574–2578, IEEE, 2016.

[111] A. Soysal and S. Ulukus, “Age of information in G/G/1/1 systems,” *arXiv preprint arXiv:1805.12586*, 2018.

[112] P. Zou, O. Ozel, and S. Subramaniam, “Waiting before serving: A companion to packet management in status update systems,” *IEEE Transactions on Information Theory*, 2019.

[113] A. Kosta, N. Pappas, A. Ephremides, and V. Angelakis, "Queue management for age sensitive status updates," in *2019 IEEE International Symposium on Information Theory (ISIT)*, pp. 330–334, IEEE, 2019.

[114] E. Najm and E. Telatar, "Status updates in a multi-stream M/G/1/1 preemptive queue," in *IEEE Infocom 2018-IEEE Conference On Computer Communications Workshops (Infocom Wkshps)*, pp. 124–129, IEEE, 2018.

[115] A. Kosta, N. Pappas, A. Ephremides, and V. Angelakis, "Age of information performance of multiaccess strategies with packet management," *Journal of Communications and Networks*, vol. 21, no. 3, pp. 244–255, 2019.

[116] Z. Jiang, B. Krishnamachari, S. Zhou, and Z. Niu, "Can decentralized status update achieve universally near-optimal age-of-information in wireless multiaccess channels?," in *2018 30th International Teletraffic Congress (ITC 30)*, vol. 1, pp. 144–152, IEEE, 2018.

[117] A. Maatouk, S. Kriouile, M. Assaad, and A. Ephremides, "On the optimality of the whittle's index policy for minimizing the age of information," *arXiv preprint arXiv:2001.03096*, 2020.

[118] I. Kadota, A. Sinha, and E. Modiano, "Scheduling algorithms for optimizing age of information in wireless networks with throughput constraints," *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1359–1372, 2019.

[119] R. Talak, S. Karaman, and E. Modiano, "Optimizing information freshness in wireless networks under general interference constraints," *IEEE/ACM Transactions on Networking*, 2019.

[120] Q. He, D. Yuan, and A. Ephremides, "Optimal link scheduling for age minimization in wireless systems," *IEEE Transactions on Information Theory*, vol. 64, no. 7, pp. 5381–5394, 2017.

[121] Y.-P. Hsu, E. Modiano, and L. Duan, “Age of information: Design and analysis of optimal scheduling algorithms,” in *2017 IEEE International Symposium on Information Theory (ISIT)*, pp. 561–565, IEEE, 2017.

[122] Z. Jiang, B. Krishnamachari, X. Zheng, S. Zhou, and Z. Niu, “Timely status update in massive iot systems: Decentralized scheduling for wireless uplinks,” *arXiv preprint arXiv:1801.03975*, 2018.

[123] Z. Jiang, B. Krishnamachari, X. Zheng, S. Zhou, and Z. Niu, “Timely status update in wireless uplinks: Analytical solutions with asymptotic optimality,” *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3885–3898, 2019.

[124] I. Kadota, A. Sinha, E. Uysal-Biyikoglu, R. Singh, and E. Modiano, “Scheduling policies for minimizing age of information in broadcast wireless networks,” *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2637–2650, 2018.

[125] R. D. Yates and S. K. Kaul, “The age of information: Real-time status updating by multiple sources,” *IEEE Transactions on Information Theory*, vol. 65, no. 3, pp. 1807–1827, 2019.

[126] N. K. Jaiswal, *Priority queues*. Elsevier, 1968.

[127] I. Adan, O. J. Boxma, and J. A. C. Resing, “Queueing models with multiple waiting lines,” *Queueing Systems*, vol. 37, no. 1-3, pp. 65–98, 2001.

[128] S. K. Kaul and R. D. Yates, “Age of information: Updates with priority,” in *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 2644–2648, IEEE, 2018.

[129] V. G. Kulkarni, *Modeling and analysis of stochastic systems*. Chapman and Hall/CRC, 2016.

[130] R. W. Conway, W. L. Maxwell, and L. W. Miller, *Theory of scheduling*. Courier Corporation, 2003.

[131] T. Takenaka, “Analysis of a nonpreemptive $\Sigma M_i/G/1(\Sigma N_i)$ system,” *Electronics and Communications in Japan (Part I: Communications)*, vol. 72, no. 3, pp. 75–84, 1989.

[132] T. Takenaka, "Buffer management schemes for a heterogeneous packet switching system," *Electronics and Communications in Japan (Part I: Communications)*, vol. 67, no. 11, pp. 46–54, 1984.

[133] T. Takenaka, T. Akaike, and K. Takami, "Characteristics and approximation methods of a nonpreemptive $\Sigma M_i/G/1(\Sigma N_i)$ system," *Electronics and Communications in Japan (Part I: Communications)*, vol. 72, no. 3, pp. 85–94, 1989.

[134] A. M. Bedewy, Y. Sun, S. Kompella, and N. B. Shroff, "Age-optimal sampling and transmission scheduling in multi-source systems," *arXiv preprint arXiv:1812.09463*, 2018.

[135] O. Kella and U. Yechiali, "Priorities in M/G/1 queue with server vacations," *Naval Research Logistics (NRL)*, vol. 35, no. 1, pp. 23–34, 1988.

[136] E. W. Cheney and D. R. Kincaid, *Numerical mathematics and computing*. Cengage Learning, 2012.

[137] A. M. Bedewy, Y. Sun, and N. B. Shroff, "The age of information in multihop networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 1248–1257, 2019.

[138] J. Zhong, R. D. Yates, and E. Soljanin, "Two freshness metrics for local cache refresh," in *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 1924–1928, IEEE, 2018.

[139] C. Song, K. Liu, and X. Zhang, "A generic framework for multisensor degradation modeling based on supervised classification and failure surface," *IISE Transactions*, vol. 51, no. 11, pp. 1288–1302, 2019.

[140] Y. Cheng and M. A. Jafari, "Vision-based online process control in manufacturing applications," *IEEE Transactions on Automation Science and Engineering*, vol. 5, no. 1, pp. 140–153, 2008.

[141] B. Yao and H. Yang, "Constrained markov decision process modeling for sequential optimization of additive manufacturing build quality," *IEEE Access*, vol. 6, pp. 54786–54794, 2018.

[142] X. Guo, Z. Niu, S. Zhou, and P. Kumar, "Delay-constrained energy-optimal base station sleeping control," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 5, pp. 1073–1085, 2016.

[143] J. Wu, S. Zhou, and Z. Niu, "Traffic-aware base station sleeping control and power matching for energy-delay tradeoffs in green cellular networks," *IEEE Transactions on Wireless Communications*, vol. 12, no. 8, pp. 4196–4209, 2013.

[144] A. K. Mohapatra, N. Gautam, and R. L. Gibson, "Combined routing and node replacement in energy-efficient underwater sensor networks for seismic monitoring," *IEEE Journal of Oceanic Engineering*, vol. 38, no. 1, pp. 80–90, 2012.

[145] J. Heidemann, W. Ye, J. Wills, A. Syed, and Y. Li, "Research challenges and applications for underwater sensor networking," in *IEEE Wireless Communications and Networking Conference, 2006. WCNC 2006.*, vol. 1, pp. 228–235, IEEE, 2006.

[146] J. Heidemann, M. Stojanovic, and M. Zorzi, "Underwater sensor networks: applications, advances and challenges," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 370, no. 1958, pp. 158–175, 2012.

[147] I. Vasilescu, K. Kotay, D. Rus, M. Dunbabin, and P. Corke, "Data collection, storage, and retrieval with an underwater sensor network," in *Proceedings of the 3rd international conference on Embedded networked sensor systems*, pp. 154–165, 2005.

[148] C. Doukas and I. Maglogiannis, "Managing wearable sensor data through cloud computing," in *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pp. 440–445, IEEE, 2011.

[149] S. Majumder, T. Mondal, and M. J. Deen, "Wearable sensors for remote health monitoring," *Sensors*, vol. 17, no. 1, p. 130, 2017.

[150] H. Takagi, "Analysis of finite-capacity polling systems," *Advances in Applied Probability*, vol. 23, no. 2, pp. 373–387, 1991.

[151] S. W. Fuhrmann and R. B. Cooper, "Stochastic decompositions in the M/G/1 queue with generalized vacations," *Operations research*, vol. 33, no. 5, pp. 1117–1129, 1985.

[152] V. J. Maccio and D. G. Down, "On optimal policies for energy-aware servers," *Performance Evaluation*, vol. 90, pp. 36–52, 2015.

[153] T. T. Lee, "M/G/1/N queue with vacation time and limited service discipline," *Performance Evaluation*, vol. 9, no. 3, pp. 181–190, 1989.

[154] T. T. Lee, "M/G/1/N queue with vacation time and exhaustive service discipline," *Operations Research*, vol. 32, no. 4, pp. 774–784, 1984.

[155] S. Fuhrmann, "A note on the M/G/1 queue with server vacations," *Operations research*, vol. 32, no. 6, pp. 1368–1373, 1984.

[156] A. Frey and Y. Takahashi, "A note on an M/GI/1/N queue with vacation time and exhaustive service discipline," *Operations Research lLetters*, vol. 21, no. 2, pp. 95–100, 1997.

[157] T. Y. S. Lee and J. Sunjaya, "Exact analysis of asymmetric random polling systems with single buffers and correlated input process," *Queueing Systems*, vol. 23, no. 1-4, pp. 131–156, 1996.

[158] T. Takine, Y. Takahashi, and T. Hasegawa, "Exact analysis of asymmetric polling systems with single buffers," *IEEE Transactions on Communications*, vol. 36, no. 10, pp. 1119–1127, 1988.

[159] H. Chung, C. K. Un, and W. Y. Jung, "Performance analysis of markovian polling systems with single buffers," *Performance Evaluation*, vol. 19, no. 4, pp. 303–315, 1994.

[160] B. Mukherjee, C. K. Kwok, A. C. Lantz, and W.-H. Moh, "Comments on" exact analysis of asymmetric polling systems with single buffers," *IEEE Transactions on Communications*, vol. 38, no. 7, pp. 944–946, 1990.

[161] E. Najm, R. Yates, and E. Soljanin, “Status updates through M/G/1/1 queues with HARQ,” in *2017 IEEE International Symposium on Information Theory (ISIT)*, pp. 131–135, IEEE, 2017.

[162] M. Moltafet, M. Leinonen, and M. Codreanu, “On the age of information in multi-source queueing models,” *arXiv preprint arXiv:1911.07029*, 2019.

[163] C. Kam, S. Kompella, G. D. Nguyen, J. E. Wieselthier, and A. Ephremides, “On the age of information with packet deadlines,” *IEEE Transactions on Information Theory*, vol. 64, no. 9, pp. 6419–6428, 2018.

[164] A. Maatouk, M. Assaad, and A. Ephremides, “The age of updates in a simple relay network,” in *2018 IEEE Information Theory Workshop (ITW)*, pp. 1–5, IEEE, 2018.

[165] E. L. Lehmann, “Some concepts of dependence,” *The Annals of Mathematical Statistics*, pp. 1137–1153, 1966.

[166] D. Kofman, “Blocking probability, throughput and waiting time in finite capacity polling systems,” *Queueing Systems*, vol. 14, no. 3-4, pp. 385–411, 1993.

[167] H. Takagi, “Analysis and application of polling models,” in *Performance Evaluation: Origins and Directions*, pp. 423–442, Springer, 2000.

[168] O. J. Boxma and J. A. Weststrate, “Waiting times in polling systems with markovian server routing,” in *Messung, Modellierung und Bewertung von Rechensystemen und Netzen*, pp. 89–104, Springer, 1989.

APPENDIX A

APPENDIX FOR CHAPTER 3

A.1 Proof for Theorem 3.2.1

In this section we mainly provide the proof for Theorem 1 of our original paper. We first introduce a fact that will be useful later.

Fact A.1.1. *For positive numbers* $\{a_i, b_i\}_{i=1}^n$*, we have* $\frac{\sum_{i=1}^n a_i}{\sum_{i=1}^n b_i} \le \max_{i=1}^n\{\frac{a_i}{b_i}\}$.

Proof. Without loss of generality, assume $\frac{a_n}{b_n} = \max_{i=1}^n\{\frac{a_i}{b_i}\}$, then for any i we have $\frac{a_n}{b_n} \ge \frac{a_i}{b_i}$, thus $a_n b_i \ge a_i b_n$ holds for all $i = 1, ..., n$. Since $\frac{a_n}{b_n} - \frac{\sum_{i=1}^n a_i}{\sum_{i=1}^n b_i} = \frac{a_n \sum_{i=1}^n b_i - b_n \sum_{i=1}^n a_i}{b_n(\sum_{i=1}^n b_i)} = \frac{\sum_{i=1}^n (a_n b_i - a_i b_n)}{b_n(\sum_{i=1}^n b_i)} \ge 0$, we have $\frac{\sum_{i=1}^n a_i}{\sum_{i=1}^n b_i} \le \frac{a_n}{b_n} = \max_{i=1}^n\{\frac{a_i}{b_i}\}$. $\square$

In Theorem 1 we want to show $\sup_I \frac{C^e(I)}{C^*(I)} \le \rho$ for some constant ρ. However, by Fact A.1.1 we only need to show that this inequality holds for the instance processed in each busy period. Here we first introduce the concept of busy periods. If there is at least one job in the system, we say the system is busy, otherwise it is empty. When the system is empty, the server is not serving under the online policy. The status of the system under the online policy can be described as a busy period following an empty period, and then following by a busy period, and so on. There are two types of busy periods that we are interested in. Type I busy period (denoted as I-B) is the busy period in which the server under online policy resumes work without setting up. This is because the server was idling at the last queue it served (say queue i) after the previous busy period, and the first arrival in the new busy period also occurs at queue i. Type II busy period (denote as II-B) is the busy period in which the server resumes work with a setting up, which is because the new arrival occurs at a queue different from the queue that the server was idling at. We will consider these two types of busy period separately in the proof. To show the online policy has competitive ratio ρ, from Fact A.1.1 we only need to show that $\sup_I \frac{C^e(I)}{C^*(I)} \le \rho$ holds for any instance I processed in a busy period.

In the following we only focus our discussion in a single busy period. Since the server in the online policy serves queues in a cyclic way, without loss of generality, we assume that the server serves from queue 1 to queue k, and then switches back to queue 1, and so on. A cycle (round) starts when the server visits queue 1 and ends when it visits queue 1 the next time. If at some time point a queue, say queue i, is empty and skipped by the server, we still say that queue i has been visited in this cycle, with setup time 0. We define the job instance served by the server in its w^{th} visit to queue as b_i^w (for $i = 1, ..., k$), and we call each b_i^w a *batch*. Batch b_i^w is a subset of job instance I. In each cycle, there are k batches served by the online policy, and some of them may be empty but not all of them (if all of them are empty then the system is empty and the server would idle at queue 1). We let $I^w = \cup_{i=1}^k b_i^w$. For each batch b_i^w with the number of jobs $n(b_i^w) = n_i^w$, S_i^w is the earliest time when a job from b_i^w starts being processed under the online policy, and R_i^w is the earliest release date (arrival time) over all jobs from batch b_i^w. Notice that $R_i^w \le S_i^w$. Each batch b_i^w may be processed by the optimal offline policy in a different way from the online policy. Suppose E_i^{w*} is the earliest time when a job in batch b_i^w starts service under the optimal offline policy. Note E_i^{w*} may differ from S_i^w. Before time S_i^w, we know all the batches $(\cup_{j=1}^{w-1} I^j) \cup (\cup_{l=1}^{i-1} b_l^w)$ have been served by the online policy. However in the optimal offline policy, only some jobs from these batches have been served. We suppose q_i^w number of jobs in $(\cup_{j=1}^{w-1} I^j) \cup (\cup_{l=1}^{i-1} b_l^w)$ have been served by the optimal offline policy before time E_i^{w*}. Note that q_i^w is just a number instead of a job instance. We let $C^e(I)$ be the cumulative completion times of all jobs in job instance I under online policy $e \in \Pi_1$, and $C^*(I)$ be the cumulative completion times of all jobs in I under the offline optimal policy. For convenience, we let $g(I) = \frac{n(I)(n(I)+1)}{2}$ for job instance I, which is the sum of arithmetic sequence from 1 to $n(I)$. Also, $g(I)$ can be regarded as the completion time of I when 1) all the jobs are available at time 0, 2) each of them are of processing time 1, and 3) no setup time is considered. All the notations are summarized in Table A.1 of this document. Before going to the proof of Theorem 3.2.1, we first provide an example to show how the total completion time is characterized.

Example A.1.2. Suppose there is a job instance I with $n(I) = n_1 + n_2$ jobs which arrives at time

R. Each of the job has processing time 1. Under policy π the server starts to process the first n_1 jobs, and idles for time W, and then processes the rest of n_2 jobs without idling. The total completion time for I is given by

$$\begin{aligned} C^{\pi}(I) &= (R+1)+(R+2)+...+(R+n_1)+(R+n_1+W+1) \\ &\quad +(R+n_1+W+2)+...+(R+n_1+W+n_2) \\ &= (n_1+n_2)R+n_2W+g(I). \end{aligned}$$

Notice the completion time of I is made up of three components. The first component $(n_1+n_2)R$ is because all the jobs in I arrive at time R. The second term n_2W is because the rest n_2 jobs wait for another W amount of time. The third term $g(I)$ is the pure completion time if we process jobs one by one without idling.

Having showed the idea of calculating total completion times in Example A.1.2, now we move on to show the proof of Theorem 3.2.1. We next introduce the idea of *truncated optimal schedule*. Notice that the optimal policy may not always be work-conserving (i.e., never idles when there are jobs in the system). The optimal policy may wait at some queue in order to receive more jobs which will arrive in the future. The truncated optimal solution is defined by the completion time for the optimal offline problem with subtracting the completion time caused by idling, which is shown in Figure A.1. There is a waiting (idling) period W between b_1^1 and b_1^2 in Figure A.1. The truncated optimal solution is given by $C^*(b_1^1 \cup b_1^2 \cup b_2^1 \cup b_3^1) - W(n_1^2+n_2^1+n_3^1)$. We use $C^t(I)$ to denote the total completion time for instance I under the truncated optimal schedule. Note that the truncated optimal solution is always a lower bound for the real optimal solution.

Lemma A.1.3. *Suppose I is a job instance, b is a batch and $p_{min}=1$, then $C^t(I\cup b) \geq C^t(I) + g(b) + En(b) + n(b)(n(I)-q)$, where E is the time when the server starts serving batch b in the truncated optimal solution, and q is the number of jobs in I that are served before time E.*

Proof. We suppose the optimal solution is given, and now we consider the total completion time

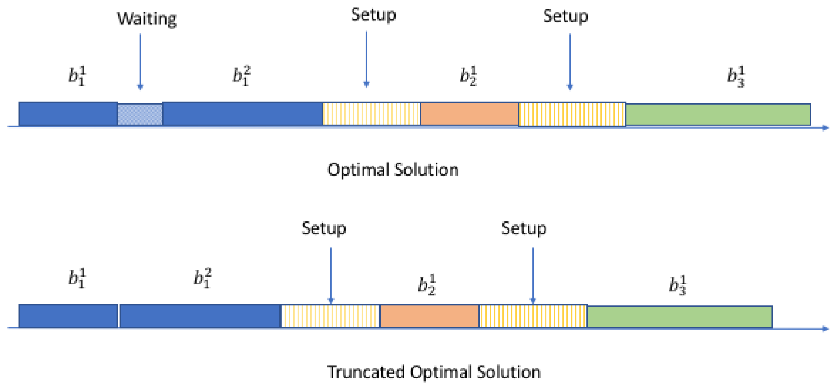


Figure A.1: Truncated Optimal Solution

of $I \cup b$ under truncated optimal solution. If all the jobs from b are served after I in the optimal solution, then we have $C^t(I \cup b) \geq C^t(I) + g(b) + En(b)$. If not, we let $\delta(b) = C^t(I \cup b) - C^t(I)$ be the additional completion time incurred by inserting b into I. Notice that $\delta(b)$ is minimized when all jobs in b has $p_{min} = 1$. If we can show that $\delta(b) \geq g(b) + En(b) + n(b)(n(I) - q)$ with every job in b having $p_{min} = 1$, we can then prove the lemma. So we assume here that every job in b has $p_{min} = 1$. Since the earliest time to process batch b in the truncated optimal solution is E, if we combine all jobs in b altogether and serve them in one batch from time E to time $E + n(b)$, then $\delta(b)$ is again minimized since all the jobs in b have the smallest processing time. So in the following we show that by inserting batch b at time E, the additional completion time incurred is at least $g(b) + En(b) + n(b)(n(I) - q)$. By inserting batch b into I from time E to $E + n(b)$, some jobs from I served after E in the original truncated optimal solution (with total number $(n(I) - q)$) are moved after batch b, resulting an increase of delay $n(b)(n(I) - q)$ for these jobs. Besides, inserting a batch b at time E increases the total completion time by $g(b) + En(b)$. So inserting a batch b can increase at least $g(b) + En(b) + n(b)(n(I) - q)$ amount of completion time. We thus prove the lemma. □

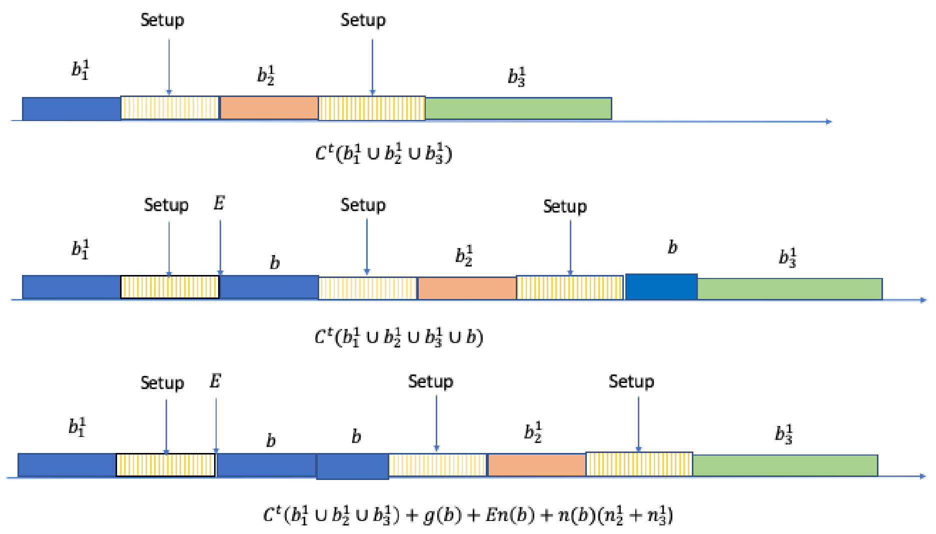

Setup
Setup
b_1^1
b_2^1
b_3^1
$C^t(b_1^1 \cup b_2^1 \cup b_3^1)$
Setup
E
Setup
Setup
b_1^1
b
b_2^1
b
b_3^1
$C^t(b_1^1 \cup b_2^1 \cup b_3^1 \cup b)$
Setup
E
Setup
Setup
b_1^1
b
b
b_2^1
b_3^1
$C^t(b_1^1 \cup b_2^1 \cup b_3^1) + g(b) + En(b) + n(b)(n_2^1 + n_3^1)$

Figure A.2: Insert a Batch

To make the proof of Lemma A.1.3 easier to understand, an example is given in Figure A.2. The first schedule in Figure A.2 is the truncated optimal for the batch $b_1^1 \cup b_2^1 \cup b_3^1$. The second schedule is the truncated optimal schedule for $b_1^1 \cup b_2^1 \cup b_3^1 \cup b$, where b is separated into two parts. If all jobs in b are of workload $p_{min} = 1$, it is always beneficial to schedule all jobs of b in the same batch, which is shown as the third schedule in Figure A.2. Notice that in Figure A.2, $q = n(b_1^1 + b_1^2) = n_1^1 + n_1^2$.

Lemma A.1.4. *Suppose I is a job instance, b is a batch and $p_{min} = 1$, then $C^*(I \cup b) \geq C^*(I) + g(b) + E^* n(b)$, where E^* is the time when the server starts serving batch b in the optimal solution.*

Proof. Since the earliest service time in the optimal solution for b is at E^*, we have the minimal total completion time for b is $g(b) + E^* n(b)$. Hence proved. □

We now introduce a benchmark for the online policy by combining the optimal solution and the truncated optimal solution. We let $C^m(I) = \alpha C^*(I) + (1 - \alpha) C^t(I)$ be the benchmark, where $\alpha = \frac{1}{1+k}$. We notice that $C^*(I) \geq C^m(I) \geq C^t(I)$ from the fact that $C^*(I) \geq C^t(I)$. If we have $\frac{C^e(I)}{C^m(I)} \leq \rho$, then we can show that $\frac{C^e(I)}{C^*(I)} \leq \rho$.

Next we restate Theorem 3.2.1 in the main paper and describe the proof.

Theorem A.1.5. ***(Theorem 3.2.1 in the main paper)*** *Any policy in* Π_1 *has competitive ratio of* $\kappa = \max\{\frac{3}{2}\gamma, k+1\}$ *for the polling system* $1 \mid r_i, \tau, p_{max} \leq \gamma p_{min} \mid \sum C_i$*. When* $\frac{3}{2}\gamma \leq k+1$*, for arbitrary* $\epsilon > 0$*, there is an instance* I *such that* $\frac{C^e(I)}{C^*(I)} > \kappa - \epsilon$*.*

Proof. We prove the theorem by induction. In this proof we only consider an instance I that the cyclic policy $e \in \Pi_1$ serves in a busy period. By induction we can finally conclude that $\frac{C^e(I)}{C^m(I)} \leq \kappa$ for instance I, which also implies that $\frac{C^e(I)}{C^*(I)} \leq \kappa$. We first show that the batches served by the online policy in the first cycle, i.e., $I^1 \in I$, satisfies $\frac{C^e(I^1)}{C^m(I^1)} \leq \kappa$. We next prove that if the result holds true for $\cup_{j=1}^{w-1} I^j$, then it also holds for $(\cup_{j=1}^{w-1} I^j) \cup b_1^w$. We next show the result holds true for $(\cup_{j=1}^{w-1} I^j) \cup (\cup_{i=1}^{l+1} b_i^w)$ if the result holds for $(\cup_{j=1}^{w-1} I^j) \cup (\cup_{i=1}^{l} b_i^w)$.

To start, we first assume $p_{min} = 1$ and $p_{max} = \gamma$ (the case where $p_{min} = 0$ is similar). Notice that $I^1 = \cup_{i=1}^k b_i^1$ is the union of batches served in the first cycle under the online policy. Without loss of generality, we assume the server serves from queue 1 to queue k in each cycle. Knowing that $I^1 = \cup_{i=1}^k b_i^1$, we let $n_{(k)}^1 \geq n_{(k-1)}^1 \geq ... \geq n_{(1)}^1$ be the descending permutation of $(n_1^1, ..., n_k^1)$, and $E^1 = \min_{i=1}^k E_i^1$, $S^1 = \min_{i=1}^k S_i^1$. Then we have (with explanation given later)

$$C^t(I^1) \quad \geq \quad g(I^1) + \tau \sum_{i=2}^{k} \sum_{j=i}^{k} n_{(k-j+1)}^1 + E^1 \sum_{i=1}^{k} n_i^1, \tag{A.1}$$

and

$$C^e(I^1) \quad \leq \quad \gamma g(I^1) + \tau \sum_{i=2}^{k} \sum_{j=i}^{k} n_{(j)}^1 + S^1 \sum_{i=1}^{k} n_i^1. \tag{A.2}$$

The RHS of Inequality (A.1) is the minimal completion time of a list which has the same number of jobs in each queue as I^1 and all of these jobs arrive at time E^1 with each job having processing time 1. The first term $g(I^1)$ is the pure completion time. The second term is because $n_{(k)}^1 \geq n_{(k-1)}^1 \geq ... \geq n_{(1)}^1$, if all batches are available at time E^1 and there is no further arrivals, the best order of serving the batches is to serve from the longest one to the shortest one. Note that the optimal policy may start without setting up since it may be the same queue that the server was idling at and resumed with. In the case where there is no setup for the first queue, the completion time incurred by setup is $\tau \sum_{i=2}^{k} \sum_{j=i}^{k} n^1_{(k-j+1)}$. The third term in RHS of Inequality (A.1) is because the entire service process for the optimal solution starts from E^1. Therefore, the RHS of Inequality (A.1) is a lower bound for $C^t(I^1)$. The RHS of inequality (A.2) is the upper bound for the online policy, which says that the online policy may serve batches from the shortest to the longest, starting from time point S^1, and all jobs are of the maximal workload γ. Since we consider I-B in this case, the server in the online policy does not set up for the first batch as the server was idling in the same queue as the new arrival. So the completion time resulted by setup is upper bounded by $\tau \sum_{i=2}^{k} \sum_{j=i}^{k} n^1_{(j)}$.

We let $Z(k) = \sum_{i=1}^{k} \sum_{j=i}^{k} n^1_{(j)}$ and $Z^t(k) = \sum_{i=1}^{k} \sum_{j=i}^{k} n^1_{(k-j+1)}$ then

$$\frac{Z(k)}{Z^t(k)} = \frac{kn^1_{(k)} + (k-1)n^1_{(k-1)} + ... + n^1_{(1)}}{kn^1_{(1)} + (k-1)n^1_{(2)} + ... + n^1_{(k)}} \leq \frac{kn^1_{(k)} + kn^1_{(k-1)} + ... + kn^1_{(1)}}{n^1_{(1)} + n^1_{(2)} + ... + n^1_{(k)}} \leq k.$$

From Fact A.1.1 we have

$$\begin{aligned} \frac{C^e(I^1)}{C^t(I^1)} &\leq \frac{\gamma g(I^1) + \tau \sum_{i=2}^{k} \sum_{j=i}^{k} n^1_{(i)} + S^1 \sum_{i=1}^{k} n^1_i}{g(I^1) + \tau \sum_{i=2}^{k} \sum_{j=i}^{k} n^1_{(k-j+1)} + E^1 \sum_{i=1}^{k} n^1_i} \\ &\leq \frac{\gamma g(I^1) + \tau(Z(k) - \sum_{i=1}^{k} n^1_i) + E^1 \sum_{i=1}^{k} n^1_i}{g(I^1) + \tau(Z^t(k) - \sum_{i=1}^{k} n^1_i) + E^1 \sum_{i=1}^{k} n^1_i} \\ &\leq \max\{\gamma, k, 1\} < \kappa. \end{aligned} \tag{A.3}$$

The Inequality (A.3) follows from the fact that $\tau \leq \min_{i=1}^{k} S^1_i = \min_{i=1}^{k}\{R^1_i\} \leq \min_{i=1}^{k} E^1_i$ because there must be a busy period happening before an I-B. So far we have shown $\frac{C^e(I^1)}{C^m(I^1)} \leq \kappa$ from the fact that $C^t(I^1) \leq C^m(I^1)$. Now we prove that $C^e((\cup_{j=1}^{w-1} I^j) \cup b^w_1) \leq \kappa C^m((\cup_{j=1}^{w-1} I^1) \cup b^w_1)$. Clearly if $n^w_1 = 0$ then the conclusion holds. Now we let $\bar{n} = \sum_{j=1}^{w-1} \sum_{i=1}^{k} n^j_i + n^w_1$ and suppose $n_1^w \neq 0$, we then have

$$C^e((\cup_{j=1}^{w-1} I^j) \cup b_1^w) \;\; \leq \;\; C^e(\cup_{j=1}^{w-1} I^j) + \gamma g(b_1^w) + n_1^w \left(\gamma \sum_{i=1}^{k} n_i^{w-1} + k\tau + S_1^{w-1} \right), \quad \text{(A.4)}$$

and

$$\begin{aligned} C^m((\cup_{j=1}^{w-1} I^j) \cup b_1^w) \;\; &= \;\; \alpha C^*((\cup_{j=1}^{w-1} I^j) \cup b_1^w) + (1-\alpha) C^t((\cup_{j=1}^{w-1} I^j) \cup b_1^w) \\ &\geq \;\; \alpha C^*(\cup_{j=1}^{w-1} I^j) + \alpha E_1^{w*} n_1^w + \alpha g(b_1^w) \\ &\quad\;\; +(1-\alpha) C^t(\cup_{j=1}^{w-1} I^j) + (1-\alpha) g(b_1^w) \\ &\quad\;\; +(1-\alpha) E_1^w n_1^w + (1-\alpha) n_1^w (\bar{n} - q_1^w), \end{aligned} \quad \text{(A.5)}$$

where

$$E_1^w \geq \;\; S^1 + q_1^w, \quad \text{(A.6)}$$

and

$$E_1^{w*} \geq R_1^w \geq S_1^{w-1} + n_1^{w-1}. \quad \text{(A.7)}$$

The RHS of Inequality (A.4) is because the completion time of batch b_1^w is bounded by γ times its pure completion time $g(b_1^w)$, which is the maximal pure completion time for b_1^w (if all the workload in b_1^w is $p_{max} = \gamma$), plus n_1^w times the maximal starting time $\gamma \sum_{i=1}^{k} n_i^{w-1} + k\tau + S_1^{w-1}$. Inequality A.5 follows from Lemma A.1.3 and A.1.4 directly. Inequality (A.6) holds because before E_1^w, the server has served q_1^w number of jobs. Inequality (A.7) is because the earliest time to serve b_1^w is no earlier than R_1^w. From Inequalities (A.4,A.5,A.6 and A.7) we have

$$
\begin{aligned}
& \frac{C^e((\cup_{j=1}^{w-1} I^j) \cup b_1^w)}{C^m((\cup_{j=1}^{w-1} I^j) \cup b_1^w)} \\
\leq\ & \frac{C^e(\cup_{j=1}^{w-1} I^j) + \gamma g(b_1^w) + n_1^w \left(\gamma \sum_{i=1}^k n_i^{w-1} + k\tau + S_1^{w-1}\right)}{\alpha C^*(\cup_{j=1}^{w-1} I^j) + (1-\alpha) C^t(\cup_{j=1}^{w-1} I^j) + \alpha S_1^{w-1} n_1^w + g(b_1^w) + (1-\alpha) n_1^w \bar{n} + (1-\alpha) n_1^w \tau} \\
\leq\ & \max\{\frac{C^e(\cup_{j=1}^{w-1} I^j)}{C^m(\cup_{j=1}^{w-1} I^j)}, \frac{\gamma g(b_1^w)}{g(b_1^w)}, \frac{k\tau + S_1^{w-1}}{\alpha S_1^{w-1} + (1-\alpha)\tau}, \frac{\gamma \bar{n}}{(1-\alpha)\bar{n}}\} \\
\leq\ & \max\{\kappa, \gamma, k+1, \frac{\gamma}{1-\alpha}\}.
\end{aligned}
$$

Notice that $\max\{\kappa, \gamma, k+1, \frac{\gamma}{1-\alpha}\} = \max\{\kappa, \gamma, k+1, \gamma \frac{k+1}{k}\} \leq \max\{\kappa, \gamma, k+1, \frac{3}{2}\gamma\} = \kappa$ from the fact that $k \geq 2$ and $\alpha = \frac{1}{k+1}$.

Now suppose the result holds for $\bar{b} = \left(\cup_{j=1}^{w-1} I^j\right) \cup \left(\cup_{i=1}^{l} b_i^w\right)$ where $w \geq 2$, and we want to show it also holds for $\bar{b} \cup b_{l+1}^w = \left(\cup_{j=1}^{w-1} I^j\right) \cup \left(\cup_{i=1}^{l+1} b_i^w\right)$ for $l < k$ by induction, where $n_{l+1}^w \neq 0$. We abuse the notion by letting $\bar{n} = \sum_{j=1}^{w-1} \sum_{i=1}^{k} n_i^j + \sum_{i=1}^{l} n_i^w$ be the number of jobs served before b_{l+1}^w, and $\bar{n}_{l+1}^w = \sum_{j=l+1}^{k} n_j^{w-1} + \sum_{j=1}^{l} n_j^w$ be the number of jobs served between S_{l+1}^{w-1} and S_{l+1}^w. Because the server stays in queue i at the k^{th} visit for no more than time γn_i^k and serves n_i^k jobs ,we have

$$
C^e(\bar{b} \cup b_{l+1}^w) \leq C^e(\bar{b}) + \gamma g(b_{l+1}^w) + n_{l+1}^w \left(\gamma \bar{n}_{l+1}^w + k\tau + S_{l+1}^{w-1}\right),
$$

and

$$
\begin{aligned}
C^m(\bar{b} \cup b_{l+1}^w) &= \alpha C^*(\bar{b} \cup b_{l+1}^w) + (1-\alpha) C^t(\bar{b} \cup b_{l+1}^w) \\
&\geq \alpha C^*(\bar{b}) + \alpha E_{l+1}^{w*} n_{l+1}^w + \alpha g(b_{l+1}^w) \\
&\quad + (1-\alpha) C^t(\bar{b}) + (1-\alpha) g(b_{l+1}^w) \\
&\quad + (1-\alpha) E_{l+1}^w n_{l+1}^w + (1-\alpha) n_{l+1}^w (\bar{n} - q_{l+1}^w),
\end{aligned}
$$

where

$$E^w_{l+1} \geq \tau + q^w_{l+1},$$

and

$$E^{w*}_{l+1} \geq R^w_{l+1} > S^{w-1}_{l+1} + n^{w-1}_{l+1} \quad .$$

Similar to our discussion above, we have

$$\begin{aligned}
& \frac{C^e(\bar{b} \cup b^w_{l+1})}{C^m(\bar{b} \cup b^w_{l+1})} \\
\leq \ & \frac{C^e(\bar{b}) + \gamma g(b^w_{l+1}) + n^w_{l+1}\left(\gamma \bar{n}^w_{l+1} + k\tau + S^{w-1}_{l+1}\right)}{\alpha C^*(\bar{b}) + (1-\alpha)C^t(\bar{b}) + \alpha S^{w-1}_{l+1} n^w_{l+1} + g(b^w_{l+1}) + (1-\alpha)n^w_{l+1}\bar{n} + (1-\alpha)n^w_1\tau} \\
\leq \ & \max\{\frac{C^e(\bar{b})}{C^m(\bar{b})}, \frac{\gamma g(b^w_{l+1})}{g(b^w_{l+1})}, \frac{k\tau + S^{w-1}_{l+1}}{\alpha S^{w-1}_{l+1} + (1-\alpha)\tau}, \frac{\gamma \bar{n}}{(1-\alpha)\bar{n}}\} \\
\leq \ & \max\{\kappa, \gamma, k+1, \frac{\gamma}{1-\alpha}\} = \kappa.
\end{aligned}$$

Now we show that results hold for the second type of busy period, i.e., II-B. For II-B, the first arrival occurs in a different queue from where the server was idling, thus the server starts this period with a setup. Note that the very first busy period is also a II-B. If I^1 belongs to the first busy period, then

$$C^t(I^1) \geq g(I^1) + \tau \sum_{i=1}^{k} \sum_{j=i}^{k} n^1_{(k-j+1)}.$$

$$C^e(I^1) \leq \gamma g(I^1) + \tau \sum_{i=1}^{k} \sum_{j=i}^{k} n^1_{(j)} + \tau \sum_{i=1}^{k} n^1_i.$$

Thus

$$\frac{C^e(I^1)}{C^t(I^1)} \leq \frac{\gamma g(I^1) + \tau Z(k) + \tau \sum_{i=1}^k n_i^1}{g(I^1) + \tau Z^t(k)} \leq \max\{\gamma, k+1\} \leq \kappa.$$

If I^1 does not belong to the first busy period, then

$$C^t(I^1) \geq g(I^1) + \tau \sum_{i=2}^{k} \sum_{j=i}^{k} n_{(k-j+1)}^1 + E^1 \sum_{i=1}^{k} n_i^1,$$

and

$$C^e(I^1) \leq \gamma g(I^1) + \tau \sum_{i=2}^{k} \sum_{j=i}^{k} n_{(j)}^1 + S^1 \sum_{i=1}^{k} n_i^1.$$

Thus if $R^1 \geq 2\tau$, then $\frac{R^1}{R^1-\tau} \leq 2$, then

$$\begin{aligned}
\frac{C^e(I^1)}{C^t(I^1)} &\leq \frac{\gamma g(I^1) + \tau \sum_{i=2}^k \sum_{j=i}^k n_{(j)}^1 + S^1 \sum_{i=1}^k n_i^1}{g(I^1) + \tau \sum_{i=2}^k \sum_{j=i}^k n_{(k-j+1)}^1 + E^1 \sum_{i=1}^k n_i^1} \\
&\leq \frac{\gamma g(I^1) + \tau \sum_{i=2}^k \sum_{j=i}^k n_{(j)}^1 + (R^1+\tau) \sum_{i=1}^k n_i^1}{g(I^1) + \tau \sum_{i=2}^k \sum_{j=i}^k n_{(k-j+1)}^1 + R^1 \sum_{i=1}^k n_i^1} \qquad \text{(A.8)} \\
&\leq \frac{\gamma g(I^1) + \tau Z(k) + R^1 \sum_{i=1}^k n_i^1}{g(I^1) + \tau Z^t(k) + (R^1 - \tau) \sum_{i=1}^k n_i^1} \\
&\leq \max\{\gamma, k, 2\} \\
&< \kappa.
\end{aligned}$$

Inequality (A.8) follows from $E^1 \geq R^1 = \min_{i=1}^k R_i^1 \geq \tau$ and $S^1 = R^1 + \tau$ because the server would immediately set up the queue where a new arrival occurs after an idling period. If $R^1 \leq \tau$ then I^1 belongs to the very first busy period, which we have discussed. If $\tau < R^1 < 2\tau$, then the online policy has only scheduled at most one batch before R^1. Since the new busy period belongs to II-B, both online and optimal policy in this busy period start from processing a queue different from the queue processed in the previous busy period. We then have $E^1 \geq 2\tau$ and

$$\begin{aligned}
\frac{C^e(I^1)}{C^t(I^1)} &\leq \frac{\gamma g(I^1) + \tau \sum_{i=2}^k \sum_{j=i}^k n_{(j)}^1 + (R^1 + \tau) \sum_{i=1}^k n_i^1}{g(I^1) + \tau \sum_{i=2}^k \sum_{j=i}^k n_{(k-j+1)}^1 + 2\tau \sum_{i=1}^k n_i^1} \\
&\leq \frac{\gamma g(I^1) + \tau Z(k) + R^1 \sum_{i=1}^k n_i^1}{g(I^1) + \tau Z^t(k) + \tau \sum_{i=1}^k n_i^1} \\
&\leq \max\{\gamma, k, 2\} < \kappa.
\end{aligned}$$

Discussion for $(\cup_{j=1}^{w-1} I_j) \cup b_{l+1}^w$ for $l = 0, ..., k-1$ is similar to our discussion for I-B.

Now we prove the approximate tightness argument of this theorem by constructing a special instance I. When $\frac{3}{2}\gamma \leq (k+1)$, we have $\kappa = k+1$. Suppose at time 0 there is one job with workload γ at queue 1 and one job with $p = 1$ at the other queues. At time $\gamma + \tau + \epsilon_1$ (with small $\epsilon_1 > 0$) a batch b_1^2 arrives at queue 1 and each job in b_1^2 has workload $p = 1$. We thus have

$$C^e(I) \geq g(I) + n_1^2(k+1)\tau + \frac{k(k+1)}{2}\tau,$$

and

$$C^*(I) \leq \gamma g(I) + (n_1^2 + k - 1)(\gamma - 1) + n_1^2(\tau + \epsilon_1) + \frac{k(k+1)}{2}\tau + (k-1)\epsilon_1.$$

Thus when $\tau = (n_1^2)^2$ there is an n_1^2 such that for arbitrary ϵ,

$$\frac{C^e(I)}{C^*(I)} > 1 + k - \epsilon.$$

The theorem also holds for $p_{max} = 0$, for simplicity we do not show the proof here. $\square$

A.2 Proof for Theorem 3.2.2

Theorem A.2.1. ***(Theorem 3.2.2 in the main paper)*** *Any policy in Π_2 has competitive ratio of $\kappa = \max\{\frac{3}{2}\gamma, k+1\}$ for the polling system $1 \mid r_i, \tau, p_{max} \leq \gamma p_{min} \mid \sum C_i$. When $\frac{3}{2}\gamma \leq k+1$, for arbitrary $\epsilon > 0$, there is an instance I such that $\frac{C^e(I)}{C^*(I)} > \kappa - \epsilon$.*

Proof. The proof is similar to the one for Cyclic with Skipping the Empty Queue, however this time we only need to show $C^{ew}(I) \leq \kappa C^m(I)$ for any busy period I where $C^{ew}(I)$ is the completion time by a policy from Π_2. Notice we no longer need to consider different cases for I-B and II-B because even when the system is idling, the cyclic policy still keeps setting up queues in cycle. When a new arrival occurs after the system being empty for some time, we simply regard this time R^1 as the beginning of a busy period. Without loss of generality, we assume the server begins serving with queue 1 in this busy period. Let $n^1_{(k)} \geq n^1_{(k-1)} \geq ... \geq n^1_{(1)}$ be the descending permutation of $(n^1_1, ..., n^1_k)$, $R^1 = \min_{i=1}^k R^1_i$, $E^1 = \min_{i=1}^k E^1_i$ and $S^1 = \min_{i=1}^k S^1_i$. Notice some of $n^1_{(i)}$ may be zero (not all of them) but the server still sets up the queue even if the queue is empty. We have

$$C^m(I_1) \quad \geq \quad g(I_1) + \tau \sum_{i=2}^{k} \sum_{j=i}^{k} n^1_{(k-j+1)} + E^1 \sum_{i=1}^{k} n^1_i,$$

and

$$C^{ew}(I_1) \quad \leq \quad \gamma g(I_1) + \tau k \sum_{i=1}^{k} n^1_{(i)} + R^1 \sum_{i=1}^{k} n^1_i.$$

To show $\frac{C^{ew}(I_1)}{C^m(I_1)} \leq \kappa$, by abusing the notation a little, we first let $Z(k) = k \sum_{i=1}^k n^1_{(i)}$ and $Z^t(k) = \sum_{i=1}^k \sum_{j=i}^k n^1_{(k-j+1)}$, so

$$\frac{Z(k)}{Z^t(k)} = \frac{kn^1_{(k)} + kn^1_{(k-1)} + ... + kn^1_{(1)}}{kn^1_{(1)} + (k-1)n^1_{(2)} + ... + n^1_{(k)}} \le \frac{kn^1_{(k)} + kn^1_{(k-1)} + ... + kn^1_{(1)}}{n^1_{(1)} + n^1_{(2)} + ... + n^1_{(k)}} \le k.$$

Thus

$$\begin{aligned} \frac{C^{ew}(I_1)}{C^m(I_1)} &\le \frac{\gamma g(I_1) + \tau k \sum_{i=1}^k n^1_{(i)} + R^1 \sum_{i=1}^k n^1_i}{g(I_1) + \tau \sum_{i=2}^k \sum_{j=i}^k n^1_{(k-j+1)} + E^1 \sum_{i=1}^k n^1_i} \\ &= \frac{\gamma g(I_1) + \tau k \sum_{i=1}^k n^1_{(i)} + R^1 \sum_{i=1}^k n^1_i}{g(I_1) + \tau \sum_{i=2}^k \sum_{j=i}^k n^1_{(k-j+1)} + \max\{\tau, R^1\} \sum_{i=1}^k n^1_i} \\ &\le \max\{\gamma, k+1\} \qquad \text{(A.9)} \\ &\le \kappa. \end{aligned}$$

Inequality (A.9) holds because if $R^1 \le \tau$, then

$$\begin{aligned} \frac{\gamma g(I_1) + \tau k \sum_{i=1}^k n^1_{(i)} + R^1 \sum_{i=1}^k n^1_i}{g(I_1) + \tau \sum_{i=2}^k \sum_{j=i}^k n^1_{(k-j+1)} + E^1 \sum_{i=1}^k n^1_i} &\le \frac{\gamma g(I_1) + \tau k \sum_{i=1}^k n^1_{(i)} + \tau \sum_{i=1}^k n^1_i}{g(I_1) + \tau \sum_{i=2}^k \sum_{j=i}^k n^1_{(k-j+1)} + \tau \sum_{i=1}^k n^1_i} \\ &\le \max\{\gamma, k+1\}. \end{aligned}$$

And if $R^1 > \tau$, then

$$\begin{aligned} &\frac{\gamma g(I_1) + \tau k \sum_{i=1}^k n^1_{(i)} + R^1 \sum_{i=1}^k n^1_i}{g(I_1) + \tau \sum_{i=2}^k \sum_{j=i}^k n^1_{(k-j+1)} + E^1 \sum_{i=1}^k n^1_i} \\ \le\ &\frac{\gamma g(I_1) + \tau (k+1) \sum_{i=1}^k n^1_{(i)} + (R^1 - \tau) \sum_{i=1}^k n^1_i}{g(I_1) + \tau \sum_{i=1}^k \sum_{j=i}^k n^1_{(k-j+1)} + (R^1 - \tau) \sum_{i=1}^k n^1_i} \\ \le\ &\max\{\gamma, k+1, 1\}. \end{aligned}$$

The discussions for $(\cup_{j=1}^{w-1} I_j) \cup b^w_{l+1}$ is similar to proof of Theorem 3.2.1. □

A.3 Proof for Theorem 3.2.3

Theorem A.3.1. ***(Theorem 3.2.3 in the Main Paper)*** *Any policy in* Π_3 *has competitive ratio of* $\kappa = \max\{\frac{3}{2}\gamma, k+1\}$ *for the polling system* $1 \mid r_i, \tau, p_{max} \le \gamma p_{min} \mid \sum C_i$*. When* $\frac{3}{2}\gamma \le k+1$*, for* *arbitrary $\epsilon > 0$, there is an instance I such that $\frac{C^e(I)}{C^*(I)} > \kappa - \epsilon$.*

Proof. We only show the proof for the policy without skipping the empty queue. The proof is similar to the proof for Theorem 3.2.2. We prove the theorem by induction. Again for simplicity, we assume $p_{min} = 1$ so that $p_{max} = \gamma$. We want to show $C^g(I) \leq \kappa C^m(I)$ holds for any instance I, where $C^g(I)$ is the completion time for any policy $g \in \Pi_3$. Again we assume the server routes from queue 1 to queue k in each cycle. Let $n^1_{(k)} \geq n^1_{(k-1)} \geq ... \geq n^1_{(1)}$ is the descending permutation of $(n^1_1, ..., n^1_k)$ and $E^1 = \min_{i=1}^k E^1_i$, $S^1 = \min_{i=1}^k S^1_i$, we have

$$C^t(I_1) \;\; \geq \;\; g(I_1) + \tau \sum_{i=2}^{k} \sum_{j=i}^{k} n^1_{(k-j+1)} + E^1 \sum_{i=1}^{k} n^1_i,$$

and

$$C^g(I_1) \;\; \leq \;\; g(I_1) + \tau k \sum_{i=1}^{k} n^1_{(i)} + S^1 \sum_{i=1}^{k} n^1_i.$$

The rest of discussions are similar to the proof of Theorem 3.2.1, except now we have $E^{w*}_{l+1} > R^w_{l+1}$ because the policy is gated.

$\square$

Notation	Meaning	Notation	Meaning
b_i^w, $i = 1, ..., k$	The job instance that are served by the cyclic online policy during the w^{th} visit (w^{th} cycle) to queue i, within a busy period	I^w, $w = 1, 2, ...$	$I^w = \cup_{i=1}^k b_i^w$, the union of instances that are served by the online policy during the w^{th} cycle within a busy period
$n_i^w = n(b_i^w)$, $i = 1, ..., k$	The number of jobs in batch b_i^w	$\alpha = \frac{1}{k+1}$	A constant
S_i^w, $i = 1, ..., k$	The time when the online policy starts to serve batch b_i^w	$S^1 = \min_{i=1}^k \{S_i^1\}$	The earliest staring time for I^1 by the online policy
R_i^w, $i = 1, ..., k$	The earliest release time (arrival time) of batch b_i^w	$R^1 = \min_{i=1}^k \{R_i^1\}$	The earliest release time of I^1
E_i^w	The time when the truncated optimal offline policy starts to serve batch b_i^w	$E^1 = \min_{i=1}^k \{E_i^1\}$	The earliest time when the truncated optimal offline policy starts to serve I^1
E_i^{w*}	The time when the optimal offline policy starts to serve batch b_i^w	q_i^w, $i = 1...k$	The jobs in $(\cup_{j=1}^{w-1} I^j) \cup (\cup_{l=1}^{i-1} b_l^w)$ that have been served by the optimal policy before E_i^w
$g(I) = \frac{n(I)(n(I)+1)}{2}$	Pure completion time for instance I	Busy period	The time period between two consecutive empty periods
I-B	Type I busy period. The server start the new busy period without setting up	II-B	Type II busy period. The server start the new busy period by setting up

Table A.1: List of Notations for Appendix

APPENDIX B

APPENDIX FOR CHAPTER 4

B.1 Proof for Theorem 5.3.7

Theorem 5.3.7. The PAoI for CBS-P is given by $\boldsymbol{E}[A_{CBS-P}] = \frac{1-H^*(\lambda)-\lambda H^{*(1)}(\lambda)+H^*(\lambda)^2}{\lambda H^*(\lambda)} + \frac{H^*(\lambda)V^{*(1)}(\lambda)-V^{*(1)}(0)}{1-V^*(\lambda)}$, and the AoI for this system is given by

$$\boldsymbol{E}[\Delta_{CBS-P}] = \frac{\frac{V^{*(2)}(0)}{1-V^*(\lambda)} + 2\frac{V^{*(1)}(0)V^{*(1)}(\lambda)}{(1-V^*(\lambda))^2} - 2\frac{V^{*(1)}(0)}{1-V^*(\lambda)}\frac{1-H^*(\lambda)}{\lambda H^*(\lambda)} + \frac{2}{\lambda H^*(\lambda)^2}\left[\frac{1}{\lambda} - \frac{H^*(\lambda)}{\lambda} + H^{*(1)}(\lambda)\right]}{2(-\frac{V^{*(1)}(0)}{1-V^*(\lambda)} + \frac{1-H^*(\lambda)}{\lambda H^*(\lambda)})}$$
$$-\frac{H^{*(1)}(\lambda)}{H^*(\lambda)} + H^*(\lambda)\left(\frac{1}{\lambda} + \frac{V^{*(1)}(\lambda)}{1-V^*(\lambda)}\right).$$

Proof. Different from the case of non-preemptive service, in the case where service is preempted by new arrivals, we decompose the age peak into three pieces

$$\boldsymbol{E}[A] = \boldsymbol{E}[D] + \boldsymbol{E}[B] + \boldsymbol{E}[L], \tag{B.1}$$

where D is the delay of a packet that is eventually processed by the server, B is the time period when the server is on vacation (the same as we defined in Theorem 5.3.2), and L is the time when the server is busy in serving. A demonstrative graph is given by Figure B.1. Note that these three components are mutually independent. Therefore the AoI of this system can be given as

$$\boldsymbol{E}[\Delta] = \frac{\boldsymbol{E}[L^2] + 2\boldsymbol{E}[L]\boldsymbol{E}[B] + \boldsymbol{E}[B^2]}{2(\boldsymbol{E}[L] + \boldsymbol{E}[B])} + \boldsymbol{E}[D]. \tag{B.2}$$

We now derive the LST of D, denoted as $D^*(s)$. We first notice that if the service time of a packet H is smaller than the inter-arrival time T, then the packet is served without being preempted. Therefore, all the packets that are eventually processed must have the service time smaller than the inter-arrival time. If the packet that we serve arrives during the last vacation, then its delay D is its

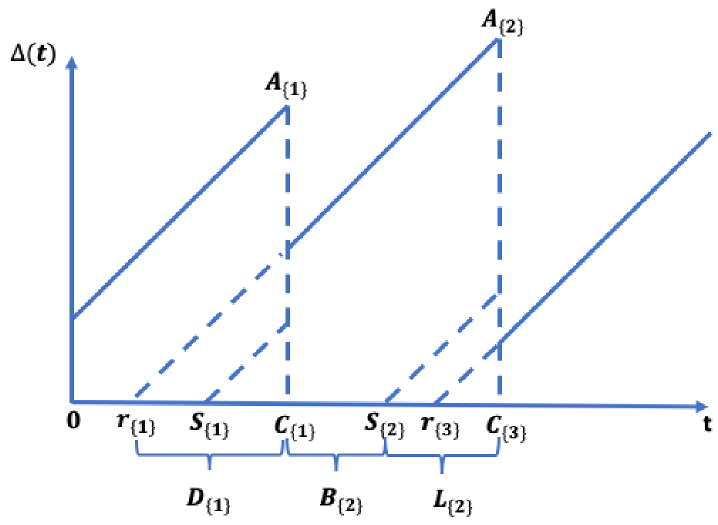


Figure B.1: Age of Information Decomposition for Preemptive Service Systems. The second age peak is decomposed into three components: $A_{\{2\}} = D_{\{1\}} + B_{\{2\}} + L_{\{2\}}$, where $D_{\{1\}}$ is the delay of packet 1, $B_{\{2\}}$ is period when the server is on vacation, and $L_{\{2\}}$ is the time period when the server is serving. Notice that in this example, packet 2 is preempted by packet 3 at time $r_{\{3\}}$, and packet 3 is not preempted by any packet.

waiting time G plus its service time. If it arrives during service (it preempts the previous packet in service), then the delay is its service time only. Thus we have $\boldsymbol{E}[e^{-sD}|H < T] = G^*(s)\hat{H}(s)$ and $\boldsymbol{E}[e^{-sD}|H \geq T] = \hat{H}(s)$, where $\hat{H}(s) = \boldsymbol{E}[e^{-sH}|H < T]$.

Since the inter-arrival time is exponential, by letting $F(x)$ be the CDF of service time H, we have $\hat{H}(s) = \frac{\int_0^\infty e^{-su} dF(u) \int_u^\infty \lambda e^{-\lambda x} dx}{\boldsymbol{P}(H<I)} = \frac{\int_0^\infty e^{-(s+\lambda)u} dF(u)}{\int_0^\infty dF(u) \int_u^\infty \lambda e^{-\lambda x} dx} = \frac{H^*(\lambda+s)}{H^*(\lambda)}$. Then we have

$$\begin{aligned} D^*(s) &= G^*(s)\frac{H^*(\lambda+s)}{H^*(\lambda)}H^*(\lambda) + \frac{H^*(\lambda+s)}{H^*(\lambda)}(1-H^*(\lambda)) \\ &= H^*(\lambda+s)\left(G^*(s) + \frac{1}{H^*(\lambda)} - 1\right). \end{aligned}$$

From the expression for $\boldsymbol{E}[G]$ in Theorem 5.3.2, we have

$$\begin{aligned}\boldsymbol{E}[D] &= -D^{*(1)}(0) = -\frac{H^{*(1)}(\lambda)}{H^*(\lambda)} - H^*(\lambda)G^{*(1)}(0) \\ &= -\frac{H^{*(1)}(\lambda)}{H^*(\lambda)} + H^*(\lambda)\left(\frac{1}{\lambda} + \frac{V^{*(1)}(\lambda)}{1-V^*(\lambda)}\right). \qquad (B.3)\end{aligned}$$

The LST of B is given in Theorem 5.3.2 as $B^*(s) = \frac{V^*(s)-V^*(s+\lambda)}{1-V^*(s+\lambda)}$, with $\boldsymbol{E}[B] = -B^{*(1)}(0) = -\frac{V^{*(1)}(0)}{1-V^*(\lambda)}$ and $\boldsymbol{E}[B^2] = B^{*(2)}(0) = \frac{V^{*(2)}(0)}{1-V^*(\lambda)} + 2\frac{V^{*(1)}(0)V^{*(1)}(\lambda)}{(1-V^*(\lambda))^2}$.

Now we derive the LST for L, i.e., $L^*(s)$. Notice that if the inter-arrival time T is greater than service time H, then the packet is processed without being preempted. If the inter-arrival time T is smaller than H, then a new period L is started after T. We then have $\boldsymbol{E}[e^{-sL}|H < T] = \hat{H}(s)$ and $\boldsymbol{E}[e^{-sL}|H \geq T] = \boldsymbol{E}[e^{-sT}L(s)|H \geq T]$. Thus

$$\begin{aligned}L^*(s) &= \int_0^\infty e^{-su}dF(u)\int_u^\infty \lambda e^{-\lambda x}dx + L(s)\int_0^\infty dF(u)\int_0^u e^{-sx}\lambda e^{-\lambda x}dx \\ &= H^*(s+\lambda) + L^*(s)\frac{\lambda}{s+\lambda}(1-H^*(s+\lambda)).\end{aligned}$$

We can then get

$$L^*(s) = \frac{H^*(s+\lambda)}{\frac{s}{s+\lambda} + \frac{\lambda}{s+\lambda}H^*(s+\lambda)}, \qquad (B.4)$$

$$\boldsymbol{E}[L] = -L^{*(1)}(0) = \frac{1-H^*(\lambda)}{\lambda H^*(\lambda)}, \qquad (B.5)$$

and

$$\boldsymbol{E}[L^2] = L^{*(2)}(0) = \frac{2}{\lambda H^*(\lambda)^2}\left(\frac{1}{\lambda} - \frac{H^*(\lambda)}{\lambda} + H^{*(1)}(\lambda)\right).$$

The PAoI for the system can now be given as

$$\begin{aligned}
\boldsymbol{E}[A] &= \boldsymbol{E}[D]+\boldsymbol{E}[B]+\boldsymbol{E}[L] \\
&= -\frac{H^{*(1)}(\lambda)}{H^*(\lambda)}+H^*(\lambda)\left(\frac{1}{\lambda}+\frac{V^{*(1)}(\lambda)}{1-V^*(\lambda)}\right)-\frac{V^{*(1)}(0)}{1-V^*(\lambda)}+\frac{1-H^*(\lambda)}{\lambda H^*(\lambda)} \\
&= \frac{1-H^*(\lambda)-\lambda H^{*(1)}(\lambda)+H^*(\lambda)^2}{\lambda H^*(\lambda)}+\frac{H^*(\lambda)V^{*(1)}(\lambda)-V^{*(1)}(0)}{1-V^*(\lambda)}.
\end{aligned}$$

□

B.2 Proof for Theorem 5.3.9

Lemma B.2.1. *It holds true for any LST function $V^*(s)$ that $\frac{V^{*(1)}(\lambda)}{1-V^*(\lambda)} \geq -\frac{1}{\lambda}$ for any positive λ.*

Proof. A non-rigorous but intuitive way of proving this is that by the fact that $\boldsymbol{E}[G] = \frac{1}{\lambda} + \frac{V^{*(1)}(\lambda)}{1-V^*(\lambda)} \geq 0$ in CBS, then $\frac{V^{*(1)}(\lambda)}{1-V^*(\lambda)} \geq -\frac{1}{\lambda}$ must hold. We now prove this inequality in another way without using the property of $\boldsymbol{E}[G]$.

Since

$$\begin{aligned}
\frac{V^{*(1)}(\lambda)}{1-V^*(\lambda)} &= \frac{-\boldsymbol{E}[Ve^{-\lambda V}]}{1-\boldsymbol{E}[e^{-\lambda V}]} \\
&= \frac{-\boldsymbol{E}[\lambda V-\lambda^2V^2+\frac{\lambda^3V^3}{2!}-\frac{\lambda^4V^4}{3!}+\frac{\lambda^5V^5}{4!}-...]}{\lambda\boldsymbol{E}[\lambda V-\frac{\lambda^2V^2}{2!}+\frac{\lambda^3V^3}{3!}-\frac{\lambda^4V^4}{4!}+...]},
\end{aligned}$$

we only need to show that $\lambda V-\lambda^2V^2+\frac{\lambda^3V^3}{2!}-\frac{\lambda^4V^4}{3!}+\frac{\lambda^5V^5}{4!}-... \leq \lambda V-\frac{\lambda^2V^2}{2!}+\frac{\lambda^3V^3}{3!}-\frac{\lambda^4V^4}{4!}+\frac{\lambda^5V^5}{5!}-...$ for any $V \geq 0$ and $\lambda \geq 0$. By letting

$$\beta(x) = (1-\frac{1}{2!})x^2-(\frac{1}{2!}-\frac{1}{3!})x^3+(\frac{1}{3!}-\frac{1}{4!})x^4-...,$$

we now only need to show that $\beta(x) \geq 0$ for any $x \geq 0$. Notice that

$$\begin{aligned}
\beta(x) &= \frac{1}{2}x^2 - \frac{2}{3!}x^3 + \frac{3}{4!}x^4 - \frac{4}{5!}x^5 + ... \\
&= x(x - \frac{1}{2}x^2 + \frac{1}{3!}x^3 - \frac{1}{4!}x^4 + ...) + (-\frac{1}{2!}x^2 + \frac{1}{3!}x^3 - \frac{1}{4!}x^4 + \frac{1}{5!}x^5 + ...) \\
&= x(1 - e^{-x}) + (-e^{-x} + 1 - x) \\
&= 1 - e^{-x} - xe^{-x}.
\end{aligned}$$

Since $\frac{\partial \beta(x)}{\partial x} = e^{-x} - e^{-x} + xe^{-x} \geq 0$ and $x \geq 0$, we have $\beta(x) \geq 0$. Therefore we have $\frac{V^{*(1)}(\lambda)}{1-V^*(\lambda)} \geq -\frac{1}{\lambda}$. □

Theorem 5.3.9. If the service time is exponentially distributed, then the system CBS-P has both AoI and PAoI smaller than CBS.

Proof. We assume that the service time is exponentially distributed with parameter μ. We first show the conclusion holds true for AoI. When the service time is exponentially distributed, we have

$$\begin{aligned}
\boldsymbol{E}[\Delta_{CBS}] &= -\frac{H^{*(2)}(0) + 2H^{*(1)}(0)\frac{V^{*(1)}(0)}{1-V^*(\lambda)} + \frac{V^{*(2)}(0)}{1-V^*(\lambda)} + 2\frac{V^{*(1)}(0)V^{*(1)}(\lambda)}{(1-V^*(\lambda))^2}}{2\left(H^{*(1)}(0) + \frac{V^{*(1)}(0)}{1-V^*(\lambda)}\right)} \\
&\quad + \frac{1}{\lambda} + \frac{V^{*(1)}(\lambda)}{1-V^*(\lambda)} - H^{*(1)}(0) \\
&= \frac{\frac{2}{\mu^2} - \frac{2}{\mu}\frac{V^{*(1)}(0)}{1-V^*(\lambda)} + \frac{V^{*(2)}(0)}{1-V^*(\lambda)} + 2\frac{V^{*(1)}(0)V^{*(1)}(\lambda)}{(1-V^*(\lambda))^2}}{2\left(\frac{1}{\mu} - \frac{V^{*(1)}(0)}{1-V^*(\lambda)}\right)} + \frac{1}{\lambda} + \frac{V^{*(1)}(\lambda)}{1-V^*(\lambda)} + \frac{1}{\mu}
\end{aligned}$$

and

$$\begin{aligned}
\boldsymbol{E}[\Delta_{CBS-P}] &= \frac{\frac{V^{*(2)}(0)}{1-V^*(\lambda)} + 2\frac{V^{*(1)}(0)V^{*(1)}(\lambda)}{(1-V^*(\lambda))^2} - 2\frac{V^{*(1)}(0)}{1-V^*(\lambda)}\frac{1-H^*(\lambda)}{\lambda H^*(\lambda)} + \frac{2}{\lambda H^*(\lambda)^2}\left[\frac{1}{\lambda} - \frac{H^*(\lambda)}{\lambda} + H^{*(1)}(\lambda)\right]}{2(-\frac{V^{*(1)}(0)}{1-V^*(\lambda)} + \frac{1-H^*(\lambda)}{\lambda H^*(\lambda)})} \\
&\quad - \frac{H^{*(1)}(\lambda)}{H^*(\lambda)} + H^*(\lambda)\left(\frac{1}{\lambda} + \frac{V^{*(1)}(\lambda)}{1-V^*(\lambda)}\right) \\
&= \frac{\frac{V^{*(2)}(0)}{1-V^*(\lambda)} + 2\frac{V^{*(1)}(0)V^{*(1)}(\lambda)}{(1-V^*(\lambda))^2} - 2\frac{V^{*(1)}(0)}{1-V^*(\lambda)}\frac{1}{\mu} + \frac{2}{\mu^2}}{2(-\frac{V^{*(1)}(0)}{1-V^*(\lambda)} + \frac{1}{\mu})} \\
&\quad + \frac{\frac{\mu}{(\mu+\lambda)^2}}{\frac{\mu}{\mu+\lambda}} + \frac{\mu}{\mu+\lambda}\left(\frac{1}{\lambda} + \frac{V^{*(1)}(\lambda)}{1-V^*(\lambda)}\right).
\end{aligned}$$

Therefore, by using Lemma B.2.1, we have

$$\begin{aligned}
\boldsymbol{E}[\Delta_{CBS}] - \boldsymbol{E}[\Delta_{CBS-P}] &= \frac{1}{\mu} + \frac{\lambda}{\mu+\lambda}\frac{V^{*(1)}(\lambda)}{1-V^*(\lambda)} \\
&\geq \frac{1}{\mu} - \frac{\lambda}{\mu+\lambda}\frac{1}{\lambda} \\
&= \frac{\lambda}{\mu(\mu+\lambda)} \geq 0.
\end{aligned}$$

Now we show the result holds true for PAoI. Since we have

$$\boldsymbol{E}[A_{CBS}] = \frac{1}{\lambda} + \frac{V^{*(1)}(\lambda) - V^{*(1)}(0)}{1-V^*(\lambda)} + \frac{2}{\mu}$$

and

$$\begin{aligned}
\boldsymbol{E}[A_{CBS-P}] &= \frac{1 - H^*(\lambda) - \lambda H^{*(1)}(\lambda) + H^*(\lambda)^2}{\lambda H^*(\lambda)} + \frac{H^*(\lambda)V^{*(1)}(\lambda) - V^{*(1)}(0)}{1-V^*(\lambda)} \\
&= \frac{1 - \frac{\mu}{\mu+\lambda} + \frac{\lambda\mu}{(\mu+\lambda)^2} + (\frac{\mu}{\mu+\lambda})^2}{\frac{\lambda\mu}{\mu+\lambda}} + \frac{\frac{\mu}{\mu+\lambda}V^{*(1)}(\lambda) - V^{*(1)}(0)}{1-V^*(\lambda)},
\end{aligned}$$

then

$$\begin{aligned}\boldsymbol{E}[A_{CBS}]-\boldsymbol{E}[A_{CBS-P}] &= \frac{1}{\lambda}+\frac{1}{\mu}-\frac{1}{\lambda}+\frac{\lambda}{\mu+\lambda}\frac{V^{*(1)}(\lambda)}{1-V^{*}(\lambda)}\\ &\geq \frac{1}{\mu}-\frac{1}{\mu+\lambda}\geq 0.\end{aligned}$$

□

B.3 Proof for Theorem 5.3.10

Theorem 5.3.10. If the service time H satisfies $\boldsymbol{E}[H]\geq\frac{1-H(s)}{\lambda H(s)}$ for all $s>0$, then CBS-P always has smaller PAoI than CBS.

Proof. We first have

$$\begin{aligned}\boldsymbol{E}[A_{CBS}]-\boldsymbol{E}[A_{CBS-P}] &= \frac{1}{\lambda}+\frac{V^{*(1)}(\lambda)-V^{*(1)}(0)}{1-V^{*}(\lambda)}+2\boldsymbol{E}[H]\\ &\quad -\frac{1-H^{*}(\lambda)-\lambda H^{*(1)}(\lambda)+H^{*}(\lambda)^{2}}{\lambda H^{*}(\lambda)}-\frac{H^{*}(\lambda)V^{*(1)}(\lambda)-V^{*(1)}(0)}{1-V^{*}(\lambda)}\\ &= (1-H^{*}(\lambda))\left(\frac{1}{\lambda}+\frac{V^{*(1)}(\lambda)}{1-V^{*}(\lambda)}\right)+2\boldsymbol{E}[H]\\ &\quad -\frac{1-H^{*}(\lambda)-\lambda H^{*(1)}(\lambda)}{\lambda H^{*}(\lambda)}.\end{aligned}$$

From Lemma B.2.1 we know that $\frac{1}{\lambda}+\frac{V^{*(1)}(\lambda)}{1-V^{*}(\lambda)}\geq 0$ and $1-H^{*}(\lambda)\geq-\lambda H^{*(1)}(\lambda)$, then we have

$$\boldsymbol{E}[A_{CBS}]-\boldsymbol{E}[A_{CBS-P}] \geq 2\boldsymbol{E}[H]-2\frac{1-H^{*}(\lambda)}{H^{*}(\lambda)}\geq 0.$$

□

B.4 Proof for Theorem 5.3.11

Theorem 5.3.11. For M/G/1/1 we have $\boldsymbol{E}[A_{M/G/1/1}]=\frac{1}{\lambda}-2H^{*(1)}(0)$, $\boldsymbol{E}[\Delta_{M/G/1/1}]=\frac{\frac{2}{\lambda^{2}}-\frac{2}{\lambda}H^{*(1)}(0)+H^{*(2)}(0)}{\frac{1}{\lambda}-H^{*(1)}(0)}-H^{*(1)}(0)$ and $Var(A_{M/G/1/1})=\frac{1}{\lambda^{2}}+2H^{*(2)}(0)-2\{H^{*(1)}(0)\}^{2}$.

For M/G/1/1/preemptive we have $\boldsymbol{E}[A_{M/G/1/1/preemptive}] = \frac{-H^{*(1)}(\lambda)}{H^{*}(\lambda)} + \frac{1}{\lambda H^{*}(\lambda)}$, $\boldsymbol{E}[\Delta_{M/G/1/1/preemptive}] = \frac{1}{\lambda H^{*}(\lambda)}$, and $Var(A_{M/G/1/1/preemptive}) = \frac{H^{*(2)}(\lambda)}{H^{*}(\lambda)} - \frac{\{H^{*(1)}(\lambda)\}^2}{H^{*}(\lambda)^2} + \frac{1}{\lambda^2 H^{*}(\lambda)^2} + \frac{2H^{*(1)}(\lambda)}{\lambda H^{*}(\lambda)^2}$.

For M/G/1/2* we have $\boldsymbol{E}[A_{M/G/1/2^*}] = -2H^{*(1)}(0) + \frac{1}{\lambda} + H^{*(1)}(\lambda)$, $\boldsymbol{E}[\Delta_{M/G/1/2^*}] = \frac{\frac{1}{2}H^{*(2)}(0) + \frac{1}{\lambda^2}H^{*}(\lambda) - \frac{1}{\lambda}H^{*(1)}(\lambda)}{-H^{*(1)}(0) + \frac{H^{*}(\lambda)}{\lambda}} + \frac{1}{\lambda} - \frac{1}{\lambda}H^{*}(\lambda) + H^{*(1)}(\lambda) - H^{*(1)}(0)$, and $Var(A_{M/G/1/2^*}) = 2H^{*(2)}(0) - 2H^{*(1)}(0) + \frac{2H^{*}(\lambda)[H^{*(1)}(0)+H^{*(1)}(\lambda)]}{\lambda} + \frac{2H^{*}(\lambda)(1-H^{*}(\lambda))}{\lambda^2} + \frac{1}{\lambda^2} - H^{*(2)}(\lambda) - \frac{2}{\lambda}H^{*(1)}(\lambda) - H^{*(1)}(\lambda)^2$.

Proof. We first show the variance of peak age in M/G/1/1. Realizing that in M/G/1/1 system, once a packet arrives, the server will start processing it immediately. Thus there is no waiting time for all packets. Then the LST of peak age in M/G/1/1 can be given as $A^*(s) = I^*(s)H^*(s)$. The inter-service time I can be further decomposed into the idling time T (exponentially distributed) and service time H, i.e., $I = T + H$. We thus have $A^*(s) = T^*(s)H^*(s)^2$. By some simple algebra we have $\boldsymbol{E}[A_{M/G/1/1}] = \frac{1}{\lambda} - 2H^{*(1)}(0)$, $\boldsymbol{E}[\Delta_{M/G/1/1}] = \frac{\frac{2}{\lambda^2} - \frac{2}{\lambda}H^{*(1)}(0) + H^{*(2)}(0)}{\frac{1}{\lambda} - H^{*(1)}(0)} - H^{*(1)}(0)$ and $Var(A_{M/G/1/1}) = \frac{1}{\lambda^2} + 2H^{*(2)}(0) - 2\{H^{*(1)}(0)\}^2$.

Similarly, for M/G/1/1/preemptive system, there is no waiting time for packets. Thus by the argument in Appendix B.1, we have $D^*(s) = \frac{H^*(s+\lambda)}{H^*(\lambda)}$. Then the LST of peak age can be given as $A^*(s) = D^*(s)T^*(s)L^*(s)$, where $L^*(s)$ is given by Equation (B.4). We then have $\boldsymbol{E}[A_{M/G/1/1/preemptive}] = \frac{-H^{*(1)}(\lambda)}{H^{*}(\lambda)} + \frac{1}{\lambda H^{*}(\lambda)}$ and $\boldsymbol{E}[\Delta_{M/G/1/1/preemptive}] = \frac{1}{\lambda H^{*}(\lambda)}$. And the variance of peak age of M/G/1/1/preemptive system is given by $Var(A_{M/G/1/1/preemptive}) = \frac{H^{*(2)}(\lambda)}{H^{*}(\lambda)} - \frac{\{H^{*(1)}(\lambda)\}^2}{H^{*}(\lambda)^2} + \frac{1}{\lambda^2 H^{*}(\lambda)^2} + \frac{2H^{*(1)}(\lambda)}{\lambda H^{*}(\lambda)^2}$.

For M/G/1/2* system, the inter-service time is H if there is an arrival during processing time. If there is no arrival during processing time, the next service starts when the next arrival occurs. By memoryless property of Poisson arrivals, we have $I = T$ in this case. Therefore $I = \max\{H, T\}$.

To calculate the LST of I, by letting $F(h)$ be the CDF of H, we have

$$\begin{aligned}
I^*(s) = \boldsymbol{E}[e^{-sI}] &= \int_{h=0}^{\infty}\int_{t=h}^{\infty}\lambda e^{-\lambda t}e^{-st}dF(h)dt + \int_{h=0}^{\infty}\int_{t=0}^{h}e^{-sh}\lambda e^{-\lambda t}dF(h)dt \\
&= \frac{\lambda}{\lambda+s}H^*(s+\lambda) + H^*(s) - H^*(s+\lambda) \\
&= H^*(s) - \frac{s}{\lambda+s}H^*(s+\lambda).
\end{aligned}$$

We can then have

$$I^{*(1)}(0) = H^{*(1)}(0) - \frac{H^*(\lambda)}{\lambda},$$

and

$$I^{*(2)}(0) = H^{*(2)}(0) + \frac{2}{\lambda^2}H^*(\lambda) - \frac{2}{\lambda}H^{*(1)}(\lambda).$$

The waiting time only occurs when there is an arrival during processing time H, so that $W = \max\{H-T,0\}$. The LST of W is thus be given as

$$\begin{aligned}
W^*(s) &= \int_{h=0}^{\infty}\int_{t=0}^{h}e^{-s(h-t)}dF(h)\lambda e^{-\lambda t}dt + \int_{h=0}^{\infty}dF(h)\int_{t=h}^{\infty}\lambda e^{-\lambda t}dt \\
&= \frac{\lambda}{\lambda-s}(H^*(s)-H^*(\lambda)) + H^*(\lambda) \\
&= \frac{\lambda}{\lambda-s}H^*(s) - \frac{s}{\lambda-s}H^*(\lambda).
\end{aligned}$$

From Lemma 5.3.1 we have

$$\begin{aligned}
G^*(s) &= \frac{\lambda}{\lambda+s} + \frac{s}{\lambda+s}W^*(\lambda+s) \\
&= \frac{\lambda}{\lambda+s} - \frac{\lambda}{\lambda+s}H^*(\lambda+s) + H^*(\lambda).
\end{aligned}$$

By taking the first and second derivative of $G^*(s)$, we have

$$G^{*(1)}(0) = -\frac{1}{\lambda} + \frac{1}{\lambda}H^*(\lambda) - H^{*(1)}(\lambda)$$

and

$$G^{*(2)}(0) = \frac{2}{\lambda^2} - \frac{2}{\lambda^2}H^*(\lambda) + \frac{2}{\lambda}H^{*(1)}(\lambda) - H^{*(2)}(\lambda).$$

By Equation (5.1) and (5.2), we have

$$\boldsymbol{E}[A_{M/G/1/2^*}] = -2H^{*(1)}(0) + \frac{1}{\lambda} + H^{*(1)}(\lambda)$$

and

$$\boldsymbol{E}[\Delta_{M/G/1/2^*}] = \frac{\frac{1}{2}H^{*(2)}(0) + \frac{1}{\lambda^2}H^*(\lambda) - \frac{1}{\lambda}H^{*(1)}(\lambda)}{-H^{*(1)}(0) + \frac{H^*(\lambda)}{\lambda}} + \frac{1}{\lambda} - \frac{1}{\lambda}H^*(\lambda) + H^{*(1)}(\lambda) - H^{*(1)}(0).$$

Using Equation (5.4), we have the variance of peak age

$$\begin{aligned}
Var(A_{M/G/1/2^*}) &= \frac{2}{\lambda^2} - \frac{2}{\lambda^2}H^*(\lambda) + \frac{2}{\lambda}H^{*(1)}(\lambda) - H^{*(2)}(\lambda) - \left[-\frac{1}{\lambda} + \frac{1}{\lambda}H^*(\lambda) - H^{*(1)}(\lambda)\right]^2 \\
&\quad + H^{*(2)}(0) + \frac{2}{\lambda^2}H^*(\lambda) - \frac{2}{\lambda}H^{*(1)}(\lambda) - \left[H^{*(1)}(0) - \frac{H^*(\lambda)}{\lambda}\right]^2 \\
&\quad + H^{*(2)}(0) - H^{*(1)}(0)^2 \\
&= 2H^{*(2)}(0) - 2H^{*(1)}(0) + \frac{2H^*(\lambda)[H^{*(1)}(0) + H^{*(1)}(\lambda)]}{\lambda} \\
&\quad + \frac{2H^*(\lambda)(1 - H^*(\lambda))}{\lambda^2} + \frac{1}{\lambda^2} - H^{*(2)}(\lambda) - \frac{2}{\lambda}H^{*(1)}(\lambda) - H^{*(1)}(\lambda)^2.
\end{aligned}$$

□

B.5 Exact Solution for PAoI in CBS-P with Dependent Vacation

Notice that in CBS-P, the server's vacation time B can be divided into $B = T + W$, where T is the inter-arrival time of packets which is exponentially distributed, and W is the time when the buffer is occupied (the same as their definition in Section 5.4). Because of the memoryless property of exponential distribution, we have $\boldsymbol{E}[B] = \frac{1}{\lambda} + \boldsymbol{E}[W]$.

By Equation (5.6), (B.1), (B.3), and (B.5), the PAoI for CBS-P can be written as

$$\begin{aligned}
\boldsymbol{E}[A] &= \boldsymbol{E}[D] + \boldsymbol{E}[B] + \boldsymbol{E}[L] \\
&= -\frac{H^{*(1)}(\lambda)}{H^*(\lambda)} + H^*(\lambda)\frac{1}{\lambda}(1 - W^*(\lambda)) + \frac{1}{\lambda} + \boldsymbol{E}[W] + \frac{1 - H^*(\lambda)}{\lambda H^*(\lambda)} \\
&= -\frac{H^{*(1)}(\lambda)}{H^*(\lambda)} + H^*(\lambda)\frac{1}{\lambda}(1 - W^*(\lambda)) + \boldsymbol{E}[W] + \frac{1}{\lambda H^*(\lambda)}.
\end{aligned}$$

B.6 Proof for Theorem 5.4.1

Theorem 5.4.1. If the service time for each queue is exponentially distributed, then CBPS-P will always have smaller PAoI than CBPS.

Proof. When the service time is exponentially distributed, from Equation (5.9), we have

$$L_j^*(s) = \frac{H_j^*(s+\lambda_j)}{\frac{s}{s+\lambda_j} + \frac{\lambda_j}{s+\lambda_j} H_j^*(s+\lambda_j)} = \frac{\frac{\mu_j}{s+\lambda_j+\mu_j}}{\frac{s}{s+\lambda_j} + \frac{\lambda_j}{s+\lambda_j}\frac{\mu_j}{s+\lambda_j+\mu_j}} = \frac{\mu_j}{s+\mu_j}.$$

So that the expressions for $\tilde{H}_j^*$ in Equation (5.8) are identical for CBPS and CBPS-P. Both systems will have the same $F_j(z_1, ..., z_k)$ for all j after solving for Equation (5.7). Similarly, since $\frac{1-H_j^*(\lambda_j)}{\lambda_j H^*(\lambda_j)} = \frac{1}{\mu_j}$, both CBPS and CBPS-P will have the same expression for γ_j in Equation (5.10) for all j. Therefore, CBPS and CBPS-P have the same expressions for $W_j^*(\lambda_j)$ and $\boldsymbol{E}[W_j]$ for all queue j.

We then have

$$
\begin{aligned}
& \boldsymbol{E}[A_j^{CBPS}] - \boldsymbol{E}[A_j^{CBPS-P}] \\
= \; & -\frac{1}{\lambda_j} W_j^*(\lambda_j) + \frac{2}{\lambda_j} + \boldsymbol{E}[W_j] + 2\boldsymbol{E}[H_j] \\
& - \left[-\frac{H_j^{*(1)}(\lambda_j)}{H_j^*(\lambda_j)} + H_j^*(\lambda_j) \frac{1}{\lambda_j}(1 - W_j^*(\lambda_j)) + \frac{1}{\lambda_j} + \boldsymbol{E}[W_j] + \frac{1 - H_j^*(\lambda_j)}{\lambda_j H_j^*(\lambda_j)} \right] \\
= \; & \big(1 - H_j^*(\lambda_j)\big) \frac{1}{\lambda_j} \big(1 - W_j^*(\lambda_j)\big) + \frac{1}{\mu_j} - \frac{1}{\mu_j + \lambda_j} \geq 0.
\end{aligned}
$$

$\square$