

Information Freshness Optimization in Real-time Network Applications

Zhongdong Liu

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Applications

Bo Ji, Chair

Wenjing Lou

Y. Thomas Hou

Taejoong Chung

Jie Wu

Yu Wang

April 9, 2024

Blacksburg, Virginia

Keywords: Information freshness, Age-of-Information, latency, transmission cost, Internet of Things, optimization, machine learning algorithm, algorithm design

Copyright 2024, Zhongdong Liu

Information Freshness Optimization in Real-time Network Applications

Zhongdong Liu

(ABSTRACT)

In recent years, the remarkable development in ubiquitous communication networks and smart portable devices spawned a wide variety of real-time applications that require timely information updates (e.g., autonomous vehicular systems, industrial automation systems, and live streaming services). These real-time applications all have one thing in common: they desire their knowledge of the information source to be as fresh as possible. In order to measure the freshness of information, a new metric, called the *Age-of-Information (AoI)* is proposed. AoI is defined as the time elapsed since the generation time of the freshest delivered update. This metric is influenced by both the inter-arrival time and the delay of the updates. As a result of these dependencies, the AoI metric exhibits distinct characteristics compared to traditional delay and throughput metrics.

In this dissertation, our goal is to optimize AoI under various real-time network applications. Firstly, we investigate a fundamental problem of how exactly various scheduling policies impact AoI performance. Though there is a large body of work studying the AoI performance under different scheduling policies, the use of the update-size information and its combinations with other information (such as arrival-time information and service pre-emption) to reduce AoI has still not been explored yet. Secondly, as a recently introduced measure of freshness, the relationship between AoI and other performance metrics remains largely ambiguous. We analyze the tradeoffs between AoI and additional performance metrics, including service performance and update cost, within real-world applications.

This dissertation is organized into three parts. In the first part, we realize that scheduling

policies leveraging the update-size information can substantially reduce the delay, one of the key components of AoI. However, it remains largely unknown how exactly scheduling policies (especially those making use of update-size information) impact the AoI performance. To this end, we conduct a systematic and comparative study to investigate the impact of scheduling policies on the AoI performance in single-server queues and provide useful guidelines for the design of AoI-efficient scheduling policies.

In the second part, we analyze the tradeoffs between AoI and other performance metrics in real-world systems. Specifically, we focus on the following two important tradeoffs. (i) The tradeoff between service performance and AoI that arises in the data-driven real-time applications (e.g., Google Maps and stock trading applications). In these applications, the computing resource is often shared for processing both updates from information sources and queries from end users. Hence there is a natural tradeoff between service performance (e.g., response time to queries) and AoI (i.e., the freshness of data in response to user queries). To address this tradeoff, we begin by introducing a simple single-server two-queue model that captures the coupled scheduling between updates and queries. Subsequently, we design threshold-based scheduling policies to prioritize either updates or queries. Finally, we conduct a rigorous analysis of the performance of these threshold-based scheduling policies. (ii) The tradeoff between update cost and AoI that appear in the crowdsensing-based applications (e.g., Google Waze and GasBuddy). On the one hand, users are not satisfied if the responses to their requests are stale; on the other side, there is a cost for the applications to update their information regarding certain points of interest since they typically need to make monetary payments to incentivize users. To capture this tradeoff, we first formulate an optimization problem with the objective of minimizing the sum of the staleness cost (which is a function of the AoI) and the update cost, then we obtain a closed-form optimal threshold-based policy by reformulating the problem as a Markov decision process (MDP).

In the third part, we study the minimization of data freshness and transmission costs (e.g., energy cost) under an (arbitrary) time-varying wireless channel without and with machine learning (ML) advice. We consider a discrete-time system where a resource-constrained source transmits time-sensitive data to a destination over a time-varying wireless channel. Each transmission incurs a fixed cost, while not transmitting results in a staleness cost measured by the AoI. The source needs to balance the tradeoff between these transmission and staleness costs. To tackle this challenge, we develop a robust online algorithm aimed at minimizing the sum of transmission and staleness costs, ensuring a worst-case performance guarantee. While online algorithms are robust, they tend to be overly conservative and may perform poorly on average in typical scenarios. In contrast, ML algorithms, which leverage historical data and prediction models, generally perform well on average but lack worst-case performance guarantees. To harness the advantages of both approaches, we design a learning-augmented online algorithm that achieves two key properties: (i) *consistency*: closely approximating the optimal offline algorithm when the ML prediction is accurate and trusted; (ii) *robustness*: providing a worst-case performance guarantee even when ML predictions are inaccurate.

Information Freshness Optimization in Real-time Network Applications

Zhongdong Liu

(GENERAL AUDIENCE ABSTRACT)

In recent years, the rapid growth of communication networks and smart devices has spurred the emergence of real-time applications like autonomous vehicles and industrial automation systems. These applications share a common need for timely information. The freshness of information can be measured using a new metric called Age-of-Information (AoI). This dissertation aims to optimize AoI across various real-time network applications, organized into three parts. In the first part, we explore how scheduling policies (particularly those considering update size) impact the AoI performance. Through a systematic and comparative study in single-server queues, we provide useful guidelines for the design of AoI-efficient scheduling policies. The second part explores the tradeoff between update cost and AoI in crowdsensing applications like Google Waze and GasBuddy, where users demand fresh responses to their requests; however, updating information incurs update costs for applications. We aim to minimize the sum of staleness cost (a function of AoI) and update cost. By reformulating the problem as a Markov decision process (MDP), we design a simple threshold-based policy and prove its optimality. In the third part, we study the minimization of data freshness and transmission costs (e.g., energy cost) under a time-varying wireless channel. We first develop a robust online algorithm that achieves a competitive ratio of 3, ensuring a worst-case performance guarantee. Furthermore, when advice is available, e.g., predictions from machine learning (ML) models, we design a learning-augmented online algorithm that exhibits two desired properties: (i) consistency: closely approximating the optimal offline algorithm when the ML prediction is accurate and trusted; (ii) robustness: guaranteeing

worst-case performance even with inaccurate ML prediction. While this dissertation marks a significant advancement in AoI research, numerous open problems remain. For instance, our learning-augmented online algorithm treats ML predictions as external inputs. Exploring the co-design and training of ML and online algorithms to improve performance could yield interesting insights. Additionally, while AoI typically assesses update importance based solely on timestamps, the content of updates also holds significance. Incorporating considerations of both age and semantics of information is imperative in future research.

Dedication

To my beloved family.

Acknowledgments

First of all, I would like to express my most sincere gratitude to my esteemed Ph.D. advisor, Dr. Bo Ji, the College of Engineering Faculty Fellow at Virginia Tech. I was fortunate to be admitted to Dr. Ji's group and given the opportunity to pursue a Ph.D. degree. During the past seven years, Dr. Ji has consistently exhibited extraordinary dedication, support, and patience toward my academic pursuits. He invested numerous hours collaborating with me to discuss, define, and refine research problems and solutions. Under his guidance, I not only learned to conduct academic research to high standards but also gained insights into cultivating responsibility, professionalism, and a forward-thinking approach essential for my future career beyond the realm of doctoral studies. The most profound lesson I learned from the extensive hours spent working alongside Dr. Ji is the way of looking at problems and solutions. This encompasses crucial aspects such as discerning the big picture, identifying the essence of the problem, and grasping the intuition behind potential solutions. Completing this dissertation and all my research endeavors would have been impossible without the indispensable assistance and guidance provided by Dr. Ji. The invaluable experience of learning and collaborating with him stands as the most precious aspect of my Ph.D. journey.

My deepest gratitude also extends to Dr. Bin Li (The Pennsylvania State University), who collaborated closely with me throughout my Ph.D. studies and provided precious assistance with this dissertation. His hands-on teaching has been immensely beneficial to me. I also want to thank my Ph.D. committee members: Dr. Wenjing Lou, Dr. Y. Thomas Hou, Dr. Taejoong Chung, Dr. Jie Wu (Temple University), and Dr. Yu Wang (Temple University), for generously dedicating their time and providing valuable feedback on this dissertation. Their insightful advice has significantly enhanced the quality of this work in multiple aspects.

I would also like to thank all my current and former colleagues in the SNAIL Lab for their help in my research endeavors and their cherished friendship: Fengjiao Li, Keyuan Zhang, Yu Sang, Gamal Sallam, Duo Cheng, Osama Bajaber, and Matthew Corbett. I am particularly indebted to our group's alumni, Yu Sang and Gamal Sallam, who have provided me with guidance and support since my inception into the SNAIL Lab, for which I am immensely grateful. Furthermore, I wish to express my sincere thanks to the friends I have made in Blacksburg: Shaoran Li, Yan Huang, Yubo Wu, Chengzhang Li, Yongce Chen, and Mingsi Liao. Their kindness and support have been instrumental as I navigated the beginnings of my life in Blacksburg.

Last but not least, I wish to express my deepest gratitude to my parents, Shanxing Liu and Yingmei Yu, for their unwavering love and dedication in raising me. Their tireless efforts and diligence provided me with a joyful and nurturing childhood, laying the foundation for all the fulfillment I experience in adulthood. This dissertation stands as a testament to the profound impact of their unconditional love. Additionally, I extend my heartfelt thanks to my sister, Chunfang Liu, for her love, understanding, and encouragement that has been constant throughout my life.

Funding Acknowledgments

This research was supported in part by the National Science Foundation (NSF) under Grants CCF-1657162 and CNS-2112694, and ONR MURI Grant N00014-19-1-2621.

Contents

List of Figures	xvi
List of Tables	xxiii
1 Introduction	1
1.1 Motivation and Objectives	1
1.2 Dissertation Outline and Contributions	4
2 Anti-Aging Scheduling in Single-Server Queues: A Systematic and Comparative Study	8
2.1 Introduction	8
2.2 Related work	12
2.3 System Model	14
2.4 Size-based policies	15
2.5 AoI-based policies	24
2.6 Preemptive, informative, AoI-based Policies	28
2.6.1 Informative Policies	29
2.6.2 Preemptive, Informative, AoI-based Policies	35
2.7 Chapter Summary	39

3	Towards the Tradeoff Between Service Performance and Information Freshness	41
3.1	Introduction	41
3.2	Related Work	43
3.3	System Model	45
3.4	Scheduling Policies	48
3.4.1	The FCFS Policy	48
3.4.2	The Query- k Policy	51
3.4.3	The Update- k Policy	55
3.4.4	The Joint- (M, N) Policy	57
3.5	Numerical Results	58
3.6	Chapter Summary	60
4	Towards Optimal Tradeoff Between Data Freshness and Update Cost in Information-update Systems	62
4.1	Introduction	62
4.2	Related Work	65
4.3	System Model and Problem Formulation	67
4.4	Guidelines for Algorithm Design	69
4.4.1	Reactive Policies	70
4.4.2	Capped Reactive Policies	71

4.5	MDP Formulation and Threshold structure	73
4.6	Optimal Threshold-based Policy	76
4.7	Numerical and Experimental Results	79
4.8	Chapter Summary	85
5	Learning-augmented Online Minimization of Age of Information and Trans-	
	mission Costs	86
5.1	Introduction	86
5.2	Related Work	89
5.3	System Model and Problem Formulation	90
5.4	Robust Online Algorithm	93
	5.4.1 Problem Reformulation	93
	5.4.2 Primal-dual Online Algorithm Description and Analysis	96
5.5	Learning-augmented Online Algorithm	100
	5.5.1 Machine Learning Predictions	100
	5.5.2 Learning-augmented Online Algorithm Description	101
	5.5.3 Learning-augmented Online Algorithm Analysis	103
5.6	Numerical Results	106
	5.6.1 Online Algorithm	107
	5.6.2 Learning-augmented Online Algorithm	108
5.7	Chapter Summary	113

6 Summary and Future Work	114
6.1 Summary	114
6.2 Future Work	116
Appendices	119
Appendix A Appendix of Chapter 2	120
A.1 Additional Simulation Results for the G/G/1 Queue	120
Appendix B Proofs of Chapter 4	124
B.1 Proof of Lemma 4.1	124
B.2 Proof of Lemma 4.2	126
B.3 Proof of Theorem 4.3	128
B.4 Proof of Lemma B.1	131
B.5 Proof of Lemma B.2	134
B.6 Proof of Theorem 4.5	138
Appendix C Proofs of Chapter 5	141
C.1 Proof of Lemma 5.1	141
C.2 Proof of Theorem 5.3	144
C.3 Proof of Lemma 5.6	153
C.4 Proof of Lemma 5.7	161

List of Figures

1.1	Dissertation structure	4
2.1	Our position in the design space of AoI-efficient scheduling policies for a G/G/1 queue	9
2.2	An example of the AoI evolution under the FCFS policy	15
2.3	Comparisons of the average AoI performance under several common scheduling policies	18
2.4	Comparisons of the average PAoI performance under several common scheduling policies	18
2.5	An example of the AoI/PAoI evolution where the interarrival time has a large variability	22
2.6	An example of the AoI/PAoI evolution where the service time has a large variability	24
2.7	The AoI evolution under three AoI-based policies: ADE (red), ADS (blue), and ADM (green)	25
2.8	An example of the AoI evolution under ADE and SJF	27
2.9	Comparisons of the average AoI performance: AoI-based policies vs. non-AoI-based policies	28
2.10	Comparisons of the average PAoI performance: AoI-based policies vs. non-AoI-based policies	28

2.11	Comparisons of the average AoI performance: informative policies vs. non-informative policies	30
2.12	Comparisons of the average PAoI performance: informative policies vs. non-informative policies	30
2.13	Part of sample path of $\hat{S}_{\text{LCFS_I}}(t)$ and $\hat{S}_{\text{LCFS}}(t)$ in different cases	33
2.14	Comparisons of the average AoI performance: preemptive, informative, AoI-based policies vs. others	39
2.15	Comparisons of the average PAoI performance: preemptive, informative, AoI-based policies vs. others	39
3.1	An example of the AoI evolution	46
3.2	Performance comparisons of different policies with varying update load ($\lambda_q = 0.1$ and $\mu_q = \mu_u = 1$)	58
3.3	Performance comparisons of different policies with varying query load ($\lambda_u = 0.1$ and $\mu_q = \mu_u = 1$)	59
3.4	Impact of the threshold on the single-threshold-based policies ($\lambda_u = \lambda_q = 1/3$ and $\mu_u = \mu_q = 1$)	60
3.5	Impact of different values of the thresholds M and N on the Joint- (M, N) policy ($\lambda_u = \lambda_q = 1/3$ and $\mu_u = \mu_q = 1$)	60
4.1	An illustration of our system model. Upon receiving a request from the users, the server can either first update the data and then reply (red path: A1-A4) or simply reply with local data (blue path: B1-B2).	63

4.2	An illustration of the AoI evolution at the server. There are two updates (in time-slots u_1 and u_2) during the process of serving three requests (in time-slots r_1 , r_2 , and r_3).	69
4.3	The relationship between online policies Π , reactive policies Π^R , capped reactive policies Π^{R+} , threshold-based policies Π^T , an overall optimal policy π^* (see Theorem 4.3), and an optimal threshold-based policy $\pi(\tau^*)$ (see Corollary 4.6). Note that policy $\pi(\tau^*)$ is also an overall optimal policy and could be the same as policy π^* in some cases.	72
4.4	Average cost under threshold-based policies with different thresholds when $f(\Delta) = \Delta$, where $\lambda = 0.1$ and $p = 100$.	80
4.5	Average cost under threshold-based policies with different thresholds when $f(\Delta) = \Delta^2$, where $\lambda = 0.1$ and $p = 100$.	80
4.6	Performance comparisons of different policies with different request arrival rate λ and different update cost p when $f(\Delta) = \Delta$, respectively.	82
4.7	Performance comparisons of different policies using trace dataset, where update cost $p = 25$, request arrival rate $\lambda = 0.4$, and $f(\Delta) = \Delta$.	82
4.8	Performance comparisons of different policies with different request arrival rate λ and different update cost p when $f(\Delta) = \Delta^2$, respectively.	82
4.9	Performance comparisons of different policies using trace dataset, where update cost $p = 50$, request arrival rate $\lambda = 0.4$, and $f(\Delta) = \Delta^2$.	82

5.1	The updates of primal variables $z_i(t)$ and dual variables $y_i(t)$ in the k -th ACK interval $[t_k + 1, t_{k+1}]$ of PDOA, where channels are OFF during $[t_k + 5, t_k + 7]$. The x-axis represents time and the y-axis represents the packet id. PDOA makes two ACKs at slot t_k and slot t_{k+1} , where the ACK cost $c = 18$. The primal variables $z_i(t)$ and dual variables $y_i(t)$ are updated from slot $t_k + 1$ to slot t_{k+1} ; and in slot t , packets are updated from packet $t_k + 1$ to packet t . The red bold italic 1 denotes when the ACK marker equals or is larger than 1. In Fig. 5.1(a), the grey areas denote the updates due to Line 15; In Fig. 5.1(b), the grey areas denote the updates due to Lines 19-27.	98
5.2	The updates of variables in the k -th ACK interval $[t_k + 1, t_{k+1}]$ of LAPDOA, where channels are OFF during $[t_k + 5, t_k + 7]$. LAPDOA makes two ACKs at t_k and t_{k+1} , and the ML prediction $\mathcal{P}$ makes its i -th ACK at slot $t_k + 4$. The red bold italic value denotes when the ACK marker $M \geq 1$. Let $m_1 = \lambda/c$, $m_2 = 1/\lambda c$, $y_1 = \lambda$, and $y_2 = 1$. The light grey area denotes the small updates, the white area (without background) denotes the big updates, and the dark grey area denotes the zero updates.	104
5.3	Performance comparison of online algorithms under different datasets.	107
5.4	Performance comparison of LAPDOA under different trust parameters using synthetic dataset.	110
5.5	Average cost ratio vs. empirical CR of LAPDOA when the pattern sequence percentage is 99%. The direction of the arrow indicates that λ becomes larger.	110
5.6	Average cost ratio vs. empirical CR of LAPDOA using real trace dataset. The direction of the arrow indicates that λ becomes larger.	112

A.1	Comparisons of the average AoI performances of several common scheduling policies under different distributions	120
A.2	Comparisons of the average PAoI performances of several common scheduling policies under different distributions	121
A.3	Comparisons of the average AoI performance under different distributions: AoI-based policies vs. non-AoI-based policies	121
A.4	Comparisons of the average PAoI performance under different distributions: AoI-based policies vs. non-AoI-based policies	121
A.5	Comparisons of the avg. AoI performance under different distributions: informative policies vs. non-informative policies	122
A.6	Comparisons of the avg. PAoI performance under different distributions: informative policies vs. non-informative policies	122
A.7	Comparisons of the avg. AoI performance under different distributions: preemptive, informative, AoI-based policies vs. others	122
A.8	Comparisons of the avg. PAoI performance under different distributions: preemptive, informative, AoI-based policies vs. others	123
B.1	An illustration of the advantage of a constructed reactive policy $\pi' \in \Pi^R$ over an arbitrary given non-reactive policy $\pi \in \Pi \setminus \Pi^R$, where the solid black curve and the dashed red curve denote the AoI trajectories under policies π and π' , respectively.	126

B.2	An illustration of the advantage of a constructed capped reactive policy $\pi' \in \Pi^{R+}$ over an arbitrary given non-capped reactive policy $\pi \in \Pi^R \setminus \Pi^{R+}$, where the solid black curve and the dashed red curve denote the AoI trajectories under policies π and π' , respectively.	126
C.1	The updates of primal variables $z_i(t)$ and dual variables $y_i(t)$ in the k -th ACK interval $[t_k + 1, t_{k+1}]$, where channels are OFF during $[t' - 1, t' + 1]$. The red bold italic 1 denotes when the ACK marker equals or is larger than 1. In addition, $T_1(k)$ is an equilateral triangle made of 1 (the underlined 1's with yellow background), $T_2(k)$ is an equilateral triangle made of 1 (the bold 1's with green background), and $R_1(k)$ is a rectangle made of 1 and 0 (the regular 1's and 0's without background).	149
C.2	An illustration of $C_H^1(k), C_H^2(k), C_H^3(k)$ and $D^1(k)$ when $j \neq t'$. $C_H^1(k)$ is an equilateral triangle made of 1 (the underlined 1's with yellow background in Fig. C.2(a)), $C_H^2(k)$ is an equilateral triangle made of 1 (the bold 1's with green background in Fig. C.2(a)), $C_H^3(k)$ is a rectangle made of 1 (the regular 1's without background in Fig. C.2(a)), and $D^1(k)$ is an equilateral triangle made of 1 (the bold 1's with green background in Fig. C.2(b)).	159
C.3	An illustration of $C_H^1(k), C_H^2(k), C_H^3(k)$ and $D^1(k)$ when $j = t'$. $C_H^1(k)$ is an equilateral triangle made of 1 (the underlined 1's with yellow background in Fig. C.3(a)), $C_H^2(k)$ is an equilateral triangle made of 1 (the bold 1's with green background in Fig. C.3(a)), $C_H^3(k)$ is a rectangle made of 1 (the regular 1's without background in Fig. C.3(a)), and $D^1(k)$ is an equilateral triangle made of 1 (the bold 1's with green background in Fig. C.3(b)).	159

C.4	An illustration of $C_{H,z}(k)$, $C_{H,z}^1(k)$, and $C_{H,z}^2(k)$ in two different cases ($j \neq t'$ and $j = t'$). $C_{H,z}(k)$ is the sum of the underlined 1's, $C_{H,z}^1(k)$ is an equilateral triangle made of 1 (the bold underlined 1's with green background), and $C_{H,z}^2(k)$ is a rectangle made of 1 (the sum of some regular 1's and some underlined 1's with orange background).	164
-----	--	-----

List of Tables

2.1	Guidelines for the design of AoI-efficient scheduling policies for a G/G/1 queue	10
3.1	Summary of Key Notations	47

Chapter 1

Introduction

1.1 Motivation and Objectives

The last decades have witnessed significant advances in the computing capabilities of mobile and sensing devices. Simultaneously, the communication capacity of networks has increased significantly, by orders of magnitude. These developments have spawned a wide variety of real-time applications that require timely information updates. A practically important example of such emerging applications is autonomous vehicular systems and intelligent transportation systems, where real-time vehicular status information (location, velocity, acceleration, etc.) needs to be shared with nearby vehicles and road-side facilities to enable full self-driving capability [1, 2]. More such examples include sensor networks for environmental monitoring, weather or news update applications, and live streaming services.

For such real-time services that require timely updated information, the primary concern is the freshness of the data delivered to the receiver or controller. To this end, a timeliness metric called *Age-of-Information (AoI)* is recently proposed [3], which is defined as the time elapsed since the most recently received update was generated. In this dissertation, our goal is to optimize AoI under various real-time network applications. Ever since the concept of AoI was introduced [3], the study on AoI has attracted a lot of research interest (see [4, 5, 6] for a survey). Although researchers actively delving into the exploration of AoI, there remain lots of areas worthy of further investigation.

First, though there is a large body of work that provides detailed analyses on the AoI performance of information-update systems under different queueing models and scheduling policies [7, 8], however, almost all of these scheduling policies explore the design based on three aspects: arrival-time information, service preemption, and informative updates. Another important aspect, the update-size information, has not received sufficient attention yet. It has been shown that scheduling policies that leverage the update-size information can substantially reduce the delay, one of the key components of AoI. In addition, since the update-size information is “orthogonal” to the above three aspects, more AoI-efficient scheduling policies can be possibly designed by combining update-size information with other information (e.g., update-size information and arrival-time information). Therefore, it remains largely unknown how exactly scheduling policies (especially those making use of update-size information) impact the AoI performance.

Second, as a novel timeliness metric, AoI can serve to measure data freshness in numerous real-time systems reliant on timely information updates. Given the intricacy of such systems and the diverse range of requirements they entail, it becomes imperative to consider the tradeoffs between AoI and other performance metrics. In particular, we delve into two important tradeoffs prevalent in real-world applications.

- The tradeoff between service performance and AoI in the data-driven real-time applications (e.g., Google Maps or stock trading applications). These applications need to quickly inject data streams and compute updated analytics results in real-time. For such applications, both the service performance (e.g., response time) received by end users and the freshness of the information received by the end users are important. On the one hand, unilaterally optimizing service performance without considering the freshness of information may lead to users receiving stale information, which may be of greatly reduced value or even useless. On the other hand, if the goal is to opti-

mize for freshness, it might choose to process new updates first and then respond with fresher information, but the end users may be unsatisfied since they have to wait a long time before getting responses. Apparently, there exists a tradeoff between service performance (e.g., response time) and information freshness.

- The tradeoff between update cost and AoI in the crowdsensing-based applications (e.g., Google Waze and GasBuddy). These applications provide services to users by resorting to the user community to sense and send back real-time information (e.g., traffic conditions and gas prices). On the one hand, there is a cost for the applications to update their knowledge of points of interest since they often need to make monetary payments to incentivize users to send their feedback. On the other hand, users are not satisfied if the responses to their requests are outdated. Naturally, there is a tradeoff between update cost and data freshness.

Third, we find that most of the existing work analyzes the AoI performance based on stochastic models with some stationary assumptions. For example, the memoryless property is often used in the queueing system [3, 7, 9, 10, 11], Poisson (or Bernoulli) arrival process is widely taken in the information-update system [12, 13], and communication channels are usually assumed to follow some probability distribution in the wireless network [14, 15, 16]. Although these assumptions lead to tractable performance analysis and optimization, such assumptions may not hold in many practical scenarios. We aim to develop a robust online algorithm for optimizing AoI under a time-varying wireless channel that has a worst-case performance guarantee. That is, different from typical studies with stationary network assumptions, the ratio between the cost of our online algorithm and the cost of the optimal offline algorithm is always bounded under the worst channel state. While online algorithms are robust, they are usually overly conservative and may have a poor average performance in typical scenarios. In contrast, by leveraging historical data and prediction models, machine

learning (ML) algorithms perform well in average cases. However, they typically lack worst-case performance guarantees. Therefore, we also aim to develop a learning-augmented online algorithm that achieve the best of both approaches: excelling in average scenarios while ensuring a guaranteed performance in worst-case scenarios.

1.2 Dissertation Outline and Contributions

Keeping the previously mentioned objectives in focus, this dissertation delves into three categories of AoI optimization problems in real-world applications. There are four chapters covering different research problems. Fig. 1.1 shows the interconnection and contributions among these four chapters.

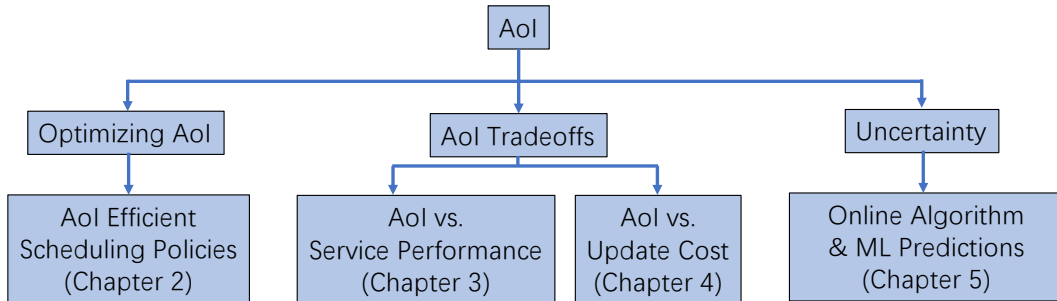


Figure 1.1: Dissertation structure

In Chapter 2, we conduct a systematic and comparative study to investigate the impact of scheduling policies on the AoI performance in single-server queues and provide useful guidelines for the design of AoI-efficient scheduling policies. Specifically, we first perform extensive simulations to demonstrate that the update-size information can be leveraged for achieving a substantially improved AoI compared to non-size-based (or arrival-time-based) policies. Then, by utilizing both the update-size and arrival-time information, we propose three AoI-based policies. Observing improved AoI performance of policies that allow service preemption and that prioritize informative updates, we further propose preemptive,

informative, AoI-based scheduling policies. Our simulation results show that such policies empirically achieve the best AoI performance among all the considered policies. However, compared to the best delay-efficient policies (such as Shortest-Remaining-Processing-Time (SRPT)), the AoI improvement is rather marginal in the settings with exogenous arrivals. Interestingly, we also prove sample-path equivalence between some size-based policies and AoI-based policies. This provides an intuitive explanation for why some size-based policies (such as SRPT) achieve a very good AoI performance.

In Chapter 3, we study the tradeoff between service performance and information freshness in emerging data-driven applications that require rapidly injecting data streams and computing updated analytics results in real-time (e.g., Google Maps or stock trading applications). In many such applications, computing resources are often shared for processing both updates from information sources and queries from end users. This requires joint scheduling of updates and queries because the service provider needs to make a critical decision upon receiving a user query: either it responds immediately with currently available but possibly stale information, or it first processes new updates and then responds with fresher information. Hence, the tradeoff between *service performance* (e.g., response time) and *information freshness* naturally arises in this context. To that end, we propose a simple *single-server two-queue model* that captures the coupled scheduling of updates and queries and aims to design scheduling policies that can properly address the important tradeoff between performance and freshness. Specifically, we consider the *response time* as a performance metric and the *AoI* as a freshness metric. After demonstrating the limitations of the simplest First-Come-First-Served (FCFS) policy, we propose two *threshold-based policies*: the Query- k policy that prioritizes queries and the Update- k policy that prioritizes updates. Then, we rigorously analyze both the response time and the *Peak AoI (PAoI)* of the threshold-based policies. Further, we propose the Joint- (M, N) policy, which allows

flexibly prioritizing updates or queries through choosing different values of two thresholds M and N . Finally, we conduct simulations to evaluate the response time and the PAoI of the proposed policies. The results show that our proposed threshold-based policies can effectively control the balance between performance and freshness.

In Chapter 4, we study the tradeoff between data freshness and update cost in information-update systems. Consider a discrete-time information-update system, where a service provider can proactively retrieve information from the information source to update its data and users query the data at the service provider. One example is crowdsensing-based applications. In order to keep users satisfied, the application desires to provide users with fresh data, where the freshness is measured by the AoI. However, maintaining fresh data requires the application to update its database frequently, which incurs an update cost (e.g., incentive payment). Hence, there exists a natural tradeoff between the AoI and the update cost at the service provider who needs to make update decisions. To capture this tradeoff, we formulate an optimization problem with the objective of minimizing the total cost, which is the sum of the staleness cost (which is a function of the AoI) and the update cost. Then, we provide two useful guidelines for the design of efficient update policies. Following these guidelines and assuming that the aggregated request arrival process is Bernoulli, we prove that there exists a threshold-based policy that is optimal among all online policies and thus focus on the class of threshold-based policies. Furthermore, we derive the closed-form formula for computing the long-term average cost under any threshold-based policy and obtain the optimal threshold. Finally, we perform extensive simulations using both synthetic data and real traces to verify our theoretical results and demonstrate the superior performance of the optimal threshold-based policy compared with several baseline policies.

In Chapter 5, we consider a discrete-time system where a resource-constrained source (e.g., a small sensor) transmits its time-sensitive data to a destination over a time-varying

wireless channel. Each transmission incurs a fixed transmission cost (e.g., energy cost), and no transmission results in a staleness cost represented by the AoI. The source must balance the tradeoff between transmission and staleness costs. To address this challenge, we develop a robust online algorithm to minimize the sum of transmission and staleness costs, ensuring a worst-case performance guarantee. While online algorithms are robust, they are usually overly conservative and may have a poor average performance in typical scenarios. In contrast, by leveraging historical data and prediction models, ML algorithms perform well in average cases. However, they typically lack worst-case performance guarantees. To achieve the best of both worlds, we design a learning-augmented online algorithm that exhibits two desired properties: (i) *consistency*: closely approximating the optimal offline algorithm when the ML prediction is accurate and trusted; (ii) *robustness*: ensuring worst-case performance guarantee even ML predictions are inaccurate. Finally, we perform extensive simulations to show that our online algorithm performs well empirically and that our learning-augmented algorithm achieves both consistency and robustness.

Chapter 2

Anti-Aging Scheduling in Single-Server Queues: A Systematic and Comparative Study

2.1 Introduction

Recently the study of information freshness has received increasing attention, especially for time-sensitive applications that require real-time information/status updates, such as road congestion alerts, stock quotes, and weather forecasts. In order to measure the freshness of information, a new metric, called the *Age-of-Information (AoI)* is proposed. The AoI is defined as the time elapsed since the generation of the freshest update among those that have been received by the destination [3]. Prior studies reveal that the AoI depends on both the inter-arrival time and the delay of the updates. Due to the dependency between the inter-arrival time and the delay, this new AoI metric exhibits very different characteristics than the traditional delay metric and is generally much harder to analyze (see, e.g., [3]).

Although it is well-known that scheduling policies play an important role in reducing the delay in single-server queues, it remains largely unknown how exactly scheduling policies impact the AoI performance. To that end, we aim to holistically study the impact of various

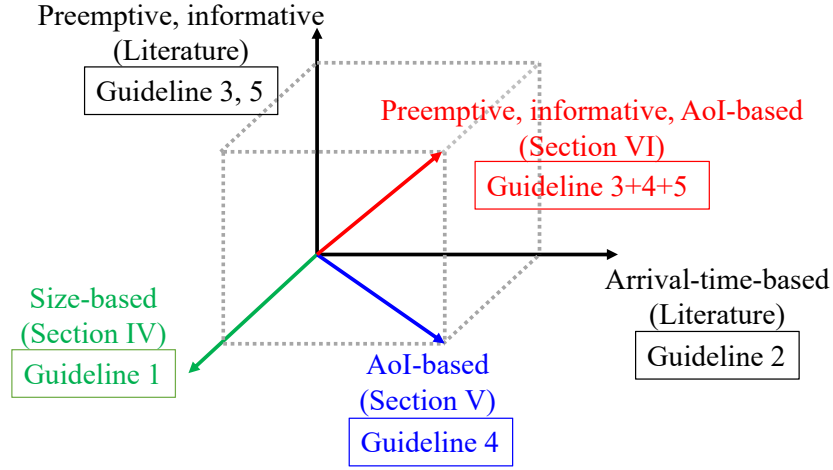


Figure 2.1: Our position in the design space of AoI-efficient scheduling policies for a $G/G/1$ queue

aspects of scheduling policies on the AoI performance in single-server queues and provide useful guidelines for the design of scheduling policies that can achieve a small AoI.

While much research effort has already been exerted to the design and analysis of scheduling policies aiming to reduce the AoI, almost all of these policies are only based on the arrival time of updates, such as First-Come-First-Served (FCFS) and Last-Come-First-Served (LCFS), assuming that the update-size information is unavailable. Here, the size of an update is the amount of time required to serve the update if there were no other updates around. In some applications, such as smart grid and traffic monitoring, the update-size information can be obtained or fairly well estimated [17]. It has been shown that scheduling policies that leverage the size information can substantially reduce the delay, especially when the system load is high or when the size variability is large [18]. This motivates us to investigate the AoI performance of size-based policies in a $G/G/1$ queue. Note that the update-size information is “orthogonal” to the arrival-time information, both of which could significantly impact the AoI performance. Therefore, it is quite natural to further consider AoI-based policies that use both the update-size and arrival-time information of updates.

Guideline	Summary	Representative policies
1	Prioritizing small updates	SJF, SJF_P, SRPT
2	Prioritizing recent updates	LCFS, LCFS_P
3	Allowing service preemption	PS, LCFS_P, SJF_P, SRPT
4	AoI-based designs	ADE, ADS, ADM
5	Prioritizing informative updates	Informative version of the above policies

Table 2.1: Guidelines for the design of AoI-efficient scheduling policies for a G/G/1 queue

In addition, prior work has revealed that scheduling policies that allow service *preemption* and that prioritize *informative* updates (also called *effective* updates, which are those that lead to a reduced AoI once delivered; see Section 2.6.1 for a formal definition) yield a good AoI performance [7, 11, 19]. Intuitively, preemption prevents fresh updates from being blocked by a large and/or stale update in service; informative policies discard stale updates, which do not bring new information but may block fresh updates. To that end, we also consider AoI-based scheduling designs that both allow service preemption and prioritize informative updates.

In Fig. 2.1, we position our work in the literature by summarizing various design aspects of scheduling policies for a G/G/1 queue. Existing work mostly explores the design based on the arrival-time information along with considering service preemption and informative updates. We point out that the size-based design is an orthogonal dimension of great importance, which somehow has not received sufficient attention yet. Unsurprisingly, designing AoI-efficient policies requires the consideration of all these dimensions. In Table 2.1, we summarize several useful guidelines for the design of AoI-efficient policies, which are also labeled in Fig. 2.1. To the best of our knowledge, this is the first work that conducts a systematic and comparative study to investigate the design of AoI-efficient scheduling policies for a G/G/1 queue. In the following, we summarize our key contributions along with an explanation of Fig. 2.1 and Table 2.1.

First, we investigate the AoI performance of size-based scheduling policies (i.e., the green arrow in Fig. 2.1), which is an orthogonal approach to the arrival-time-based design studied in most existing work. We conduct extensive simulations to show that size-based policies that prioritize small updates significantly improve AoI performance. We also explain interesting observations from the simulation results and summarize useful guidelines (i.e., Guidelines 1, 2, and 3 in Table 2.1) for the design of AoI-efficient policies.

Second, leveraging both the update-size and arrival-time information, we introduce Guideline 4 and propose AoI-based scheduling policies (i.e., the blue arrow in Fig. 2.1). These AoI-based policies attempt to optimize the AoI at a specific future time instant from three different perspectives: the AoI-Drop-Earliest (ADE) policy, which makes the AoI drop the earliest; the AoI-Drop-to-Smallest (ADS) policy, which makes the AoI drop to the smallest; the AoI-Drop-Most (ADM) policy, which makes the AoI drop the most. The simulation results show that such AoI-based policies indeed have a good AoI performance.

Third, we observe that informative policies can significantly improve the AoI performance compared to their non-informative counterparts, which leads to Guideline 5. Integrating all the guidelines, we propose preemptive, informative, AoI-based policies (i.e., the red arrow in Fig. 2.1). The simulation results show that such policies empirically achieve the best AoI performance among all the considered policies.

Finally, we prove the sample-path equivalence between some size-based policies and AoI-based policies. These results provide an intuitive explanation for why some size-based policies, such as Shortest-Remaining-Processing-Time (SRPT), achieve a very good AoI performance.

To summarize, our study reveals that among the various aspects of scheduling policies we investigated, prioritizing small updates, allowing service preemption, and prioritizing

informative updates play the most important role in the design of AoI-efficient scheduling policies. However, compared to the best delay-efficient policies (such as SRPT), the AoI improvement of the preemptive, informative, and AoI-based policies is rather marginal in the settings with exogenous arrivals. Moreover, when the AoI requirement is not stringent or the update-size information is not available, some simple delay-efficient policies (such as LCFS with preemption (LCFS_P)) are also good candidates for AoI-efficient policies.

The rest of this chapter is organized as follows. We first discuss related work in Section 2.2. Then, we describe our system model in Section 2.3. In Section 2.4, we evaluate the AoI performance of size-based scheduling policies. We further propose AoI-based scheduling policies in Section 2.5. In addition, we evaluate the AoI performance of preemptive, informative, AoI-based policies in Section 2.6. Finally, we make concluding remarks in Section 2.7.

2.2 Related work

The traditional queueing literature on single-server queues is largely focused on delay analysis. In [20], the authors prove that all non-preemptive scheduling policies that do not make use of job size information have the same distribution of the number of jobs in the system. The work of [21, 22] proves that for a work-conserving queue, the SRPT policy minimizes the number of jobs in the system at any point and is therefore delay-optimal. The work of [23] derives a formula of the average delay for several common scheduling policies (which will be discussed in Section 2.4).

On the other hand, although the AoI research is still in a nascent stage, it has already attracted a lot of interest (see [4, 5] for a survey). Here we only discuss the most relevant work, which is focused on the AoI-oriented queueing analysis. Much of the existing work considers scheduling policies that are based on the arrival time (such as FCFS and LCFS).

The AoI is introduced in [3], where the authors study the average AoI in the M/M/1, M/D/1, and D/M/1 queues under the FCFS policy. In [24], the AoI performance of the FCFS policy in the M/M/1/1 and M/M/1/2 queues is studied, where new arrivals are discarded if the buffer is full. In [25], the authors study the average AoI performance of a multi-source FCFS M/G/1 queue. They derive the exact expression and three approximations of the average AoI for a special case of an M/M/1 queue and a general case of an M/G/1 queue, respectively. The average AoI of the LCFS policy in the M/M/1 queue is also discussed in [24].

There has been some work that aims to reduce the AoI by making use of service preemption. In [9], the average AoI of LCFS in the M/M/1 queue with and without service preemption is analyzed. The work of [26] is quite similar to [9], but it considers the average AoI in the M/M/2 queue. In [27], the average AoI for the M/G/1/1 preemptive system with a multi-stream updates source is derived. The age-optimality of the preemptive LCFS (LCFS_P) policy is proved in [19], where the service times are exponentially distributed.

In addition to taking advantage of service preemption, some of the prior studies also consider the strategy of prioritizing informative updates for reducing the AoI. The work of [7, 11] reveals that the AoI performance can be improved by prioritizing informative updates and discarding non-informative policies when making scheduling decisions. In [28], the authors consider a G/G/1 queue with informative updates and derive the stationary distribution of the AoI, which is in terms of the stationary distribution of the delay and the Peak AoI (PAoI). With the AoI distribution, one can analyze the mean or higher moments of the AoI in GI/GI/1, M/GI/1, and GI/M/1 queues under several scheduling policies (e.g., FCFS and LCFS).

Recent research effort has also been exerted to understanding the relation between the AoI and the delay. In [29], the authors analyze the tradeoff between the AoI and the delay in a single-server M/G/1 system under a specific scheduling policy without knowing the service

time of each individual update. In [30], the violation probability of the delay and the PAoI is investigated under an additive white Gaussian noise (AWGN) channel, but the update size is assumed to be identical.

2.3 System Model

In this section, we consider a single-server queueing system and give the definitions of the Age-of-Information (AoI) and the Peak AoI (PAoI).

We model the information-update system as a G/G/1 queue where a single source generates updates (which contain current state of a measurement or observation of the source) with rate λ . The updates enter the queueing system immediately after they are generated. Hence, the generation time is the same as the arrival time. We use S to denote the size of an update (i.e., the amount of time required for the update to complete service), which has a general distribution with mean $\mathbb{E}[S] = 1/\mu$. The system load is defined as $\rho \triangleq \lambda/\mu$.

We use t_i and t'_i to denote the time at which the i -th update was generated at the source and the time at which it leaves the server, respectively. The AoI at time t is then defined as $\Delta(t) \triangleq t - U(t)$, where $U(t) \triangleq \max \{t_i : t'_i \leq t\}$ is the generation time of the freshest update among those that have been processed by the server. An example of the AoI evolution under the FCFS policy is shown in Fig. 2.2. Then, the average AoI can be defined as

$$\Delta = \lim_{t \rightarrow \infty} \frac{1}{t} \int_0^t \Delta(\tau) d\tau. \quad (2.1)$$

In general, the analysis of the average AoI is quite difficult since it is determined by two dependent quantities: the inter-arrival time and the delay of updates[3]. We define the inter-arrival time between the i -th update and $(i-1)$ -th update as $X_i \triangleq t_i - t_{i-1}$ and define

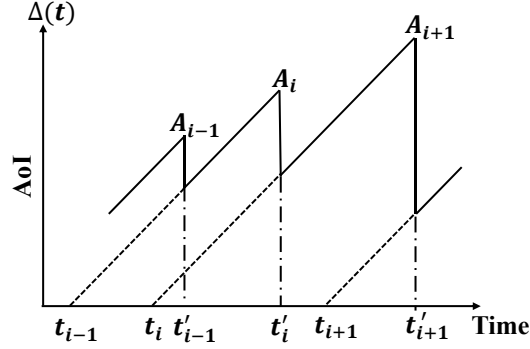


Figure 2.2: An example of the AoI evolution under the FCFS policy

the delay of the i -th update as $T_i \triangleq t'_i - t_i$. Alternatively, the Peak AoI (PAoI) is also proposed as an information freshness metric [7], which is defined as the maximum value of the AoI before it drops due to a newly delivered fresh update. Let A_i be the i -th PAoI. From Fig. 2.2, we can see $A_i = t'_i - t_{i-1}$. This can be rewritten as the sum of the inter-arrival time between the i -th update and the previous update (i.e., X_i) and the delay of the i -th update (i.e., T_i). Therefore, the PAoI of the i -th update can also be expressed as $A_i = X_i + T_i$, and its expectation is $\mathbb{E}[A_i] = \mathbb{E}[X_i] + \mathbb{E}[T_i]$.

2.4 Size-based policies

In this section, we investigate the AoI performance of several common scheduling policies, including size-based policies and non-size-based policies, via extensive simulations. Note that these common scheduling policies may serve the non-informative updates (which do not lead to a reduced AoI). This is because in some applications, such as news and social networks, obsolete updates are still useful and need to be served [19]. In Section 2.6, we will discuss the case where obsolete updates are discarded.

Following [18], we first give the definitions of several common scheduling policies that can be divided into four types: depending on whether they are size-based or not, where the

size-based policies use the update-size information (which is available in some applications, such as smart grid [17]) for making scheduling decisions; depending on whether they are preemptive or not. The definition of preemption is given below. In this chapter, we do not consider the cost of preemption.

Definition 2.1. A policy is preemptive if an update may be stopped partway through its execution and then restarted at a later time without losing intermediary work.

The first type consists of policies that are non-preemptive and blind to the update size:

- *First-Come-First-Served (FCFS)*: When the server frees up, it chooses to serve the update that arrived first if any.
- *Last-Come-First-Served (LCFS)*: When the server frees up, it chooses to serve the update that arrived last if any.
- *Random-Order-Service (RANDOM)*: When the server frees up, it randomly chooses one update to serve if any.

The second type consists of policies that are non-preemptive and make scheduling decisions based on the update size:

- *Shortest-Job-First (SJF)*: When the server frees up, it chooses to serve the update with the smallest size if any.

The third type consists of policies that are preemptive and blind to the update size:

- *Processor-Sharing (PS)*: All the updates in the system are served simultaneously and equally (i.e., each update receives an equal fraction of the available service capacity).

- *Preemptive Last-Come-First-Served (LCFS_P)*: This is the preemptive version of the LCFS policy. Specifically, a preemption happens when there is a new update.

The fourth type consists of policies that are preemptive and make scheduling decisions based on the update size:

- *Preemptive Shortest-Job-First (SJF_P)*: This is the preemptive version of the SJF policy. Specifically, a preemption happens when there is a new update that has the smallest size.
- *Shortest-Remaining-Processing-Time (SRPT)*: When the server frees up, it chooses to serve the update with the smallest remaining size. In addition, a preemption happens only when there is a new update whose size is smaller than the remaining size of the update in service.

Previous work (see, e.g., [18, Section VII]) reveals that size-based policies can greatly improve the delay performance. Due to such results, we conjecture that size-based policies also achieve a better AoI performance given that the AoI is dominantly determined by the delay when the system load is high or when the size variability is large [3]. As we mentioned earlier, it is in general very difficult to obtain the exact expression of the average AoI except for some special cases (e.g., FCFS and LCFS) [3, 28]. Therefore, we attempt to investigate the AoI performance of size-based policies through extensive simulations.

In Figs. 2.3 and 2.4, we present the simulation results of the average AoI and PAoI performance under the scheduling policies we introduced above, respectively. There are three commonly used methods to conduct the simulation: Independent Replications, Batch Means, and Regeneration. Here, we use the Independent Replications for the following reasons: (i) the replication means are independent; (ii) it allows to start the individual replications in

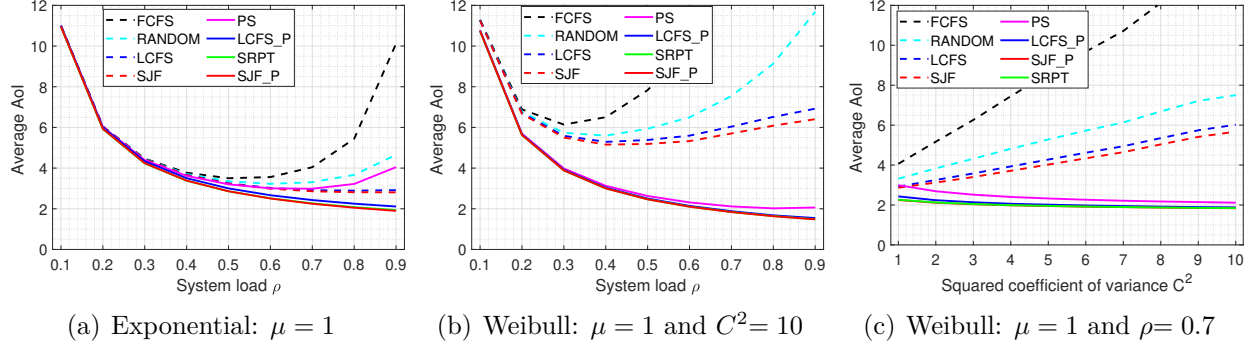


Figure 2.3: Comparisons of the average AoI performance under several common scheduling policies

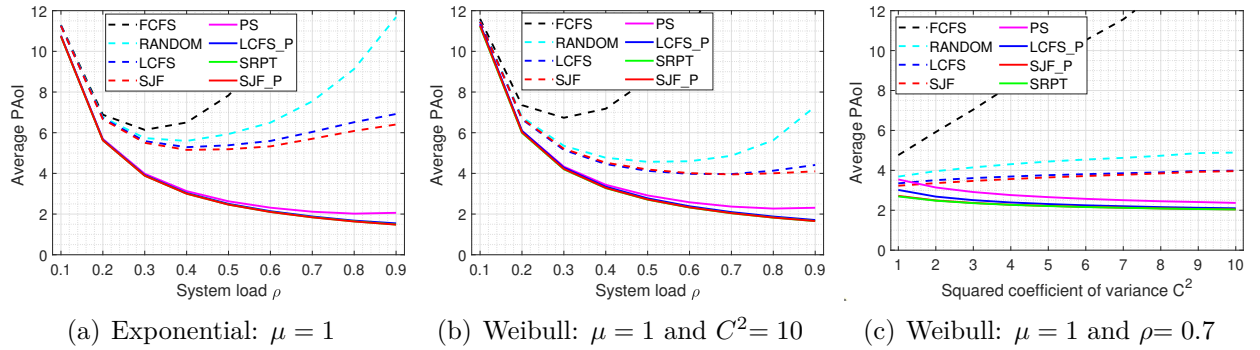


Figure 2.4: Comparisons of the average PAoI performance under several common scheduling policies

different initial states such that various different sample paths of the underlying stochastic process can be observed. Specifically, we conduct 50 simulation runs and take the average values. In each simulation run, we consider a total number of 10^5 updates to ensure that the steady state is reached. All the random numbers are generated using the default pseudorandom number generator (i.e., the Mersenne Twister) in the Python standard library. Here, we assume that a single source generates updates according to a Poisson process with rate λ , and the update size is independent and identically distributed (*i.i.d.*). In Fig. 2.3(a), we assume that the update size follows an exponential distribution with mean $1/\mu = 1$.

In Figs. 2.3(b) and 2.3(c), we assume that the update size follows a Weibull distribution¹ with mean $1/\mu = 1$. We define the squared coefficient of variation of the update size as $C^2 \triangleq \text{Var}(S)/\mathbb{E}[S]^2$, i.e., the variance normalized by the square of the mean [18]. Hence, a larger C^2 means a larger variability. In Fig. 2.3(b), we fix $C^2 = 10$ and change the value of system load ρ , while in Fig. 2.3(c), we fix system load $\rho = 0.7$ and change the value of C^2 . Note that throughout the chapter, these simulation settings are used as default settings unless otherwise specified. In addition, the 95% confidence intervals of Figs. 2.3 and 2.4 are also provided in our online technical report [31], in which we observe that the margin of error is only a very small portion of the average (about 1%).

In the following, we will discuss key observations from the simulation results and propose useful guidelines for the design of AoI-efficient policies.

Observation 1. Size-based policies achieve a better average AoI/PAoI performance than non-size-based policies in both non-preemptive and preemptive cases.

In Fig. 2.3, we can see that for the non-preemptive case, SJF has a better average AoI performance than FCFS, RANDOM, and LCFS in various settings. Similarly, for the preemptive case, SJF_P and SRPT have a better average AoI performance than PS and LCFS_P. Similar observations can be made for the average PAoI performance in Fig. 2.4.

Observation 2. Under preemptive, size-based policies, the average AoI/PAoI decreases as the system load increases.

In Figs. 2.3(a) and 2.3(b), we can see that under SJF, SJF_P, and SRPT, the average AoI decreases as the system load ρ increases. There are two reasons. First, when ρ increases, there will be more updates with small size arriving to the queue. Therefore, size-based policies

¹The Weibull distribution is a heavy-tailed distribution with pdf $f(x; \alpha, \beta) = \frac{\alpha}{\beta} (\frac{x}{\beta})^{\alpha-1} e^{-(x/\beta)^\alpha}$ for $x > 0$, where $\alpha > 0$ is the shape parameter and $\beta > 0$ is the scale parameter.

that prioritize updates with small size lead to more frequent AoI drops. Second, preemption operations prevent fresh updates from being blocked by a large or stale update in service. Similar observations can be made for the average PAoI performance in Figs. 2.4(a) and 2.4(b).

Observations 1 and 2 lead to the following guideline:

Guideline 1. When the update-size information is available, one should prioritize updates with a small size.

However, in certain application scenarios, the update-size information may not be available or is difficult to estimate. Hence, the scheduling decisions have to be made without the update-size information. In such scenarios, we make the following observations from Figs. 2.3 and 2.4.

Observation 3. LCFS and LCFS_P achieve the best average AoI performance among non-preemptive, non-size-based policies and preemptive, non-size-based policies, respectively.

Observation 4. Under LCFS_P, the average AoI/PAoI decreases as the system load increases.

Observations 3 and 4 have also been made in previous work [19, 24, 32]. It is quite intuitive that when the update-size information is unavailable, one should give a higher priority to more recent updates. This is because while all the updates have the same expected service time, the most recent update arrives the last and thus leads to the smallest AoI once delivered. Therefore, Observations 3 and 4 lead to the following guideline:

Guideline 2. When the update-size information is unavailable, one should prioritize recent updates.

Note that Observations 2 and 4 also suggest that under preemptive policies, the average AoI/PAoI decreases as the system load ρ increases. This is because preemptions prevent

fresh updates from being blocked by a large or stale update in service. In addition, we have also observed the following nice properties of preemptive policies.

Observation 5. Not only do preemptive policies achieve a better average AoI/PAoI performance than non-preemptive policies, but they are also less sensitive when the update-size variability changes, i.e., they are more robust.

In Figs. 2.3(a) and 2.3(b), we can see that preemptive policies (e.g., LCFS_P, SJF_P, and SRPT) generally have a better average AoI performance than non-preemptive ones (e.g., FCFS, RANDOM, LCFS, and SJF), especially when the system load is high. In Fig. 2.3(c), we can see that the advantage of preemptive policies becomes larger as the update-size variability (i.e., C^2) increases. Moreover, the AoI performance of preemptive policies is only very slightly impacted when the update-size variability changes, while that of non-preemptive policies varies significantly. Therefore, Observations 2, 4, and 5 lead to the following guideline:

Guideline 3. Service preemption should be employed when it is allowed.

Note that the above observations not only hold for the M/G/1 queue but also can be made for the G/G/1 queue. More simulation results for the G/G/1 queue (i.e., Figs. A.1-A.8) can be found in Appendix A.1 and our technical report [31]. In addition, we make the following interesting observations regarding the average PAoI and AoI in a G/G/1 queue.

Observation 6. The average PAoI could be much smaller than the average AoI when the interarrival time has a large variability.

In Figs. A.1(a) and A.2(a), we can see that the average PAoI is much smaller than the average AoI for all the common scheduling policies we considered. This is due to the interarrival time has a large variability. We present an example in Fig. 2.5 to illustrate

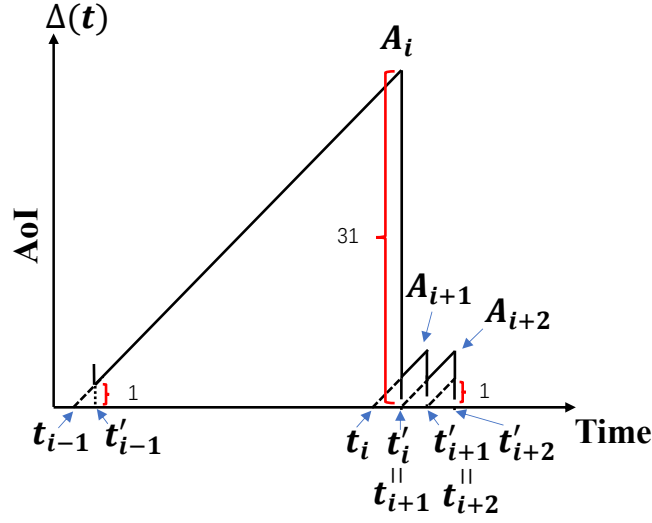


Figure 2.5: An example of the AoI/PAoI evolution where the interarrival time has a large variability

that this phenomenon comes from the large variability of the interarrival time. We consider three updates: the i -th, the $(i + 1)$ -st and $(i + 2)$ -nd updates, which are served in sequence during (t'_{i-1}, t'_{i+2}) . Their interarrival times are as follows: $t_i - t_{i-1} = 30$, $t_{i+1} - t_i = 1$, and $t_{i+2} - t_{i+1} = 1$; and their system times are as follows: $t'_i - t_i = 1$, $t'_{i+1} - t_{i+1} = 1$, and $t'_{i+2} - t_{i+2} = 1$. In addition, we also assume $t'_{i-1} - t_{i-1} = 1$. Therefore, the average AoI and the average PAoI during (t'_{i-1}, t'_{i+2}) are $\frac{31^2 + 2^2 + 2^2 - 3 \times 1^2}{2 \times (30 + 1 + 1)} \approx 15.09$ and $\frac{31 + 2 + 2}{3} \approx 11.67$, respectively. In this case, the average PAoI is indeed smaller than the average AoI.

The importance of Observation 6 can be summarized as follows. First, in certain settings (e.g., where the interarrival time has a large variability), the average AoI can actually be higher than the average PAoI. This observation is counterintuitive, given that the computation of the average PAoI includes the peak values of the AoI only. Second, given that the average AoI and the average PAoI exhibit different relationships in different settings, an AoI-efficient scheduling policy may not necessarily achieve the desired PAoI performance, and vice versa. In other words, one must carefully study the design of AoI-efficient scheduling

policies with different goals in mind (i.e., minimizing the average AoI or the average PAoI).

Observation 7. While the average AoI performance of several non-preemptive policies (such as RANDOM, LCFS, and SJF) is sensitive to the update-size variability, their average PAoI performance is not.

In Fig.2.4(c), we observe that while the average PAoI performance of FCFS is sensitive to the update-size variability, under several non-preemptive policies (such as RANDOM, LCFS, and SJF), the average PAoI performance is much less sensitive. An explanation for this observation is the following.

First, we explain why the average PAoI under FCFS is still sensitive to the update-size variability. Note that a key difference between FCFS and other non-preemptive policies is that under FCFS, every update leads to an AoI drop and thus corresponds to an AoI peak². When a large update is in service, it will block all the following updates that are waiting in the queue, which results in a large delay for all such updates and thus a large PAoI corresponding to these updates. In contrast, under RANDOM, LCFS, and SJF, the impact of such a blocking issue is minimal for the updates that lead to an AoI drop.

Next, we explain why under RANDOM, LCFS, and SJF, while the average AoI is sensitive to the update-size variability, the average PAoI is not. We first consider LCFS. In the setting we consider, there is a high chance that the newest update has a small size. Serving such small-size updates leads to a small PAoI. When the newest update has a large size, the corresponding PAoI would also be large. However, this happens less often. Therefore, the AoI trajectory would consist of a smaller percentage of large AoI peaks with many small AoI peaks in between. As the update-size variability increases, there will be fewer but larger AoI peaks. In such cases, while the average AoI is sensitive to the large AoI peaks (which come

²Consider a non-preemptive policy, the LCFS policy, as an example. Under LCFS, there may be older updates waiting in the queue when a new update is being served. After this new update finishes service, those older updates waiting in the queue become outdated, and the delivery of any of these older updates will not lead to an AoI drop.

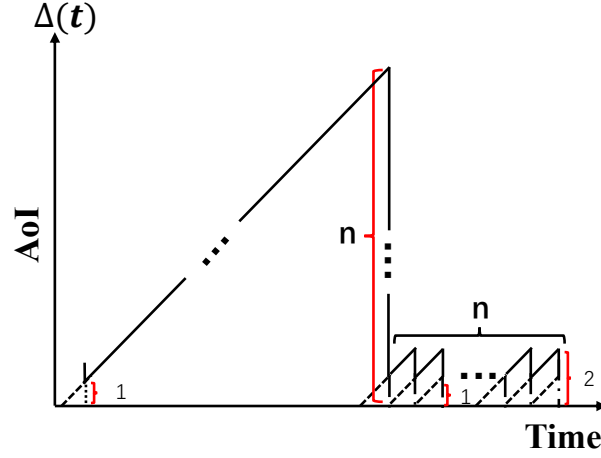


Figure 2.6: An example of the AoI/PAoI evolution where the service time has a large variability

from the large update-size variability), the average PAoI is much less sensitive. To illustrate this fact, we provide an example in Fig. 2.6, where there is a large update of size $n - 1$, immediately followed by n small updates of size 1. In this case, we can compute the average AoI as $\Delta = \left[1 \times \left(\frac{n^2}{2} - \frac{1^2}{2} \right) + n \times \left(\frac{2^2}{2} - \frac{1^2}{2} \right) \right] / ((n - 1) + n) = \frac{n^2 + 3n - 1}{4n - 2} = \mathcal{O}(n)$ and compute the average PAoI as $A = \frac{n + 2 \times n}{n + 1} = \frac{3n}{n + 1} = \mathcal{O}(3)$. This example shows that a larger update-size variability (i.e., a larger n in this example) results in a larger average AoI but only minimally affects the average PAoI. A similar explanation also applies to SJF and RANDOM.

2.5 AoI-based policies

In Section 2.4, we have demonstrated that size-based policies achieve a better average AoI/-PAoI performance than non-size-based policies. However, size-based policies do not utilize the arrival-time information, which also plays an important role in reducing the AoI. In this section, we propose three AoI-based scheduling policies, which leverage both the update-size and arrival-time information to reduce the AoI. Our simulation results show that these AoI-based policies outperform non-AoI-based policies.

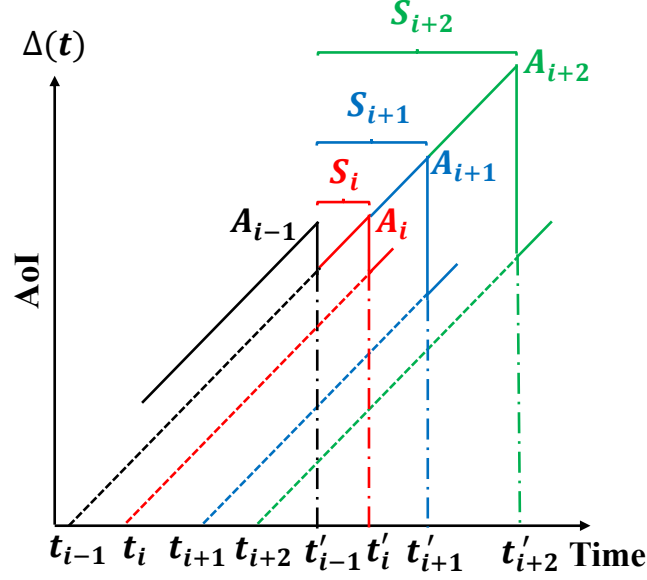


Figure 2.7: The AoI evolution under three AoI-based policies: ADE (red), ADS (blue), and ADM (green)

We begin with the definitions of three AoI-based policies that *attempt to optimize the AoI at a specific future time instant* from three different perspectives:

- *AoI-Drop-Earliest (ADE)*: When the server frees up, it chooses to serve an update such that once it is delivered, the AoI drops as soon as possible.
- *AoI-Drop-to-Smallest (ADS)*: When the server frees up, it chooses to serve an update such that once it is delivered, the AoI drops to a value as small as possible.
- *AoI-Drop-Most (ADM)*: When the server frees up, it chooses to serve an update such that once it is delivered, the AoI drops as much as possible.

If all updates waiting in the queue are obsolete, then the above policies choose to serve an update with the smallest size.

Although all of these AoI-based policies are quite intuitive, they behave very differently. In order to explain the differences between these AoI-based policies, we present an example

in Fig. 2.7 to show how the AoI evolves under these policies. Suppose that when the $(i-1)$ -st update is being served, three new updates (i.e., the i -th, $(i+1)$ -st, and $(i+2)$ -nd updates) arrive in sequence at times t_i , t_{i+1} , and t_{i+2} , respectively. The sizes of these updates satisfy $S_i < S_{i+1} < S_{i+2}$. When the server frees up after it finishes serving the $(i-1)$ -st update at time t'_{i-1} , ADE, ADS, and ADM choose to serve the i -th, $(i+1)$ -st, and $(i+2)$ -nd updates, respectively. This is because serving the i -th update leads to the earliest AoI drop at time t'_i (following the red curve), serving the $(i+1)$ -st update leads to the AoI dropping to the smallest at time t'_{i+1} (following the blue curve), and serving the $(i+2)$ -nd update leads to the largest AoI drop at time t'_{i+2} (following the green curve). Clearly, ADE, ADS, and ADM aim to optimize AoI at a specific future time instant (i.e., the future delivery time of the chosen update) with different myopic goals. Note that at first glance, ADS and ADM may look the same. Indeed, they would be equivalent if the events of the AoI drop had happened at the same time instant. However, these two policies are different as the time instants at which the AoI drops are not necessarily the same (e.g., t'_{i+1} vs. t'_{i+2} in Fig. 2.7). In addition, ADE and SJF may also look the same at first glance. Indeed, these two policies would make the same decision (i.e., choose the smallest update to serve) when the smallest update leads to an AoI drop. However, they make different decisions when the smallest update does not lead to an AoI drop. An example is provided in Fig. 2.8 to illustrate the key difference. In Fig. 2.8, after the $(i-1)$ -st update completes service at time t'_{i-1} , there are two updates waiting to be served: the $(i-2)$ -nd update and the i -th update. Suppose that the update size and the arrival time of these two updates satisfy the following: $S_{i-2} < S_i$ and $t_{i-2} < t_{i-1} < t_i$. Clearly, ADE chooses to serve the i -th update that leads to an earlier AoI drop (see Fig. 2.8(a)), while SJF chooses to serve the $(i-2)$ -nd update that has a smaller size (see Fig. 2.8(b)).

Next, we conduct extensive simulations to investigate the AoI performance of these AoI-

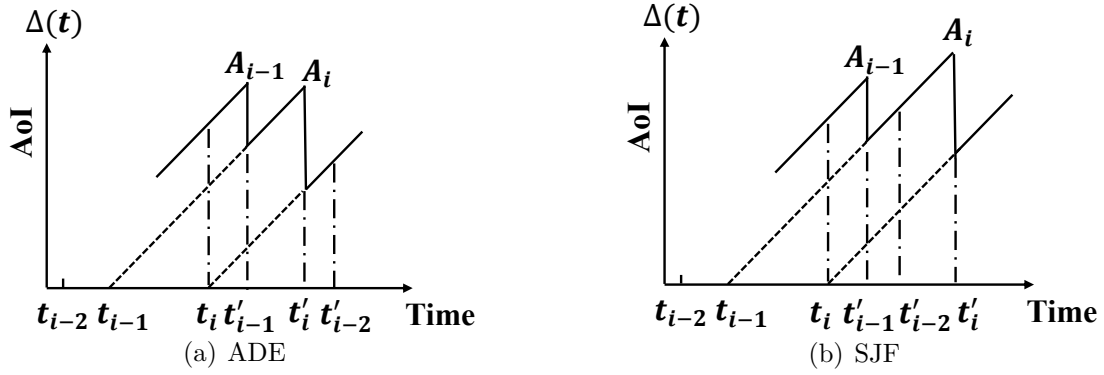


Figure 2.8: An example of the AoI evolution under ADE and SJF

based policies. In Fig. 2.9, we present the simulation results of the average AoI performance of the AoI-based policies compared to a representative arrival-time-based policy (i.e., LCFS) and a representative size-based-policy (i.e., SJF). All the policies considered here are non-preemptive; the preemptive cases will be discussed in Section 2.6.

In Fig. 2.9(a), we observe that most AoI-based policies are slightly better than non-AoI-based policies, although their performances are very close. Among the AoI-based policies, ADE is the best, ADM is the worst, and ADS is in-between. This is not surprising that ADM is the worst: although ADM has the largest AoI drop, this is at the cost that it may have to wait until the AoI becomes large first. ADE being the best suggests that giving a higher priority to small updates (so that the AoI drops as soon as possible) is a good strategy. In Figs. 2.9(b) and 2.9(c), similar observations can be made for update size following Weibull distributions.

The above observations lead to the following guideline:

Guideline 4. Leveraging both the update-size and arrival-time information can further improve the AoI performance. However, the benefit seems marginal.

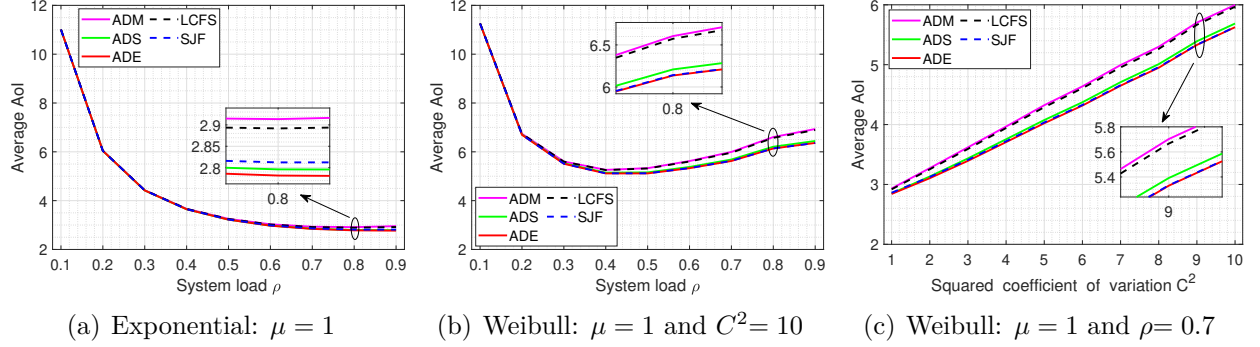


Figure 2.9: Comparisons of the average AoI performance: AoI-based policies vs. non-AoI-based policies

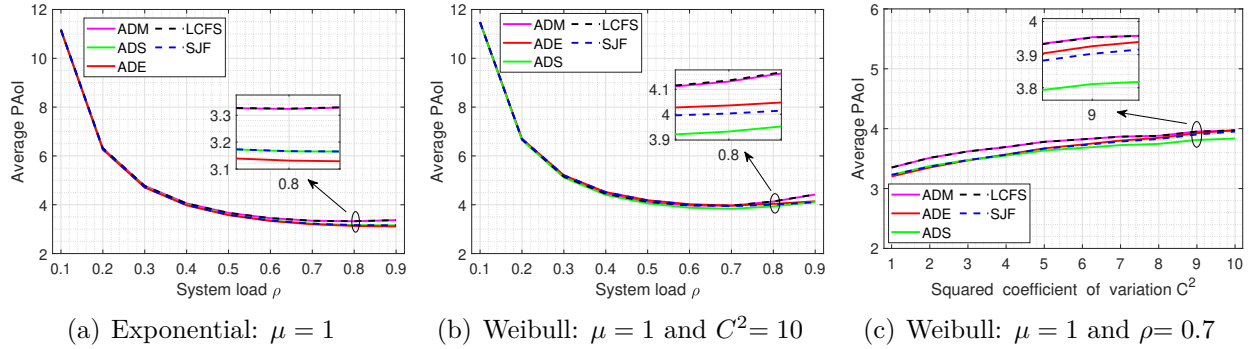


Figure 2.10: Comparisons of the average PAoI performance: AoI-based policies vs. non-AoI-based policies

2.6 Preemptive, informative, AoI-based Policies

In Section 2.4, we have observed that preemptive policies have several advantages and perform better than non-preemptive policies. In this section, we first demonstrate that policies that prioritize informative updates (i.e., those that can lead to AoI drops once delivered) perform better than non-informative policies. Then, by integrating the guidelines we have, we consider preemptive, informative, AoI-based policies and evaluate their performances through simulations.

2.6.1 Informative Policies

As far as the AoI is concerned, there are two types of updates: informative updates and non-informative updates [33]. *Informative updates lead to AoI drops once delivered while non-informative updates do not.* In some applications, such as autonomous vehicles and stock quotes, it is reasonable to discard non-informative updates (which do not help reduce the AoI but may block new updates). In this subsection, we introduce the “informative” versions of various policies, which prioritize informative updates and discard non-informative updates. Then, we use simulation results to demonstrate that informative policies generally have a better average AoI/PAoI performance than the original (non-informative) ones. Furthermore, we rigorously prove that in a G/M/1 queue, the informative version of LCFS is stochastically better than the original LCFS policy.

We use π_I to denote the informative version³ of policy π . All the scheduling policies we consider have their informative versions. In some cases, the informative version is simply the same as the original policy (e.g., FCFS and LCFS_P).

In Fig. 2.11, we show the simulation results of the average AoI performance of several informative policies compared to their non-informative counterparts. In order to evaluate the benefit of informative policies, we plot the informative AoI gain, which is the ratio of the difference between the average AoI of the non-informative version and the informative version to the average AoI of the non-informative version. Hence, a larger informative gain means a larger benefit of the informative version. One important observation from Fig. 2.11 is as follows.

Observation 8. Informative policies achieve a better average AoI performance than their non-informative counterparts. The informative gain is larger for non-preemptive policies and

³For simplicity, we omit the additional “_” in the policy name if policy π is a preemptive policy ending with “_P”. For example, we use LCFS_PI to denote the informative version of LCFS_P.

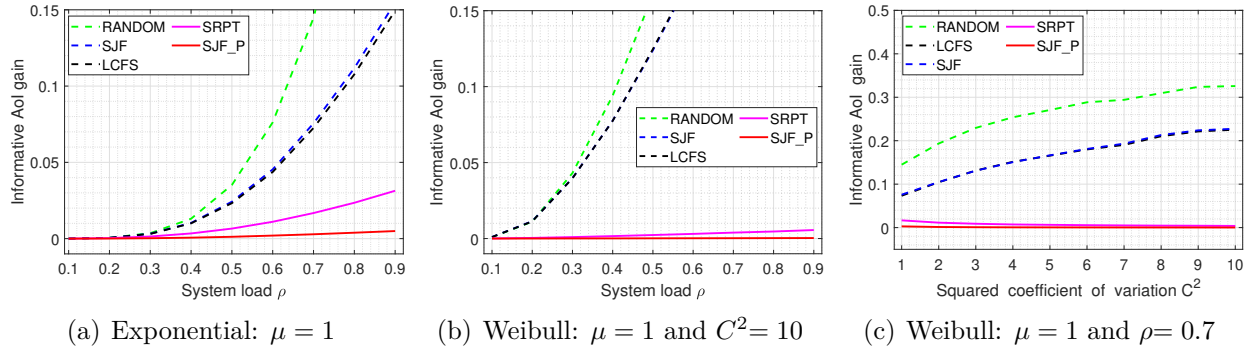


Figure 2.11: Comparisons of the average AoI performance: informative policies vs. non-informative policies

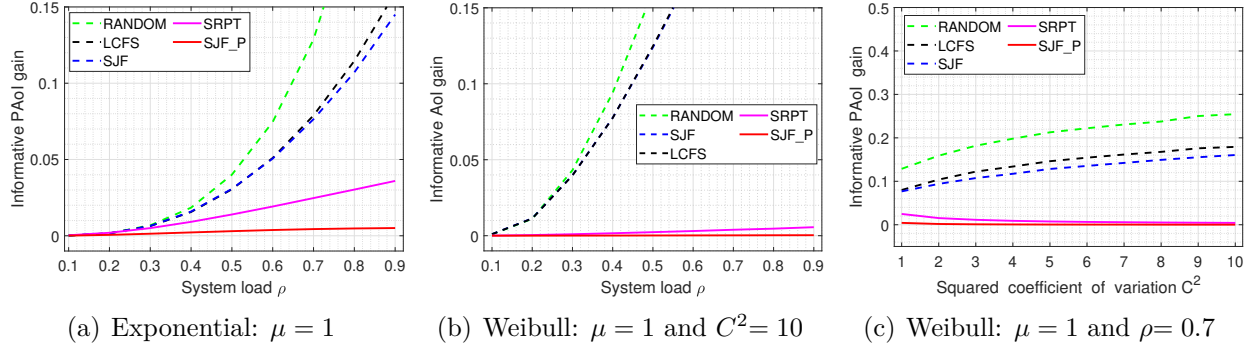


Figure 2.12: Comparisons of the average PAoI performance: informative policies vs. non-informative policies

increases as the system load increases.

Intuitively, informative policies are expected to outperform their non-informative counterparts because serving non-informative updates cannot reduce the AoI but may block new updates. The simulation results verify this intuition as the informative AoI gain is always non-negative. Second, we can see that most non-preemptive policies (e.g., RANDOM, LCFS, and SJF) benefit more from prioritizing informative updates. Third, as the system load ρ increases, the informative AoI gain increases under most considered policies, especially those non-preemptive ones. This is because as the system load increases, the number of non-informative updates also increases, which has a larger negative impact on the AoI

performance for non-preemptive, non-informative policies.

Observation 8 leads to the following guideline:

Guideline 5. The server should prioritize informative updates and discard non-informative updates when it is allowed.

Based on Observation 8, we conjecture that an informative policy is at least as good as its non-informative counterpart. As a preliminary result, we prove that this conjecture is indeed true for LCFS in a G/M/1 queue. In the following, we introduce the stochastic ordering notion, which will be used in the statement of Proposition 2.3.

Definition 2.2. Stochastic Ordering of Stochastic Processes [34, Ch.6.B.7]: Let $\{X(t), t \in [0, \infty)\}$ and $\{Y(t), t \in [0, \infty)\}$ be two stochastic processes. Then, $\{X(t), t \in [0, \infty)\}$ is said to be stochastically less than $\{Y(t), t \in [0, \infty)\}$, denoted by $\{X(t), t \in [0, \infty)\} \leq_{\text{st}} \{Y(t), t \in [0, \infty)\}$, if, for all choices of integer n and $t_1 < t_2 < \dots < t_n$ in $[0, \infty)$, the following holds for all upper sets⁴ $S^U \subseteq \mathbb{R}^n$:

$$\mathbb{P}(\vec{X} \in S^U) \leq \mathbb{P}(\vec{Y} \in S^U), \quad (2.2)$$

where $\vec{X} \triangleq (X(t_1), X(t_2), \dots, X(t_n))$ and $\vec{Y} \triangleq (Y(t_1), Y(t_2), \dots, Y(t_n))$. Stochastic equality can be defined in a similar manner and is denoted by $\{X(t), t \in [0, \infty)\} =_{\text{st}} \{Y(t), t \in [0, \infty)\}$.

Roughly speaking, Eq. (2.2) implies that $\vec{X}$ is less likely than $\vec{Y}$ to take on large values, where “large” means any value in an upper set S^U . We also use $\Delta_\pi(t)$ to denote the AoI process under policy π . Furthermore, we define a set of parameters $\mathcal{I} = \{n, (t_i)_{i=1}^n\}$, where n is the number of updates and t_i is the generation time of update i . Having these definitions and notations, we are now ready to state Proposition 2.3.

⁴A set $S^U \subseteq \mathbb{R}^n$ is an upper set if $\vec{y} \in S^U$ whenever $\vec{y} \geq \vec{x}$ and $\vec{x} \in S^U$, where $\vec{x} = (x_1, \dots, x_n)$ and $\vec{y} = (y_1, \dots, y_n)$ are two vectors in $\mathbb{R}^n$ and $\vec{y} \geq \vec{x}$ if $y_i \geq x_i$ for all $i = 1, 2, \dots, n$.

Proposition 2.3. *In a $G/M/1$ queue, for all $\mathcal{I}$, the AoI under LCFS-I is stochastically smaller than that under LCFS, i.e.,*

$$[\{\Delta_{\text{LCFS-I}}(t), t \in [0, \infty)\} | \mathcal{I}] \leq_{\text{st}} [\{\Delta_{\text{LCFS}}(t), t \in [0, \infty)\} | \mathcal{I}]. \quad (2.3)$$

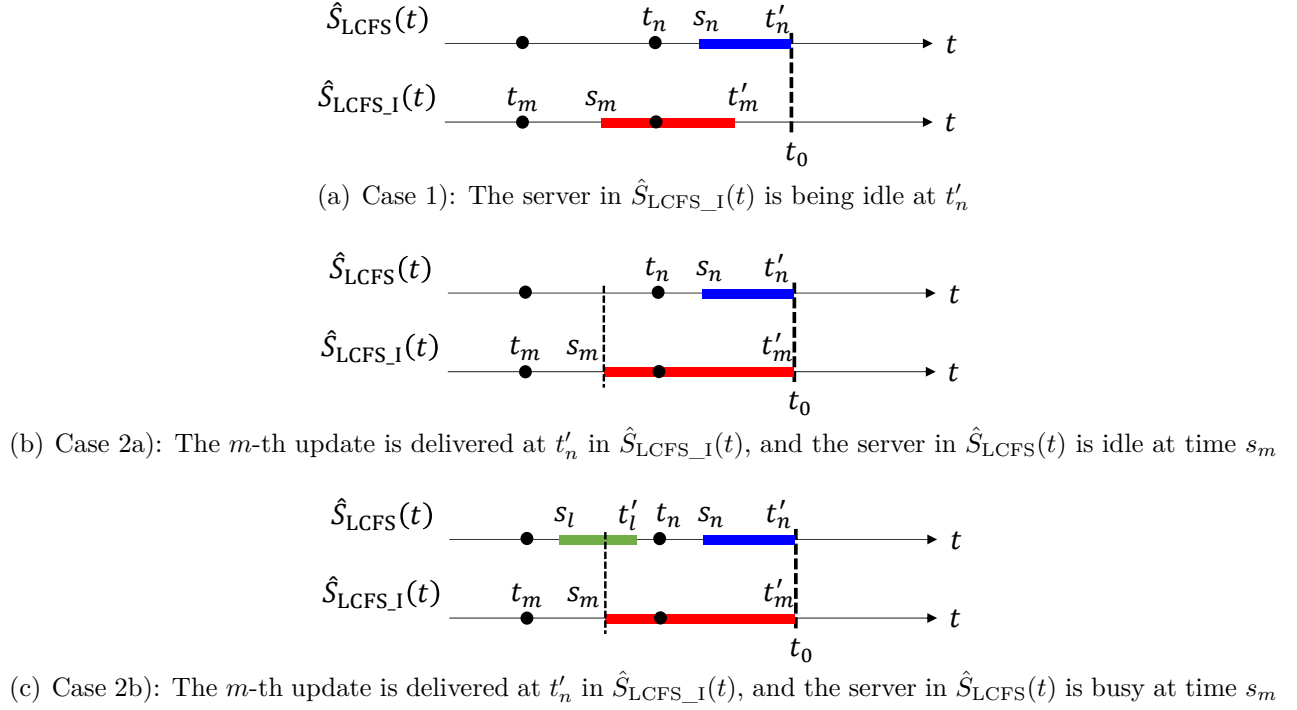
Proof. Recall that we use t_i and t'_i to denote the arrival time and the delivery time of the i -th update, respectively. In addition, we use s_i to denote the service start time of the i -th update.

We define the system state at time t under policy π as $S_\pi(t) \triangleq U_\pi(t)$, where $U_\pi(t)$ is the largest arrival time of the updates that have been served under policy π by time t . Let $\{S_\pi(t), t \in [0, \infty)\}$ be the state process under policy π . By the definition of AoI, Eq. (2.3) holds if the following holds:

$$[\{S_{\text{LCFS-I}}(t), t \in [0, \infty)\} | \mathcal{I}] \geq_{\text{st}} [\{S_{\text{LCFS}}(t), t \in [0, \infty)\} | \mathcal{I}]. \quad (2.4)$$

Next, we prove Eq. (2.4) by contradiction through a coupling argument. Suppose that stochastic processes $\hat{S}_{\text{LCFS-I}}(t)$ and $\hat{S}_{\text{LCFS}}(t)$ have the same stochastic laws as $S_{\text{LCFS-I}}(t)$ and $S_{\text{LCFS}}(t)$, respectively. We couple $\hat{S}_{\text{LCFS-I}}(t)$ and $\hat{S}_{\text{LCFS}}(t)$ in the following manner: If an update i is delivered at t'_i in $\hat{S}_{\text{LCFS}}(t)$, then the update j being served at t'_i (if any) in $\hat{S}_{\text{LCFS-I}}(t)$ is also delivered at the same time. This coupling is reasonable because: (i) the updates served in $\hat{S}_{\text{LCFS-I}}(t)$ are not chosen based on update size; (ii) the service time of an update in both $\hat{S}_{\text{LCFS-I}}(t)$ and $\hat{S}_{\text{LCFS}}(t)$ is exponentially distributed and has the memoryless property. By Theorem 6.B.30 in [34], Eq. (2.4) holds if the following holds:

$$\mathbb{P}(\hat{S}_{\text{LCFS-I}}(t) \geq \hat{S}_{\text{LCFS}}(t), t \in [0, \infty) | \mathcal{I}) = 1. \quad (2.5)$$

Figure 2.13: Part of sample path of $\hat{S}_{LCFS_I}(t)$ and $\hat{S}_{LCFS}(t)$ in different cases

In the following, we want to show that $\hat{S}_{LCFS_I}(t) \geq \hat{S}_{LCFS}(t)$ holds conditionally on an arbitrary sample path $\mathcal{I}$, which trivially implies Eq. (2.5). We prove it by contradiction. For the sake of contradiction, suppose that $\hat{S}_{LCFS_I}(t) < \hat{S}_{LCFS}(t)$ does happen and that it happens for the first time at time t_0 (see Fig. 2.13 for illustration). Let m and n be the index of the served updates with the largest arrival time by t_0 in $\hat{S}_{LCFS_I}(t)$ and $\hat{S}_{LCFS}(t)$, respectively. Then, we have $U_{LCFS_I}(t_0) = t_m$ and $U_{LCFS}(t_0) = t_n$. Note that we also have $t_m < t_n$ due to $\hat{S}_{LCFS_I}(t_0) < \hat{S}_{LCFS}(t_0)$ (i.e., $U_{LCFS_I}(t_0) < U_{LCFS}(t_0)$). Since t_0 is the first time when $\hat{S}_{LCFS_I}(t) < \hat{S}_{LCFS}(t)$ happens, a crucial observation is that t_0 must be immediately after an update is delivered in $\hat{S}_{LCFS}(t)$. Hence, we have $t_0 = (t'_n)^+$, where $(t'_n)^+$ denotes the time immediately after t'_n .

Due to the coupling between $\hat{S}_{LCFS}(t)$ and $\hat{S}_{LCFS_I}(t)$, there are two cases in $\hat{S}_{LCFS_I}(t)$: 1) the server is being idle at t'_n ; 2) an update is delivered at t'_n too. We discuss these two

cases separately and show that there is a contradiction in both cases.

Case 1): The server in $\hat{S}_{\text{LCFS_I}}(t)$ is being idle at t'_n (see Fig. 2.13(a)). Then, the most recently delivered update in $\hat{S}_{\text{LCFS_I}}(t)$ (i.e., the m -th update) must be delivered before t'_n . Hence, we have $t'_m < t'_n$ and that the server in $\hat{S}_{\text{LCFS_I}}(t)$ stays in the idle state during $(t'_m, t'_n]$. Then, the server in $\hat{S}_{\text{LCFS_I}}(t)$ could have started serving a newer update that arrives later than the m -th update immediately after t'_m . (Such a newer update must exist as the n -th update is a valid candidate due to $t_m < t_n$.) This results in a contradiction with the server being idle during $(t'_m, t'_n]$.

Case 2): An update is delivered at t'_n in $\hat{S}_{\text{LCFS_I}}(t)$. This delivered update is the m -th update. Note that we must have $s_m < t_n$. This is because if $s_m \geq t_n$, then the server in $\hat{S}_{\text{LCFS_I}}(t)$ would have chosen to serve the n -th update or a fresher update that arrives later than t_n at time s_m since this selected update is a newer update (due to $t_m < t_n$). There are two subcases for the server in $\hat{S}_{\text{LCFS}}(t)$ at time s_m : 2a) idle; 2b) busy. Again, we discuss these two subcases separately and show that there is a contradiction in both cases.

Case 2a): The server in $\hat{S}_{\text{LCFS}}(t)$ is idle at time s_m (see Fig. 2.13(b)). In this case, the m -th update must have already been delivered by time s_m in $\hat{S}_{\text{LCFS}}(t)$. Otherwise, the server in $\hat{S}_{\text{LCFS}}(t)$ would have started serving the m -th update (or a newer update) at or before s_m . This implies that $\hat{S}_{\text{LCFS_I}}(t) < \hat{S}_{\text{LCFS}}(t)$ happens before s_m , which results in a contradiction with that t_0 is the first time at which $\hat{S}_{\text{LCFS_I}}(t) < \hat{S}_{\text{LCFS}}(t)$ happens.

Case 2b): The server in $\hat{S}_{\text{LCFS}}(t)$ is busy at time s_m (see Fig. 2.13(c)). Assume that the l -th update is being served at s_m in $\hat{S}_{\text{LCFS}}(t)$. In this case, the l -th update must be delivered by time s_n in $\hat{S}_{\text{LCFS}}(t)$. This is because the n -th update starts service at s_n in $\hat{S}_{\text{LCFS}}(t)$. Then, the m -th update must also be delivered by time s_n in $\hat{S}_{\text{LCFS_I}}(t)$, due to the coupling between $\hat{S}_{\text{LCFS}}(t)$ and $\hat{S}_{\text{LCFS_I}}(t)$. This results in a contradiction that the m -th update is

delivered at t'_n .

Combining all the cases, we show that $\hat{S}_{\text{LCFS_I}}(t) \geq \hat{S}_{\text{LCFS}}(t)$ holds conditionally on an arbitrary sample path $\mathcal{I}$. This trivially implies Eq. (2.5), which further implies Eq. (2.4) by Theorem 6.B.30 in [34]. This completes the proof. $\square$

2.6.2 Preemptive, Informative, AoI-based Policies

So far, we have demonstrated the advantages of preemptive policies, AoI-based policies, and informative policies. In this subsection, we want to integrate all of these three ideas and propose preemptive, informative, AoI-based policies.

We first consider the preemptive, informative version of three AoI-based policies: ADE_PI, ADS_PI, and ADM_PI. We can show equivalence between ADE_PI and SRPT_I (i.e., the informative version of SRPT) and between ADE_I and SJF_I (i.e., the informative version of ADE and SJF, respectively) in the sample-path sense. These results are stated in Propositions 2.4 and 2.5.

Proposition 2.4. *ADE_PI and SRPT_I are equivalent in every sample path.*

Proof. We use strong induction to prove that under the same sample path, ADE_PI and SRPT_I always choose the same update to serve at the same time. In the following, we only consider informative updates since non-informative updates are discarded under both ADE_PI and SRPT_I.

Suppose that when ADE_PI needs to choose the n -th update to serve at time $t_{\text{ADE_PI}}(n)$, it chooses the update with index $d_{\text{ADE_PI}}(n)$. Similarly, SRPT_I chooses the update with index $d_{\text{SRPT_I}}(n)$ as its n -th update to serve at $t_{\text{SRPT_I}}(n)$.

Claim: ADE_PI and SRPT_I always serve the same update at the same time, i.e.,

$(d_{\text{ADE_PI}}(n), t_{\text{ADE_PI}}(n)) = (d_{\text{SRPT_I}}(n), t_{\text{SRPT_I}}(n))$ for all n .

Base case: When $n = 1$, both ADE_PI and SRPT_I serve the first update when it arrives. Hence, we have $(d_{\text{ADE_PI}}(1), t_{\text{ADE_PI}}(1)) = (d_{\text{SRPT_I}}(1), t_{\text{SRPT_I}}(1))$.

Induction step: Suppose that for $n = k$ ($k \geq 1$), we have $(d_{\text{ADE_PI}}(m), t_{\text{ADE_PI}}(m)) = (d_{\text{SRPT_I}}(m), t_{\text{SRPT_I}}(m))$ for the m -th update for all $1 \leq m \leq k$. We want to show that $(d_{\text{ADE_PI}}(n), t_{\text{ADE_PI}}(n)) = (d_{\text{SRPT_I}}(n), t_{\text{SRPT_I}}(n))$ still holds for $n = k+1$. Note that there are two cases for the $(k+1)$ -st update: 1) the $(k+1)$ -st update preempts the k -th update; 2) the $(k+1)$ -st update does not preempt the k -th update, i.e., the $(k+1)$ -st update starts service from the idle state or immediately after the k -th update is delivered. We discuss these two cases separately and show that $(d_{\text{ADE_PI}}(k+1), t_{\text{ADE_PI}}(k+1)) = (d_{\text{SRPT_I}}(k+1), t_{\text{SRPT_I}}(k+1))$ holds in both cases.

Case 1): The $(k+1)$ -st update preempts the k -th update. During the service of the k -th update, the $(k+1)$ -st update arrives. Under ADE_PI, in order to make AoI drop as early as possible, the server compares the remaining service time of the k -th update with the original service time of the $(k+1)$ -st update and chooses to serve the update with a smaller remaining service time. This is exactly the same as what SRPT_I does. Therefore, we have $(d_{\text{ADE_PI}}(k+1), t_{\text{ADE_PI}}(k+1)) = (d_{\text{SRPT_I}}(k+1), t_{\text{SRPT_I}}(k+1))$.

Case 2): The $(k+1)$ -st update does not preempt the k -th update. On the one hand, if the $(k+1)$ -st update starts service from the idle state, then by the induction hypothesis, both ADE_PI and SRPT_I finish serving the k -th update at the same time and then go through a period of being idle. Therefore, ADE_PI and SRPT_I will also serve the same $(k+1)$ -st update at the same time, i.e., $(d_{\text{ADE_PI}}(k+1), t_{\text{ADE_PI}}(k+1)) = (d_{\text{SRPT_I}}(k+1), t_{\text{SRPT_I}}(k+1))$. On the other hand, if the $(k+1)$ -st update starts service

immediately after the service of k -th update, then by the induction hypothesis, ADE_PI and SRPT_I will start service at the same time, i.e., $t_{\text{ADE_PI}}(k+1) = t_{\text{SRPT_I}}(k+1)$. SRPT_I will select the $(k+1)$ -st update with the shortest remaining size. However, this selected $(k+1)$ -st update must have not been served before. Otherwise, this update is no longer informative it was preempted by another update. Thus, SRPT_I ends up choosing an update with the shortest original size, which will also be selected by ADE_PI. This implies $d_{\text{ADE_PI}}(k+1) = d_{\text{SRPT_I}}(k+1)$. Therefore, we have $(d_{\text{ADE_PI}}(k+1), t_{\text{ADE_PI}}(k+1)) = (d_{\text{SRPT_I}}(k+1), t_{\text{SRPT_I}}(k+1))$. $\square$

Proposition 2.5. *ADE_I and SJF_I are equivalent in every sample path.*

Proof. Similar to the proof of Proposition 2.4, we use strong induction to show that under the same sample path, ADE_I and SJF_I always choose the same update to serve at the same time. Here, we also only consider the informative updates.

Suppose that when ADE_I needs to choose the n -th update to serve at time $t_{\text{ADE_I}}(n)$, it chooses the update with index $d_{\text{ADE_I}}(n)$. Similarly, SJF_I chooses the update with index $d_{\text{SJF_I}}(n)$ as its n -th update to serve at $t_{\text{SJF_I}}(n)$.

Claim: ADE_I and SJF_I always serve the same update at the same time, i.e., $(d_{\text{ADE_I}}(n), t_{\text{ADE_I}}(n)) = (d_{\text{SJF_I}}(n), t_{\text{SJF_I}}(n))$ for all n .

Base case: When $n = 1$, both ADE_I and SJF_I serve the first update when it arrives. Hence, we have $(d_{\text{ADE_I}}(1), t_{\text{ADE_I}}(1)) = (d_{\text{SJF_I}}(1), t_{\text{SJF_I}}(1))$.

Induction step: Suppose that for $n = k$ ($k \geq 1$), we have $(d_{\text{ADE_I}}(m), t_{\text{ADE_I}}(m)) = (d_{\text{SJF_I}}(m), t_{\text{SJF_I}}(m))$ for the m -th update for $1 \leq m \leq k$. We want to show that $(d_{\text{ADE_I}}(n), t_{\text{ADE_I}}(n)) = (d_{\text{SJF_I}}(n), t_{\text{SJF_I}}(n))$ still holds for $n = k+1$. Note that there are two cases for the $(k+1)$ -st update: 1) the $(k+1)$ -st update starts service from the idle

state; 2) the $(k + 1)$ -st update starts service immediately after the k -th update is delivered. We discuss these two cases separately and show that $(d_{\text{ADE_I}}(k + 1), t_{\text{ADE_I}}(k + 1)) = (d_{\text{SJF_I}}(k + 1), t_{\text{SJF_I}}(k + 1))$ holds in both cases.

Case 1): The $(k + 1)$ -st update starts service from the idle state. By the induction hypothesis, both ADE_I and SJF_I finish serving the k -th update at the same time and then go through a period of being idle. Therefore, ADE_I and SJF_I will also serve the same $(k + 1)$ -st update at the same time, i.e., $(d_{\text{ADE_I}}(k + 1), t_{\text{ADE_I}}(k + 1)) = (d_{\text{SJF_I}}(k + 1), t_{\text{SJF_I}}(k + 1))$.

Case 2): The $(k + 1)$ -st update starts service immediately after the k -th update is delivered. By the induction hypothesis, ADE_I and SJF_I will start service at the same time, i.e., $t_{\text{ADE_I}}(k + 1) = t_{\text{SJF_I}}(k + 1)$. SJF_I will choose the $(k + 1)$ -st update that has the smallest update size, which will also be selected by ADE_I since this update can make AoI drop earliest. This implies $d_{\text{ADE_PI}}(k + 1) = d_{\text{SJF_I}}(k + 1)$. Therefore, we have $(d_{\text{ADE_I}}(k + 1), t_{\text{ADE_I}}(k + 1)) = (d_{\text{SJF_I}}(k + 1), t_{\text{SJF_I}}(k + 1))$. $\square$

Propositions 2.4 and 2.5 imply that although SRPT_I and SJF_I do not explicitly follow an AoI-based design, they are essentially AoI-based policies. This provides an intuitive explanation for why size-based policies, such as variants of SRPT and SJF, have a good empirical AoI performance.

In Fig. 2.14, we present the simulation results for the average AoI performance of the preemptive, informative, AoI-based policies (ADE_PI) compared to several other policies. We observe that in various settings we consider, ADE_PI achieves the best AoI performance. However, compared to the best delay-efficient policies (such as SRPT), the AoI improvement of the preemptive, informative, and AoI-based policies is rather marginal in the settings with exogenous arrivals.

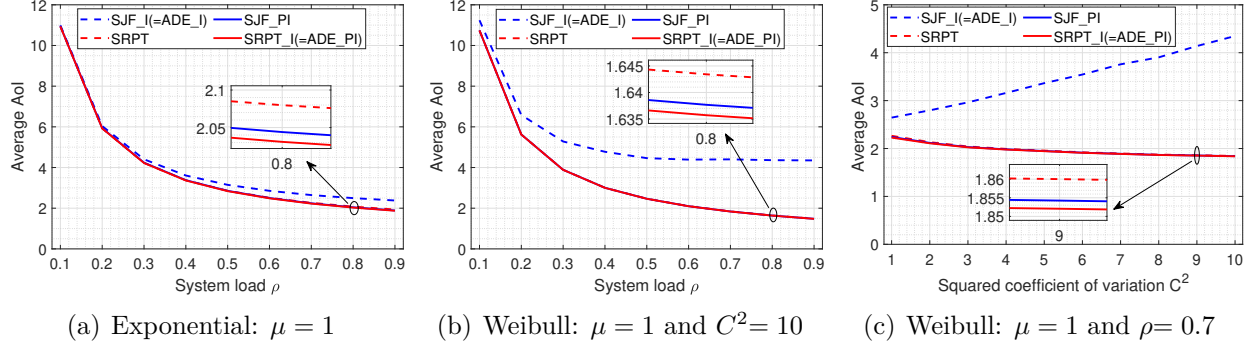


Figure 2.14: Comparisons of the average AoI performance: preemptive, informative, AoI-based policies vs. others

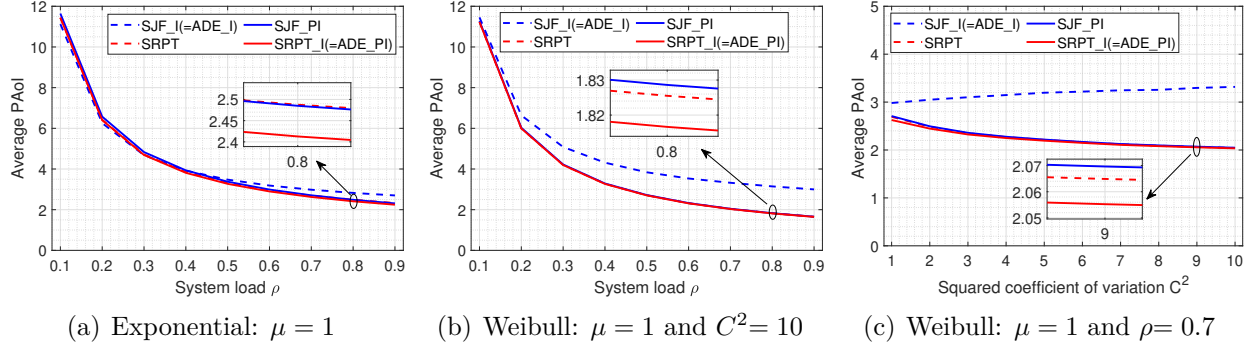


Figure 2.15: Comparisons of the average PAoI performance: preemptive, informative, AoI-based policies vs. others

2.7 Chapter Summary

In this chapter, we systematically studied the impact of various aspects of scheduling policies on AoI performance and provided several useful guidelines for the design of AoI-efficient scheduling policies. Our study reveals that among the various aspects of scheduling policies we investigated, prioritizing small updates, allowing service preemption, and prioritizing informative updates play the most important role in the design of AoI-efficient scheduling policies. It turns out that common scheduling policies like SRPT and SJF_P and their informative variants can achieve a very good AoI performance, although they do not explicitly make scheduling decisions based on the AoI. This can be partially explained by the

equivalence between such size-based policies and some AoI-based policies. Moreover, when the AoI requirement is not stringent or the update-size information is not available, some simple delay-efficient policies (such as LCFS_P) are also good candidates for AoI-efficient policies.

Our findings also raise several interesting questions that are worth investigating as future work. One important direction is to pursue more theoretical results beyond the simulation results we provided in this chapter. For example, it would be interesting to see whether one can rigorously prove that any informative policy always outperforms its non-informative counterpart, which is consistently observed in the simulation results.

Chapter 3

Towards the Tradeoff Between Service Performance and Information Freshness

3.1 Introduction

The last decade has witnessed an unprecedented growth in the demand for data-driven real-time services (built on frameworks such as Apache Storm [35]). These services are fueled by emerging applications that require rapidly injecting data streams and computing updated analytics results in *real-time* (or *near-real-time*). For such applications, *service performance* (e.g., response time) perceived by end users is typically a primary concern and has been extensively studied in the literature. Yet, *freshness* of the information received by end users, another equally or even more important concern, has not received enough attention. Unilaterally optimizing service performance without accounting for information freshness could render users receive stale information, which is potentially of much less value or even useless. For example, upon receiving a user query, in order to minimize the response time the service provider may respond immediately with currently available but possibly outdated information. On the other hand, it may choose to first process new updates and then responds with fresher information if the goal is to optimize freshness. Hence, *there*

exists a natural tradeoff between service performance (e.g., response time) and information freshness.

In this chapter, we consider the *response time* as a performance metric and the *Age of Information (AoI)* [3] as a freshness metric. While the response time has been shared as a standard performance metric, the AoI, which is defined as the time elapsed since the generation of the freshest update among those that have been delivered to the receiver (see Section 3.3 for the formal definition), is a recently proposed freshness/timeliness metric [3]. Note that there is a limited body of existing work (see, e.g., [36, 37, 38]) that investigates the important tradeoff between performance and freshness as we do. However, all of these studies provide heuristic solutions only and fall short of theoretical results with rigorous analysis.

To that end, in this chapter, we aim to fill this important gap and design efficient policies that can properly address the critical tradeoff between performance and freshness. We summarize the main contributions of this chapter as follows.

First, we propose a simple *single-server two-queue model* that captures the coupled scheduling of updates and queries. Second, after demonstrating the limitations of the First-Come-First-Served (FCFS) policy, we propose two *threshold-based policies*: the Query- k policy that prioritizes queries and the Update- k policy that prioritizes updates. Then, we rigorously analyze the response time and the Peak AoI (PAoI) (i.e., the maximum value of the AoI at the server immediately before a new update is processed) [7] of these two policies. To the best of our knowledge, *this is the first analytical work that systematically studies the tradeoff between performance and freshness in a rigorous manner*. Further, we propose the Joint- (M, N) policy, which allows flexibly prioritizing updates or queries through choosing different values of two thresholds M and N . Finally, we conduct simulations to evaluate the response time and the PAoI of the proposed policies. The results show that our pro-

posed threshold-based policies can effectively control the balance between performance and freshness.

The rest of this chapter is organized as follows. We first discuss related work in Section 3.2. Then, we describe our proposed model in Section 3.3. In Section 3.4, we analyze the response time and the PAoI of our proposed threshold-based policies, followed by a discussion on the simulation results in Section 3.5. Finally, we make concluding remarks in Section 3.6.

3.2 Related Work

Research on the performance (e.g., response time) started very early. In [39], it studies under what condition such that the performance in FCFS policy is better than that in *Process-Sharing (PS)* policy. The authors conclude that a special task assignment which has the ability to inspect incoming tasks and assign them to hosts for service can achieve this goal. Further, the performance comparison between the *Shortest-Remaining-Processing-Time (SRPT)* policy and the PS policy is studied in [40], it shows SRPT has better performance than PS when the service load is high. Based on the SRPT policy, the work of [41] improves the performance by giving preference to the queries whose remaining size or original size is small. The simulation results show that even the queries for large files suffer little in this SRPT-based scheduling. The first analytical study of performance and robustness in threshold-based resource allocation policies appears in [42], where the authors conclude that using multiple thresholds does not always provide benefits to performance and robustness. However, there is still no analytical work considering the freshness in these studies.

The notion of AoI is formally introduced in [3], where the authors analyze the time average AoI in M/M/1, M/D/1, and D/M/1 systems under the FCFS policy. Since this seminal work, the study on the AoI has attracted a lot of research interest. There is a large

body of work that focuses on the analysis of the AoI under a number of queueing models. For example, the work of [7, 12, 33] focuses on the model where the updates arrive according to the Poisson process and are served by a single server. There is another body of work that considers how to minimize the AoI by carefully designing scheduling policies in different scenarios (e.g., wireless networks [43, 44] and energy harvesting networks [45, 46]). In [10], the authors propose the Pull model for investigating the expected AoI at the user's side and discover a new tradeoff between different levels of information freshness and different response times across the servers. Besides the above work that focuses on the analysis and optimization of the AoI, several other work also considers applications where the AoI is highly relevant (see, e.g., [47, 48]).

Despite the aforementioned studies on service performance and information freshness, the tradeoff between them has often been neglected in the literature (partially due to the nature of the considered applications), except for the following limited work. In [36], the tradeoff of performance and freshness has been considered for database-driven web servers, where the goal is to optimize performance under the freshness constraint. The work of [37] proposes to combine performance and freshness into a single compound metric and addresses the tradeoff between them through optimizing the compound metric. Further, the work of [37] has been extended to account for user preference for performance and freshness [38]. In stark contrast to these studies that provide heuristic solutions only, in this chapter we aim to systematically understand this tradeoff by providing theoretical results with rigorous analysis.

3.3 System Model

In this section, we describe the single-server two-queue model and give the formal definition of the AoI and the PAoI.

We consider a queueing system where a single server is shared by two M/M/1 queues. One is the update queue that buffers updates coming from the information source, and the other is the query queue that buffers queries from the user. We assume that the arrival processes of the updates and the queries are both Poisson with the rate λ_u and λ_q , respectively. Also, we assume that the service times of the updates and the queries are both exponentially distributed with mean $1/\mu_u$ and $1/\mu_q$, respectively. Therefore, the loads of the update queue and query queue can be denoted by $\rho_u = \lambda_u/\mu_u$ and $\rho_q = \lambda_q/\mu_q$, respectively. In addition, we assume that the server does not remain at an empty queue if the other queue is nonempty. Further, let $X_{u,i}$ be the inter-arrival time between the i -th update and the $(i-1)$ -th update, let $S_{u,i}$ be the service time of the i -th update, let $T_{u,i}$ be the system time of the i -th update, and let $N_{u,i}^u$ (resp., $N_{u,i}^q$) be the number of updates (resp., queries) seen by the i -th update upon its arrival. More generally, we drop subscript i and use X_u , S_u , and T_u to denote the corresponding quantities for an ordinary update. For example, X_u denotes the inter-arrival time of an update. Similarly, we define $X_{q,i}$, $S_{q,i}$, $T_{q,i}$, $N_{q,i}^u$, $N_{q,i}^q$, X_q , S_q , and T_q for queries. Also, we use N_u (resp., N_q) to denote the number of updates (resp., queries) in the system.

Next, we give the formal definition of the AoI and the PAoI. Let $U(t)$ denote the generation time of the freshest update among those that have been processed by the server. We use $\Delta(t)$ to denote the AoI at time t , which is defined as the time elapsed since the generation of this freshest update, i.e., $\Delta(t) \triangleq t - U(t)$. An example of the AoI evolution is shown in Fig. 3.1. The AoI increases linearly as time goes on until a new update is completely processed. For example, consider the i -th update, which is generated at time t_i and finishes

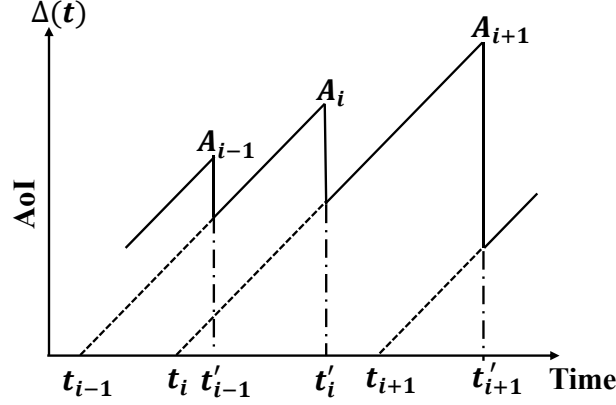


Figure 3.1: An example of the AoI evolution

processing at time t'_i . When the server finishes processing the i -th update, the AoI drops to the value of $t'_i - t_i$, i.e., the system time of the i -th update. Then, the average AoI can be defined as

$$\Delta = \lim_{\tau \rightarrow \infty} \frac{1}{\tau} \int_0^\tau \Delta(t) dt. \quad (3.1)$$

Analyzing the average AoI involves two important quantities: the inter-arrival time and the system time of the updates. The fact that the latter is dependent on the former often renders the analysis of the average AoI quite challenging except for some simple settings (e.g., M/M/1 queue) [3]. On the other hand, the analysis of the average PAoI is usually more tractable. The PAoI is the maximum value of the AoI achieved immediately before a new update is processed. Let A_i be the i -th PAoI. From Fig. 3.1, we can see $A_i = t'_i - t_{i-1}$. This can be rewritten as the sum of the inter-arrival time between the i -th update and the previous update (i.e., $t_i - t_{i-1}$) and the system time of the i -th update (i.e., $t'_i - t_i$). Therefore, the expected PAoI can be expressed as

$$\mathbb{E}[A] = \mathbb{E}[X_u] + \mathbb{E}[T_u], \quad (3.2)$$

where $\mathbb{E}[\cdot]$ is the expectation operator and A is the PAoI corresponding to an update.

Table 3.1: Summary of Key Notations

Symbol	Meaning
λ_u	Arrival rate of the updates
μ_u	Service rate of the updates
ρ_u	Load of the update queue (i.e., $\rho_u = \lambda_u/\mu_u$)
$X_{u,i}$	Inter-arrival time between the i -th and $(i-1)$ -th updates
$S_{u,i}$	Service time of the i -th update
$T_{u,i}$	System time of the i -th update
$N_{u,i}^u$	Number of updates seen by the i -th update upon arrival
$N_{u,i}^q$	Number of queries seen by the i -th update upon arrival
N_u	Number of updates in the system
A_i	The i -th PAoI
λ_q	Arrival rate of the queries
μ_q	Service rate of the queries
ρ_q	Load of the query queue (i.e., $\rho_q = \lambda_q/\mu_q$)
$X_{q,i}$	Inter-arrival time between the i -th and $(i-1)$ -th queries
$S_{q,i}$	Service time of the i -th query
$T_{q,i}$	System time (or response time) of the i -th query
$N_{q,i}^u$	Number of updates seen by the i -th query upon arrival
$N_{q,i}^q$	Number of queries seen by the i -th query upon arrival
N_q	Number of queries in the system
ρ	Total load (i.e., $\rho = \rho_u + \rho_q$)

While computing the first term of the right hand side (RHS) of Eq. (3.2) is trivial, i.e., $\mathbb{E}[X_u] = 1/\lambda_u$, computing the second term $\mathbb{E}[T_u]$ is more involved as it depends on the underlying scheduling policy.

To measure the service performance, we consider the average response time, i.e., the system time of the queries T_q .

For quick reference, we provide a summary of the key notations of this chapter in Table 3.1.

3.4 Scheduling Policies

In this section, we first consider a simple scheduling policy, the FCFS policy, and explain its limitation in balancing service performance and information freshness. Then, we propose two threshold-based policies: the Query- k policy that prioritizes queries and the Update- k policy that prioritizes updates, and rigorously analyze the response time and the PAoI under these policies. Further, we propose the Joint- (M, N) policy, where we jointly set thresholds M and N for the updates and the queries, respectively. The Joint- (M, N) policy generalizes the Query- k policy and the Update- k policy and allows flexibly prioritizing updates or queries through choosing different values of M and N .

3.4.1 The FCFS Policy

We first consider the FCFS policy, a simple policy that serves updates and queries according to the order of their arrivals. Preemption is not allowed during the service. The main results for the FCFS policy are stated in Proposition 3.1.

Proposition 3.1. *Under the FCFS policy, the expected response time is*

$$\mathbb{E}[T_q] = \frac{\rho_u/\mu_u + (1 - \rho_u)/\mu_q}{1 - \rho_u - \rho_q}, \quad (3.3)$$

and the expected PAoI is

$$\mathbb{E}[A] = \frac{1}{\lambda_u} + \frac{\rho_q/\mu_q + (1 - \rho_q)/\mu_u}{1 - \rho_u - \rho_q}. \quad (3.4)$$

Proof. Consider an arbitrary update, say the i -th update. It is easy to see that its system

time consists of the following three components: (i) the total service time of all the other updates that are already in the system upon its arrival; (ii) the total service time of all the queries are already in the system upon its arrival; (iii) its own service time. Taking the sum of these three components, we have

$$T_{u,i} = \sum_{j=i_u^*}^{i_u^*+N_{u,i}^u-1} S_{u,j} + \sum_{k=i_q^*}^{i_q^*+N_{u,i}^q-1} S_{q,k} + S_{u,i}, \quad (3.5)$$

where i_u^* (resp., i_q^*) is the index of the first update (resp., query) seen by the i -th update upon its arrival to the system.

Taking the expectation of both sides of Eq. (3.5), we have

$$\mathbb{E}[T_{u,i}] = \mathbb{E}\left[\sum_{j=i_u^*}^{i_u^*+N_{u,i}^u-1} S_{u,j}\right] + \mathbb{E}\left[\sum_{k=i_q^*}^{i_q^*+N_{u,i}^q-1} S_{q,k}\right] + \mathbb{E}[S_{u,i}]. \quad (3.6)$$

Note that in Eq. (3.6), the $S_{u,j}$'s are *i.i.d.* with mean $\mathbb{E}[S_u]$, and they are independent of $N_{u,i}^u$. Similarly, the $S_{q,k}$'s are *i.i.d.* with mean $\mathbb{E}[S_q]$, and they are independent of $N_{u,i}^q$. Hence, by applying Wald's equation to the first term and the second term of the RHS of Eq. (3.6), respectively, we have

$$\mathbb{E}\left[\sum_{k=i_q^*}^{i_q^*+N_{u,i}^q-1} S_{q,k}\right] = \mathbb{E}[S_q] \cdot \mathbb{E}[N_q] \quad (3.7)$$

and

$$\mathbb{E}\left[\sum_{j=i_u^*}^{i_u^*+N_{u,i}^u-1} S_{u,j}\right] = \mathbb{E}[S_u] \cdot \mathbb{E}[N_u]. \quad (3.8)$$

Then, by plugging Eqs. (3.7) and (3.8) into Eq. (3.6) and using the fact that $\mathbb{E}[S_{u,i}] = \mathbb{E}[S_u]$,

we have

$$\mathbb{E}[T_u] = \mathbb{E}[N_u] \cdot \mathbb{E}[S_u] + \mathbb{E}[N_q] \cdot \mathbb{E}[S_q] + \mathbb{E}[S_u]. \quad (3.9)$$

By applying the Little's Law (i.e., $\mathbb{E}[N_u] = \lambda_u \mathbb{E}[T_u]$ and $\mathbb{E}[N_q] = \lambda_q \mathbb{E}[T_q]$) and using the fact that $\mathbb{E}[S_u] = 1/\mu_u$ and $\mathbb{E}[S_q] = 1/\mu_q$, we simplify Eq. (3.9) as

$$\mathbb{E}[T_u] = \rho_u \mathbb{E}[T_u] + \rho_q \mathbb{E}[T_q] + \frac{1}{\mu_u}. \quad (3.10)$$

Similarly, the expected system time of the queries can be expressed as

$$\mathbb{E}[T_q] = \rho_u \mathbb{E}[T_u] + \rho_q \mathbb{E}[T_q] + \frac{1}{\mu_q}. \quad (3.11)$$

Solving Eqs. (3.10) and (3.11) yields Eq. (3.3) and the following:

$$\mathbb{E}[T_u] = \frac{\rho_q/\mu_q + (1 - \rho_q)/\mu_u}{1 - \rho_u - \rho_q}. \quad (3.12)$$

Then, plugging Eq. (3.12) into Eq. (3.2) and using the fact that $\mathbb{E}[X_u] = 1/\lambda_u$, we obtain Eq. (3.4) and complete the proof. $\square$

The FCFS policy is a simple algorithm and is easy to implement in practice. However, a key limitation is that the FCFS policy does not provide a knob for prioritizing either updates or queries and thus cannot achieve a desired balance between service performance and information freshness. To that end, in the following subsections, we will propose threshold-based policies that can prioritize either queries or updates and thus control the tradeoff between the response time and the PAoI.

3.4.2 The Query- k Policy

In this subsection, we propose the Query- k policy that sets a threshold k for the query queue and prioritizes the queries whenever the length of the query queue reaches k . We will analyze the response time and the PAoI under this policy.

Specifically, the Query- k policy functions in the following manner: (i) there is one single threshold k for the query queue; (ii) when the server is currently serving the update queue, the server has to switch from the update queue to the query queue instantly either if the number of queries reaches the threshold k (thus preemption is allowed in this policy) or the update queue becomes empty; (iii) no work of updates is lost due to the switches; (iv) once the server switches to the query queue, it needs to empty all queries waiting in the queue before it switches back to the update queue; (v) within each queue, FCFS is applied.

In the following, we will discuss three cases of the threshold value: 1) $k = 1$, 2) $1 < k < \infty$, and 3) $k = \infty$.

3.4.2.1 Threshold $k=1$

In this case, the server processes queries as long as the query queue is non-empty. Hence, the query queue always has a higher priority than the update queue. This model is equivalent to a preemptive priority queue with two classes of jobs [18, Ch. 32].

Proposition 3.2. *Under the Query-1 policy, the expected response time is*

$$\mathbb{E}[T_q] = \frac{1}{\mu_q} + \frac{\rho_q/\mu_q}{1 - \rho_q}, \quad (3.13)$$

and the expected P AoI is

$$\mathbb{E}[A] = \frac{1}{\lambda_u} + \frac{1/\mu_u}{1 - \rho_q} + \frac{\rho_q/\mu_q + \rho_u/\mu_u}{(1 - \rho_q)(1 - \rho_q - \rho_u)}. \quad (3.14)$$

Proof. Consider a more general model where jobs are classified into N classes [18, Ch. 32]. Assume that Class 1 has the highest priority, Class 2 has the second highest priority, and so on. A job of the higher class can preempt a job of the lower class, but no work is lost due to preemption. Then, the expected system time for a job of Class n (where $n = 1, 2, \dots$), denoted by $T(n)$, can be expressed as

$$\mathbb{E}[T(n)] = \frac{\mathbb{E}[S_n]}{1 - \sum_{i=1}^{n-1} \rho_i} + \frac{\sum_{i=1}^n \rho_i \frac{\mathbb{E}[S_i^2]}{2\mathbb{E}[S_i]}}{(1 - \sum_{i=1}^{n-1} \rho_i)(1 - \sum_{i=1}^n \rho_i)}, \quad (3.15)$$

where S_i is the service time of jobs of Class i and ρ_i is the load due to jobs of Class i . In our model, there are two classes: the queries belong to Class 1, and the updates belong to Class 2. Hence, by plugging $n = 1$ and $n = 2$ into Eq. (3.15) and using the fact that $\mathbb{E}[T_q] = \mathbb{E}[T(1)]$ and $\mathbb{E}[T_u] = \mathbb{E}[T(2)]$, respectively, we obtain Eq. (3.13) and the following:

$$\mathbb{E}[T_u] = \frac{1/\mu_u}{1 - \rho_q} + \frac{\rho_q/\mu_q + \rho_u/\mu_u}{(1 - \rho_q)(1 - \rho_q - \rho_u)}. \quad (3.16)$$

Then, by plugging Eq. (3.16) into Eq. (3.2) and using the fact that $\mathbb{E}[X_u] = 1/\lambda_u$, we obtain Eq. (3.14) and complete the proof. $\square$

3.4.2.2 Threshold $1 < k < \infty$

Since the threshold k is now larger than 1, the query queue has a higher priority than the update queue only when the threshold $k > 1$ is reached. In other words, the query queue

no longer has an absolute priority over the update queue. Hence, the analysis techniques used for the case of $k = 1$ are not applicable here. Instead, we will exploit the techniques developed in [49] to analyze the response time and the PAoI.

The work of [49] studies a similar two-queue model and considers a threshold-based policy. The authors propose a method for calculating the probability-generating functions, which can be used for calculating the expected queue length. We will use the method of [49] in the analyses of the response time and the PAoI under the Query- k policy.

We first define the following steady-state probabilities:

$$\begin{aligned} p_{ij} &= \mathbb{P}(N_q = i, N_u = j, Z = 1), i \geq 1, j \geq 0, \\ q_{ij} &= \mathbb{P}(N_q = i, N_u = j, Z = 2), 0 \leq i < k, j \geq 1, \\ r_{00} &= \mathbb{P}(N_q = 0, N_u = 0), \end{aligned} \tag{3.17}$$

where $Z = 1$ if the server is serving the query queue and $Z = 2$ if the server is serving the update queue. Apparently, the steady-state probabilities should satisfy the following:

$$r_{00} + \sum_{i=1}^{\infty} \sum_{j=0}^{\infty} p_{ij} + \sum_{i=0}^{k-1} \sum_{j=1}^{\infty} q_{ij} = 1. \tag{3.18}$$

The corresponding probability-generating functions are

$$\begin{aligned} P(x, y) &= \sum_{i=1}^{\infty} \sum_{j=0}^{\infty} p_{ij} x^{i-1} y^j, \\ Q_i(y) &= \sum_{j=1}^{\infty} q_{ij} y^{j-1}, 0 \leq i < k. \end{aligned} \tag{3.19}$$

Using the probability-generating functions in Eq. (3.19), we can express the expected queue

length of the query queue as

$$\mathbb{E}[N_q] = \frac{d}{dx} xP(x, y=1) \Big|_{x=1} + \sum_{i=0}^{k-1} iQ_i(y=1). \quad (3.20)$$

We are now ready to state the main results for the Query- k policy with $1 < k < \infty$.

Proposition 3.3. *Under the Query- k policy with $1 < k < \infty$, the expected response time is*

$$\mathbb{E}[T_q] = \mathbb{E}[N_q]/\lambda_q, \quad (3.21)$$

and the expected PAoI is

$$\mathbb{E}[A] = \frac{1}{\lambda_u} + \frac{\mu_u}{\lambda_u} \cdot \left(\frac{\lambda_q/\mu_q^2 + \lambda_u/\mu_u^2}{1 - \rho} - \frac{\mathbb{E}[N_q]}{\mu_q} \right), \quad (3.22)$$

where $\mathbb{E}[N_q]$ is given by Eq. (3.20).

Proof. First, simply applying Little's Law for the query queue (i.e., $\mathbb{E}[N_q] = \lambda_q \mathbb{E}[T_q]$) yields Eq. (3.21).

Next, by applying the Conservation Law [50, pp. 236–238], we can also calculate the expected queue length of the update queue as

$$\mathbb{E}[N_u] = \mu_u \cdot \left(\frac{\lambda_q/\mu_q^2 + \lambda_u/\mu_u^2}{1 - \rho} - \frac{\mathbb{E}[N_q]}{\mu_q} \right). \quad (3.23)$$

Then, by applying the Little's Law for the update queue (i.e., $\mathbb{E}[N_u] = \lambda_u \mathbb{E}[T_u]$) in Eq. (3.23) to get $\mathbb{E}[T_u]$, plugging it into Eq. (3.2), and using the fact that $\mathbb{E}[X_u] = 1/\lambda_u$, we obtain Eq. (3.22) and complete the proof. $\square$

Remark: To compute $\mathbb{E}[N_q]$ in Eqs. (3.21) and (3.22), we need to solve the global balance

equations along with Eq. (3.18), express the probability-generating functions $P(x, y)$ and $Q_i(y)$'s in terms of λ_q , λ_u , μ_q , and μ_u , and plug them into Eq. (3.20). The detailed derivation can be found in [49] and is omitted here.

3.4.2.3 Threshold $k = \infty$

In this case, since the threshold of the query queue is infinity, the server switches to serving the queries only when the update queue becomes empty. Then, it keeps serving the queries. Only when the query queue becomes empty, the server switches to serving the updates. Therefore, the system reduces to the classical two-queue model with exhaustive service at both queues (i.e., all jobs waiting in the current queue will be served before the server turns to the other queue) [51]. The work of [51] presents a method for deriving the distribution of waiting time for updates and queries. By using their method, we can obtain the expected system time for updates and queries, respectively, which can further be used to analyze the response time and the PAoI.

3.4.3 The Update- k Policy

Similar to the Query- k policy that prioritizes the queries, we propose another threshold-based policy, called the Update- k policy, which prioritizes the updates. Similarly, we will discuss three cases: 1) $k = 1$, 2) $1 < k < \infty$, and 3) $k = \infty$.

3.4.3.1 Threshold $k=1$

In this case, the server always gives a higher priority to the update queue. Hence, the updates now belong to Class 1, and the queries belong to Class 2. We state the following proposition and omit the proof as it is similar to that of Proposition 3.2.

Proposition 3.4. *Under the Update-1 policy, the expected response time is*

$$\mathbb{E}[T_q] = \mathbb{E}[T(2)] = \frac{1/\mu_q}{1 - \rho_u} + \frac{\rho_q/\mu_q + \rho_u/\mu_u}{(1 - \rho_u)(1 - \rho_q - \rho_u)}, \quad (3.24)$$

and the expected PAoI is

$$\mathbb{E}[A] = \mathbb{E}[X_u] + \mathbb{E}[T(1)] = \frac{1}{\lambda_u} + \frac{1}{\mu_u} + \frac{\rho_u/\mu_u}{1 - \rho_u}. \quad (3.25)$$

3.4.3.2 Threshold $1 < k < \infty$

This case is similar to the case of the Query- k policy with $1 < k < \infty$. Following the same line of analysis as that in the proof of Proposition 3.3, we can compute $\mathbb{E}[N_u]$ using the techniques developed in [49] and analyze the expected response time and the PAoI. We state the main results in Proposition 3.5 and omit the detailed proof.

Proposition 3.5. *Under the Update- k policy with $1 < k < \infty$, the expected response time is*

$$\mathbb{E}[T_q] = \frac{\mu_q}{\lambda_q} \cdot \left(\frac{\lambda_q/\mu_q^2 + \lambda_u/\mu_u^2}{1 - \rho} - \frac{\mathbb{E}[N_u]}{\mu_u} \right), \quad (3.26)$$

and the expected PAoI is

$$\mathbb{E}[A] = 1/\lambda_u + \mathbb{E}[N_u]/\lambda_u. \quad (3.27)$$

3.4.3.3 Threshold $k = \infty$

Same as the Query- k policy, the system reduces to the classical two-queue model with exhaustive service at both queues. The analysis will be exactly the same as that of the Query- k policy with $k = \infty$.

3.4.4 The Joint- (M, N) Policy

In the previous two subsections, we have been focused on threshold-based policies that prioritize either queries or updates. The analyses reveal the following insights: the priority is given to the queue with a threshold; the lower the threshold, the higher the degree of priority. Take the Query- k policy for example. When $k = 1$, the query queue always has a higher priority; when $k = \infty$, the query queue no longer has a higher priority, because the system reduces to the classical two-queue model with exhaustive service at both queues. Hence, one limitation of the single-threshold-based policies is that the priority is given to one queue only.

Next, we introduce the Joint- (M, N) policy, where we jointly set thresholds M and N for the updates and the queries, respectively. This policy generalizes the Query- k policy and the Update- k policy and allows flexibly prioritizing updates or queries through choosing different values of M and N .

Specifically, the Joint- (M, N) policy functions in the following manner: (i) the update queue has a threshold M , and the query queue has a threshold N ; (ii) the server immediately switches to the queue whose queue length reaches its threshold and continues to serve this queue as long as the threshold of the other queue is not reached; (iii) if both thresholds are reached, the server will serve the queue with a new arrival.

The Query- k policy and the Update- k policy are two special cases of the Joint- (M, N) policy, where $(M = \infty, N = k)$ and $(M = k, N = \infty)$, respectively. When $1 < M < \infty$ and $1 < N < \infty$, the Joint- (M, N) policy becomes more flexible in prioritizing updates and queries. We leave the analyses of the general Joint- (M, N) policy as our future work. However, in Section 3.5 we provide simulation results to demonstrate its advantages compared to the one-threshold-based policies.

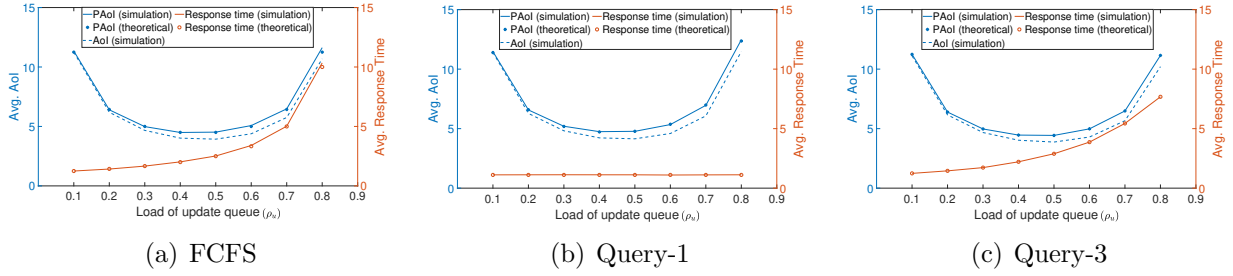


Figure 3.2: Performance comparisons of different policies with varying update load ($\lambda_q = 0.1$ and $\mu_q = \mu_u = 1$)

3.5 Numerical Results

In this section, we conduct simulations to evaluate the response time and the PAoI of the proposed policies. We first consider the FCFS policy and demonstrate its limitations. Then, we show that the single-threshold-based policies (i.e., the Query- k policy and the Update- k policy) have the ability to effectively control the tradeoff between the response time and the PAoI. Finally, we demonstrate the flexibility of the Joint- (M, N) policy. We implement and simulate these policies in Java. In the simulation results, each data point is the average of 10 runs, and each run lasts 20,000 time units. We also include our analytical results computed using Wolfram Mathematica for the purpose of comparison.

We first simulate the FCFS policy and assume $\lambda_q = 0.1$ and $\mu_q = \mu_u = 1$. The results are presented in Fig. 3.2(a). The results show that both the average PAoI and the average AoI decrease first and then increase as the update load increases. When the update load is low, the PAoI and the AoI are large due to large inter-arrival times of the updates; when the update load is high, the PAoI and the AoI are also large due to large queueing delays. On the other hand, the response time keeps increasing as a larger update load can only worsen the congestion condition for queries. Hence, when the update load is high, the response time and the PAoI can both be poor since the FCFS policy does not prioritize either queries or

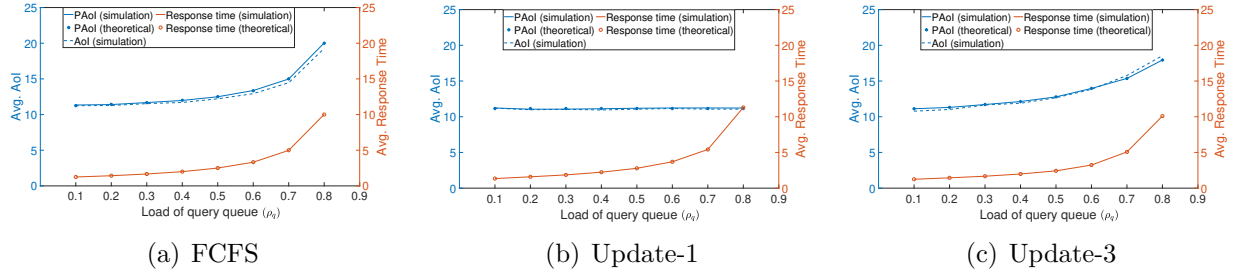


Figure 3.3: Performance comparisons of different policies with varying query load ($\lambda_u = 0.1$ and $\mu_q = \mu_u = 1$)

updates.

Next, we consider the Query- k policy and assume $\lambda_q = 0.1$ and $\mu_q = \mu_u = 1$. The results are presented in Figs. 3.2(b) and 3.2(c). We can observe from Fig. 3.2(b) that the average response time remains unchanged under the Query-1 policy since the queries are always given a higher priority than the updates, while the PAoI is only slightly larger than that under the FCFS policy (e.g., 12.46 vs. 11.62 when $\rho_u = 0.8$). Fig. 3.2(c) shows that under the Query-3 policy, the response time keeps increasing as the update load increases, but it is still better than that under the FCFS policy. Compared to the Query-1 policy, while the PAoI is a little smaller (e.g., 11.15 vs. 12.46 when $\rho_u = 0.8$), the response time becomes much worse. Therefore, one needs to carefully choose the value of the threshold so as to effectively control the tradeoff between the response time and the PAoI. Note that the PAoI does not vary much under different policies due to a small query rate of 0.1.

Similarly, we compare the FCFS policy with the Update- k policy with different values of k , by assuming $\lambda_u = 0.1$ and $\mu_q = \mu_u = 1$ and varying the query load. The results are presented in Fig. 3.3, where similar observations can be made.

Further, we investigate the impact of different values of the threshold under the threshold-based policies and present the results in Fig. 3.4. We assume $\lambda_u = \lambda_q = 1/3$ and $\mu_u = \mu_q = 1$. Fig. 3.4(a) shows that under the Query- k policy, as the threshold k increases, while

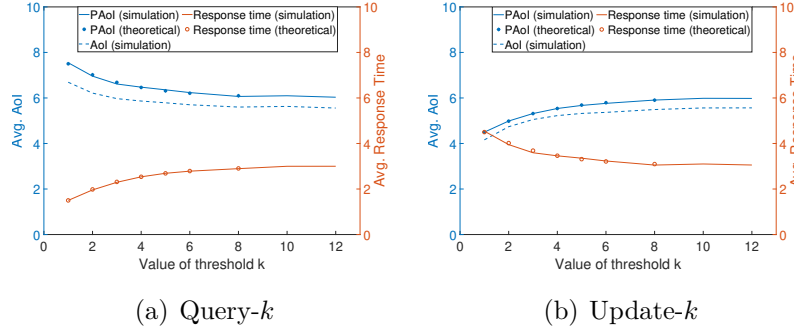


Figure 3.4: Impact of the threshold on the single-threshold-based policies ($\lambda_u = \lambda_q = 1/3$ and $\mu_u = \mu_q = 1$)

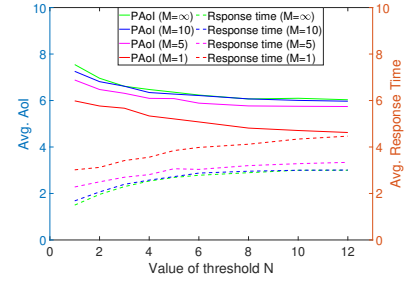


Figure 3.5: Impact of different values of the thresholds M and N on the Joint- (M, N) policy ($\lambda_u = \lambda_q = 1/3$ and $\mu_u = \mu_q = 1$)

the PAoI and the AoI decrease, the response time increases. This is because the degree of priority given to queries becomes lower as k increases. Such behavior saturates when k reaches a certain value (e.g., around $k = 8$ in Fig. 3.4(a)). Similar observations can be made in Fig. 3.4(b), which shows the results for the Update- k policy.

Finally, we also simulate the Joint- (M, N) policy. Assuming $\lambda_u = \lambda_q = 1/3$ and $\mu_u = \mu_q = 1$, we investigate the impact of different values of the thresholds M and N on the response time and the PAoI. The results are presented in Fig. 3.5. We observe that the larger (resp., smaller) the value of M (resp., N), the higher the PAoI and the lower the response time. Therefore, the Joint- (M, N) policy allows more flexibly prioritizing updates or queries through choosing different values of the two thresholds (i.e., M and N).

3.6 Chapter Summary

In this chapter, we proposed a simple single-server two-queue model that captures the coupled scheduling between updates and queries for data-driven real-time applications. Aiming to address the natural tradeoff between service performance and information freshness in

such applications, we proposed threshold-based scheduling policies that prioritize updates or queries and analyzed the response time and the PAoI in a rigorous manner. The simulation results further demonstrated that by properly choosing the values of the thresholds, the proposed policies can achieve the desired balance between service performance and information freshness.

Although this chapter provides useful insights towards the tradeoff between the response time and the PAoI, there remain some open questions, which will be investigated in our future work. For example, it would be interesting to rigorously analyze the average AoI under the threshold-based policies and to systematically study the Joint- (M, N) policy. In addition, we have implicitly assumed that there was a negligible overhead for the server to switch back and forth between the query queue and the update queue. It would be interesting to investigate and characterize the impact of switching cost.

Chapter 4

Towards Optimal Tradeoff Between Data Freshness and Update Cost in Information-update Systems

4.1 Introduction

With the remarkable development of communication networks and smart portable devices in recent years, we have witnessed significant advances in crowdsensing-based applications (e.g., Google Waze [52] and GasBuddy [53]). These applications provide services to users by resorting to the community to sense and send back real-time information (e.g., traffic conditions and gas prices) [54]. To satisfy the diverse needs of users, such applications need to maintain their knowledge of a set of distributed points of interest (PoI). For example, GasBuddy monitors gasoline prices at a large number of scattered gas stations in a certain area. In order to quickly and accurately respond to users' requests, the applications need to keep their data fresh. However, given the dynamic changes of the data, maintaining the freshness of data introduces a natural tradeoff between data freshness and update cost. On the one hand, users are unsatisfied if the responses to their requests are outdated; on the other hand, there is a cost for the applications to update their data because updating data relies on user feedback and often requires monetary payment to incentivize users [53, 54].

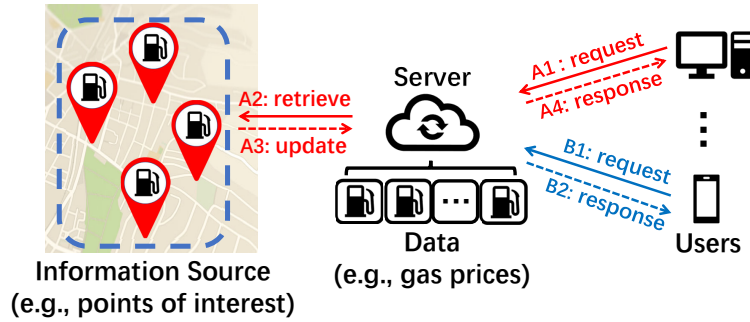


Figure 4.1: An illustration of our system model. Upon receiving a request from the users, the server can either first update the data and then reply (red path: A1-A4) or simply reply with local data (blue path: B1-B2).

In fact, the tradeoff between data freshness and update cost does not only exist in crowdsensing-based applications, but also in a wide variety of time-sensitive data-driven applications that require timely information updates [3, 55, 56, 57, 58]. For example, in stock analysis applications, the server keeps track of the prices of a large number of stocks and generates different versions of the analysis reports for the stocks at certain times, and users send requests to query these analysis reports [58]. To ensure that users receive the real-time analysis of the stock they are trading with, the server needs to retrieve timely information (e.g., various stock market indexes) from the stock market, which incurs an update cost (e.g., bandwidth resources). Similar applications also include news feeds, weather updates, and flight aggregators.

The aforementioned applications have two notable characteristics: First, the server can proactively retrieve information from the information source to update its data, and users need to query the server to obtain the data (i.e., the “Pull” model [10, 55]); Second, the responses to users’ requests (e.g., gas prices) typically do not require significant processing, and the packet size is usually small, making the packet transmission time negligible. However, retrieving the data from the information source often requires certain resources and introduces costs. These two characteristics not only distinguish such applications from other

ones whose update costs mainly come from service and communication delays [13, 59, 60] but also lead to the tradeoff between the data freshness and the update cost.

To that end, in this work, we aim to optimize the tradeoff between data freshness and update cost. Specifically, we consider a discrete-time system in the setting where a service provider can proactively retrieve information from the information source and users obtain the data at the service provider by sending requests (see Fig. 4.1). The freshness of the data received by users is measured by a popular timeliness metric called *Age-of-Information (AoI)* [3], which is defined as the time elapsed since the most recent update occurred. To represent the dissatisfaction of users receiving stale data, we introduce the *staleness cost*, which is a non-decreasing function of the AoI (see formal definition in Section 4.3). Clearly, one needs to account for both the update cost and the staleness cost when designing an online update policy.

We summarize our main contributions as follows.

First, we study the tradeoff between the data freshness and the update cost by formulating an optimization problem to minimize the sum of the staleness cost (which is a function of the AoI) and the update cost.

Second, we provide two useful guidelines for the design of optimal update policies. These guidelines suggest that 1) the service provider should update the data only at a point when it receives a request, and 2) the server should perform an update when the staleness cost is no smaller than the update cost.

Third, following these guidelines and assuming that the request arrival process is Bernoulli, we reformulate our problem as a Markov decision process (MDP) and show that there exists a threshold-based policy that is optimal among all online policies, which motivates us to focus on the class of threshold-based policies. Furthermore, we derive the closed-form expression

of the average cost under any threshold-based policy and obtain the optimal threshold.

Finally, we perform extensive simulations using both synthetic data and real traces to verify our theoretical results and evaluate the performance of our proposed policy compared with several baseline policies. Our simulation results show that the threshold-based policy outperforms the baselines in more general settings (e.g., when the request arrival process is non-Bernoulli).

The remainder of this chapter is organized as follows. We first discuss related work in Section 4.2. The system model is described in Section 4.3. Two guidelines for designing update policies are provided in Section 4.4. Then, we prove that our MDP formulation admits an optimal threshold-based policy and derive the optimal threshold in Sections 4.5 and 4.6, respectively. Finally, we present the numerical results in Section 4.7 and conclude our chapter in Section 4.8.

4.2 Related Work

Ever since the concept of AoI was introduced in [3], the study on the AoI has attracted a lot of research interest. There is a large body of work that provides detailed analyses on the AoI performance of information-update systems under different queueing models (M/M/1, M/D/1, etc.) and scheduling policies (FCFS, LCFS, etc.) [7, 8].

Another important line of research focuses on AoI minimization. One specific type of optimization problem, which is similar to our work, is the joint minimization of AoI and certain costs [61, 62, 63]. In [61], the authors consider a discrete-time system where an information source is monitored over a communication channel with a transmission cost. They investigate the optimal policy for minimizing the sum of transmission cost and the

inaccuracy of the state information at the monitor. It turns out that the optimal policies also have a threshold-based structure. Note that in their model, they assume that the source is governed by a random walk process and there are no users, which is different from ours. A similar source monitoring problem is considered in [62], where the goal is to minimize the sum of transmission costs and an unknown, time-varying penalty function of the AoI. They consider both single-source and multi-source scenarios and propose online learning algorithms with provable regret. In [63], the authors consider a source-monitor pair with stochastic arrival of updates at the source. The source pays a transmission cost to send the update, and its goal is to minimize the weighted sum of AoI and transmission costs. Under the assumption that the update arrival process is Poisson, they propose an optimal threshold-based policy. Their work differs from ours in their continuous-time setting and no user involvement.

Along this line, researchers have also considered AoI minimization with constraints (see a survey in [6]). The considered constraints can be viewed as a special type of update cost. For example, in wireless networks, the update of data consumes wireless channel resources. Therefore, the number of packets that can be transmitted depends on the interference model [44]. Similarly, for caching services, the cache server can only update certain contents at a time due to the capacity constraint [64, 65, 66, 67]. Another example is the energy constraint [68, 69, 70, 71], which is common in energy-constrained IoT systems. In these models, the update cost is usually imposed as a constraint of the optimization problem.

While the tradeoff between AoI and costs has been studied, most of them fall into the category where the costs primarily come from service (e.g., CPU cycle and storage) and/or communication (e.g., channel resources, delay, and energy consumption) caused by large packet size, which is different from our model, where the packet size is usually small and the update costs come from the retrieval of data from the information source. In addition, we

also emphasize users' perspective in our model (i.e., the “Pull” model [10, 55]), where users can proactively query the server to obtain the data.

4.3 System Model and Problem Formulation

We consider a discrete-time information-update system that consists of an information source, a service provider (or server for short), and multiple users (see Fig. 4.1). The server can communicate with the information source and update its data with the latest information. The users need to query the server to obtain the data.

We consider an aggregated arrival process formed by the requests from all the users (which will be assumed as Bernoulli process in Section 4.5 for further analysis). The requests arrive at the beginning of the time-slot, and the server replies to the requests with the most recently updated data at the end of a time-slot. We use the metric *Age-of-Information (AoI)* to measure the freshness of data, which is defined as the time elapsed since the most recent update. For ease of exhibition, we assume that the AoI drops to 0 after the update at the end of a time-slot¹. The evolution of $\Delta(t)$ is as follows:

$$\Delta(t) = \begin{cases} \Delta(t-1) + 1, & \text{if } u(t) = 0; \\ 0, & \text{if } u(t) = 1, \end{cases} \quad (4.1)$$

where $u(t)$ indicates whether the server updates the data at time-slot t . We assume that the time-slot is indexed from 1 and the initial AoI also equals 1, i.e., $\Delta(1) = 1$. Let u_i denote the i -th update time. Then, an update policy π can be denoted by the update times: $\pi \triangleq \{u_i^\pi\}_{i=1}^\infty$. An illustration of a typical AoI evolution is shown in Fig. 4.2. To reflect the

¹Some work also assumes that the AoI drops to 1 [62, 72]. We assume that the AoI drops to 0 to make the discussion concise and clear.

dissatisfaction level of the users when they receive stale data, we also introduce a *staleness cost* for each response of the server. Specifically, the staleness cost is defined as a penalty function $f(\Delta)$ of the AoI Δ , where the function $f : [0, \infty) \mapsto [0, \infty)$ is assumed to be measurable, non-negative, and non-decreasing. For simplicity, we let $f(0) = 0$.

At the beginning of each time-slot, the server can decide whether to update the data or not. If it does, it needs to pay a constant update cost p and receives the latest data from the information source at the end of time-slot. To avoid the staleness cost, the server can first update the data and then reply to the request with the latest data. Let r_j be the arrival time of the j -th request. After the server receives the request at r_j , if the server chooses to update the data before replying to the request, its AoI drops to 0 after the update and its staleness cost becomes $f(0) = 0$. In such a case, the server needs to pay an update cost p though. Otherwise, if the server does not update and replies with the current local data, the server needs to pay a staleness cost $f(\Delta(r_j))$.

Assume that the number of updates during the process of serving N requests under policy π is $U^\pi(N)$, i.e.,

$$U^\pi(N) \triangleq \max\{i | u_i^\pi \leq r_N\}. \quad (4.2)$$

Then, the total cost of serving N requests, which is the sum of update costs and the staleness costs, is defined as

$$C^\pi(N) \triangleq \sum_{j=1}^N f(\Delta(r_j)) + pU^\pi(N). \quad (4.3)$$

The objective is to find an update policy π that minimizes *the long-term average expected cost per request (or average cost for short)*, which is defined as

$$\bar{C}^\pi \triangleq \lim_{N \rightarrow \infty} \frac{\mathbb{E}[C^\pi(N)]}{N}, \quad (4.4)$$

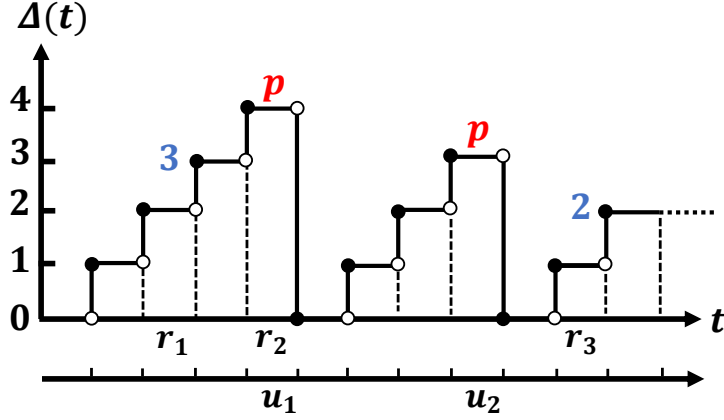


Figure 4.2: An illustration of the AoI evolution at the server. There are two updates (in time-slots u_1 and u_2) during the process of serving three requests (in time-slots r_1 , r_2 , and r_3).

where the expectation is taken over the randomness in the arrival process and the update policy. Here, we assume that the limit of average cost under policy π exists. We focus on the set of online policies, denoted by Π , under which the information available at time t for making update decisions includes the update history, the arrival times of requests that arrive until t , and the update cost p . Then, we can formulate the following optimization problem:

$$\min_{\pi \in \Pi} \bar{C}^{\pi}. \quad (4.5)$$

4.4 Guidelines for Algorithm Design

In this section, we provide two useful guidelines for the design of efficient update policies. Through a sample-path-dominance argument, we show that policies following these guidelines can achieve a lower total cost than those that do not. Therefore, we can reduce the search space of problem (4.5) to a certain class of online policies.

4.4.1 Reactive Policies

In this subsection, we present our first guideline for the design of update policies. As described in Section 4.3, the server can update the data at any time. However, we show that to achieve a lower total cost, it is sufficient for the update policy to just consider updating the data immediately upon receiving a new request. We call such policies *Reactive Policies* as the server does not need to update the data when there is no request. We use Π^R to denote the set of reactive policies:

$$\Pi^R \triangleq \{\pi \in \Pi \mid u_k^\pi \in \{r_j\}_{j=1}^\infty \text{ for all } k\}. \quad (4.6)$$

Next, we show that restricting to reactive policies does not incur any performance loss.

Lemma 4.1. *For any policy $\pi \in \Pi$, there exists a reactive policy $\pi' \in \Pi^R$ that achieves an average cost no larger than that of policy π , i.e., $\bar{C}^{\pi'} \leq \bar{C}^\pi$.*

We provide the detailed proof in Appendix B.1 and explain the key ideas as follows. Intuitively, postponing the update until a request arrives does not increase the total cost because the total number of updates remains the same, but doing this achieves a lower staleness cost since the update time is closer to the request arrival time. Therefore, reactive policies can achieve a smaller total cost than those non-reactive policies.

Lemma 4.1 implies that the search space of Problem (4.5) can be further reduced from the set of online policies Π to the set of reactive policies Π^R . Now, consider any reactive policy π . Upon receiving a request, the server needs to decide whether to update the data or not before responding. Therefore, we use I_j^π to denote the decision made by the server

upon receiving the j -th request for the data at time r_j :

$$I_j^\pi \triangleq \begin{cases} 1, & \text{if the server updates the data at time } r_j; \\ 0, & \text{otherwise.} \end{cases}$$

4.4.2 Capped Reactive Policies

In this subsection, we present the second guideline for the design of update policies. In Section 4.4.1, we show that the reactive policies achieve a smaller or equal average cost by postponing the update until a request arrives. In fact, after the server receives the request, if the staleness cost is no smaller than the update cost, it is better for the server to update the data to avoid a larger staleness cost. Doing so not only leads to a smaller cost for this request but also benefits the next few requests. We use Π^{R+} to denote the set of reactive policies that satisfy the above guideline:

$$\Pi^{R+} \triangleq \{\pi \in \Pi^R \mid I_j^\pi = 1 \text{ for all } j \text{ when } f(\Delta(r_j)) \geq p\}.$$

That is, for any policy $\pi \in \Pi^{R+}$, it must update the data when the staleness cost is no smaller than the update cost; otherwise, it can choose to update the data or not. We call such policies *Capped Reactive Policies* because the staleness cost of such policies is capped by the update cost. Fig. 4.3 illustrates the relationship between Π^R and Π^{R+} . Note that the condition $f(\Delta(r_j)) \geq p$ can also be expressed as $\Delta(r_j) \geq \Delta^*$, where Δ^* is the smallest AoI such that the staleness cost is no smaller than the update cost, i.e.,

$$\Delta^* \triangleq \min\{\Delta \mid f(\Delta) \geq p\}. \quad (4.7)$$

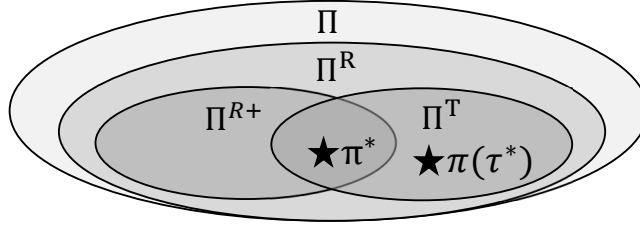


Figure 4.3: The relationship between online policies Π , reactive policies Π^R , capped reactive policies Π^{R+} , threshold-based policies Π^T , an overall optimal policy π^* (see Theorem 4.3), and an optimal threshold-based policy $\pi(\tau^*)$ (see Corollary 4.6). Note that policy $\pi(\tau^*)$ is also an overall optimal policy and could be the same as policy π^* in some cases.

In the following, we show that restricting to capped reactive policies does not incur any performance loss.

Lemma 4.2. *For any policy $\pi \in \Pi^R$, there exists a capped reactive policy $\pi' \in \Pi^{R+}$ that achieves an average cost no larger than that of policy π , i.e., $\bar{C}^{\pi'} \leq \bar{C}^\pi$.*

We provide the detailed proof in Appendix B.2 and explain the key ideas in the following. Upon receiving a request, policy π' performs an update if the staleness cost is no smaller than the update cost. Compared to policy π that does not make such an update, doing so incurs an update cost for policy π' , but it avoids a larger staleness cost. Besides, it also reduces the staleness cost for the requests that arrive thereafter. Therefore, policy π' can achieve a total cost no larger than that of policy π .

By Lemma 4.2, we can further reduce the search space to the class of capped reactive policies Π^{R+} . Therefore, Problem (4.5) can be further reduced to the following:

$$\min_{\pi \in \Pi^{R+}} \bar{C}^\pi. \quad (4.8)$$

Till this point, we do not make any assumption on the request arrival process. The aforementioned guidelines can be applied to general request arrival processes. In the following,

unless otherwise specified, we focus on the capped reactive policies Π^{R+} . *This capped property plays an important role in characterizing the threshold-based structure of an optimal policy for solving the MDP formulation in Section 4.5.*

4.5 MDP Formulation and Threshold structure

Under a capped reactive policy, the server makes update decisions upon receiving requests and pays a cost (an update cost or a staleness cost) based on the decision. Naturally, this sequential decision process can be modeled as an MDP. In this section, we assume that the request arrival process is Bernoulli with rate $\lambda \in (0, 1)$, denoted by $\text{Bernoulli}(\lambda)$, i.e., the probability that a request arrives in a time-slot is λ . Then, we reformulate Problem (4.8) as a discrete-time MDP and show that there exists a stationary threshold-based policy that solves the Bellman equation of the considered MDP and is thus optimal among all online policies.

The MDP formulation has the following key components: $\{\mathcal{N}, \mathcal{S}, \mathcal{A}_s, p(\cdot \mid s, a), c(s, a) : n \in \mathcal{N}, s \in \mathcal{S}, a \in \mathcal{A}_s\}$, where

1. $\mathcal{N} = \{1, 2, \dots\}$ is the set of decision epochs. Under a capped reactive policy, the n -th decision epoch is the time-slot when the n -th request arrives.
2. $\mathcal{S} = \{0, 1, \dots\}$ is the set of system states (which are all possible values of the AoI). We use s_n to denote the AoI value when the n -th request arrives.
3. $\mathcal{A}_s$ is the set of actions when the system state is s . Let $a \in \mathcal{A}_s$ denote the possible actions, where $a = 1$ means updating the data and $a = 0$ means not. Under a capped reactive policy, there are two sets of actions depending on the state s : when the staleness cost $f(s)$ is no smaller than the update cost p , the only available action is

to update, i.e., $\mathcal{A}_{\{s:f(s)\geq p\}} = \{1\}$; otherwise, the system can either update or not, i.e., $\mathcal{A}_{\{s:f(s)<p\}} = \{0, 1\}$.

4. The transition probability can be calculated as

$$p(z \mid s, a) = \begin{cases} (1 - \lambda)^{z-1} \lambda, & \text{if } z \geq 1 \text{ and } a = 1; \\ (1 - \lambda)^{z-s-1} \lambda, & \text{if } z > s, f(s) < p, \text{ and } a = 0; \\ 0, & \text{otherwise.} \end{cases}$$

That is, when the system is in state s , if the server updates the data, the system will enter state z ($z \geq 1$) with probability $(1 - \lambda)^{z-1} \lambda$ because the request arrival process follows Bernoulli(λ); otherwise, if the server does not update, under a capped reactive policy, the system will enter state z ($z > s$) with probability $(1 - \lambda)^{z-s-1} \lambda$ only when the staleness cost $f(s)$ is smaller than the update cost p .

5. The cost at each decision epoch can be expressed as

$$c(s, a) = \begin{cases} p, & \text{if } a = 1; \\ f(s), & \text{if } a = 0. \end{cases} \quad (4.9)$$

That is, when the system is in state s , updating the data incurs an update cost of p ; otherwise, there is a staleness cost of $f(s)$. Note that under a capped reactive policy, we always have $c(s, a) \leq p$.

The objective of the MDP is to find a stationary capped reactive stationary update policy

that minimizes the long-term average expected cost, i.e.,

$$\min_{\pi \in \Pi^{R+}} \lim_{N \rightarrow \infty} \frac{\mathbb{E}_{\pi} \left[\sum_{n=1}^N c(s_n, a_n) | s_1 = s \right]}{N}, \quad (4.10)$$

where $\mathbb{E}_{\pi}[\cdot]$ represents the conditional expectation, given that policy π is employed; s_n and a_n are the state and action taken at decision epoch n , respectively; and s is the initial state. We emphasize that, unlike traditional MDP formulations that mainly focus on optimization over time, we optimize our objective over users' requests. This allows us to reduce the state space and thus simplify the analysis. Note that the objective in Problem (4.10) is the same as that in Problem (4.8) except that we specify the initial state s in Problem (4.10). In other words, an optimal policy for Problem (4.10) is also an optimal policy for Problem (4.8). Next, we show that there exists an optimal policy for Problem (4.10) that has a threshold-based structure, which enables us to search for an optimal policy in the class of threshold-based policies (see Section 4.6). We state this result in Theorem 4.3.

Theorem 4.3. *There exists an optimal stationary capped reactive policy $\pi^* \in \Pi^{R+}$ that has a threshold-based structure.*

We provide the detailed proof in Appendix B.3 and present an outline of the proof in the following. First, we study a discounted MDP and derive its optimal value function. Second, based on the optimal value function, we derive the Bellman equation of the expected total average cost and show that the Bellman equation has a threshold-based structure. Specifically, the server needs to update the data when the current AoI value (i.e., the state) is no smaller than a certain fixed threshold s^* (see definition in Eq. (B.6)); otherwise, it does not. Now consider a stationary capped reactive policy $\pi^* \in \Pi^{R+}$ that makes update decisions based on threshold s^* . Apparently, policy π^* minimizes the Bellman equation for any state, thus it is an overall optimal policy [73, Chapter V, Theorem 2.1]. The threshold structure

of the optimal policy π^* indicates that among all threshold-based policies, there is an overall optimal policy (see Fig. 4.3). This motivates us to search for the optimal threshold-based policy in Section 4.6.

4.6 Optimal Threshold-based Policy

Theorem 4.3 tells us that we can further reduce the search space from the set of capped reactive policies to the set of capped reactive threshold-based policies. In this section, we formally define threshold-based policies and derive the closed-form expression of the average cost of the threshold-based policies. Using the closed-form expression, we can find the optimal threshold-based policy. Furthermore, we show that the optimal threshold-based policy is also an optimal policy among all online policies.

We begin with the definition of threshold-based policies.

Definition 4.4 (Threshold-based Policies). A policy in Π^R is called a threshold-based policy if it performs updates according to the following rule with a predetermined positive integer threshold τ : for the request arriving at time r_j , we have

$$I_j = \begin{cases} 1, & \Delta(r_j) \geq \tau; \\ 0, & \Delta(r_j) < \tau. \end{cases}$$

That is, the server updates the data at r_j before replying if the AoI at r_j is no smaller than threshold τ ; otherwise, the server simply replies with the current local data.

We consider an integer threshold because the values of the AoI are integers. Let $\pi(\tau)$ be the threshold-based policy with threshold τ , and let Π^T be the set of all threshold-based policies. Fig. 4.3 shows the relationship of Π^R , Π^{R+} , and Π^T .

Assuming that the request arrival process is Bernoulli, we can derive the closed-form expression of the average cost under any threshold-based policy. We state this result in Theorem 4.5.

Theorem 4.5. *Assume that the request arrival process is Bernoulli(λ), the staleness cost function is $f(\Delta)$, and the update cost is p . Then, for any policy $\pi(\tau) \in \Pi^T$ with a positive integer threshold τ , the average expected cost can be computed as follows:*

$$\bar{C}^{\pi(\tau)} = \frac{\lambda \sum_{t=1}^{\tau-1} f(t) + p}{\lambda(\tau - 1) + 1}. \quad (4.11)$$

We provide the detailed proof in Appendix B.6 and present an outline of the proof in the following. Since the request arrival process is Bernoulli, under a threshold-based update policy, the lengths of update intervals are independent and identically distributed (*i.i.d.*). Thus, the update process is a renewal process. Due to the ergodicity of the process, the expected average cost can be computed as $\bar{C}^{\pi(\tau)} = \mathbb{E}[C_k]/\mathbb{E}[N_k]$, where $\mathbb{E}[C_k]$ and $\mathbb{E}[N_k]$ are the expected total cost and the expected number of requests in the k -th update interval, respectively. In addition, by exploiting the properties of Bernoulli arrival process, we can further derive the closed-form expressions of $\mathbb{E}[C_k]$ and $\mathbb{E}[N_k]$, which are shown in the numerator and denominator of Eq. (4.11), respectively.

With the result in Theorem 4.5, we can also easily compute the optimal threshold τ^* . In fact, this optimal threshold-based policy $\pi(\tau^*)$ is also an overall optimal policy, which is shown in Corollary 4.6.

Corollary 4.6. *Assume that the request arrival process is Bernoulli(λ). Then, the threshold*

of the optimal threshold-based policy $\pi(\tau^*) \in \Pi^T$ is the following:

$$\tau^* = \left\{ \arg \min_{\tau \in \{\lfloor \tau' \rfloor, \lceil \tau' \rceil\}} \bar{C}^{\pi(\tau)} \right\}, \quad (4.12)$$

where τ' is the real number that achieves the smallest expected average cost (i.e., $\tau' = \arg \min_{\tau > 0} \bar{C}^{\pi(\tau)}$). Furthermore, the optimal threshold-based policy $\pi(\tau^*) \in \Pi^T$ is also an overall optimal policy among all online policies.

Proof. Theorem 4.3 states that there exists an overall optimal capped reactive threshold-based policy $\pi^* \in \Pi^{R+} \cap \Pi^T$. For the optimal threshold-based policy $\pi(\tau^*) \in \Pi^T$, we have $\bar{C}^{\pi(\tau^*)} \leq \bar{C}^{\pi^*}$. On the other hand, since policy π^* is an overall optimal policy, we also have $\bar{C}^{\pi(\tau^*)} \geq \bar{C}^{\pi^*}$. Therefore, we have $\bar{C}^{\pi(\tau^*)} = \bar{C}^{\pi^*}$. This implies that the optimal threshold-based policy $\pi(\tau^*) \in \Pi^T$ is also an overall optimal policy among all online policies. $\square$

Here, the real number τ' in Eq. (4.12) can either be theoretically calculated if the expression of $\sum_{t=1}^{\tau-1} f(t)$ in $\bar{C}^{\pi(\tau)}$ is known (see below for two examples) or be numerically calculated otherwise. The optimal threshold τ^* may not be unique, depending on the staleness cost function f . Also, policy $\pi(\tau^*)$ could be the same as policy π^* in some cases.

In the following, we provide the average expected cost and optimal threshold when the staleness cost is a linear function and a square function of the AoI, respectively.

Example 4.7 (A linear staleness cost function). Assume that the request arrival process is Bernoulli(λ), $f(\Delta) = \Delta$, and the update cost is p . Then, for any policy $\pi(\tau) \in \Pi^T$ with a positive integer threshold τ , we have

$$\bar{C}^{\pi(\tau)} = \frac{\lambda\tau(\tau-1)/2 + p}{\lambda(\tau-1) + 1}, \quad (4.13)$$

and the optimal threshold $\tau^* = \{\arg \min_{\tau \in \{\lfloor \tau' \rfloor, \lceil \tau' \rceil\}} \bar{C}^{\pi(\tau)}\}$, where $\tau' = (\sqrt{2p\lambda - \lambda + 1} + \lambda - 1)/\lambda$.

Example 4.8 (A quadratic staleness cost function). Assume that the request arrival process is Bernoulli(λ), $f(\Delta) = \Delta^2$, and the update cost is p . Then, for any policy $\pi(\tau) \in \Pi^T$ with a positive integer threshold τ , we have

$$\bar{C}^{\pi(\tau)} = \frac{\lambda [(\tau - 1)^3/3 + (\tau - 1)^2/2 + (\tau - 1)/6] + p}{\lambda(\tau - 1) + 1}, \quad (4.14)$$

and the optimal threshold $\tau^* = \{\arg \min_{\tau \in \{\lfloor \tau' \rfloor, \lceil \tau' \rceil\}} \bar{C}^{\pi(\tau)}\}$, where τ' is the solution of

$$1 - 6p - 6\tau + 6\tau^2 + \lambda(4\tau - 1)(\tau - 1)^2 = 0.$$

4.7 Numerical and Experimental Results

In this section, we perform extensive simulations and experiments to verify our theoretical results and compare the performance of the optimal threshold-based policy with several baseline policies using both synthetic data and real traces. Throughout this section, we consider two types of staleness cost: a linear function (i.e., $f(\Delta) = \Delta$) and a quadratic function (i.e., $f(\Delta) = \Delta^2$).

We first evaluate the performance of threshold-based policies with different thresholds when the staleness cost is a linear function of AoI in Fig. 4.4. The setting of the simulations is as follows. The request arrival process is Bernoulli with rate $\lambda = 0.1$, and the update cost is $p = 100$. The simulation results are the average of 100 simulation runs, where each run consists of $N = 10^4$ requests (which is our default setting for the synthetic simulations). We also include a breakdown of the results in terms of average staleness cost

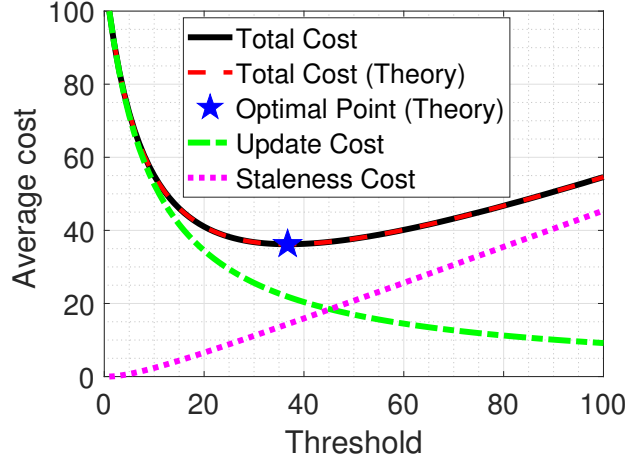


Figure 4.4: Average cost under threshold-based policies with different thresholds when $f(\Delta) = \Delta$, where $\lambda = 0.1$ and $p = 100$.

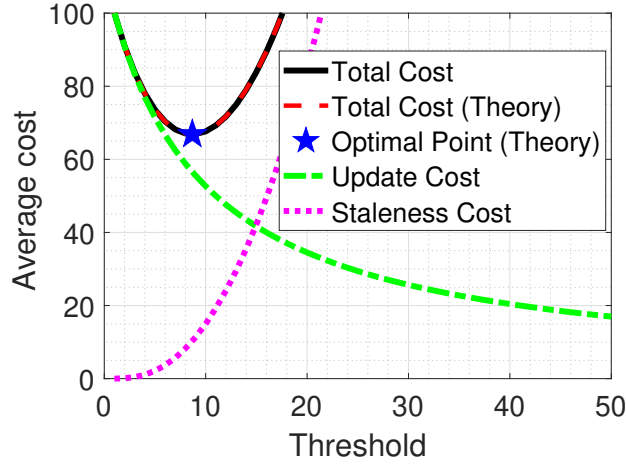


Figure 4.5: Average cost under threshold-based policies with different thresholds when $f(\Delta) = \Delta^2$, where $\lambda = 0.1$ and $p = 100$.

$\bar{a} \triangleq \sum_{j=1}^N f(\Delta(r_j))/N$ and average update cost $\bar{p} \triangleq pU^\pi(N)/N$. We observe that the simulation results of average total cost under threshold-based policies perfectly match the theoretical results in Example 4.7. Clearly, as the threshold increases, the update cost decreases, but the staleness cost increases. This is as expected because a higher threshold leads to less frequent updates, which results in a smaller update cost but a larger staleness cost. As a result, the average total cost, which is the sum of the two, first decreases and then

increases. The optimal cost $C^{\pi(\tau^*)} \approx 36.22$ is achieved at $\tau^* = \{\arg \min_{\{\lfloor \tau' \rfloor, \lceil \tau' \rceil\}} C^{\pi(\tau)}\} = 37$, where $\tau' = (\sqrt{2p\lambda - \lambda + 1} + \lambda - 1)/\lambda \approx 36.72$. Similar observations can also be made from Fig. 4.5, where the staleness cost is a quadratic function of AoI. As expected, the staleness cost increases remarkably with the threshold, since the staleness cost function is quadratic.

Next, we compare the performance of the optimal threshold-based policy with several baselines in Figs. 4.6 and 4.8, where the staleness cost function is linear and quadratic, respectively. We consider three baselines: (i) a naive policy, (ii) periodic policies, and (iii) the optimal offline policy. The naive policy is a capped reactive threshold-based policy with a threshold being equal to $\Delta^* = \lceil p \rceil$ when $f(\Delta) = \Delta$ (or $\Delta^* = \lceil \sqrt{p} \rceil$ when $f(\Delta) = \Delta^2$). That is, upon receiving a request, this policy naively updates the data when the staleness cost is no smaller than the update cost, otherwise, it does not. A periodic policy has a positive integer period d and updates the data every d time-slots, i.e., $u_i = id$ for $i = 1, 2, \dots$. Note that a periodic policy is not a reactive policy. Following a similar argument in the proof of Theorem 4.5, we can show that the average cost under a periodic policy with period d is $(p + \lambda d(d-1)/2)/\lambda d$ when $f(\Delta) = \Delta$ (or $(p + \lambda((d-1)^3/3 + (d-1)^2/2 + (d-1)/6))/\lambda d$ when $f(\Delta) = \Delta^2$). In the comparisons, we only consider the optimal periodic policy with $d^* = \lceil \sqrt{2p/\lambda} \rceil$ when $f(\Delta) = \Delta$ (or d^* being the solution of $4\lambda d^3 - 3\lambda d^2 - 6p = 0$ when $f(\Delta) = \Delta^2$). The optimal offline policy has the exact knowledge of all the request arrival times and is obtained based on the dynamic programming approach. Hence, the average cost under an optimal offline policy can be viewed as a lower bound of all online policies.

Fig. 4.6(a) shows the results for the setting with a fixed update cost $p = 50$ and a varying request arrival rate λ when the staleness cost function is linear. We observe that the optimal threshold-based policy outperforms all the other online policies and is very close to the optimal offline policy. When the request arrival rate is small, the optimal periodic policy performs poorly. This is because Bernoulli process with a small rate λ results in a

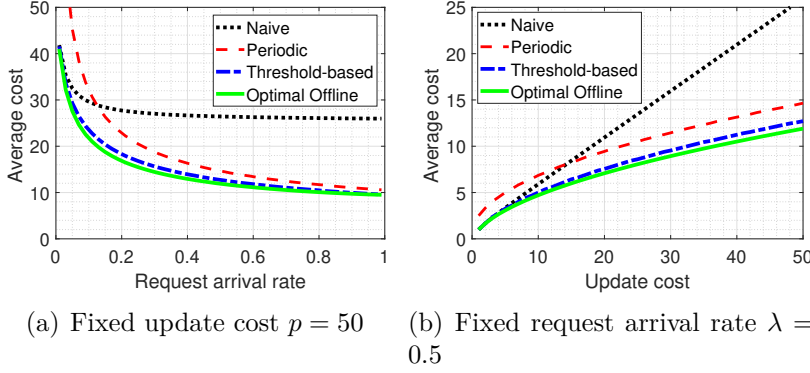


Figure 4.6: Performance comparisons of different policies with different request arrival rate λ and different update cost p when $f(\Delta) = \Delta$, respectively.

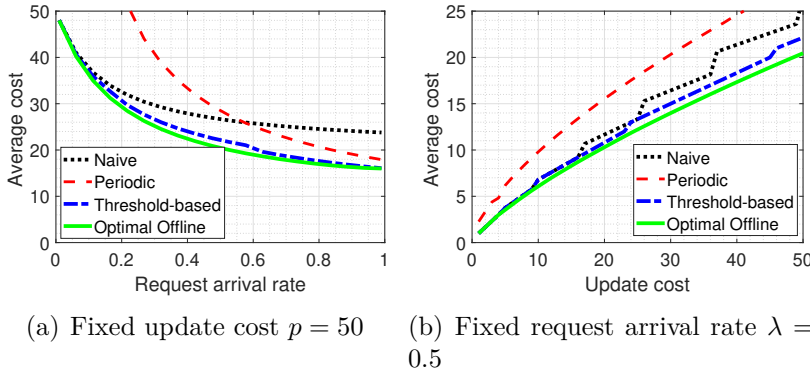


Figure 4.8: Performance comparisons of different policies with different request arrival rate λ and different update cost p when $f(\Delta) = \Delta^2$, respectively.

larger mean (i.e., $1/\lambda$) and a larger variance (i.e., $(1 - \lambda)/\lambda^2$) of the inter-arrival time of the requests. Hence, the inter-arrival time of the requests is usually larger and more random. In this case, a periodic policy that updates the data at fixed time instants results in a larger staleness cost. On the other hand, the interarrival time of requests is large when the rate λ approaches 0, resulting in a large AoI (and thus a high staleness cost) when the requests arrive. All the capped reactive policies as well as the offline optimal policy would make an

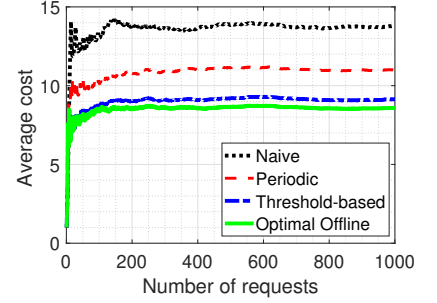


Figure 4.7: Performance comparisons of different policies using trace dataset, where update cost $p = 25$, request arrival rate $\lambda = 0.4$, and $f(\Delta) = \Delta$.

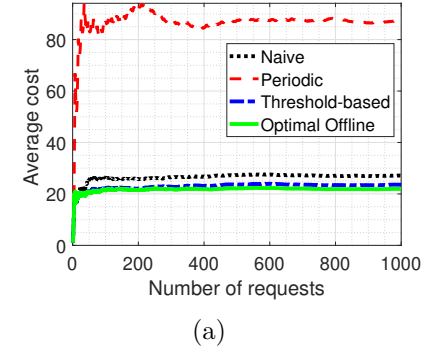


Figure 4.9: Performance comparisons of different policies using trace dataset, where update cost $p = 50$, request arrival rate $\lambda = 0.4$, and $f(\Delta) = \Delta^2$.

update decision for each request, making their average cost close to the update cost p . This aligns well with our theoretical result when we let $\lambda = 0$ in Eq. (4.13). When the request arrival rate becomes larger, the performance of the optimal periodic policy improves and is close to that of the optimal threshold-based policy. This is because a large request arrival rate λ leads to a small period d^* . In this case, the requests arrive more frequently, and the updates also occur more frequently. Hence, the staleness cost becomes small, and the update cost becomes dominant. We also observe that the naive policy performs poorly compared to the other policies when λ is large. This is because the naive policy is agnostic about the request arrival rate. The update period under a naive policy is roughly equal to p regardless of the request arrival rate. However, when λ becomes large, there could be many more requests arriving during an update interval of length p , which results in a large staleness cost. In addition, when the request arrival rate λ approaches 1, the optimal threshold-based policy and the optimal periodic policy have the same optimal threshold/period (i.e., $\tau^* = d^* = \lceil \sqrt{2p} \rceil$) as well as the same average cost (i.e., $\sqrt{2p} - 1/2$), and their knowledge about the future request arrival process is almost the same as the optimal offline policy (i.e., there is a request arriving in every time-slot), so their performance is also very close. Similar observations can also be made from Fig. 4.8(a), where the staleness cost function is quadratic. We omit the explanation since they are almost identical.

Fig. 4.6(b) shows the results for the setting with a fixed request arrival rate $\lambda = 0.5$ and a varying update cost p when the staleness cost function is linear. We observe that the optimal threshold-based policy again outperforms all the other online policies and performs closely to the optimal offline policy. The optimal periodic policy performs poorly because it is not a reactive policy, it cannot update timely according to the staleness cost of the requests, resulting in a large total staleness cost. We also observe that as the update cost increases, the naive policy performs much worse compared to the other policies. This is because the

average cost under a naive policy increases at a rate of $O(p)$, while the average cost under the other policies increases at a rate of $O(\sqrt{p})$. In Fig. 4.8(b), we show the results under the same setting except that the staleness cost function is quadratic. Again, the optimal threshold-based policy outperforms all other online policies. The optimal periodic policy performs poorly at all update costs. This is because its staleness costs become much larger due to the quadratic function and its inability to update timely upon the arrival of requests. The average cost under the naive policy has several jumps. The reason is that when the threshold Δ^* is set to $\lceil \sqrt{p} \rceil$, the integral change of $\lceil \sqrt{p} \rceil$ leads to a high increase in the average cost. For example, when the update cost p is in the range of $[26, 36]$, the threshold Δ^* under the naive policy is $\lceil \sqrt{p} \rceil = 6$ and the average cost increases linearly with respect to p according to Eq. (4.14). However, when the update cost is in the range of $[37, 49]$, the threshold Δ^* under the naive policy becomes $\lceil \sqrt{p} \rceil = 7$, which results in a much larger average cost compared to when $\Delta^* = 6$ according to Eq. (4.14). This also explains the jumps in the optimal threshold-based policy, but the jump is almost unnoticeably (e.g., when $p = 46$) since it chooses the optimal thresholds and thus mitigates the jumping effect.

In Fig. 4.7, we also compare the performance of different policies using the real trace dataset [74] when the staleness cost function is linear. The trace dataset collects around 700 billion user requests (each contains a timestamp, anonymized key, operation, etc.; see details in [74]) from 54 Twemcache clusters, which are the in-memory caching used by Twitter. In order to simplify the analysis and presentation, we focus on the user request arrival times at the cache of the 26th Twemcache cluster, whose request arrival rate is about 0.4. The request arrival process is no longer Bernoulli. In Fig. 4.7, we assume that the update cost is $p = 25$ and show the performances of different policies using the trace dataset in the first 10^3 requests. For the optimal offline policy, we still apply dynamic programming to obtain the optimal update times. For the naive policy, we simply let its threshold equal the

update cost of 25. For the threshold-based policy, we obtain the optimal threshold $\tau^* = 9$ by plugging $\lambda = 0.4$ into Eq. (4.12). Similarly, we obtain the optimal period $d^* = 11$ for the periodic policy. We observe that the optimal threshold-based policy outperforms the other online policies even though the request arrival process is now non-Bernoulli. In Fig. 4.9, we change the setting to update cost $p = 50$ and a quadratic staleness cost function. Clearly, the optimal threshold-based policy still outperforms the other online policies.

4.8 Chapter Summary

In this chapter, we considered a fundamental tradeoff between the data freshness and the update cost in a time-sensitive information-update system. We provided useful guidelines for the design of update policies. Assuming Bernoulli request arrival process, we also proposed a threshold-based update policy and proved its optimality. Our simulations based on both synthetic data and real traces corroborated the theoretical results and showed that the optimal threshold-based policy outperforms the baseline policies. For future work, one interesting direction would be to consider more general settings where users are interested in multiple contents.

Chapter 5

Learning-augmented Online Minimization of Age of Information and Transmission Costs

5.1 Introduction

In recent years, we have witnessed the swift and remarkable development of the Internet of Things (IoT), which connects billions of entities through wireless networks [75]. These entities range from small, resource-constrained sensors (e.g., temperature sensors and smart cameras) to powerful smartphones. Among various IoT applications, one most important categories is real-time IoT application, which requires timely information updates from the IoT sensors. For example, in industrial automation systems [76, 77], battery-powered IoT sensors are deployed to provide data for monitoring equipment health and product quality. On the one hand, IoT sensors are usually small and have limited battery capacity, and thus frequent transmissions drain the battery quickly; on the other hand, occasional transmissions render the information at the controller outdated, leading to detrimental decisions. In addition, wireless channels can be unreliable due to potential channel fading, interference, and the saturation of wireless networks if the traffic load generated by numerous sensors is high [78]. Clearly, under unreliable wireless networks, IoT sensors must transmit strategically to

balance the tradeoff between transmission cost (e.g., energy cost) and data freshness. Other applications include smart grids, smart cities, and so on.

To this end, in the first part of this work, we study the tradeoff between transmission cost and data freshness under a time-varying wireless channel. Specifically, we consider a discrete-time system where a device transmits data to an access point over an ON/OFF wireless channel (i.e., transmissions occur only when the channel is ON). Each transmission incurs a fixed *transmission cost*, while no transmission results in a *staleness cost* represented by the *Age-of-Information (AoI)* [3], which is defined as the time elapsed since the generation time of the freshest delivered packet. To minimize the sum of transmission costs and staleness costs, we develop a robust online algorithm that achieves a competitive ratio (CR) of 3. That is, different from typical studies with stationary network assumptions, the cost of our online algorithm is at most three times larger than that of the optimal offline algorithm under the worst channel state (see the definition of CR in Section 5.3).

While online algorithms exhibit robustness against worst-case situations, they often lean towards excessive caution and may have a subpar average performance in real-world scenarios. On the other hand, by exploiting historical data to build prediction models, machine learning (ML) algorithms can excel in average cases. Nonetheless, ML algorithms could be sensitive to disparity in training and testing data due to distribution shifts or adversarial examples, resulting in poor performance and lacking worst-case performance guarantees.

To that end, we design a novel learning-augmented online algorithm that takes advantage of both ML and online algorithms. Specifically, our learning-augmented online algorithm integrates ML prediction (a series of times indicating when to transmit) into our online algorithm, achieving two desired properties: (i) *consistency*: when the ML prediction is accurate and trusted, our learning-augmented online algorithm performs closely to the optimal offline algorithm, and (ii) *robustness*: even when the ML prediction is inaccurate, our

learning-augmented online algorithm still offers a worst-case guarantee.

Our main contributions are as follows.

First, we study the tradeoff between transmission cost and data freshness in a dynamic wireless channel by formulating an optimization problem to minimize the sum of transmission and staleness costs under an ON/OFF channel.

Second, following a similar line of analysis as in [79], we reformulate our (non-linear) optimization problem into a linear Transmission Control Protocol (TCP) acknowledgment problem [80] and propose a primal-dual-based online algorithm that achieves a CR of 3. While a similar primal-dual-based online algorithm has been claimed to asymptotically achieve a CR of $e/(e-1)$ [79], their analysis of CR relies on an (unrealistic) asymptotic setting (see Remark 5.4).

Third, by incorporating ML predictions into our online algorithm, we design a novel learning-augmented online optimization algorithm that achieves both consistency and robustness. *To the best of our knowledge, this is the first study on AoI that incorporates ML predictions into online optimization to achieve consistency and robustness.*

Finally, we perform extensive simulations using synthetic and real trace datasets. Our online algorithm outperforms the theoretical analysis, and our learning-augmented algorithm can achieve consistency and robustness.

The remainder of this chapter is organized as follows. We first discuss related work in Section 5.2. The system model and problem formulation are described in Section 5.3. In Section 5.4 and Section 5.5, we present our robust online algorithm and our learning-augmented online algorithm, respectively. Finally, we show the numerical results in Section 5.6 and conclude this chapter in Section 5.7.

5.2 Related Work

Since AoI was introduced in [3], it has sparked numerous studies on this topic (see surveys in [5, 6]). Among these AoI studies, two categories are most relevant to our work.

The first category includes studies that consider the joint minimization of AoI and certain costs [63, 81, 82, 83]. The work of [81] studies the problem of minimizing the average cost of sampling and transmission over an unreliable wireless channel subject to average AoI constraints. In [63], the authors consider a source-monitor pair with stochastic arrival of packets at the source. The source pays a transmission cost to send the packet, and its goal is to minimize the weighted sum of AoI and transmission costs. Here the packet arrival process is assumed to follow certain distributions. Although the assumptions in these studies lead to tractable performance analysis, such assumptions may not hold in practical scenarios.

The second category contains studies that focus on non-stationary settings [79, 84, 85, 86]. For example, in [84], the authors proposed online algorithms to minimize the AoI of users in a cellular network under adversarial wireless channels. In these AoI works that consider non-stationary settings, the most relevant work to ours is [79], where the authors study the minimization of the sum of download costs and AoI costs under an adversarial wireless channel. A primal-dual-based randomized online algorithm is shown to have an asymptotic CR of $e/(e - 1)$. However, this CR is attained under an (unrealistic) asymptotic setting (see Remark 5.4). To address this issue, we propose an online primal-dual-based algorithm that achieves a CR of 3 in the non-asymptotic regime. While the AoI optimization problems under non-stationary settings have been investigated, none of them considers applying ML predictions to improve the average performance of online algorithms.

In recent works [87, 88, 89, 90], researchers attempt to take advantage of both online algorithms and ML predictions, i.e., to design a learning-augmented online algorithm that

achieves consistency and robustness. In the seminal work [87], by incorporating ML predictions into the online Marker algorithm, the authors can achieve consistency and robustness for a caching problem. Following [87], a large body of research works in this direction have emerged. The most related learning-augmented work to ours is [89], where the authors design a primal-dual-based learning-augmented algorithm for the TCP acknowledgment problem. In [89], the uncertainty comes from the packet arrival times, and the controller can make decisions at any time. In our work, however, the uncertainty comes from the channel states, and no data can be transmitted when the channel is OFF. Lacking the freedom to transmit data at any time makes their algorithm inapplicable.

5.3 System Model and Problem Formulation

System Model. Consider a status-updating system where a resource-limited device sends time-sensitive data to an access point (AP) through an unreliable wireless channel. The system operates in discrete time slots, denoted by $t = 1, 2, \dots, T$, where T is finite and can be arbitrarily large. We use $s(t) \in \{0, 1\}$ to denote the channel state at time t , where $s(t) = 1$ means the channel is ON, allowing the device to access the AP; while $s(t) = 0$ means the channel is OFF, preventing access to the AP. The sequence of channel states over the time horizon is represented by $\mathbf{s} = \{s(1), \dots, s(T)\}$.

At the start of each slot, the device probes to know the current channel state and decides whether to transmit its freshest data to the AP. The transmission decision at slot t is denoted by $d(t) \in \{0, 1\}$, where $d(t) = 1$ if the device decides to transmit (i.e., generates a new update and transmits it to the AP), and $d(t) = 0$ if not. A scheduling algorithm π is denoted by $\pi = \{d^\pi(t)\}_{t=1}^T$, where $d^\pi(t)$ is the transmission decisions made by algorithm π at time t . For simplicity, we use $\pi = \{d(t)\}_{t=1}^T$ throughout this chapter. When the device decides to

transmit, it incurs a fixed *transmission cost* (e.g., energy cost) of $c > 1$, and the data on the AP will be successfully updated at the end of slot t if the channel is ON at slot t ; otherwise, the data on the AP gets staler.¹ To quantify the freshness of data on the AP side, we utilize a metric called *Age-of-Information (AoI)* [3], which measures the time elapsed since the freshest received update was generated. We use $a(t)$ to denote the AoI at time t , which evolves as

$$a(t) = \begin{cases} 0, & \text{if } s(t) \cdot d(t) = 1; \\ a(t-1) + 1, & \text{otherwise,} \end{cases} \quad (5.1)$$

where the AoI drops to 0 if the device transmits at ON slots; otherwise, it increases by 1.² We assume that $a(0) = 0$. To reflect the penalty that the AP does not get the update at time t , we introduce the *staleness cost*, which is represented by the AoI at time t .

Problem Formulation. The total cost of an algorithm π is

$$C(\mathbf{s}, \pi) \triangleq \sum_{t=1}^T (c \cdot d(t) + a(t)), \quad (5.2)$$

where the first item $c \cdot d(t)$ is the transmission cost at time t , and the second item $a(t)$ is the staleness cost at time t . In this chapter, we focus on the class of *online* scheduling algorithms, under which the information available at time t for making decisions includes the transmission cost c , the transmission history $\{d(\tau)\}_{\tau=1}^t$, and the channel state pattern $\{s(\tau)\}_{\tau=1}^t$, while the time horizon T and the future channel state $\{s(\tau)\}_{\tau=t+1}^T$ is unknown. Conversely, an *offline* scheduling algorithm has the information about the connectivity pattern $\mathbf{s}$ (and the time horizon T) beforehand.

Our goal is to develop an online algorithm π that minimizes the total cost given a channel

¹If $c \leq 1$, which is less than the staleness cost (at least 1), then the optimal policy is to transmit at every ON slot.

²Some studies let the AoI drop to 1, wherein our analysis still holds. We let the AoI drop to 0 to make the discussion concise.

state pattern $\mathbf{s}$:

$$\min_{d(t)} \quad \sum_{t=1}^T (c \cdot d(t) + a(t)) \quad (5.3a)$$

$$\text{s.t.} \quad d(t) \in \{0, 1\} \text{ for } t = 1, 2, \dots, T; \quad (5.3b)$$

$$a(t) \text{ evolves as Eq. (5.1) for } t = 1, 2, \dots, T. \quad (5.3c)$$

In Problem (5.3), the only decision variables are the transmission decisions $\{d(t)\}_{t=1}^T$, and the objective function is a non-linear function of $\{d(t)\}_{t=1}^T$ due to the dependence of $a(t)$ on $d(t)$. Specifically, based on Eq. (5.1), we can rewrite the AoI at time t as $a(t) = (1 - s(t) \cdot d(t)) \cdot (a(t-1) + 1)$. Upon rephrasing $a(t)$ with the transmission decisions $\{d(\tau)\}_{\tau=1}^t$ (i.e., rewriting $a(t-1)$ with $d(t-1)$ and $a(t-2)$, rewriting $a(t-2)$ with $d(t-2)$ and $a(t-3)$, and so forth), we can observe that $a(t)$ involves the products of the current transmission decision $d(t)$ and the previous transmission decisions $d(\tau)$ for $\tau \in [1, t-1]$, which indicates that $a(t)$ is not linear with respect to $\{d(\tau)\}_{\tau=1}^t$. This non-linearity poses a challenge to its efficient solutions. In Section 5.4.1, following a similar line of analysis as in [79], we reformulate Problem (5.3) to an equivalent TCP acknowledgment (ACK) problem, which is linear and can be solved efficiently (e.g., via the primal-dual approach [91]).

To measure the performance of an online algorithm, we use the metric *competitive ratio* (CR) [91], which is defined as the worst-case ratio of the cost under the online algorithm to the cost of the optimal offline algorithm. Formally, we say that an online algorithm π is β -competitive if there exists a constant $\beta \geq 1$ such that for any channel state pattern $\mathbf{s}$,

$$C(\mathbf{s}, \pi) \leq \beta \cdot OPT(\mathbf{s}), \quad (5.4)$$

where $OPT(\mathbf{s})$ is the cost of the optimal offline algorithm for the given channel state $\mathbf{s}$. We

desire to develop an online algorithm with a CR close to 1, which implies that our online algorithm performs closely to the optimal offline algorithm.

5.4 Robust Online Algorithm

In this section, we first reformulate our AoI Problem (5.3) to an equivalent linear TCP ACK problem. Then, this TCP ACK problem is further relaxed to a linear primal-dual-based program. Finally, a 3-competitive online algorithm is developed to solve the linear primal-dual-based program.

5.4.1 Problem Reformulation

In [79], the authors study the same non-linear Problem (5.3) and reformulate it to an equivalent linear problem. Following a similar line of analysis as in [79], we reformulate the non-linear Problem (5.3) to an equivalent linear TCP ACK Problem (5.5) as follows. Consider a TCP ACK problem, where the source reliably generates and delivers one packet to the destination in each slot $t = 1, 2, \dots, T$. Those delivered packets need to be acknowledged (for simplicity, we use “acked” instead of “acknowledged” throughout this chapter) that they are received by the destination, which requires the destination to send ACK packets (for brevity, we call it ACK) back to the source. We use $d(t) \in \{0, 1\}$ to denote the ACK decision made by the destination at slot t . Let $z_i(t) \in \{0, 1\}$ represent whether packet i (i.e., the packet sent at slot i) has been acked by slot t ($i \leq t$), where $z_i(t) = 1$ if packet i is not acked by slot t and $z_i(t) = 0$ otherwise. Once packet i is acked at slot t , then it is acked forever after slot t , i.e., $z_i(\tau) = 0$ for all $i \leq t$ and all $\tau \geq t$. The feedback channel is unreliable and its channel state in slot t is modeled by an ON/OFF binary variable $s(t) \in \{0, 1\}$. We

use $\mathbf{s} = \{s(1), \dots, s(T)\}$ to denote the entire feedback channel states. The destination can access the feedback channel state $s(t)$ at the start of each slot t . When the feedback channel is ON and the destination decides to send an ACK, all previous packets are acked, i.e., the number of unacked packets becomes 0; otherwise, the number of unacked packets increases by 1. We can see that the dynamic of the number of unacked packets is the same as the AoI dynamic.

We assume a holding cost at each slot, which is the number of unacked packets in that slot. In addition, we also assume that each ACK has an ACK cost of c . The goal of the TCP ACK problem is to develop an online scheduling algorithm $\pi = \{d(t)\}_{t=1}^T$ that minimizes the total cost given a feedback channel state pattern $\mathbf{s}$:

$$\min_{d(t), z_i(t)} \sum_{t=1}^T \left(c \cdot d(t) + \sum_{i=1}^t z_i(t) \right) \quad (5.5a)$$

$$\begin{aligned} \text{s.t. } \quad & z_i(t) + \sum_{\tau=i}^t s(\tau)d(\tau) \geq 1 \\ & \text{for } i \leq t \text{ and } t = 1, 2, \dots, T; \end{aligned} \quad (5.5b)$$

$$d(t), z_i(t) \in \{0, 1\} \text{ for } i \leq t \text{ and } t = 1, 2, \dots, T, \quad (5.5c)$$

where the first item $c \cdot d(t)$ in Eq. (5.5a) is the ACK cost at slot t , the second item $\sum_{i=1}^t z_i(t)$ in Eq. (5.5a) is the holding cost at slot t . Constraint (5.5b) states that for packet i at slot t , either this packet is not acked (i.e., $z_i(t) = 1$) or an ACK was made since its arrival (i.e., $s(\tau)d(\tau) = 1$ for some $i \leq \tau \leq t$). While Problem (5.5) is an integer linear problem, we demonstrate its equivalence to Problem (5.3) in the following.

Lemma 5.1. *Problem (5.5) is equivalent to Problem (5.3).*

We provide detailed proof in Appendix C.1 and give a proof sketch as follows. We can show that: (i) any feasible solution to Problem (5.3) can be converted to a feasible solution

to Problem (5.5), and the total costs of these two solutions are the same; (ii) any feasible solution to Problem (5.5) can be converted to a feasible solution to Problem (5.3), and the total cost of the converted solution to Problem (5.3) is no greater than the total cost of the solution to Problem (5.5). This implies that any optimal solution to Problem (5.3) is also an optimal solution to Problem (5.5), and vice versa. Therefore, these two problems are equivalent [92, Sec. 4.1.3].

To obtain a linear program of the integer Problem (5.5), we relax the integer requirement to real numbers:

$$\min_{d(t), z_i(t)} \sum_{t=1}^T \left(c \cdot d(t) + \sum_{i=1}^t z_i(t) \right) \quad (5.6a)$$

$$\begin{aligned} \text{s.t.} \quad & z_i(t) + \sum_{\tau=i}^t s(\tau) d(\tau) \geq 1 \\ & \text{for } i \leq t \text{ and } t = 1, 2, \dots, T; \end{aligned} \quad (5.6b)$$

$$d(t), z_i(t) \geq 0 \text{ for } i \leq t \text{ and } t = 1, 2, \dots, T, \quad (5.6c)$$

which is referred to as the primal problem. The corresponding dual problem of Problem (5.6) is as follows:

$$\max_{y_i(t)} \sum_{t=1}^T \sum_{i=1}^t y_i(t) \quad (5.7a)$$

$$\text{s.t.} \quad s(t) \sum_{i=1}^t \sum_{\tau=t}^T y_i(\tau) \leq c \text{ for } t = 1, 2, \dots, T; \quad (5.7b)$$

$$y_i(t) \in [0, 1] \text{ for } i \leq t \text{ and } t = 1, 2, \dots, T, \quad (5.7c)$$

which has a dual variables $y_i(t)$ for packet i and time $t \geq i$.

Remark 5.2. In [79], the authors mentioned that their reformulated problem is equivalent to the original AoI problem without proof. For the sake of completeness, we provide rigorous

proof regarding the equivalence between Problem (5.5) and Problem (5.3) in Lemma 5.1.

5.4.2 Primal-dual Online Algorithm Description and Analysis

To solve the primal-dual Problems (5.6) and (5.7), we develop the Primal-dual-based Online Algorithm (PDOA) and present it in Algorithm 1. The input is the channel state pattern $\mathbf{s}$ (revealed in an online manner), and the outputs are the primal variables $d(t)$ and $z_i(t)$, and the dual variable $y_i(t)$. Two auxiliary variables L and M are also introduced: L denotes the time when the latest ACK was made, and M denotes the ACK marker (PDOA should make an ACK when $M \geq 1$).

PDOA is a threshold-based algorithm. Assuming that the latest ACK was made at slot L , when the accumulated holding costs since slot $L + 1$ is no smaller than the ACK cost c (i.e., $M \geq 1$), PDOA will make an ACK at the next ON slot L' . Here, we call the interval $[L + 1, L']$ an ACK interval. Note that PDOA updates the primal and dual variables only for packets that are not acked in the current ACK interval $[L + 1, L']$. Specifically, consider packet i that has not been acked by the current slot $t \in [L + 1, L']$: (i) for the primal variable $z_i(t)$, if the threshold is not achieved ($M < 1$) or the channel is OFF at slot t , PDOA will update $z_i(t)$ to be 1 (in Line 4 or Line 15, respectively) since packet i is not acked by slot t ; (ii) for the dual variable $y_i(t)$, if packet i is not the last packet in the current ACK interval, PDOA will update $y_i(t)$ to be 1 to maximize the dual objective function; otherwise, PDOA will update $y_i(t)$ to $c - c \cdot M$ to ensure that when the threshold is achieved ($M \geq 1$), the sum of all the dual variables in the current ACK interval is exactly c .

In addition, at the end of slot t (i.e., Lines 19-27), if the channel is ON at slot t , PDOA will skip slot t and go to slot $t + 1$. Otherwise, the channel is OFF at slot t , and assuming that the most recent ON slot is slot $t^\dagger \triangleq \max\{\tau : \tau < t \text{ and } s(\tau) = 1\}$ ($t^\dagger \in [L, t)$), then the channels

Algorithm 1: Primal-dual-based Online Algorithm (PDOA)

Input : c, s (revealed in an online manner)
Output: $d(t), z_i(t), y_i(t)$
Init.: $d(t), z_i(t), y_i(t), L, M \leftarrow 0$ for all i and t

```

1 for  $t = 1$  to  $T$  do
2     /* Iterate all the packets arriving since the latest ACK time  $L$ . */
3     for  $i = L + 1$  to  $t$  do
4         if  $M < 1$  then                                     /* Not ready to ACK */
5              $z_i(t) \leftarrow 1$ ;
6              $M \leftarrow M + 1/c$ ;
7              $y_i(t) \leftarrow \min\{1, c - c \cdot M\}$ ;
8         end
9         if  $M \geq 1$  then                                     /* Ready to ACK */
10            if  $s(t) = 1$  then                                 /* ON channel */
11                 $d(t) \leftarrow 1$ ;
12                 $M \leftarrow 0$ ;
13                 $L \leftarrow t$ ;
14                break and go to the next slot (i.e.,  $t + 1$ );
15            else                                           /* OFF channel */
16                 $z_i(t) \leftarrow 1$ ;
17            end
18        end
19    end
20    /* At the end of slot  $t$ , update dual variable  $y_i(t)$  with  $s(i) = 0$  as: */
21    for  $i = t$  decrease to  $L + 1$  do
22        if  $s(i) = 0$  then
23            if  $y_i(t) = 0$  then
24                 $y_i(t) \leftarrow 1$ ;
25            end
26        else
27            break and go to the next slot;
28        end
29    end
30 end

```

are OFF during $[t^\dagger + 1, t]$. To maximize the dual objective function, PDOA updates the dual variables of packet $t^\dagger + 1$ to packet t to be 1 since the channels are OFF during $[t^\dagger + 1, t]$ and the updating of their dual variables does not violate constraint (5.7b) (see an illustration in Fig. 5.1(b)). Note that there may be some OFF channels before slot $t^\dagger$, but PDOA does not

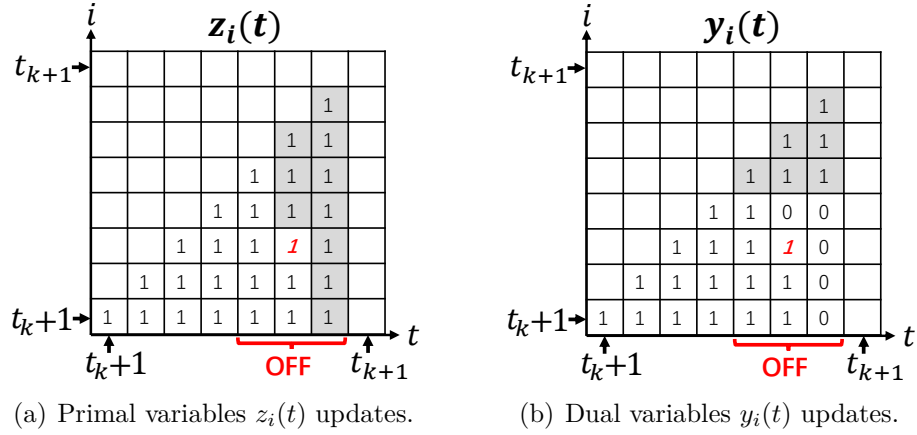


Figure 5.1: The updates of primal variables $z_i(t)$ and dual variables $y_i(t)$ in the k -th ACK interval $[t_k + 1, t_{k+1}]$ of PDOA, where channels are OFF during $[t_k + 5, t_k + 7]$. The x-axis represents time and the y-axis represents the packet id. PDOA makes two ACKs at slot t_k and slot t_{k+1} , where the ACK cost $c = 18$. The primal variables $z_i(t)$ and dual variables $y_i(t)$ are updated from slot $t_k + 1$ to slot t_{k+1} ; and in slot t , packets are updated from packet $t_k + 1$ to packet t . The red bold italic 1 denotes when the ACK marker equals or is larger than 1. In Fig. 5.1(a), the grey areas denote the updates due to Line 15; In Fig. 5.1(b), the grey areas denote the updates due to Lines 19-27.

update their dual variables to avoid the violation of constraint (5.7b). For example, assuming that slot t' ($t' < t^\dagger$) is an OFF channel and we let $y_{t'}(t) = 1$. Letting $y_{t'}(t) = 1$ has no effect on the constraint (5.7b) at slot t' since we always have $s(t') \sum_{i=1}^{\hat{t}} \sum_{\tau=t'}^T y_i(\tau) = 0$, but doing this does impact the constraint (5.7b) at slot $t^\dagger$ (i.e., increasing $s(t^\dagger) \sum_{i=1}^{t^\dagger} \sum_{\tau=t^\dagger}^T y_i(\tau)$ by 1 because $y_{t'}(t)$ is a part of $s(t^\dagger) \sum_{i=1}^{t^\dagger} \sum_{\tau=t^\dagger}^T y_i(\tau)$ as $t' < t^\dagger$ and $t^\dagger \leq t$), possibly making constraint (5.7b) at slot $t^\dagger$ violated.

Theorem 5.3. *PDOA is 3-competitive.*

We provide detailed proof in Section C.2 and explain the key ideas as follows. We first show that given any channel state $\mathbf{s}$, PDOA produces a feasible solution to primal Problem (5.6) and dual Problem (5.7). Then, we show that in any k -th ACK interval, the ratio between the primal objective value and the dual objective value (denoted by $P(k)$ and $D(k)$, respectively) is at most 3, i.e., $P(k)/D(k) \leq 3$. This implies that the ratio between the total

primal objective value (denoted by P) and the total dual objective value (denoted by D) is also at most 3, i.e., $P/D \leq 3$. By the weak duality [91], PDOA is 3-competitive.

Remark 5.4. In [79], the authors also propose a primal-dual-based online algorithm and show that their algorithm achieves a CR of $e/(e-1)$. However, this CR is achieved only in an (unrealistic) asymptotic setting (i.e., when the transmission cost c goes to infinity but the time horizon T is finite). Specifically, in their algorithm, to maximize the dual objective function, at the end of the last slot T , they update certain dual variables $y_i(t)$ that arrived at the OFF slot to be 1. In their analysis (the proof of their Theorem 7), they show that because of those dual variables updates at the end of slot T , the primal objective value satisfies $P \leq (1 + 1/((1 + 1/c)^{\lfloor c \rfloor} - 1)) \cdot D + (T(T+1)/2) \cdot (D/c)$. When c goes to infinity, their CR becomes $P/D \leq e/(e-1)$ as $(T(T+1)/2)/c$ goes to 0 since T is finite. However, the optimization problem becomes trivial in this setting since an optimal algorithm is simply not to transmit at all given that the transmission cost c can significantly exceed the total staleness cost (at most $(T(T+1)/2)$). Furthermore, when c is finite, their CR is a quadratic function of the time horizon T , which can be very large when T is large. Instead, our analysis holds for any T and c . In our algorithm, rather than updating these dual variables $y_i(t)$ at the end of slot T , we directly update them only in the current ACK interval (i.e., Lines 19-27), ensuring that the dual constraint (5.7b) is satisfied and the dual objective function is as large as possible. This enables us to focus on the analysis of $P(k)/D(k)$ in the current ACK interval and show that $P(k)/D(k) \leq 3$ for any k -th ACK interval, which implies that PDOA is 3-competitive.

5.5 Learning-augmented Online Algorithm

Online algorithms are known for their robustness against worst-case scenarios, but they can be overly conservative and may have a poor average performance in typical scenarios. In contrast, ML algorithms leverage historical data to train models that excel in average cases. However, they typically lack worst-case performance guarantees when facing distribution shifts or outliers. To attain the best of both worlds, we design a learning-augmented online algorithm that achieves both consistency and robustness.

5.5.1 Machine Learning Predictions

We consider the case where an ML algorithm provides a prediction $\mathcal{P} \triangleq \{p_1, p_2, \dots, p_n\}$ that represents the times to transmit an ACK for the destination (i.e., the prediction $\mathcal{P}$ makes a total of n ACKs and sends the i -th ACK at slot p_i). The prediction $\mathcal{P}$ is unaware of the channel state pattern $\mathbf{s}$ and can be provided either in full in the beginning (i.e., $t = 0$) or one-by-one in each slot. Furthermore, when the prediction $\mathcal{P}$ decides to send an ACK at an OFF slot, we will simply ignore the decision for this particular slot.

Provided with the prediction $\mathcal{P}$, we specify a trust parameter $\lambda \in (0, 1]$ to reflect our confidence in the prediction $\mathcal{P}$: a smaller λ means higher confidence. The learning-augmented online algorithm takes a prediction $\mathcal{P}$, a trust parameter λ , and a channel state pattern $\mathbf{s}$ (revealed in an online manner) as inputs, and outputs a solution with a cost of $C(\mathbf{s}, \mathcal{P}, \lambda)$. A learning-augmented online algorithm is said $\beta(\lambda)$ -robust ($\beta(\lambda) \geq 1$) and $\gamma(\lambda)$ -consistent ($\gamma(\lambda) \geq 1$) if its cost satisfies

$$C(\mathbf{s}, \mathcal{P}, \lambda) \leq \min\{\beta(\lambda) \cdot \text{OPT}(\mathbf{s}), \gamma(\lambda) \cdot C(\mathbf{s}, \mathcal{P})\}, \quad (5.8)$$

where $OPT(\mathbf{s})$ and $C(\mathbf{s}, \mathcal{P})$ is the cost of the optimal offline algorithm and the cost of purely following the prediction $\mathcal{P}$ under the channel state pattern $\mathbf{s}$, respectively.

We aim to design a learning-augmented online algorithm for primal Problem (5.6) that exhibits two desired properties (i) *consistency*: when the ML prediction $\mathcal{P}$ is accurate ($C(\mathbf{s}, \mathcal{P}) \approx OPT(\mathbf{s})$) and trusted, our learning-augmented online algorithm performs closely to the optimal offline algorithm (i.e., $\gamma(\lambda) \rightarrow 1$ as $\lambda \rightarrow 0$); and (ii) *robustness*: even if the ML prediction $\mathcal{P}$ is inaccurate, our learning-augmented algorithm still retains a worst-case guarantee (i.e., $C(\mathbf{s}, \mathcal{P}, \lambda) \leq \beta(\lambda) \cdot OPT(\mathbf{s})$ for any prediction $\mathcal{P}$).

5.5.2 Learning-augmented Online Algorithm Description

We present our Learning-augmented Primal-dual-based Online Algorithm (LAPDOA) in Algorithm 2. LAPDOA behaves similarly to PDOA, but the updates of primal variables and dual variables incorporate the ML prediction $\mathcal{P}$.

In LAPDOA, two additional auxiliary variables M' and y' are used, where M' denotes the increment of the ACK marker M and y' denotes the increment of the dual variables $y_i(t)$ in each iteration of update. Assuming that the current time is t , let $\alpha(t)$ denote the next time when the prediction $\mathcal{P}$ sends an ACK (i.e., $\alpha(t) \triangleq \min\{p_i : p_i \geq t\}$ and $\alpha(t) = \infty$ if $t > p_n$). For the updates of primal and dual variables of an unacked packet i at slot t , based on the relationship between the current time t and $\alpha(i)$ (which is also the time when the prediction $\mathcal{P}$ makes an ACK for packet i because packet i arrives at slot i), we classify them into three types:

- Big updates: those updates make $M' \leftarrow 1/\lambda c$, $y' \leftarrow 1$, and $z_i(t) \leftarrow 1$. The big updates are made when LAPDOA is behind the ACK scheduled by the prediction $\mathcal{P}$ (i.e., $t \geq \alpha(i)$), and it tries to catch up with the prediction $\mathcal{P}$ by making a big increase in

Algorithm 2: Learning-augmented Primal-dual-based Online Algorithm (LAP-DOA)

Input : $c, \mathcal{P}, \lambda, \mathbf{s}$ (revealed in an online manner)
Output: $d(t), z_i(t), y_i(t)$
Init.: $d(t), z_i(t), y_i(t), L, M \leftarrow 0$ for all i and t

```

1 for  $t = 1$  to  $T$  do
2   /* Iterate the packets arriving since the most recent ACK time  $L$ . */
3   for  $i = L + 1$  to  $t$  do
4     if  $M < 1$  then /* Not ready to ACK */
5       if  $t \geq \alpha(i)$  then
6         /* Big update: prediction already acked packet  $i$  */
7          $M' \leftarrow 1/\lambda c, y' \leftarrow 1$ ;
8       else
9         /* Small update: prediction did not ack packet  $i$  yet */
10         $M' \leftarrow \lambda/c, y' \leftarrow \lambda$ ;
11      end
12       $z_i(t) \leftarrow 1, M \leftarrow M + M', y_i(t) \leftarrow y'$ ;
13    end
14    if  $M \geq 1$  then /* Ready to ACK */
15      if  $s(t) = 1$  then /* ON channel */
16         $d(t) \leftarrow 1, M \leftarrow 0, L \leftarrow t$ ;
17        break and go to the next slot (i.e.,  $t + 1$ );
18      else /* OFF channel */
19        if  $z_i(t) \neq 1$  then
20          /* Zero update */
21           $z_i(t) \leftarrow 1$ ;
22        end
23      end
24    end
25  end
26  /* At the end of slot  $t$ , update dual variable  $y_i(t)$  with  $s(i) = 0$  as: */
27  for  $i = t$  decrease to  $L + 1$  do
28    if  $s(i) = 0$  then
29      if  $y_i(t) = 0$  then
30         $y_i(t) \leftarrow 1$ ;
31      end
32    else
33      break and go to the next slot;
34    end
35  end
36 end
  
```

the ACK marker.

- Small updates: those updates make $M' \leftarrow \lambda/c$, $y' \leftarrow \lambda$, and $z_i(t) \leftarrow 1$. The small updates are made when LAPDOA is ahead of the ACK scheduled by the prediction $\mathcal{P}$ (i.e., $t < \alpha(i)$), and LAPDOA tries to slow down its ACK rate by making a small increase in the ACK marker.
- Zero updates: those updates make $M' \leftarrow 0$, $y' \leftarrow 0$, and $z_i(t) \leftarrow 1$. The zero updates are made when LAPDOA is supposed to ACK at some slot t' but find that slot t' is OFF, and it has to delay its ACK to the next ON slot and pay the holding cost (i.e., $z_i(t) = 1$) along the way.

An illustration of these three types of updates is in Fig. 5.2.

In addition, at the end of slot t (i.e., Lines 22-30), if the channel is OFF at slot t , similar to PDOA, to maximize the dual objective function, LAPDOA updates the dual variables of the packets that arrive after the most recent ON slot (which is assumed to be slot $t^\dagger \triangleq \max\{\tau : \tau < t \text{ and } s(\tau) = 1\}$ ($t^\dagger \in [L, t)$)) and the updating of their dual variables does not violate constraint (5.7b) since the channels are OFF during $[t^\dagger + 1, t]$ (see an illustration in Fig. 5.2(c)).

5.5.3 Learning-augmented Online Algorithm Analysis

In this subsection, we focus on the consistency and robustness analysis of LAPDOA with $\lambda \in (0, 1]$. The special cases of LAPDOA with $\lambda = 0$ and $\lambda = 1$ correspond to the cases that LAPDOA follows the prediction $\mathcal{P}$ purely and PDOA, respectively. It is noteworthy that by choosing different values of λ , LAPDOA exhibits a crucial trade-off between consistency and robustness.

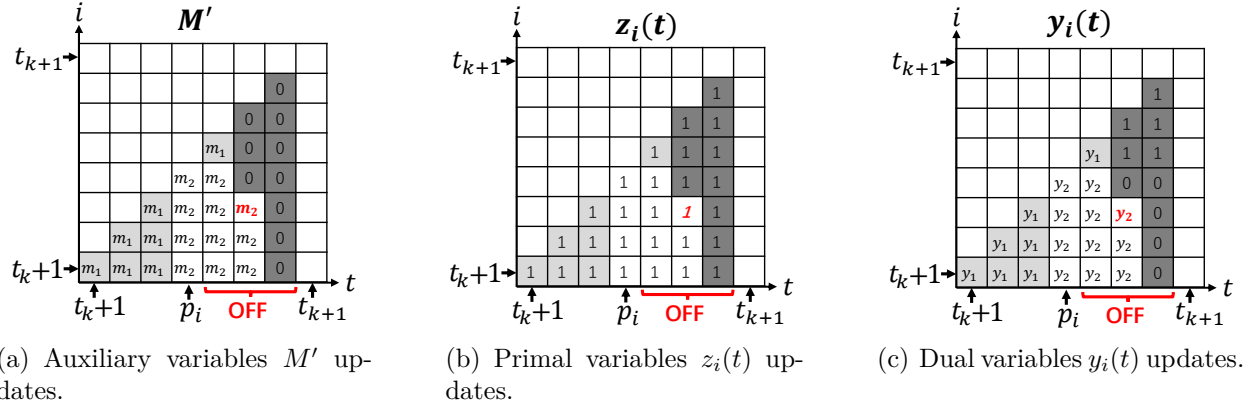


Figure 5.2: The updates of variables in the k -th ACK interval $[t_k + 1, t_{k+1}]$ of LAPDOA, where channels are OFF during $[t_k + 5, t_k + 7]$. LAPDOA makes two ACKs at t_k and t_{k+1} , and the ML prediction $\mathcal{P}$ makes its i -th ACK at slot $t_k + 4$. The red bold italic value denotes when the ACK marker $M \geq 1$. Let $m_1 = \lambda/c$, $m_2 = 1/\lambda c$, $y_1 = \lambda$, and $y_2 = 1$. The light grey area denotes the small updates, the white area (without background) denotes the big updates, and the dark grey area denotes the zero updates.

Theorem 5.5. For any channel state pattern $\mathbf{s}$, any prediction $\mathcal{P}$, any parameter $\lambda \in (0, 1]$, and any ACK cost c , LAPDOA outputs an almost feasible solution (within a factor of $c/(c+1)$) with a cost of: when $\lambda \in (0, 1/c]$,

$$C(\mathbf{s}, \mathcal{P}, \lambda) \leq \min\{(3/\lambda) \cdot ((c+1)/c) \cdot OPT(\mathbf{s}), (1+\lambda)C_H(\mathbf{s}, \mathcal{P}) + C_A(\mathbf{s}, \mathcal{P})\}, \quad (5.9)$$

and when $\lambda \in (1/c, 1]$,

$$C(\mathbf{s}, \mathcal{P}, \lambda) \leq \min\{(3/\lambda) \cdot ((c+1)/c) \cdot OPT(\mathbf{s}), (\lambda+2)C_H(\mathbf{s}, \mathcal{P}) + (1/\lambda+2) \cdot \lceil \lambda c \rceil \cdot C_A(\mathbf{s}, \mathcal{P})/c\}, \quad (5.10)$$

where $C_A(\mathbf{s}, \mathcal{P})$ and $C_H(\mathbf{s}, \mathcal{P})$ denote the total ACK costs and total holding costs of prediction $\mathcal{P}$ under $\mathbf{s}$, respectively; and $C(\mathbf{s}, \mathcal{P}) = C_A(\mathbf{s}, \mathcal{P}) + C_H(\mathbf{s}, \mathcal{P})$.

Next, we show that LAPDOA has the robustness guarantee in Lemma 5.6 and the consistency guarantee in Lemma 5.7. Combining Lemmas 5.6 and 5.7, we can conclude Theorem 5.5.

Lemma 5.6. (*Robustness*) *For any ON/OFF input instance $\mathbf{s}$, any prediction $\mathcal{P}$, any parameter $\lambda \in (0, 1]$, and any ACK cost c , LAPDOA outputs a solution which has a cost of*

$$C(\mathbf{s}, \mathcal{P}, \lambda) \leq (3/\lambda) \cdot ((c+1)/c) \text{OPT}(\mathbf{s}). \quad (5.11)$$

We provide detailed proof in Appendix C.3 and explain the key ideas as follows. We first show that LAPDOA produces a feasible primal solution and an almost feasible dual solution (with a factor of $c/(c+1)$). Then, we show that in any k -th ACK interval, LAPDOA achieves $P(k)/D(k) \leq 3/\lambda$. This implies that LAPDOA also achieves $P/D \leq 3/\lambda$ on the entire instance. Finally, by scaling down all dual variables $y_i(t)$ generated by LAPDOA by a factor of $c/(c+1)$, we obtain a feasible dual solution with a dual objective value of $(c/(c+1)) \cdot D$. By the weak duality, we have $P/\text{OPT} \leq P/((c/(c+1)) \cdot D) = (P/D) \cdot ((c+1)/c) \leq (3/\lambda) \cdot ((c+1)/c)$.

Lemma 5.7. (*Consistency*) *For any channel state pattern $\mathbf{s}$, any prediction $\mathcal{P}$, any parameter $\lambda \in (0, 1]$, and any ACK cost c , LAPDOA outputs a solution with a cost of: when $\lambda \in (0, 1/c]$,*

$$C(\mathbf{s}, \mathcal{P}, \lambda) \leq (1 + \lambda)C_H(\mathbf{s}, \mathcal{P}) + C_A(\mathbf{s}, \mathcal{P}), \quad (5.12)$$

and when $\lambda \in (1/c, 1]$,

$$\begin{aligned} C(\mathbf{s}, \mathcal{P}, \lambda) \leq \\ (\lambda + 2)C_H(\mathbf{s}, \mathcal{P}) + (1/\lambda + 2) \cdot \lceil \lambda c \rceil \cdot C_A(\mathbf{s}, \mathcal{P})/c, \end{aligned} \quad (5.13)$$

where $C_A(\mathbf{s}, \mathcal{P})$ and $C_H(\mathbf{s}, \mathcal{P})$ denote the total ACK cost and total holding cost of prediction $\mathcal{P}$ under $\mathbf{s}$, respectively; and $C(\mathbf{s}, \mathcal{P}) = C_A(\mathbf{s}, \mathcal{P}) + C_H(\mathbf{s}, \mathcal{P})$.

We provide detailed proof in Appendix C.4 and give a proof sketch as follows. In general, LAPDOA generates three types of updates: big updates, small updates, and zero updates. Our idea is to bound the total cost of each type of update by the cost of the algorithm that purely follows the prediction. In the case of $\lambda \in (0, 1/c]$, we show that the total number of big updates is $C_A(\mathbf{s}, \mathcal{P})/c$, and each big update increases the primal objective value by c , so the total cost of big updates in LAPDOA is $C_A(\mathbf{s}, \mathcal{P})$. In addition, the total number of small updates and zero updates can be shown to be bounded by $C_H(\mathbf{s}, \mathcal{P})$, and each small update or zero update incurs a cost at most $1 + \lambda$, thus the total cost of small and zero updates in LAPDOA is at most $(1 + \lambda)C_H(\mathbf{s}, \mathcal{P})$. In summary, the total cost of LAPDOA in the case of $\lambda \in (0, 1/c]$ is bounded by $(1 + \lambda)C_H(\mathbf{s}, \mathcal{P}) + ((c + 1)/c)C_A(\mathbf{s}, \mathcal{P})$. A similar bound can be also obtained for the case of $\lambda \in (1/c, 1]$.

Remark 5.8. When we trust the ML prediction (i.e., $\lambda \rightarrow 0$) and the ML prediction is accurate at the same time ($C(\mathbf{s}, \mathcal{P}) \approx OPT(\mathbf{s})$), our learning-augmented algorithm also performs nearly to the optimal offline algorithm, achieving consistency.

Remark 5.9. With any $\lambda \in (0, 1]$, the CR of LAPDOA is at most $(3/\lambda) \cdot ((c + 1)/c)$, regardless of the prediction quality. This indicates that our learning-augmented algorithm has the worst-case performance guarantees, achieving robustness.

5.6 Numerical Results

In this section, we perform simulations using both synthetic data and real trace data to show that our online algorithm PDOA outperforms the State-of-the-Art online algorithm and that

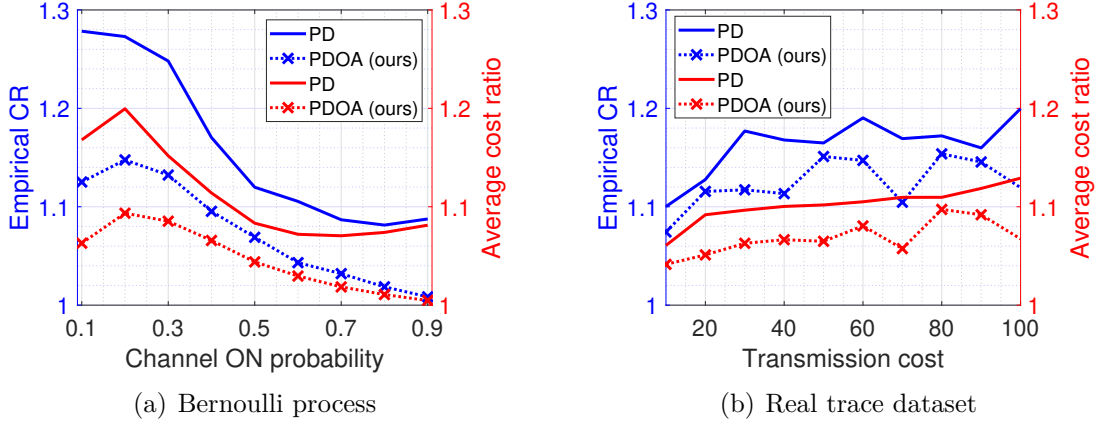


Figure 5.3: Performance comparison of online algorithms under different datasets.

our learning-augmented online algorithm LAPDOA achieves consistency and robustness.

5.6.1 Online Algorithm

In Fig. 5.3, we compare PDOA with the State-of-the-Art online algorithm proposed in [79] (which is referred to as “PD” in Fig. 5.3). Two datasets are considered: (i) The synthetic dataset in Fig. 5.3(a). We adopt the same settings as in [79], where the channel state is a Bernoulli process with varying channel ON probability and the transmission cost $c = 15$. The number of simulation runs is 100, each has 1000 slots; (ii) The real trace dataset [93] in Fig. 5.3(b). This dataset contains the channel measurement (i.e., reference signal received quality (RSRQ)) of the commercial mmWave 5G services in a major U.S. city. Specifically, located in the Minneapolis downtown region, the researchers in [93] repeatedly conduct walking tests on the 1300m loop area. Throughout these walking tests, they utilized a 5G monitoring tool installed on an Android smartphone to collect RSRQ information. The RSRQ values fluctuate as the tester moves, being higher in proximity to the mmWave 5G tower and decreasing as the tester moves away. There are 11 simulation runs, each with 750 slots. For the ON/OFF channel determination, a threshold is established for the RSRQ

(-13dB): the channel is considered ON when the RSRQ exceeds the threshold; otherwise, it is deemed OFF. Here we vary the transmission cost from 10 to 100. In both datasets, the performance metrics are the empirical CR and the average cost ratio (i.e., the worst cost ratio and the average cost ratio under the online algorithm and the optimal offline algorithm over multiple simulation runs).

Fig. 5.3 illustrates that our online algorithm PDOA consistently outperforms the State-of-the-Art online algorithm PD in both datasets, i.e., PDOA achieves a lower empirical CR and a lower average cost ratio than PD. In addition, the empirical CR of PDOA outperforms the theoretical analysis (with a CR of 3), validating our theoretical results.

5.6.2 Learning-augmented Online Algorithm

In this subsection, we study the performance of LAPDOA under different prediction qualities using both synthetic and real trace datasets. We first explain how to generate ML predictions based on the training dataset. Then, we shift the distribution of the testing dataset to deviate from the training dataset and showcase the performance of LAPDOA on the testing datasets.

The Synthetic Dataset. In Fig. 5.3(a), PDOA demonstrates strong performance under the Bernoulli process. However, a specific training dataset reveals its suboptimal performance. In this training dataset, the transmission cost $c = 16$, and the channel state sequence is constituted by an independently repeating pattern $[X \times \text{OFF}, Y \times \text{ON}]$, where $X \sim B(13, 0.9)$ and $Y \sim B(6, 0.9)$ ($B(n, p)$ represents the binomial distribution with parameters n and p). Under this pattern, in most cases, PDOA only makes one transmission at the first ON slot of these Y ON slots (i.e., after a long consecutive X OFF slots, the ACK marker M will be larger than 1, and PDOA will transmit at the first ON slot. However, after this transmission, during the short remaining $(Y - 1)$ ON slots, the ACK marker M is

unable to be increased to 1). This results in a high AoI increase for these next X OFF slots. For the optimal offline algorithm, to have a lower AoI during OFF slots, it will transmit at both the first ON slot and the last ON slot among those Y ON slots. To generate a sequence of channel states of the required length, we repeat the pattern enough times independently and concatenate them together.

Recall that LAPDOA incorporates an ML prediction $\mathcal{P}$ that provides the transmission decision at each slot. To generate such an ML prediction $\mathcal{P}$, we train an *Long Short-term Memory (LSTM)* network, which has three LSTM layers (each layer has 20 hidden states) followed by one fully connected layer. The input of our LSTM network is the current channel state, and its output is the transmission probability at that slot. For training, we manually create 300 sequences, each with a length of 100 slots consisting of repeating patterns introduced earlier (we call these constructed sequences “pattern sequences”). Optimal offline transmission decisions for the training datasets are obtained through dynamic programming. We use the mean squared error between the LSTM network output and the optimal offline algorithm output as the loss function and employ the Adam optimizer to train the weights. In the end, to convert the output of our LSTM network (i.e., transmission probability) to the real transmission decisions, a threshold (i.e., 0.5) is set, and transmission occurs when the output of the LSTM network exceeds the threshold.

In Fig. 5.4, we illustrate LAPDOA’s performance under varying prediction qualities, influenced by a distribution shift between the training and testing datasets. The training dataset only contains the sequences fully composed of the pattern (i.e., the percentage of the pattern sequences is 100%). However, in the testing dataset, we reduce the percentage of the pattern sequence by replacing some pattern sequences with a Bernoulli process sequence of a length of 100 with an ON probability of 0.32 (close to the pattern ON probability). While the training dataset and the testing dataset share the same channel ON probability, they

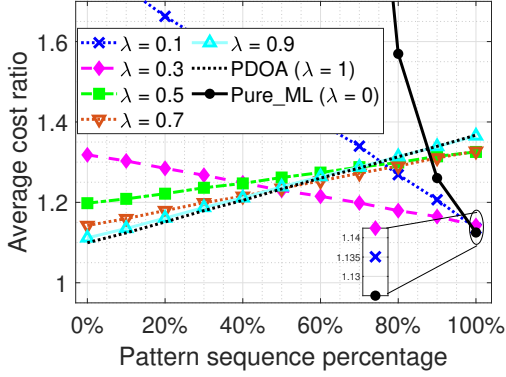


Figure 5.4: Performance comparison of LAPDOA under different trust parameters using synthetic dataset.

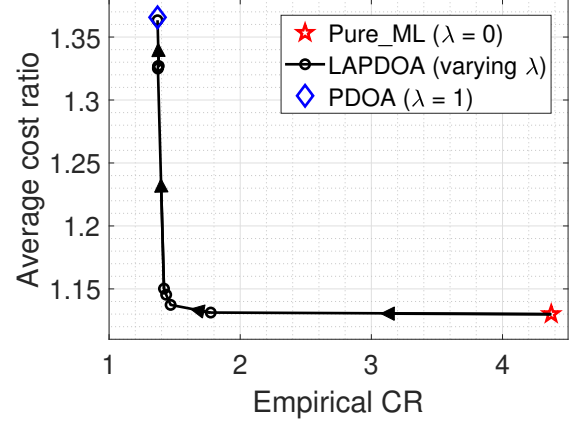


Figure 5.5: Average cost ratio vs. empirical CR of LAPDOA when the pattern sequence percentage is 99%. The direction of the arrow indicates that λ becomes larger.

exhibit shifts in distribution. The magnitude of this shift amplifies as the percentage of the pattern sequence decreases. As we can observe in Fig. 5.4, when the distribution shift is small (100% or 90% pattern sequence percentage), our trained ML algorithm (“Pure_ML” in the figure) outperforms PDOA (recall that Pure_ML is a special case of LAPDOA with $\lambda = 0$, and PDOA is a special case of LAPDOA with $\lambda = 1$). Learning-augmented algorithms trusting the prediction ($\lambda \in \{0.1, 0.3\}$) closely match the ML algorithm’s performance. Conversely, with a substantial distribution shift (0 or 10% pattern sequence percentage), the ML algorithm performs poorly while PDOA performs well. In this case, learning-augmented algorithms not trusting the prediction ($\lambda \in \{0.7, 0.9\}$) closely resemble PDOA. Furthermore, with different values of λ , LAPDOA provides different tradeoff curves for consistency and robustness.

Though the trained ML algorithm performs well in the average case when the distribution shift is small (i.e., Pure_ML achieves a low average cost ratio in Fig. 5.4 when the pattern sequence percentage is high), it may lack worst-case performance guarantees. In Fig. 5.5, we

show the comparison between the average cost ratio and the empirical CR under LAPDOA when the pattern sequence percentage is 99% (i.e., there exists at least a sequence that is not the pattern sequence). Here we consider the LAPDOA algorithms with $\lambda \in \{0, 0.1, \dots, 0.9, 1\}$. Pure_ML achieves the smallest average cost ratio; however, its empirical CR significantly surpasses that of PDOA, indicating that it lacks performance robustness. In addition, as the trust parameter λ increases, the empirical CR performance improves, while the performance of the average cost ratio worsens. In this scenario, selecting λ as 0.3 appears to be beneficial, as it not only yields a low empirical CR but also sustains a low average cost ratio concurrently.

The Real Trace Dataset. We still use the real trace dataset [93]. In addition to the walking tests we introduced before, this dataset also contains the RSRQ measurement of the driving tests. Throughout these driving tests, the researchers mounted the smartphone on the car’s windshield and repeatedly drove on the same 1300m loop area to collect RSRQ information. Again, we set a threshold for RSRQ (-13dB) to determine the ON/OFF channels and let the transmission cost $c = 16$. The differences between the driving datasets and the walking datasets are that: (i) the time length of one driving loop is much shorter than that of one walking loop (i.e., 250 seconds vs 750 seconds); (ii) the proportion of ON slots in the driving dataset is less than that in the walking dataset. As explained in [93], this phenomenon primarily arises due to signal attenuation caused by the car’s body components, such as windshields or side windows. Additionally, the swift movement of the car leads to frequent handoffs between 5G panels and towers, which further degrades signal strength.

Similar to the synthetic dataset, the ML prediction $\mathcal{P}$ is also generated by an LSTM network (with the same architecture as introduced in the synthetic dataset). To train this LSTM network, we use a 5 loop of driving tests as our training dataset. We consider two different testing datasets: (i) a 3 loop of driving tests in Fig. 5.6(a), and (ii) a 3 loop of walking tests in Fig. 5.6(b). The distribution shift between the first testing dataset and

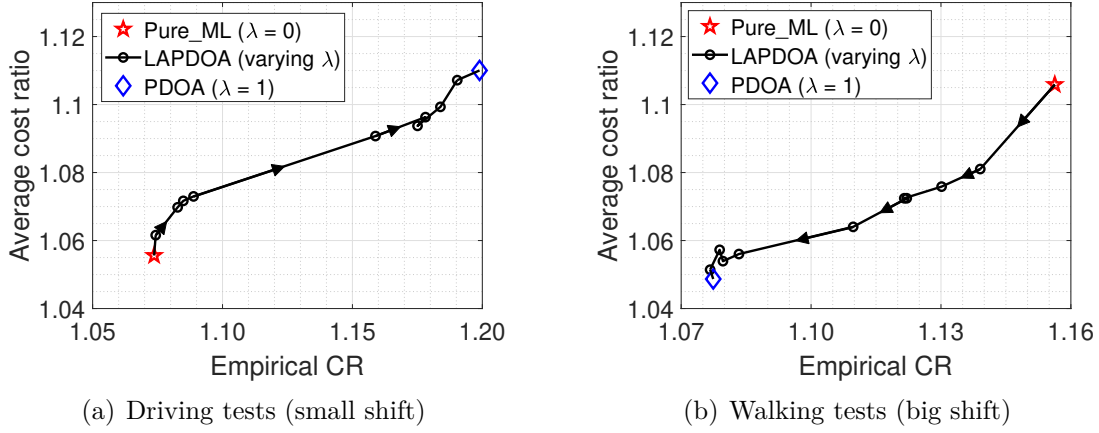


Figure 5.6: Average cost ratio vs. empirical CR of LAPDOA using real trace dataset. The direction of the arrow indicates that λ becomes larger.

the training dataset is small as the data is collected under the same scenario (i.e., driving), while the distribution shift between the second testing dataset and the training dataset is large as the data is collected under the two different scenarios (i.e., walking vs. driving). In Fig. 5.6(a), when the testing dataset is the driving dataset, which has a small distribution shift compared to the training dataset, the Pure_ML demonstrates superior performance not only for average cost ratio but also empirical CR. We conjecture that Pure_ML achieves a low empirical CR because this testing dataset is highly identical to the training datasets (i.e., with the same 1300m loop area, the signal strength measured in one driving loop does not appear to change dramatically in another driving loop, and thus those driving loops share a similar signal strength pattern). However, in Fig. 5.6(b), when the testing dataset significantly deviates from the training dataset, the performance of Pure_ML diminishes, resulting in both a high average cost ratio and a high empirical CR. In contrast, the PDOA online algorithm excels in this scenario in terms of both the average cost ratio and the empirical CR. Upon analyzing these two testing datasets, we learn that, on the one hand, if we understand the characteristics of the testing dataset, we can select our trust parameters correspondingly. For example, if we are aware that the testing dataset deviates from the

training dataset greatly, we should choose a lower trust parameter. On the other hand, when uncertainty shrouds the testing dataset, selecting an appropriate trust parameter (e.g., $\lambda = 0.3$) enables LAPDOA to strike a good trade-off between consistency and robustness.

5.7 Chapter Summary

In this chapter, we studied the minimization of data freshness and transmission costs under a time-varying wireless channel. After reformulating our original problem to a TCP ACK problem, we developed a 3-competitive primal-dual-based online algorithm. Realizing the pros and cons of online algorithms and ML algorithms, we designed a learning-augmented online algorithm that takes advantage of both approaches and achieves consistency and robustness. Finally, simulation results validate the superiority of our online algorithm and highlight the consistency and robustness achieved by our learning-augmented algorithm. For future work, one interesting direction would be to consider how to adaptively select the trust parameter λ to achieve the best performance.

Chapter 6

Summary and Future Work

6.1 Summary

In this dissertation, we studied AoI, a recently proposed timeliness measure, from the following three aspects: (i) the performance of AoI under various scheduling policies (Chapter 2), (ii) the tradeoffs between AoI and other performance metrics (i.e., service performance and update cost) in real-life applications (Chapters 3 and 4), and (iii) robust online algorithms for AoI optimization without and with ML predictions (Chapter 5). A summary for each chapter is given below:

The performance of AoI under various scheduling policies. Since the concept of AoI is introduced in [3], there is plenty of work analyzing the performance of AoI under different scheduling policies. However, as one important design aspect, the update-size information somehow has not been given enough attention yet and thus making it unclear how exactly scheduling policies (especially those making use of update-size information) impact the AoI performance. In Chapter 2, we systematically studied the impact of various aspects of scheduling policies on the AoI performance and provided several useful guidelines for the design of AoI-efficient scheduling policies. Our study reveals that among various aspects of scheduling policies we investigated, prioritizing small updates, allowing service preemption, and prioritizing informative updates play the most important role in the design of AoI-efficient scheduling policies. It turns out that common scheduling policies like SRPT

and SJF_P and their informative variants can achieve a very good AoI performance, although they do not explicitly make scheduling decisions based on the AoI. This can be partially explained by the equivalence between such size-based policies and some AoI-based policies.

The tradeoff between PAoI and service performance. We observe that in data-driven real-time applications (e.g., Google Maps or stock trading applications), the computing resource is often shared for processing both updates from information sources and queries from end users. In this case, there is a natural tradeoff between service performance (e.g., response time to queries) and information freshness. In Chapter 3, we proposed a simple single-server two-queue model that captures the coupled scheduling between updates and queries for data-driven real-time applications. To address the tradeoff, we proposed threshold-based scheduling policies that prioritize updates or queries and analyzed the response time and the PAoI in a rigorous manner. The simulation results further demonstrated that by properly choosing the values of the thresholds, the proposed policies can achieve the desired balance between service performance and information freshness.

The tradeoff between AoI and update cost. For the crowdsensing-based applications (e.g., Google Waze and GasBuddy), on the one hand, users are not satisfied if the responses to their requests are stale; on the other side, there is a cost for the applications to update their information regarding certain points of interests since they usually need to make monetary payments to incentivize users to send their feedback. In Chapter 4, we considered this fundamental tradeoff between the data freshness and the update cost. We provided useful guidelines for the design of update policies. Assuming Bernoulli request arrival process, we also proposed a threshold-based update policy and proved its optimality. Our simulations based on both synthetic data and real traces corroborated the theoretical results and showed that the optimal threshold-based policy outperforms the baseline policies.

Robust online algorithms for AoI optimization without and with ML predic-

tions. In Chapter 5, we studied the minimization of data freshness and transmission costs under a time-varying wireless channel. After reformulating our original problem to a TCP ACK problem, we developed a 3-competitive primal-dual-based online algorithm. Realizing the pros and cons of online algorithms and ML algorithms, we designed a learning-augmented online algorithm that takes advantage of both approaches and achieves consistency and robustness. Finally, simulation results validate the superiority of our online algorithm and highlight the consistency and robustness achieved by our learning-augmented algorithm.

6.2 Future Work

While this dissertation represents a significant leap forward in expanding the current understanding of AoI research, many unresolved problems await exploration. The following section delves into some of these open problems.

Rigorously analyzing AoI under different scheduling policies. In Chapter 2, a systematic examination was conducted on the influence of different facets of scheduling policies on AoI performance, yielding several valuable insights for crafting scheduling policies optimized for AoI efficiency. Moreover, our findings pose intriguing questions warranting further investigation in future endeavors. One notable direction is the pursuit of theoretical underpinnings beyond the scope of the simulation results presented in this chapter. For instance, it would be compelling to ascertain through rigorous proof whether informative policies consistently outperform their non-informative counterparts, a phenomenon consistently observed in our simulations.

The balance between AoI and service performance. Although Chapter 3 provides useful insights towards the tradeoff between the response time and the PAoI, there remain some open questions, which are worthy of further investigation in future work. For example,

it would be interesting to rigorously analyze the average AoI under the threshold-based policies and to systematically study the Joint- (M, N) policy. In addition, we have implicitly assumed that there was a negligible overhead for the server to switch back and forth between the query queue and the update queue. It would be interesting to investigate and characterize the impact of switching cost.

Leveraging ML predictions for AoI optimization. In Chapter 5, we designed a learning-augmented online algorithm that takes advantage of both online algorithms and ML algorithms and achieves consistency and robustness. However, there are still some open questions that deserve further research. For example, one interesting direction would be to consider how to adaptively select the trust parameter λ to achieve the best performance. In addition, in the optimization problem we consider, the ML prediction is solely treated as an external input, and we lack the capability to generate or control it by ourselves. It would be interesting to consider the co-design of ML algorithms and online algorithms to further improve the performance of the learning-augmented online algorithm.

Appendices

Appendix A

Appendix of Chapter 2

A.1 Additional Simulation Results for the G/G/1 Queue

We present additional simulation results for the G/G/1 queue in Figs. A.1-A.8. For all these simulations, we assume that the interarrival time follows a Weibull distribution with $C^2 = 10$. In subfigure (a), we assume that the update size follows an Exponential distribution with mean $1/\mu = 1$; in subfigures (b) and (c), we assume that the update size follows a Weibull distribution with mean $1/\mu = 1$. Note that in subfigures (a) and (b), we change the value of the system load ρ ; in subfigure (c), we change the value of C^2 for the update size while fixing the system load at $\rho = 0.7$. Observations 1-8 can also be made for the setting of G/G/1 queue.

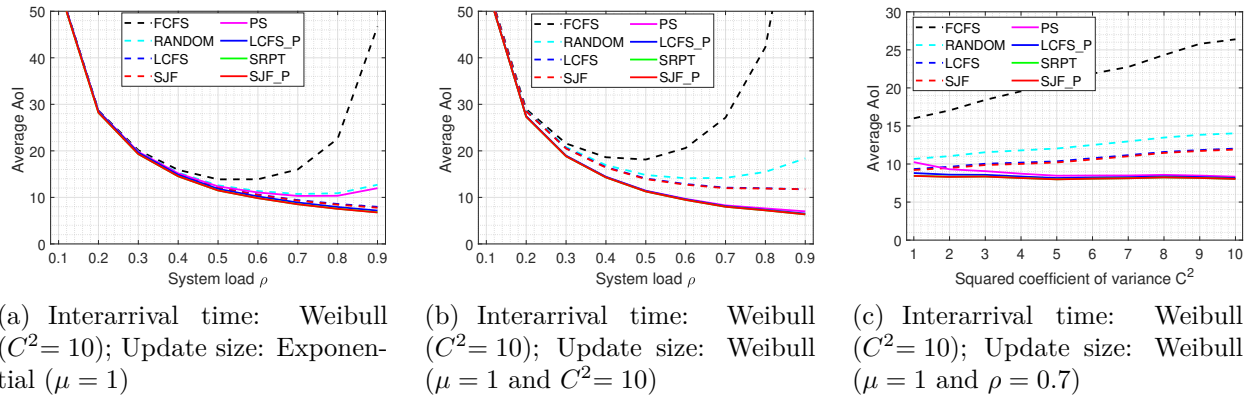
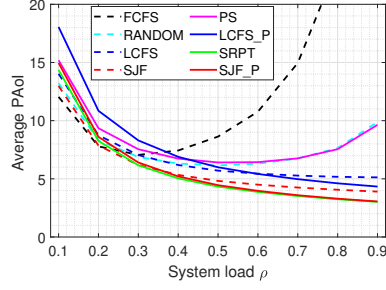
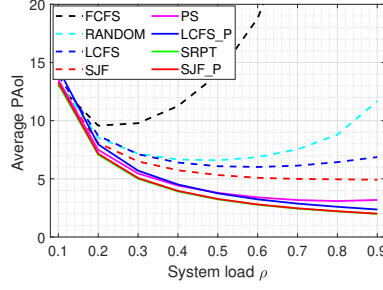


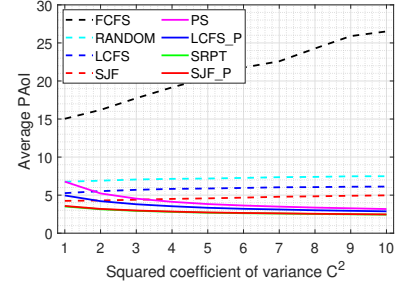
Figure A.1: Comparisons of the average AoI performances of several common scheduling policies under different distributions



(a) Interarrival time: Weibull ($C^2 = 10$); Update size: Exponential ($\mu = 1$)

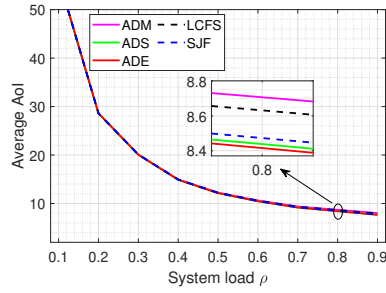


(b) Interarrival time: Weibull ($C^2 = 10$); Update size: Weibull ($\mu = 1$ and $C^2 = 10$)

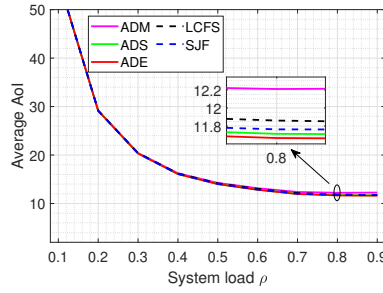


(c) Interarrival time: Weibull ($C^2 = 10$); Update size: Weibull ($\mu = 1$ and $\rho = 0.7$)

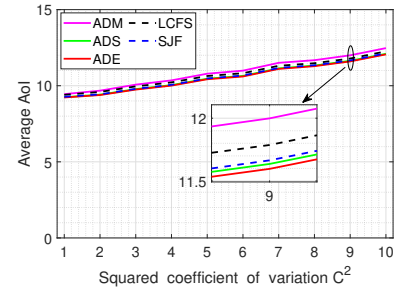
Figure A.2: Comparisons of the average PAoI performances of several common scheduling policies under different distributions



(a) Interarrival time: Weibull ($C^2 = 10$); Update size: Exponential ($\mu = 1$)

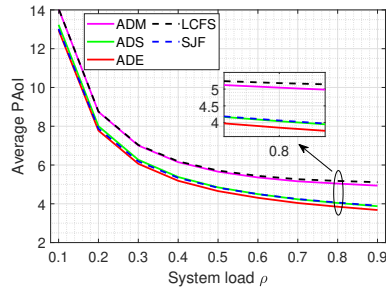


(b) Interarrival time: Weibull ($C^2 = 10$); Update size: Weibull ($\mu = 1$ and $C^2 = 10$)

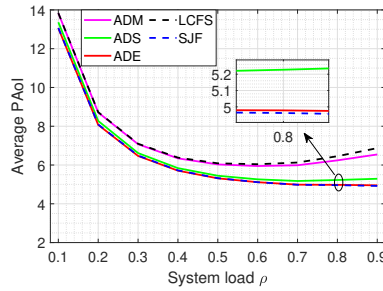


(c) Interarrival time: Weibull ($C^2 = 10$); Update size: Weibull ($\mu = 1$ and $\rho = 0.7$)

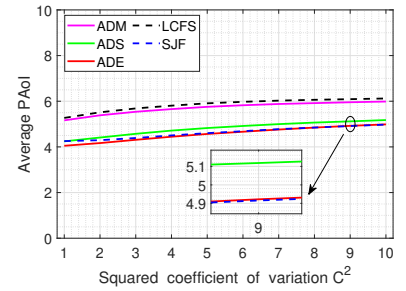
Figure A.3: Comparisons of the average AoI performance under different distributions: AoI-based policies vs. non-AoI-based policies



(a) Interarrival time: Weibull ($C^2 = 10$); Update size: Exponential ($\mu = 1$)



(b) Interarrival time: Weibull ($C^2 = 10$); Update size: Weibull ($\mu = 1$ and $C^2 = 10$)



(c) Interarrival time: Weibull ($C^2 = 10$); Update size: Weibull ($\mu = 1$ and $\rho = 0.7$)

Figure A.4: Comparisons of the average PAoI performance under different distributions: AoI-based policies vs. non-AoI-based policies

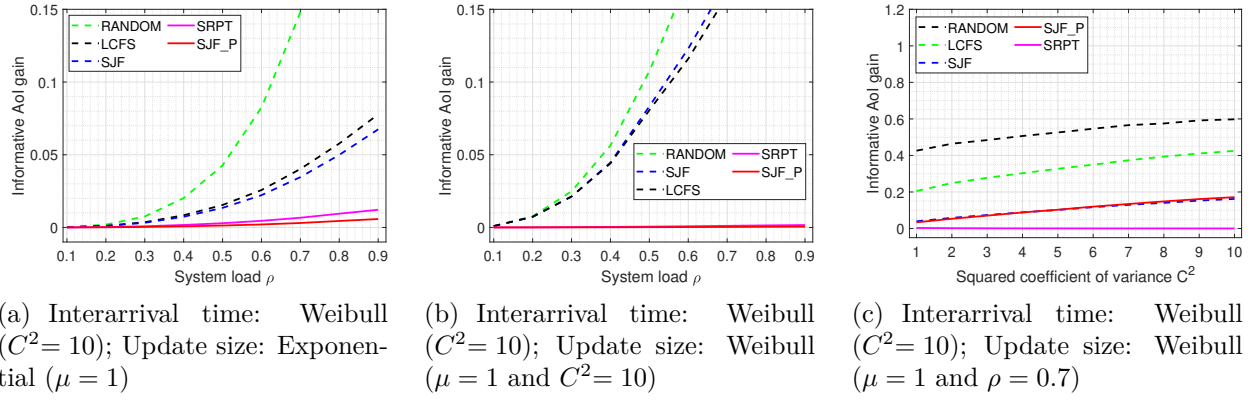


Figure A.5: Comparisons of the avg. AoI performance under different distributions: informative policies vs. non-informative policies

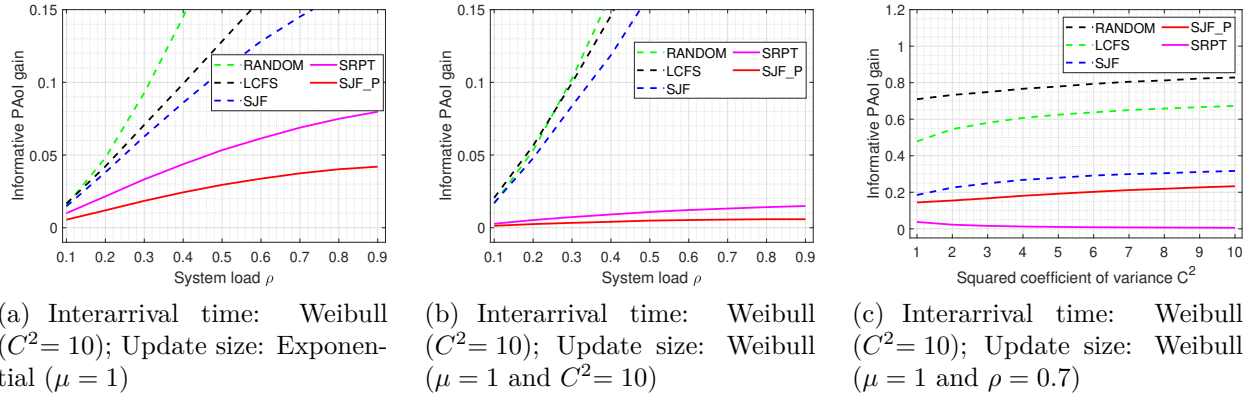


Figure A.6: Comparisons of the avg. PAoI performance under different distributions: informative policies vs. non-informative policies

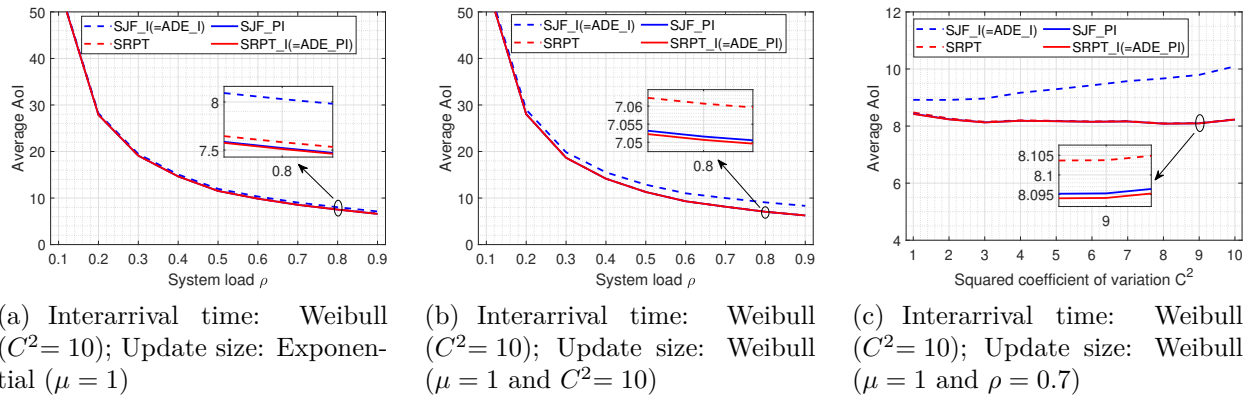
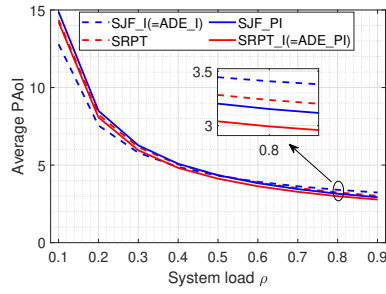
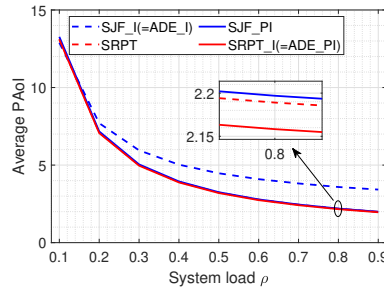


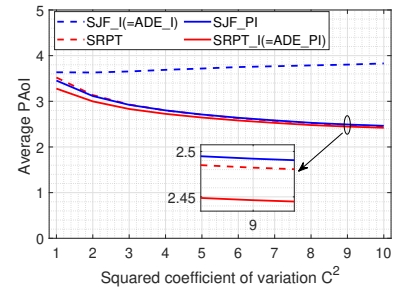
Figure A.7: Comparisons of the avg. AoI performance under different distributions: preemptive, informative, AoI-based policies vs. others



(a) Interarrival time: Weibull ($C^2=10$); Update size: Exponential ($\mu=1$)



(b) Interarrival time: Weibull ($C^2=10$); Update size: Weibull ($\mu=1$ and $C^2=10$)



(c) Interarrival time: Weibull ($C^2=10$); Update size: Weibull ($\mu=1$ and $\rho=0.7$)

Figure A.8: Comparisons of the avg. PAoI performance under different distributions: pre-emptive, informative, AoI-based policies vs. others

Appendix B

Proofs of Chapter 4

B.1 Proof of Lemma 4.1

Proof. For any given policy $\pi \in \Pi$, there are two cases: (i) $\pi \in \Pi^R$ and (ii) $\pi \in \Pi \setminus \Pi^R$.

Case (i) is trivial as we can simply choose $\pi' = \pi$.

In Case (ii), we construct a reactive policy $\pi' \in \Pi^R$ in the following manner. Consider any sample path with N requests: 1) for all the updates performed by policy π at some request arrival times, policy π' also has such updates; 2) for each update performed by policy π that is not at any of the request arrival times, policy π' postpones the update to the time instant when the next request arrives. Clearly, policy π' constructed in the above manner is a valid reactive policy.

Next, we want to prove that in Case (ii), policy π' achieves a total cost smaller than that of policy π by induction. Note that at time 1, the AoI under policy π' is no larger than that of policy π . (Although since the AoI at time 1 is equal under both policies, we want to use “no larger than” instead of “equal” here so that the argument can later be repeatedly applied.) Let u_k be the first time at which policy π updates the data when no request arrives, and let r_l be the arrival time of the first request that arrives after time u_k . Fig. B.1 provides an illustration of such a scenario, where the solid black curve and the dashed red curve denote the AoI trajectories under policies π and π' , respectively. We want to show that policy π'

has a smaller total cost than that of policy π during interval $[1, r_l]$.

We first compare the total cost during interval $[1, u_k)$ under both policies. Note that under policy π , all the updates performed before time u_k are at request arrival times because u_k is the first time at which policy π updates the data when no request arrives. Due to Step 1) of the above construction, policy π' must update the data in a way that is exactly the same as policy π during interval $[1, u_k)$. Hence, the total update cost during interval $[1, u_k)$ must be the same under both policies. Also, the AoI under policy π' must remain no larger than that under policy π during interval $[1, u_k)$. Hence, policy π' must have a total staleness cost no larger than that of policy π during interval $[1, u_k)$. This implies that policy π' has a total cost (including both update cost and staleness cost) no larger than that of policy π during interval $[1, u_k)$.

We now compare the total cost during the interval $[u_k, r_l]$ under both policies. Due to Step 2) of the construction of policy π' , it does not update the data at time u_k but postpones the update to time r_l . Then, under policy π' the total cost during interval $[u_k, r_l]$ is equal to p because the only update is performed at time r_l at which the only request arrives. On the other hand, policy π has a total cost strictly larger than p because it performs one update at time u_k and needs to pay either a staleness cost at time r_l or at least an additional update cost during interval $(u_k, r_l]$. Hence, policy π' has a total cost smaller than that of policy π during interval $[u_k, r_l]$.

Combining the above discussions, we show that policy π' achieves a total cost smaller than that of policy π during interval $[1, r_l]$.

Since the AoI under policy π' becomes 1 at time r_{l+1} , the AoI under policy π' is apparently no larger than that of policy π at time r_{l+1} . Then, we can view time r_{l+1} as a new starting point and inductively apply the above argument. This completes the proof. $\square$

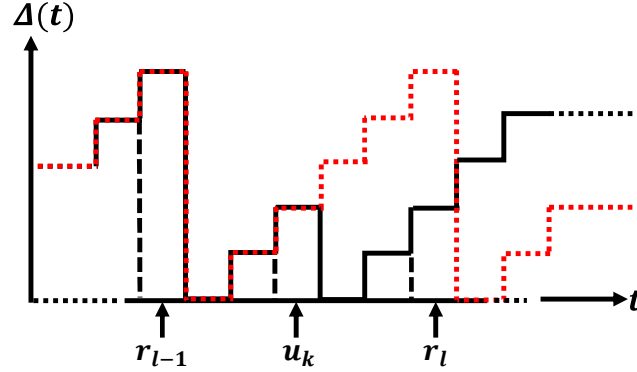


Figure B.1: An illustration of the advantage of a constructed reactive policy $\pi' \in \Pi^R$ over an arbitrary given non-reactive policy $\pi \in \Pi \setminus \Pi^R$, where the solid black curve and the dashed red curve denote the AoI trajectories under policies π and π' , respectively.

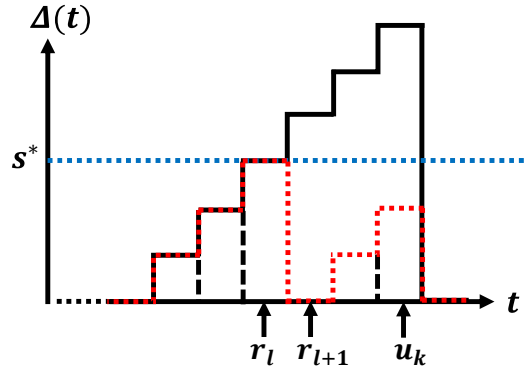


Figure B.2: An illustration of the advantage of a constructed capped reactive policy $\pi' \in \Pi^{R+}$ over an arbitrary given non-capped reactive policy $\pi \in \Pi^R \setminus \Pi^{R+}$, where the solid black curve and the dashed red curve denote the AoI trajectories under policies π and π' , respectively.

B.2 Proof of Lemma 4.2

Proof. For any given policy $\pi \in \Pi^R$, there are two cases: (i) $\pi \in \Pi^{R+}$ and (ii) $\pi \in \Pi^R \setminus \Pi^{R+}$.

Case (i) is trivial as we can simply choose $\pi' = \pi$. In Case (ii), we construct a capped reactive policy $\pi' \in \Pi^{R+}$ in the following manner. Consider any sample path with N requests: 1) for all the updates performed by policy π , policy π' also has such updates; 2) for every request r_j such that $\Delta(r_j) \geq s^*$ (see definition of s^* in Eq. (4.7)), policy π' updates the data at r_j regardless of policy π 's update decision at r_j . Clearly, policy π' constructed in the

above manner is a valid capped reactive policy.

Next, we want to show that in Case (ii), policy π' achieves a total cost smaller than that of policy π by induction. Recall that the system starts with $\Delta(1) = 1$. Let r_l be the first request arrival time at which $\Delta(r_j) \geq s^*$ but policy π does not update the data. Then, the AoI trajectories before time r_l are exactly the same under policies π' and π . Due to Step 2) of the above construction, policy π' must update the data at r_l because $\Delta(r_j) \geq s^*$. Let u_k be the first update performed after r_l under policy π . Then, policy π' must also update the data at time u_k due to Step 1) of the above construction. Fig. B.2 provides an illustration of such a scenario, where the solid black curve and the dashed red curve denote the AoI trajectories under policies π and π' , respectively. It is easy to see that policy π' has a smaller AoI than that under policy π during interval (r_l, u_k) .

We now compare the total cost during interval $[1, u_k]$ under both policies. First, note that the AoI trajectories before time r_l are exactly the same under policies π' and π , i.e., both policies have the same cost during interval $[1, r_l)$. Then, it is easy to see that at time r_l , policy π' has an update cost p but no staleness cost, and policy π has a staleness cost $\Delta(r_j)$ but no update cost. Hence, policy π' achieves a smaller total cost than that of policy π at time r_l because $\Delta(r_j) \geq s^*$. Also, note that policy π' has a smaller AoI than that of policy π during interval (r_l, u_k) and that there is no update under both policies during interval (r_l, u_k) . Hence, policy π' achieves a smaller total cost than that of policy π during interval (r_l, u_k) . At time u_k , both policies have an update cost p but no staleness cost since neither of them updates the data. Combining the above discussions, we show that policy π' achieves a smaller total cost than that of policy π during interval $[1, u_k]$.

Since $\Delta(u_k)$ drops to 0 immediately at time u_k under both policies, we can view time $(u_k + 1)$ as a new starting point and repeatedly apply the above argument. This completes the proof. $\square$

B.3 Proof of Theorem 4.3

Proof. Our proof includes two steps: 1) We study a discounted MDP and show that the optimal value function of the discounted MDP is non-decreasing in the initial state s ; 2) Based on the optimal value function of the discounted MDP, we derive the Bellman equation of the average expected cost and show that it has a threshold-based structure, where the threshold is based on the AoI. This implies that there exists an optimal threshold-based stationary capped reactive policy for the average expected cost.

Step 1): In general, the derivation and properties of the Bellman equation of the average expected cost are not easy to obtain, and we usually rely on the study of the discounted MDP to get some insights towards the design of an optimal policy [73].

The expected total α -discounted cost of a capped reactive policy $\pi \in \Pi^{R+}$ is defined as

$$C_\alpha^\pi(s) \triangleq \mathbb{E}_\pi \left[\sum_{n=1}^{\infty} \alpha^{n-1} c(s_n, a_n) | s_1 = s \right], \quad (\text{B.1})$$

where $0 \leq \alpha < 1$ is the discount factor. Here, $C_\alpha^\pi(s)$ is well defined, given that for any n , we have $\mathbb{E}_\pi[c(s_n, a_n) | s_1 = s] \leq p$ under a capped reactive policy π , and thus, we have

$$C_\alpha^\pi(s) \leq \sum_{n=1}^{\infty} \alpha^{n-1} p = \frac{p}{1-\alpha}. \quad (\text{B.2})$$

Let $C_\alpha(s) \triangleq \min_{\pi} C_\alpha^\pi(s)$ be the optimal value function. Then, we can obtain the Bellman equation of the α -discounted MDP with $C_\alpha(s)$ [73], which is

$$C_\alpha(s) = \min_{a \in \mathcal{A}_s} \left\{ c(s, a) + \alpha \sum_{z \in \mathcal{S}} p(z | s, a) C_\alpha(z) \right\}. \quad (\text{B.3})$$

The Bellman equation Eq. (B.3) states that the value of the initial state s (i.e., $C_\alpha(s)$) equals

the expected return of the best action, which is the discounted expected value of the next state (i.e., $\alpha \sum_{z \in S} p(z | s, a) C_\alpha(z)$), plus the immediate cost along the way (i.e., $c(s, a)$). In the following, we show that $C_\alpha(s)$ is non-decreasing in s . This property enables us to show that the Bellman equation of the average cost (i.e., Lemma B.2) has a threshold-based structure.

Lemma B.1. *The optimal value function $C_\alpha(s)$ is non-decreasing in the initial state s .*

We provide the proof of Lemma B.1 in Appendix B.4 and explain the key ideas in the following. The goal is to construct a sequence $\{C_{\alpha,n}(s)\}$ that is non-decreasing in s for any n , where $C_{\alpha,n}(s)$ is the minimal expected discounted cost in an n -stages problem. Then, we show that $C_\alpha(s) = \lim_{n \rightarrow \infty} C_{\alpha,n}(s)$, which implies that $C_\alpha(s)$ is also non-decreasing in s .

Step 2): With the optimal value function of the α -discounted MDP (i.e., $C_\alpha(s)$), we can derive the Bellman equation of the average cost as follows.

Lemma B.2. *Let $h(s) \triangleq \lim_{\alpha \rightarrow 1} [C_\alpha(s) - C_\alpha(1)]$ and $g \triangleq \lim_{\alpha \rightarrow 1} (1 - \alpha) C_\alpha(1)$. Then, the Bellman equation of the average cost is given by the following:*

$$h(s) + g = \begin{cases} p + \sum_{z=1}^{\infty} (1 - \lambda)^{z-1} \lambda h(z), & \text{if } s \geq \Delta^*; \\ \min \left\{ p + \sum_{z=1}^{\infty} (1 - \lambda)^{z-1} \lambda h(z), \right. \\ \left. f(s) + \sum_{z=s+1}^{\infty} (1 - \lambda)^{z-s-1} \lambda h(z) \right\}, & \text{if } s < \Delta^*. \end{cases} \quad (\text{B.4})$$

We provide the proof of Lemma B.2 in Appendix B.5 and explain the key ideas in the following. First, given the definitions of $h(s)$ and g , we show that Eq. (B.4) does hold. To this end, we define $h_\alpha(s) \triangleq C_\alpha(s) - C_\alpha(1)$ and substitute $h_\alpha(s)$ into the Bellman equation of the α -discounted MDP Eq. (B.3). Then, we prove that we can find a sequence $\{\alpha_m\} \rightarrow 1$

such that $\lim_{m \rightarrow \infty} h_{\alpha_m}(s) = h(s)$ for any s and $\lim_{m \rightarrow \infty} (1 - \alpha_m)C_{\alpha_m}(1) = g$. Taking the limit $m \rightarrow \infty$ on both sides of the Bellman equation of $h_{\alpha}(s)$, we obtain Eq. (B.4). Second, we show that Eq. (B.4) is the Bellman equation for the average expected cost. This can be done by applying the same techniques used in [73, Chapter V, Theorem 2.1].

Next, we show that the Bellman equation Eq. (B.4) has a threshold structure, which guides us to find the optimal threshold-based stationary capped reactive policy.

Assume that the current state is s . Based on the Bellman equation Eq. (B.4), it is optimal to update when $s \geq \Delta^*$; and when $s < \Delta^*$, it is optimal to update if

$$\begin{aligned} f(s) + \alpha \sum_{z=s+1}^{\infty} (1 - \lambda)^{z-s-1} \lambda h(z) \geq \\ p + \alpha \sum_{z=1}^{\infty} (1 - \lambda)^{z-1} \lambda h(z), \end{aligned} \quad (\text{B.5})$$

where the right hand side is a constant. It is easy to check that $h(s)$ is non-decreasing in s given that $C_{\alpha}(s)$ is non-decreasing in s . Hence, we can find s^* as follows:

$$\begin{aligned} s^* \triangleq \min \left\{ s : f(s) + \alpha \sum_{z=s+1}^{\infty} (1 - \lambda)^{z-s-1} \lambda h(z) \geq \right. \\ \left. p + \alpha \sum_{z=1}^{\infty} (1 - \lambda)^{z-1} \lambda h(z) \right\}. \end{aligned} \quad (\text{B.6})$$

Set $s^* = \Delta^*$ when $s^* \geq \Delta^*$. Now consider a capped reactive policy $\pi^* \in \Pi^{R+}$: upon receiving a request, policy π^* updates the data if the current state is no smaller than s^* ; otherwise, it does not update the data and replies with the current local data. Clearly, policy π^* is a stationary threshold-based policy. Besides, policy π^* selects the action that minimizes the right hand side of the Bellman equation Eq. (B.4) for any state. Thus, it is an optimal policy [73, Chapter V, Theorem 2.1]. $\square$

B.4 Proof of Lemma B.1

Proof. Our proof idea is to construct a sequence $\{C_{\alpha,n}(s)\}$ that is non-decreasing in s for any n , where $C_{\alpha,n}(s)$ is the minimal expected discounted cost in an n -stages problem. Then, we show that $C_\alpha(s) = \lim_{n \rightarrow \infty} C_{\alpha,n}(s)$, which implies that $C_\alpha(s)$ is also non-decreasing in s .

First, we show how to construct the sequence $\{C_{\alpha,n}(s)\}$. Consider an n -stage problem of our α -discounted MDP. Denote the minimal expected discounted cost of this n -stage problem by

$$\begin{aligned} C_{\alpha,n}(s) &\triangleq \min_{a \in A_s} \left\{ c(s, a) + \alpha \sum_{z \in S} p(z \mid s, a) C_{\alpha,n-1}(z) \right\} \\ &= \begin{cases} p + \alpha \sum_{z=1}^{\infty} (1 - \lambda)^{z-1} \lambda C_{\alpha,n-1}(z), & \text{if } s \geq \Delta^*; \\ \min \left\{ p + \alpha \sum_{z=1}^{\infty} (1 - \lambda)^{z-1} \lambda C_{\alpha,n-1}(z), \right. \\ \left. f(s) + \alpha \sum_{z=s+1}^{\infty} (1 - \lambda)^{z-s-1} \lambda C_{\alpha,n-1}(z) \right\}, & \text{if } s < \Delta^*, \end{cases} \end{aligned}$$

where the terminal cost is $C_{\alpha,1}(s) \triangleq \min\{p, f(s)\}$.

Then, we prove by induction that our constructed sequence $\{C_{\alpha,n}(s)\}$ is non-decreasing in s for any n . Obviously, $C_{\alpha,1}(s)$ is non-decreasing in s . We assume that $C_{\alpha,n-1}(s)$ is non-decreasing in s . Next, we show that $C_{\alpha,n}(s)$ is non-decreasing in s . When $s \geq \Delta^*$, $C_{\alpha,n}(s)$ is a constant and is independent of s . On the other hand, when $s < \Delta^*$, to better present our discussion, we denote

$$c \triangleq p + \alpha \sum_{z=1}^{\infty} (1 - \lambda)^{z-1} \lambda C_{\alpha,n-1}(z)$$

and

$$C'_{\alpha,n-1}(s) \triangleq f(s) + \alpha \sum_{z=s+1}^{\infty} (1 - \lambda)^{z-s-1} \lambda C_{\alpha,n-1}(z)$$

$$= f(s) + \alpha \sum_{k=1}^{\infty} (1 - \lambda)^{k-1} \lambda C_{\alpha, n-1}(s + k),$$

where the last step follows by setting $k = z - s$. We want to show that $C'_{\alpha, n-1}(s)$ is non-decreasing in s . To see this, consider two states: $j \geq i > 0$. Then, we have

$$\begin{aligned} & C'_{\alpha, n}(j) - C'_{\alpha, n}(i) \\ &= f(j) - f(i) \\ &+ \alpha \sum_{k=1}^{\infty} (1 - \lambda)^{k-1} \lambda (C_{\alpha, n-1}(j + k) - C_{\alpha, n-1}(i + k)). \end{aligned}$$

Since $f(s)$ is non-decreasing in s , we have $f(j) \geq f(i)$. By inductive hypothesis that $C_{\alpha, n-1}(s)$ is non-decreasing in s , we have $C_{\alpha, n-1}(j + k) \geq C_{\alpha, n-1}(i + k)$ for any $k > 0$. Therefore, $C'_{\alpha, n}(j) - C'_{\alpha, n}(i) \geq 0$, i.e., $C_{\alpha, n}(s)$ is non-decreasing in s . This, along with c being a constant, implies that $C_{\alpha, n}(s) = \min\{c, C'_{\alpha, n-1}(s)\}$ is also non-decreasing in s when $s < \Delta^*$. Till this point, we have shown that $C_{\alpha, n}(s)$ is non-decreasing in s when $s < \Delta^*$ and when $s \geq \Delta^*$, respectively. In addition, since $C_{\alpha, n}(s)$ achieves a smaller value when $s < \Delta^*$ (i.e., $\min\{c, C'_{\alpha, n-1}(s)\}$) compared to the case of $s \geq \Delta^*$ (i.e., c), this implies that $C_{\alpha, n}(s)$ is non-decreasing in s for any n .

Finally, given any non-negative integer s , by the definition of $C_{\alpha, n}(s)$ and the fact that the cost in each stage is non-negative, we know that $C_{\alpha, n}(s)$ is non-decreasing in n . In addition, $C_{\alpha, n}(s)$ is bounded. Indeed, the cost in each stage is bounded by the update cost p under our considered policy. As such, $C_{\alpha, n}(s) \leq \sum_{i=1}^n \alpha^{i-1} p = (1 - \alpha^n)p / (1 - \alpha) \leq p / (1 - \alpha)$. Therefore, by the monotone convergence theorem, we have

$$\lim_{n \rightarrow \infty} C_{\alpha, n}(s) = C_{\alpha}(s), \tag{B.7}$$

holding for any state $s = 0, 1, 2, \dots$. It still remains to prove that $C_\alpha(s)$ is non-decreasing in s . We prove this by contradiction. Suppose that $C_\alpha(s)$ is decreasing in s , i.e., there exists $i, j \in \mathcal{S}$ such that $i < j$ and $C_\alpha(i) > C_\alpha(j)$. Let

$$r = C_\alpha(i) - C_\alpha(j) > 0. \quad (\text{B.8})$$

Note that $C_{\alpha,n}(s)$ converges to $C_\alpha(s)$ pointwise for any s , there exists positive integers N_1 and N_2 such that for any $n > \max\{N_1, N_2\}$,

$$|C_\alpha(i) - C_{\alpha,n}(i)| < \frac{r}{3} \quad (\text{B.9})$$

$$\text{and } |C_\alpha(j) - C_{\alpha,n}(j)| < \frac{r}{3}, \quad (\text{B.10})$$

which are equivalent to

$$-\frac{r}{3} < C_\alpha(i) - C_{\alpha,n}(i) < \frac{r}{3} \quad (\text{B.11})$$

$$\text{and } -\frac{r}{3} < C_\alpha(j) - C_{\alpha,n}(j) < \frac{r}{3}. \quad (\text{B.12})$$

By subtracting Eq. (B.11) from Eq. (B.12), we have

$$-\frac{2r}{3} < (C_\alpha(j) - C_{\alpha,n}(j)) - (C_\alpha(i) - C_{\alpha,n}(i)) < \frac{2r}{3}, \quad (\text{B.13})$$

Note that

$$\begin{aligned} & (C_\alpha(j) - C_{\alpha,n}(j)) - (C_\alpha(i) - C_{\alpha,n}(i)) \\ &= (C_{\alpha,n}(i) - C_{\alpha,n}(j)) - (C_\alpha(i) - C_\alpha(j)) \\ &= (C_{\alpha,n}(i) - C_{\alpha,n}(j)) - r. \end{aligned} \quad (\text{B.14})$$

Then, we have

$$C_{\alpha,n}(i) - C_{\alpha,n}(j) > \frac{r}{3} > 0, \quad (\text{B.15})$$

which contradicts the fact that $C_{\alpha,n}(s)$ is non-decreasing in s for any n . Therefore, $C_\alpha(s)$ must be non-decreasing in s . $\square$

B.5 Proof of Lemma B.2

Proof. The proof consists of two steps: 1) given the definitions of $h(s)$ and g , we show that Eq. (B.4) does hold; 2) we show that Eq. (B.4) is the Bellman equation for the average cost.

Step 1): As we can see that the Bellman equation Eq. (B.4) consists of two cases: $s \geq \Delta^*$ and $s < \Delta^*$. In this proof, we only prove the case of $s < \Delta^*$, and a similar proof can also be applied to the case of $s \geq \Delta^*$.

Define

$$h_\alpha(s) \triangleq C_\alpha(s) - C_\alpha(1), \quad (\text{B.16})$$

and substituting $C_\alpha(s)$ into the Bellman equation of the α -discounted MDP Eq. (B.3) gives

$$\begin{aligned} & h_\alpha(s) + (1 - \alpha)C_\alpha(1) \\ &= \min \left\{ p + \alpha \sum_{z=1}^{\infty} (1 - \lambda)^{z-1} \lambda h_\alpha(z), \right. \\ & \quad \left. s + \alpha \sum_{z=s+1}^{\infty} (1 - \lambda)^{z-s-1} \lambda h_\alpha(z) \right\}. \end{aligned} \quad (\text{B.17})$$

Our idea is to find a sequence $\{\alpha_m\} \rightarrow 1$ (where $0 \leq \alpha_m < 1$ for any m) such that

$$\lim_{m \rightarrow \infty} h_{\alpha_m}(s) = h(s) \text{ for any } s \text{ and } \lim_{m \rightarrow \infty} (1 - \alpha_m)C_{\alpha_m}(1) = g.$$

From Eq. (B.3), we know that

$$\begin{aligned}
h_\alpha(s) &= C_\alpha(s) - C_\alpha(1) \\
&\leq p + \alpha \sum_{z=1}^{\infty} (1-\lambda)^{z-1} \lambda C_\alpha(z) - C_\alpha(1) \\
&= p + \alpha \sum_{z=2}^{\infty} (1-\lambda)^{z-1} \lambda C_\alpha(z) + (\alpha\lambda - 1)C_\alpha(1) \\
&\leq p + \alpha \sum_{z=2}^{\infty} (1-\lambda)^{z-1} \lambda C_\alpha(z),
\end{aligned} \tag{B.18}$$

where the last quantity is a constant and we denote it as $M \triangleq p + \alpha \sum_{z=2}^{\infty} (1-\lambda)^{z-1} \lambda C_\alpha(z)$. This immediately gives $|h_\alpha(s)| \leq M$ because $C_\alpha(s)$ is non-decreasing in s and thus $h_\alpha(s) \geq 0$, which also implies that $h_\alpha(s)$ is uniformly bounded for all s and α . Because every bounded sequence contains a convergent subsequence [94, Bolzano–Weierstrass Theorem], we can find a sequence $\{\alpha_m^{(1)}\}$ with $\lim_{m \rightarrow \infty} \alpha_m^{(1)} = 1$ such that $\lim_{m \rightarrow \infty} h_{\alpha_m^{(1)}}(1) \triangleq h(1)$ exists. Furthermore, because $h_{\alpha_m^{(1)}}(2)$ is also bounded, we can find a subsequence $\{\alpha_m^{(2)}\} \subseteq \{\alpha_m^{(1)}\}$ such that $\lim_{m \rightarrow \infty} h_{\alpha_m^{(2)}}(2) \triangleq h(2)$ exists. Similarly, we can continue this argument for $h(3)$ and so on. Finally, let $\{\alpha_m\} = \{\alpha_m^{(m)}\}$, and when $m \rightarrow \infty$, it follows that $\lim_{m \rightarrow \infty} h_{\alpha_m}(1) = h(1)$, $\lim_{m \rightarrow \infty} h_{\alpha_m}(2) = h(2)$, and so on. Therefore, $\lim_{m \rightarrow \infty} h_{\alpha_m}(s) = h(s)$ for any s .

Under a capped reactive policy π , we have $\mathbb{E}_\pi[c(s_n, a_n) | s_1 = s] \leq p$. Then, for any α_m , we obtain

$$\begin{aligned}
C_{\alpha_m}(1) &= \mathbb{E}_\pi \left[\sum_{n=1}^{\infty} \alpha_m^{n-1} c(s_n, Y_n) | s_1 = 1 \right] \\
&\leq \sum_{n=1}^{\infty} \alpha_m^{n-1} p \\
&= \frac{p}{1 - \alpha_m},
\end{aligned} \tag{B.19}$$

which indicates $C_{\alpha_m}(1)$ is bounded. This also implies that both of $C_{\alpha_m}(1)$ and $(1 - \alpha_m)C_{\alpha_m}(1)$ are uniformly bounded for all α_m . Hence, there exists a subsequence $\{\alpha_{\bar{m}}\} \subseteq \{\alpha_m\}$ such

that $\lim_{\bar{m} \rightarrow \infty} (1 - \alpha_{\bar{m}})C_{\alpha_{\bar{m}}}(1) \triangleq g$ exists. Note that $\lim_{\bar{m} \rightarrow \infty} h_{\alpha_{\bar{m}}}(s) = h(s)$ also hold for any s .

Plug $\alpha_{\bar{m}}$ into the Bellman equation Eq. (B.17), we have

$$\begin{aligned} h_{\alpha_{\bar{m}}}(s) + (1 - \alpha_{\bar{m}})C_{\alpha_{\bar{m}}}(1) = \\ \min \left\{ p + 1 + \alpha_{\bar{m}} \sum_{j=1}^{\infty} (1 - \lambda)^{j-1} \lambda h_{\alpha_{\bar{m}}}(j), \right. \\ \left. s + 1 + \alpha_{\bar{m}} \sum_{j=s+1}^{\infty} (1 - \lambda)^{j-s-1} \lambda h_{\alpha_{\bar{m}}}(j) \right\}. \end{aligned} \quad (\text{B.20})$$

By the Lebesgue's Bounded Convergence Theorem [94] and the boundedness of $h_{\alpha_{\bar{m}}}(s)$, it follows that

$$\lim_{\bar{m} \rightarrow \infty} \sum_{j=k}^{\infty} (1 - \lambda)^{j-1} \lambda h_{\alpha_{\bar{m}}}(j) = \sum_{j=k}^{\infty} (1 - \lambda)^{j-1} \lambda h(j) \quad (\text{B.21})$$

for any k . Finally, Eq. (B.4) holds by taking the limit as $\bar{m} \rightarrow \infty$ at both sides of Eq. (B.20).

Step 2): To prove that Eq. (B.4) is the Bellman equation for the average expected cost, we apply the same techniques used in [73, Chapter V, Theorem 2.1]. For the ease of expression, we rewrite Eq. (B.4) as a compact form, i.e.,

$$h(s) + g = \min_{a \in A_s} \left\{ c(s, a) + \sum_{z \in S} p(z \mid s, a) h(z) \right\}. \quad (\text{B.22})$$

First of all, we claim that g is the optimal average expected cost, i.e.,

$$g = \min_{\pi \in \Pi^{R+}} \lim_{N \rightarrow \infty} \mathbb{E}_{\pi} \left[\sum_{n=1}^N c(s_n, a_n) \mid s_1 = s \right] / N. \quad (\text{B.23})$$

To see this, let $H_n \triangleq (s_1, a_1, \dots, s_n, a_n)$ denote the history of the process up to the decision

epoch n . By the iterated expectation, for any decision epoch i under policy π , we have

$$\mathbb{E}_\pi[h(s_i)] = \mathbb{E}_\pi[\mathbb{E}_\pi[h(s_i)|H_{i-1}]], \quad (\text{B.24})$$

which gives

$$\mathbb{E}_\pi \left[\sum_{i=1}^N [h(s_i) - \mathbb{E}_\pi[h(s_i)|H_{i-1}]] \right] = 0. \quad (\text{B.25})$$

Based on the definition of $\mathbb{E}_\pi[h(s_i)|H_{i-1}]$, we have

$$\begin{aligned} & \mathbb{E}_\pi[h(s_i)|H_{i-1}] \\ &= \sum_{z \in S} p(z|s_{i-1}, a_{i-1})h(z) \\ &= c(s_{i-1}, a_{i-1}) + \sum_{z \in S} p(z|s_{i-1}, a_{i-1})h(z) - c(s_{i-1}, a_{i-1}) \\ &\stackrel{(a)}{\geq} \min_{a \in A_s} \left[c(s_{i-1}, a) + \sum_{z \in S} p(z|s_{i-1}, a)h(z) \right] - c(s_{i-1}, a_{i-1}) \\ &\stackrel{(b)}{=} g + h(s_{i-1}) - c(s_{i-1}, a_{i-1}), \end{aligned} \quad (\text{B.26})$$

where (a) becomes equation under the optimal policy π^* , and (b) holds because of Eq. (B.22).

Taking Eq. (B.26) into Eq. (B.25), we obtain

$$0 \leq \mathbb{E}_\pi \left[\sum_{i=1}^N [h(s_i) - g - h(s_{i-1}) + c(s_{i-1}, a_{i-1})] \right], \quad (\text{B.27})$$

which can be rewritten as

$$g \leq \frac{\mathbb{E}_\pi[h(s_n)]}{N} - \frac{\mathbb{E}_\pi[h(s_1)]}{N} + \frac{\mathbb{E}_\pi \left[\sum_{i=1}^N c(s_{i-1}, a_{i-1}) \right]}{N}, \quad (\text{B.28})$$

where the inequality becomes equality under the optimal policy π^* . Taking the limit as

$N \rightarrow \infty$ and using the fact that $h(s)$ is bounded for any s , we have

$$g \leq \lim_{N \rightarrow \infty} \mathbb{E}_\pi \left[\sum_{n=1}^N c(s_n, a_n) | s_1 = s \right] / N, \quad (\text{B.29})$$

with equality for the optimal policy π^* and any initial value s_1 . That is, g is the optimal average expected cost. This also implies that Eq. (B.4) is the Optimality Equation since any $h(s)$ and g satisfying Eq. (B.4) result in the optimal average expected cost g . This completes the proof. $\square$

B.6 Proof of Theorem 4.5

Proof. We start with some additional notations. For the k -th update interval $[u_{k-1}^\pi + 1, u_k^\pi]$, we use N_k and C_k to denote the number of requests that arrive in $[u_{k-1}^\pi + 1, u_k^\pi]$ and the total cost serving these N_k requests, respectively.

Since the request arrival process is Bernoulli, under a threshold-based update policy, the lengths of update intervals are *i.i.d.*, so the update process is a renewal process. By the ergodicity of the process, the average cost can be rewritten as

$$\bar{C}^{\pi(\tau)} = \mathbb{E}[C_k] / \mathbb{E}[N_k]. \quad (\text{B.30})$$

To calculate $\mathbb{E}[N_k]$, we consider the requests that arrive in $[u_{k-1}^\pi + 1, u_k^\pi]$. Apparently, there is only one request in $[u_{k-1}^\pi + \tau, u_k^\pi]$, which arrives exactly at u_k^π because of the threshold-based policy. Besides, the expected number of requests arriving in $[u_{k-1}^\pi + 1, u_{k-1}^\pi + \tau - 1]$ is

$\lambda(\tau - 1)$ according to the Bernoulli process. Therefore, we have

$$\mathbb{E}[N_k] = \lambda(\tau - 1) + 1. \quad (\text{B.31})$$

The total cost in an update interval is composed of an update cost and some staleness costs, i.e.,

$$C_k = p + \sum_{n=1}^{N_k} f(\Delta(r_n)) \mathbb{1}_{\{N_k > 0\}}, \quad (\text{B.32})$$

where $\mathbb{1}_{\{\cdot\}}$ is the indicator function. Here, we slightly abuse the notation of n and use it to denote the index of requests arriving in $[u_{k-1}^\pi + 1, u_k^\pi]$ (i.e., r_n is the arrival time of the n -th request in $[u_{k-1}^\pi + 1, u_k^\pi]$). The staleness costs can be rewritten as

$$\begin{aligned} \sum_{n=1}^{N_k} f(\Delta(r_n)) \mathbb{1}_{\{N_k > 0\}} &= \sum_{n=1}^{\infty} f(\Delta(r_n)) \mathbb{1}_{\{n \leq N_k\}} \\ &= \sum_{n=1}^{\infty} f(\Delta(r_n)) \mathbb{1}_{\{\Delta(r_n) \leq \tau - 1\}}. \end{aligned} \quad (\text{B.33})$$

For the expected staleness cost of the n -th arrival of the requests, we have

$$\mathbb{E}[f(\Delta(r_n)) \mathbb{1}_{\{\Delta(r_n) \leq \tau - 1\}}] = \sum_{t=1}^{\tau-1} f(\Delta(t)) p_n(t), \quad (\text{B.34})$$

where by slightly abusing the notation, we let t denote the index of time-slot after u_{k-1}^π , and let $p_n(t)$ be the probability mass function that the n -th request arrives at time-slot t . Here $p_n(t)$ follows the negative binomial distribution [95], i.e.,

$$p_n(t) = \begin{cases} \binom{t-1}{n-1} \lambda^n (1-\lambda)^{t-n}, & t = n, n+1, \dots; \\ 0, & \text{otherwise.} \end{cases} \quad (\text{B.35})$$

Plugging Eq. (B.35) into Eq. (B.34), we obtain

$$\begin{aligned} & \mathbb{E}[f(\Delta(r_n))\mathbb{1}_{\{\Delta(r_n) \leq \tau-1\}}] \\ &= \sum_{t=1}^{\tau-1} f(\Delta(t)) \binom{t-1}{n-1} \lambda^n (1-\lambda)^{t-n}, \end{aligned} \quad (\text{B.36})$$

where the item in the summation equals 0 when $t < n$.

We rewrite the expected total cost in an update interval as

$$\begin{aligned} \mathbb{E}[C_k] &= p + \sum_{n=1}^{N_k} \mathbb{E}[f(\Delta(r_n))\mathbb{1}_{\{N_k > 0\}}] \\ &= p + \sum_{n=1}^{\infty} \sum_{t=1}^{\tau-1} f(\Delta(t)) \binom{t-1}{n-1} \lambda^n (1-\lambda)^{t-n} \\ &\stackrel{(a)}{=} p + \sum_{t=1}^{\tau-1} \sum_{n=1}^{\infty} f(\Delta(t)) \binom{t-1}{n-1} \lambda^n (1-\lambda)^{t-n} \\ &\stackrel{(b)}{=} p + \sum_{t=1}^{\tau-1} \sum_{n=1}^t f(\Delta(t)) \binom{t-1}{n-1} \lambda^n (1-\lambda)^{t-n} \\ &= p + \sum_{t=1}^{\tau-1} f(\Delta(t)) \lambda \sum_{n=1}^t \binom{t-1}{n-1} \lambda^{n-1} (1-\lambda)^{t-n} \\ &\stackrel{(c)}{=} p + \sum_{t=1}^{\tau-1} f(\Delta(t)) \lambda (\lambda + (1-\lambda))^{t-1} \\ &= p + \sum_{t=1}^{\tau-1} f(\Delta(t)) \lambda \\ &\stackrel{(d)}{=} p + \sum_{t=1}^{\tau-1} f(t) \lambda, \end{aligned} \quad (\text{B.37})$$

where we interchange the order of summation in (a) because the sum is finite, (b) is because the maximal number of requests cannot exceed the length of the update interval, (c) comes from the binomial theorem, and (d) is due to the definition of the AoI. Finally, plugging Eqs. (B.31) and (B.37) into Eq. (B.30) gives Eq. (4.11). $\square$

Appendix C

Proofs of Chapter 5

C.1 Proof of Lemma 5.1

Proof. Our goal is to show that: (i) any feasible solution to Problem (5.3) can be converted to a feasible solution to Problem (5.5), and these two solutions have the same total costs; (ii) any feasible solution to Problem (5.5) can be converted to a feasible solution to Problem (5.3), and the total cost of the converted solution to Problem (5.3) is no greater than the total cost of the solution to Problem (5.5). This implies that any optimal solution to Problem (5.3) is also an optimal solution to Problem (5.5), and vice versa. Therefore, these two problems are equivalent [92, Sec. 4.1.3].

We first show that any feasible solution to Problem (5.3) can be converted to a feasible solution to Problem (5.5). Given a channel state pattern $\mathbf{s}$, we assume that the solution $\pi = \{t_1, t_2, \dots, t_n\}$ is a feasible solution to Problem (5.3), where this solution makes the i -th transmission at the ON slot t_i (i.e., $s(t_i)d(t_i) = 1$), and the total number of transmission is n . We can compute the total cost of the solution π to Problem (5.3) as $C_{5.3}(s, \pi) = cn + (t_1 - 1)t_1/2 + \sum_{i=2}^n (t_i - t_{i-1} - 1)(t_i - t_{i-1})/2 + (T - t_n - 1)(T - t_n)/2$, where cn is the total transmission cost and the rest is the total staleness cost. Based on the solution π to Problem (5.3), we construct a solution π' to Problem (5.5) in the following way: (i) solution π' sends the ACKs at the same time when solution π transmits, i.e., solution π' sends ACKs at a sequence of slots $\{t_1, t_2, \dots, t_n\}$; (ii) based on the ACK decisions in step

(i), for any slot t and any packet $i \leq t$, solution π' lets $z_i(t) = 0$ if $\sum_{\tau=i}^t s(\tau)d(\tau) \geq 1$ and $z_i(t) = 1$ if $\sum_{\tau=i}^t s(\tau)d(\tau) = 0$. We denote the solution π' to Problem (5.5) by $\pi' = \{\{t_1, t_2, \dots, t_n\}, \{\{z_i(t)\}_{i=1}^t\}_{t=1}^T\}$. We can easily verify that solution π' is a feasible solution to Problem (5.5) because both constraints (5.5b) and (5.5c) are satisfied. Furthermore, according to the construction of $z_i(t)$ in step (ii), we know that once the ACK is made at some ON slot $t \in \{t_1, t_2, \dots, t_n\}$, then all previously arrived packets (packet 1 to packet t) are acked forever after slot t , i.e., $z_i(\tau) = 0$ for all $i \leq t$ and all $\tau \geq t$. This indicates that we can compute the total holding cost of the solution π' as

$$\begin{aligned}
& \sum_{t=1}^T \sum_{i=1}^t z_i(t) \\
&= \underbrace{\sum_{t=1}^{t_1-1} \sum_{i=1}^t z_i(t)}_{\text{Total holding cost of packet 1 to packet } t_1} + \sum_{t=t_1}^T \sum_{i=1}^{t_1} z_i(t) \\
& \quad + \underbrace{\sum_{t=t_1+1}^{t_2-1} \sum_{i=t_1+1}^t z_i(t)}_{\text{Total holding cost of packet } (t_1+1) \text{ to packet } t_2} + \sum_{t=t_2}^T \sum_{i=t_1+1}^{t_2} z_i(t) \\
& \quad + \cdots + \underbrace{\sum_{t=t_n+1}^T \sum_{i=t_n+1}^t z_i(t)}_{\text{Total holding cost of packet } (t_n+1) \text{ to packet } T} \\
& \stackrel{(a)}{=} \sum_{t=1}^{t_1-1} \sum_{i=1}^t 1 + \sum_{t=t_1}^T \sum_{i=1}^{t_1} 0 \\
& \quad + \sum_{t=t_1+1}^{t_2-1} \sum_{i=t_1+1}^t 1 + \sum_{t=t_2}^T \sum_{i=t_1+1}^{t_2} 0 \\
& \quad + \cdots + \sum_{t=t_n+1}^T \sum_{i=t_n+1}^t 1 \\
& = (t_1 - 1)t_1/2 + \sum_{i=2}^n (t_i - t_{i-1} - 1)(t_i - t_{i-1})/2 \\
& \quad + (T - t_n - 1)(T - t_n)/2,
\end{aligned} \tag{C.1}$$

where in (a), $z_i(t) = 1$ because packet i is not acked by slot t and $z_i(t) = 0$ otherwise. In addition, the total ACK cost of the solution π' is cn . Therefore, the total cost of the solution π' to Problem (5.5) is $C_{5.5}(s, \pi') = cn + (t_1 - 1)t_1/2 + \sum_{i=2}^n (t_i - t_{i-1} - 1)(t_i - t_{i-1})/2 + (T - t_n - 1)(T - t_n)/2$, which is the same as the total cost of solution π to Problem (5.3). In

summary, any feasible solution π to Problem (5.3) can be converted to a feasible solution π' to Problem (5.5), and those two solutions have the same total cost, i.e., $C_{5.3}(s, \pi) = C_{5.5}(s, \pi')$.

Next, we show that any feasible solution to Problem (5.5) can be converted to a feasible solution to Problem (5.3). Given a channel state pattern $\mathbf{s}$, we assume that the solution $\pi = \{\{t_1, t_2, \dots, t_n\}, \{\{z_i(t)\}_{i=1}^t\}_{t=1}^T\}$ is a feasible solution to Problem (5.5), where this solution makes the i -th ACK at the ON slot t_i (i.e., $s(t_i)d(t_i) = 1$). Though the solution π is a feasible solution to Problem (5.5), it is possible that this solution makes unnecessary cost of $z_i(t)$ (i.e., letting $z_i(t) = 1$ even though $\sum_{\tau=i}^t s(\tau)d(\tau) \geq 1$). In this case, we can always find another feasible solution $\hat{\pi} = \{\{t_1, t_2, \dots, t_n\}, \{\{\hat{z}_i(t)\}_{i=1}^t\}_{t=1}^T\}$ to Problem (5.5) that makes the same ACK decisions as the solution π but never makes the unnecessary cost of $\hat{z}_i(t)$ (i.e., letting $\hat{z}_i(t) = 1$ only when $\sum_{\tau=i}^t s(\tau)d(\tau) = 0$ and letting $\hat{z}_i(t) = 0$ only when $\sum_{\tau=i}^t s(\tau)d(\tau) \geq 1$), and their total cost in Problem (5.5) satisfies $C_{5.5}(\mathbf{s}, \hat{\pi}) \leq C_{5.5}(\mathbf{s}, \pi)$. Similar to the previous analysis (i.e., Eq. (C.1)), the total cost of the solution $\hat{\pi}$ is $C_{5.5}(s, \hat{\pi}) = cn + (t_1 - 1)t_1/2 + \sum_{i=2}^n (t_i - t_{i-1} - 1)(t_i - t_{i-1})/2 + (T - t_n - 1)(T - t_n)/2$. Based on the feasible solution $\hat{\pi}$ to Problem (5.5), we can construct a solution π' to Problem (5.3) in the following way: (i) solution π' transmits at the same time when solution π sends the ACKs, i.e., solution π' transmits at a sequence of slot $\{t_1, t_2, \dots, t_n\}$. We denote the solution π' to Problem (5.3) by $\pi' = \{t_1, t_2, \dots, t_n\}$. We can easily check the solution π' is a feasible solution to Problem (5.3) since constraint (5.3b) is satisfied. In addition, we can compute the total cost of the solution π' to Problem (5.3) as $C_{5.3}(s, \pi') = cn + (t_1 - 1)t_1/2 + \sum_{i=2}^n (t_i - t_{i-1} - 1)(t_i - t_{i-1})/2 + (T - t_n - 1)(T - t_n)/2$, which is the same as the total cost of solution $\hat{\pi}$ to Problem (5.5). In conclusion, any feasible solution π to Problem (5.5) can be converted to a feasible solution π' to Problem (5.3), and their total cost satisfies $C_{5.5}(s, \pi) \geq C_{5.3}(s, \pi')$.

Finally, we show that for any optimal solution of Problem (5.3), it can be converted to

an optimal solution to Problem (5.5), and vice versa. Assuming that the solution π_* is an optimal solution to Problem (5.3). From the above analysis, we can construct a feasible solution π'_* to Problem (5.5), and their total cost satisfies $C_{5.3}(\mathbf{s}, \pi_*) = C_{5.5}(\mathbf{s}, \pi'_*)$. We claim that π'_* is also an optimal solution to Problem (5.5). Otherwise, there must be an optimal solution π''_* to Problem (5.5) such that $\pi''_* \neq \pi'_*$ and $C_{5.5}(\mathbf{s}, \pi'_*) > C_{5.5}(\mathbf{s}, \pi''_*)$. Again, from the previous analysis, we know that the optimal solution π''_* to Problem (5.5) can be converted to a feasible solution $\pi_*^\dagger$ to Problem (5.3), and their total cost satisfies $C_{5.5}(\mathbf{s}, \pi''_*) \geq C_{5.3}(\mathbf{s}, \pi_*^\dagger)$. However, this indicates the solution π_* is not an optimal solution to Problem (5.3) since we have $C_{5.3}(\mathbf{s}, \pi_*^\dagger) < C_{5.3}(\mathbf{s}, \pi_*)$, which contradicts with our assumption. Therefore, the solution π_* is also an optimal solution to Problem (5.5). Similarly, we can show that any optimal solution to Problem (5.5) is also an optimal solution to Problem (5.3). This completes the proof. $\square$

C.2 Proof of Theorem 5.3

Our proof outline is as follows. We first demonstrate that PDOA produces a feasible solution to primal Problem (5.6) and dual Problem (5.7) in Lemma C.1. Then, we explain the usefulness of the primal-dual problem [91] for competitive analysis. Finally, we establish that our online primal-dual-based PDOA is 3-competitive.

To begin with, we first introduce two key observations of PDOA that will be widely used in the proofs.

Observation 9. Assuming that PDOA makes the latest ACK at some ON slot L and the current time slot is t ($t > L$), then at slot t , before the threshold is achieved ($M < 1$), PDOA updates the primal variable $z_i(t)$ and dual variable $y_i(t)$ of packet $(L + 1)$ to packet t ; however, once the threshold is achieved ($M \geq 1$), PDOA only updates the primal variable

$z_i(t)$ of the unacked packets if the channel is OFF at slot t .

Observation 10. Once PDOA makes an ACK at some ON slot t , all the packets arriving no later than slot t (packet 1 to packet t) are acked forever after slot t , and their primal variables and dual variables will never be changed after slot t , i.e., $z_i(\tau) = 0$ and $y_i(\tau) = 0$ for all $i \leq t$ and all $\tau > t$.

With Observations 9 and 10, we can prove the feasibility of the solutions produced by PDOA in the following lemma.

Lemma C.1. *PDOA produces a feasible solution to both primal Problem (5.6) and dual Problem (5.7).*

Proof. The primal constraint (5.6c) and the dual constraint (5.7c) are clearly satisfied. For the primal constraint (5.6b), it is easy to verify that for the i -th packet at slot t ($i \leq t$), if PDOA made an ACK during $[i, t]$, then constraint (5.6b) is satisfied; otherwise, since the i -th packet is not acked by slot t , PDOA will update $z_i(t)$ to be 1, so constraint (5.6b) is also satisfied. When the channels are OFF, the dual constraints (5.7b) are automatically satisfied. Now, consider an ON slot t and its dual constraint (5.7b) $\sum_{i=1}^t \sum_{\tau=t}^T y_i(\tau) \leq c$. This constraint requires that for all the packets arriving no later than slot t (packet 1 to packet t), the sum of their dual variables beyond slot t should not exceed c . Assuming that this ON slot t falls into the k -th ACK interval $[t_k + 1, t_{k+1}]$ of PDOA, i.e., $t_k + 1 \leq t \leq t_{k+1}$, where PDOA makes two ACKs at the ON slot t_{k+1} ($t_{k+1} > t_k + 1$) and the ON slot t_k (when the ON slot t falls into the last ACK interval $[t_K + 1, T]$, where PDOA makes the last ACK at the ON slot t_K , our following analysis can be easily extended to this case). According to Observations 9 and 10, packet 1 to packet t_k are not updated after slot t_k , and packet

$(t_k + 1)$ to packet t are not updated after slot t_{k+1} , then we have

$$\begin{aligned}
& \sum_{i=1}^t \sum_{\tau=t}^T y_i(\tau) \\
&= \sum_{i=1}^{t_k} \sum_{\tau=t}^T y_i(\tau) + \sum_{i=t_k+1}^t \sum_{\tau=t}^{t_{k+1}} y_i(\tau) \\
&\quad + \sum_{i=t_k+1}^t \sum_{\tau=t_{k+1}+1}^T y_i(\tau) \\
&= 0 + \sum_{i=t_k+1}^t \sum_{\tau=t}^{t_{k+1}} y_i(\tau) + 0 \\
&= \sum_{i=t_k+1}^t \sum_{\tau=t}^{t_{k+1}} y_i(\tau).
\end{aligned} \tag{C.2}$$

Next, we discuss the value of $\sum_{i=t_k+1}^t \sum_{\tau=t}^{t_{k+1}} y_i(\tau)$ in two cases: (i) $t = t_k + 1$, and (ii) $t_k + 1 < t \leq t_{k+1}$.

Case i): $t = t_k + 1$. In this case, we have $\sum_{i=t_k+1}^t \sum_{\tau=t}^{t_{k+1}} y_i(\tau) = \sum_{\tau=t_k+1}^{t_{k+1}} y_{t_k+1}(\tau)$. We assume that the threshold is achieved after the updating packet j ($t_k + 1 \leq j \leq t_{k+1}$) at slot $t^\dagger$ ($t_k + 1 < t^\dagger \leq t_{k+1}$), i.e.,

$$\begin{aligned}
& \sum_{\tau=t_k+1}^{t_{k+1}} y_{t_k+1}(\tau) \\
&= \sum_{\tau=t_k+1}^{t^\dagger} y_{t_k+1}(\tau) + \sum_{\tau=t^\dagger+1}^{t_{k+1}} y_{t_k+1}(\tau) \\
&\stackrel{(a)}{=} \sum_{\tau=t_k+1}^{t^\dagger} y_{t_k+1}(\tau) + 0 \\
&\leq \sum_{\tau=t_k+1}^{t^\dagger-1} \sum_{i=t_k+1}^{\tau} y_i(\tau) + \sum_{i=t_k+1}^j y_i(t^\dagger) \\
&= c,
\end{aligned} \tag{C.3}$$

where (a) is because the threshold is achieved at slot $t^\dagger$ and packet $(t_k + 1)$ is never updated after slot $t^\dagger$.

Case ii): $t_k + 1 < t \leq t_{k+1}$. We assume that during the interval $[t, t_{k+1}]$, all packets arriving between $[t_k + 1, t]$ (packet $t_k + 1$ to packet t) make a total number m updates, i.e., $\sum_{i=t_k+1}^t \sum_{\tau=t}^{t_{k+1}} y_i(\tau) \leq m$. Next, we discuss the value of m regarding the value of

the ACK marker M . On the one hand, consider the case where before m increases to $\lceil c \rceil - 1$, the ACK marker M already becomes no smaller than 1. In this case, all the dual variables of packet $t_k + 1$ to packet t are not updated according to Observation 9, and we have $\sum_{i=t_k+1}^t \sum_{\tau=t}^{t_k+1} y_i(\tau) \leq m < \lceil c \rceil - 1 < (c + 1) - 1 = c$. On the other hand, consider the case where before m increases to $\lceil c \rceil - 1$, the ACK marker M is smaller than 1. At the point when $m = \lceil c \rceil - 1$, the increment of the ACK marker M due to those m updates (denoted by $M(m)$) is $M(m) = (\lceil c \rceil - 1)/c$. Now the ACK marker M becomes $M = N' + M(m) \geq 1/c + M(m) = \lceil c \rceil / c \geq 1$, where N' is the increment of the ACK marker M due to packet $(t_k + 1)$ to packet $t - 1$ before slot t (N' is at least $1/c$ since packet $(t_k + 1)$ is not acked at slot $(t_k + 1)$, which increases N' by $1/c$). Given that the ACK marker M now is no smaller than 1, all the dual variables of packet $t_k + 1$ to packet t are not updated according to Observation 9. Therefore, we have $\sum_{i=t_k+1}^t \sum_{\tau=t}^{t_k+1} y_i(\tau) \leq m = \lceil c \rceil - 1 < (c + 1) - 1 = c$.

Thus, we always have $\sum_{i=1}^t \sum_{\tau=t}^T y_i(\tau) \leq c$, and the dual constraint (5.7b) is satisfied. $\square$

The primal-dual problem allows us to analyze the CR of our online algorithm without knowing the optimal offline solution. As shown in Lemma C.1, for a given channel state $\mathbf{s}$, our online algorithm outputs an integer feasible solution (denoted by π) to both primal Problem (5.6) and dual Problem (5.7). We use $P(\mathbf{s}, \pi)$ and $D(\mathbf{s}, \pi)$ to denote the primal objective value and the dual objective value under π , respectively. In addition, this integer solution π is also a feasible solution to Problem (5.5), and we use $C_{5.5}(\mathbf{s}, \pi)$ to denote the objective value of Problem (5.5) under π . Because primal Problem (5.6) and Problem (5.5) have the same objective function, we have $C_{5.5}(\mathbf{s}, \pi) = P(\mathbf{s}, \pi)$. The CR of our online algorithm π for primal Problem (5.6) satisfies

$$\underbrace{\frac{C_{5.5}(\mathbf{s}, \pi)}{OPT_{5.5}(\mathbf{s})}}_{\text{CR of Problem (5.5)}} \leq \underbrace{\frac{P(\mathbf{s}, \pi)}{OPT_{5.6}(\mathbf{s})}}_{\text{CR of primal Problem (5.6)}} \leq \frac{P(\mathbf{s}, \pi)}{D(\mathbf{s}, \pi)}, \quad (\text{C.4})$$

where $OPT_{5.5}(\mathbf{s})$ and $OPT_{5.6}(\mathbf{s})$ is the cost of the optimal offline algorithm for Problem (5.5) and primal Problem (5.6), respectively. Here we have $OPT_{5.5}(\mathbf{s}) \geq OPT_{5.6}(\mathbf{s})$ because the search space of the optimal solution in primal Problem (5.6) is larger than that in Problem (5.5). The second inequality comes from the weak duality [91]. Furthermore, if we can show that there exists a constant β such that $P(\mathbf{s}, \pi)/D(\mathbf{s}, \pi) \leq \beta$ holds for any channel state $\mathbf{s}$, then our online algorithm is β -competitive for primal Problem (5.6) and Problem (5.5).

Now with Lemma C.1, we are ready to show our main result that PDOA is 3-competitive in the following.

Proof of Theorem 5.3. For notational simplicity, let P and D be the value of the objective function of the primal and the dual solutions produced by PDOA under a given channel state $\mathbf{s}$, respectively. In the following, we show that $P/D \leq 3$. We assume that PDOA makes a sequence of ACKs $\pi = \{t_1, t_2, \dots, t_K\}$, where PDOA makes the i -th ACK at the ON slot t_i (i.e., $s(t_i)d(t_i) = 1$). Our goal is to show that for any k -th ($k \in [0, K]$) ACK interval $[t_k + 1, t_{k+1}]$ (where the first ACK interval is $[1, t_1]$ when $k = 0$ and the last ACK interval is $[t_K + 1, T]$ when $k = K$), the ratio between the primal objective value and the dual objective value in this k -th ACK interval (denoted by $P(k)$ and $D(k)$, respectively) is at most 3, i.e., $P(k)/D(k) \leq 3$. According to Observation 10, $P(k)$ and $D(k)$ are never changed when this ACK interval ends at slot t_{k+1} . This implies that PDOA also achieves $P/D \leq 3$ on the entire instance $\mathbf{s}$. This, along with Lemma C.1, concludes that PDOA is 3-competitive based on the weak duality [91].

We first discuss the relation between $P(k)$ and $D(k)$ in the first K ACK interval $[t_k + 1, t_{k+1}]$ (i.e., $k \in [0, K-1]$, where there is always an ACK made at slot t_{k+1}), and then discuss the relation between $P(K)$ and $D(K)$ in the last ACK interval $[t_K + 1, T]$ (i.e., $k = K$, where it is possible that no ACK is made at slot T) in the end.

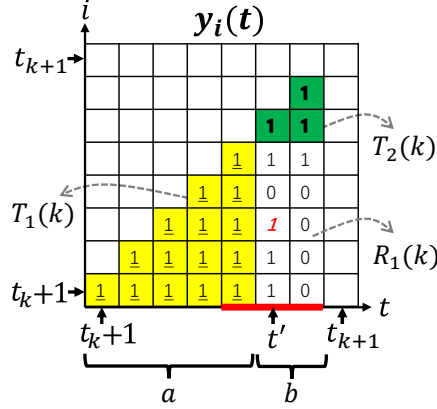


Figure C.1: The updates of primal variables $z_i(t)$ and dual variables $y_i(t)$ in the k -th ACK interval $[t_k + 1, t_{k+1}]$, where channels are OFF during $[t' - 1, t' + 1]$. The red bold italic 1 denotes when the ACK marker equals or is larger than 1. In addition, $T_1(k)$ is an equilateral triangle made of 1 (the underlined 1's with yellow background), $T_2(k)$ is an equilateral triangle made of 1 (the bold 1's with green background), and $R_1(k)$ is a rectangle made of 1 and 0 (the regular 1's and 0's without background).

Consider any k -th ($k \in [0, K - 1]$) ACK interval $[t_k + 1, t_{k+1}]$ (denoted by I_k), where PDOA makes two ACKs at the ON slots t_k and t_{k+1} , respectively. There are two cases when making an ACK at t_{k+1} : 1) the ACK marker M equals or is larger than 1 at t_{k+1} ; 2) the ACK marker M equals or is larger than 1 at some OFF slot t' ($t_k + 1 < t' < t_{k+1}$) and t_{k+1} is the very first ON slot after t' (the channels are OFF during $[t', t_{k+1} - 1]$). Note that $t' \neq t_k + 1$ because the ACK cost $c > 1$. An illustration of Case 2 is provided in Fig. C.1. We emphasize that according to Observation 10, we only need to consider the primal variable update and dual variable updates of packet $(t_k + 1)$ to packet t_{k+1} , since all previous packets (packet 1 to packet t_k) are never updated after slot t_k .

Case 1): The ACK marker M equals or is larger than 1 at t_{k+1} . In this case, Lines 3-7 in PDOA are repeated $\lceil c \rceil$ times, and the total holding cost in I_k is $\sum_{t=t_k+1}^{t_{k+1}} \sum_{i=t_k+1}^t z_i(t) = \lceil c \rceil$ and the total ACK cost in I_k is $\sum_{t=t_k+1}^{t_{k+1}} c \cdot d(t) = c \cdot d(t_{k+1}) = c$. Thus, the primal objective value is $P(k) = \lceil c \rceil + c$. Similarly, the dual objective value is the sum of the dual variables $y_i(t)$ in I_k , which is $D(k) = \sum_{t=t_k+1}^{t_{k+1}} \sum_{i=t_k+1}^t y_i(t) = c$. Therefore, we have $P(k)/D(k) =$

$$(\lceil c \rceil + c)/c \leq (c + 1 + c)/c < 3.$$

Case 2): The ACK marker M equals or is larger than 1 at some OFF slot t' ($t_k + 1 < t' < t_{k+1}$) and t_{k+1} is the very first ON slot after t' . We use $C_A(k)$ to denote the total ACK cost and use $C_H(k)$ to denote the total holding cost in I_k . Here $P(k) = C_A(k) + C_H(k)$. Then

$$\begin{aligned} P(k)/D(k) &= (C_A(k) + C_H(k))/D(k) \\ &\stackrel{(a)}{=} c/D(k) + C_H(k)/D(k) \\ &\stackrel{(b)}{\leq} c/c + C_H(k)/D(k) \\ &\stackrel{(c)}{\leq} c/c + 2\Delta D(k)/D(k) \\ &= 3, \end{aligned} \tag{C.5}$$

where (a) is because PDOA makes only one ACK at slot t_{k+1} during I_k , i.e., $C_A(k) = \sum_{t=t_k+1}^{t_{k+1}} c \cdot d(t) = c \cdot d(t_{k+1}) = c$; (b) is due to the dual objective value $D(k)$ is at least c (i.e., when the ACK marker M equals or is larger than 1, $D(k)$ equals c , and $D(k)$ can be larger than c due to the additional updates of the dual variables in Lines 19-27); and we prove (c) as follows. The total holding cost in I_k is

$$\begin{aligned} C_H(k) &= \sum_{\tau=t_k+1}^{t_{k+1}} \sum_{i=t_k+1}^{\tau} z_i(\tau) \\ &= \sum_{\tau=t_k+1}^{t_{k+1}-1} \sum_{i=t_k+1}^{\tau} z_i(\tau) + \sum_{i=t_k+1}^{t_{k+1}} z_i(t_{k+1}) \\ &\stackrel{(d)}{=} \sum_{\tau=t_k+1}^{t_{k+1}-1} \sum_{i=t_k+1}^{\tau} z_i(\tau) + 0 \\ &= \sum_{\tau=t_k+1}^{t_{k+1}-1} \sum_{i=t_k+1}^{\tau} z_i(\tau) \\ &\stackrel{(e)}{=} \sum_{\tau=t_k+1}^{t_{k+1}-1} \sum_{i=t_k+1}^{\tau} 1 \\ &= (t_{k+1} - t_k - 1)(t_{k+1} - t_k)/2, \end{aligned} \tag{C.6}$$

where in (d), $z_i(t_{k+1}) = 0$ for any $i \in [t_k + 1, t_{k+1}]$ is because all the packets in I_k are acked at slot t_{k+1} ; and in (e), $z_i(\tau) = 1$ for any $\tau \in [t_k + 1, t_{k+1} - 1]$ and $i \in [t_k + 1, \tau]$ is because the packets in I_k are not acked until slot t_{k+1} , and each of them needs to pay a holding cost, i.e., $z_i(\tau) = 1$. Similarly, the dual objective value in I_k can be computed as $D(k) = \sum_{\tau=t_k+1}^{t_{k+1}-1} \sum_{i=t_k+1}^{\tau} y_i(\tau)$. We split $D(k)$ into three parts: the triangle $T_1(k) = \sum_{\tau=t_k+1}^{t'-1} \sum_{i=t_k+1}^{\tau} y_i(\tau)$, the triangle $T_2(k) = \sum_{\tau=t'}^{t_{k+1}-1} \sum_{i=t'}^{\tau} y_i(\tau)$, and the rectangle $R_1(k) = \sum_{\tau=t'}^{t_{k+1}-1} \sum_{i=t_k+1}^{t'-1} y_i(\tau)$ (see an illustration in Fig. C.1). Here $D(k) = T_1(k) + T_2(k) + R_1(k)$.

Our goal is to show that $C_H(k) \leq 2(T_1(k) + T_2(k) + R_1(k)) = 2D(k)$. Here, we can compute $T_1(k) = \sum_{\tau=t_k+1}^{t'-1} \sum_{i=t_k+1}^{\tau} y_i(\tau) = \sum_{\tau=t_k+1}^{t'-1} \sum_{i=t_k+1}^{\tau} 1 = (t' - t_k - 1)(t' - t_k)/2$, where $y_i(\tau) = 1$ for any $\tau \in [t_k + 1, t' - 1]$ and $i \in [t_k + 1, \tau]$ comes from Lines 3-7 in PDOA since the ACK marker M equals or is larger than 1 until t' . In addition, we can compute $T_2(k) = \sum_{\tau=t'}^{t_{k+1}-1} \sum_{i=t'}^{\tau} y_i(\tau) = \sum_{\tau=t'+1}^{t_{k+1}-1} \sum_{i=t'}^{\tau} y_i(\tau) + y_{t'}(t') = \sum_{\tau=t'+1}^{t_{k+1}-1} \sum_{i=t'}^{\tau} 1 + y_{t'}(t') = [(t_{k+1} - t')(t_{k+1} - t' + 1)/2 - 1] + y_{t'}(t')$, where $y_i(\tau) = 1$ for any $\tau \in [t' + 1, t_{k+1} - 1]$ and $i \in [t', \tau]$ comes from Lines 19-27 in PDOA since the channels are OFF during $[t' + 1, t_{k+1} - 1]$. Next, we discuss the value of $T_2(k) + R_1(k)$ based on the value of $y_{t'}(t')$. On the one hand, if $y_{t'}(t') < 1$ (i.e., right after the update of packet t' at slot t' , the ACK marker M becomes no smaller than 1), then we have $R_1(k) = \sum_{\tau=t'}^{t_{k+1}-1} \sum_{i=t_k+1}^{t'-1} y_i(\tau) \geq y_{t_{k+1}}(t') = 1$, where $y_{t_{k+1}}(t') = 1$ because at slot t' , $y_i(t') = 1$ for any $i \in [t_k + 1, t')$ except that $y_{t'}(t') < 1$. In this case, we have $T_2(k) + R_1(k) \geq (t_{k+1} - t')(t_{k+1} - t' + 1)/2$. On the other hand, if $y_{t'}(t') = 1$ (i.e., the ACK marker M becomes no smaller than 1 before the update of packet t' at slot t' , and $y_{t'}(t') = 1$ due to Lines 19-27), we have $T_2(k) + R_1(k) \geq T_2(k) = (t_{k+1} - t')(t_{k+1} - t' + 1)/2$. In both cases, we always have $T_2(k) + R_1(k) \geq (t_{k+1} - t')(t_{k+1} - t' + 1)/2$. Let a be the length of $[t_k + 1, t' - 1]$ (i.e., $a = t' - t_k - 1$) and b be the length of $[t', t_{k+1} - 1]$ (i.e., $b = t_{k+1} - t'$). Now we have $T_1(k) = a(a + 1)/2$, $T_2(k) + R_1(k) \geq b(b + 1)/2$, and $C_H(k) = (a + b)(a + b + 1)/2$.

Clearly, we have

$$\begin{aligned}
& 2D(k) - C_H(k) \\
&= 2(T_1(k) + T_2(k) + R_1(k)) - C_H(k) \\
&\geq 2 \cdot [a(a+1)/2 + b(b+1)/2] - (a+b)(a+b+1)/2 \\
&= [(a-b)^2 + a+b]/2 \\
&\geq 0,
\end{aligned} \tag{C.7}$$

which completes (c) in Eq. (C.5).

In the end, we consider the last time interval $[t_K + 1, T]$ (denoted by I_K). If the last slot is an ON slot and PDOA makes the last ACK exactly at the last slot, i.e., $t_K = T$, then our previous analysis in Cases 1 and 2 still holds. Next, we consider the scenario where $t_K < T$. There are two cases at slot T : 1) T is the slot before the ACK marker M equals or is larger than 1; 2) T is the slot when or after the ACK marker M equals or is larger than 1. In both cases, there is no ACK made during I_K .

Case 1): T is the slot before the ACK marker M equals or is larger than 1. In this case, the total holding cost and the total ACK cost in I_K are $(T - t_K)(T - t_K + 1)/2$ and 0, respectively. Thus, the primal objective value is $P(K) = (T - t_K)(T - t_K + 1)/2 + 0 \cdot c = (T - t_K)(T - t_K + 1)/2$. The dual objective value is the sum of total dual variables $y_i(t)$ in I_{K+1} , which is $(T - t_K)(T - t_K + 1)/2$, i.e., $D(K) = (T - t_K)(T - t_K + 1)/2$. Therefore, we have $P(K)/D(K) = 1$.

Case 2): T is the slot when or after the ACK marker M equals or is larger than 1. We assume that the ACK marker M equals or is larger than 1 at some slot t' ($t_K < t' \leq T$). We claim that the channels are OFF during the interval $[t', T]$. Otherwise, an ACK will be made during $[t', T]$, which contradicts the fact that there is no ACK made during I_K . We

use $C_A(K)$ to denote the total ACK cost and use $C_H(K)$ to denote the total holding cost. Here $P(K) = C_A(K) + C_H(K) = 0 + C_H(K)$, where $C_A(K) = 0$ because there is no ACK made during I_K . We have

$$\begin{aligned}
 P(K)/D(K) &= (C_A(K) + C_H(K))/D(K) \\
 &= 0 + C_H(K)/D(K) \\
 &\stackrel{(a)}{\leq} 2\Delta D(K)/D(K) \\
 &= 2,
 \end{aligned} \tag{C.8}$$

where the analysis in (a) mirrors that of (c) in Eq. (C.5).

In summary, given any channel state $\mathbf{s}$, PDOA achieves $P(k)/D(k) \leq 3$ for any ACK interval I_k ($k \in [0, K]$), and thus PDOA achieves $P/D \leq 3$ on the entire instance $\mathbf{s}$. This, along with Lemma C.1 and leveraging the weak duality [91], confirms that PDOA is 3-competitive. $\square$

C.3 Proof of Lemma 5.6

Our proof outline is as follows. We first show that LAPDOA produces a feasible primal solution and an almost feasible dual solution (with a factor of $c/(c+1)$) in Lemma C.2. Then, we show that the ratio between the total primal objective value (denoted by P) and the total dual objective value (denoted by D) is at most $3/\lambda$, i.e., $P/D \leq 3/\lambda$. Scaling down all dual variables $y_i(t)$ generated by LAPDOA by a multiplicative factor of $c/(c+1)$, we obtain a feasible dual solution with a dual objective value of $(c/(c+1)) \cdot D$. By the weak duality [91], we have $P/OPT \leq P/((c/(c+1)) \cdot D) = ((c+1)/c) \cdot P/D \leq ((c+1)/c) \cdot 3/\lambda$, completing the proof.

To begin with, we introduce two key observations of LAPDOA that will be widely used in the following proofs.

Observation 11. Assuming that LAPDOA made the latest ACK at some ON slot L and the current time slot is t ($t > L$), then at slot t , before the threshold is achieved ($M < 1$), LAPDOA updates the primal variable $z_i(t)$ and the dual variable $y_i(t)$ of packet $(L + 1)$ to packet t ; however, once the threshold is achieved ($M \geq 1$), LAPDOA only updates the primal variable $z_i(t)$ of the unacked packets if the channel is OFF at slot t .

Observation 12. Once LAPDOA makes an ACK at some ON slot t , all the packets arriving no later than slot t (packet 1 to packet t) are acked forever after slot t , and their primal variables and dual variables will never be changed after slot t , i.e., $z_i(\tau) = 0$ and $y_i(\tau) = 0$ for all $i \leq t$ and all $\tau > t$.

Then, we show that LAPDOA gives an almost feasible solution in Lemma C.2.

Lemma C.2. *LAPDOA produces a feasible solution to primal Problem (5.6). In addition, let $\mathbf{y} \triangleq \{\{y_i(t)\}_{i=1}^t\}_{t=1}^T$ be the solution produced by LAPDOA to dual Problem (5.7), then $(c/(c+1))\mathbf{y}$ is a feasible solution to dual Problem (5.7).*

Proof. We omit the proof for primal constraints (5.6b)-(5.6c) and dual constraint (5.7c) because they are similar to the proof in Lemma C.1 and provide the proof for dual constraint (5.7b). Consider an ON slot t and its dual constraint (5.7b) $\sum_{i=1}^t \sum_{\tau=t}^T y_i(\tau) \leq c$. Recall that this dual constraint requires that for all the packets arriving no later than slot t , the sum of their dual variables beyond slot t should not exceed c . We assume that this ON slot t falls into the k -th ACK interval $[t_k+1, t_{k+1}]$, i.e., $t_k+1 \leq t \leq t_{k+1}$, where LAPDOA makes two ACKs at the ON slots t_k and t_{k+1} (in a special case that the ON slot t falls into the last interval $[t_K+1, T]$, i.e., $t_K+1 \leq t \leq T$, where LAPDOA makes the last ACK at the ON slot t_K , our

following analysis can be extended to this case). According to Observations 11 and 12, packet 1 to packet t_k are not updated after slot t_k , and packet $(t_k + 1)$ to packet t are not updated after slot t_{k+1} , similar to Eq. (C.2), we have $\sum_{i=1}^t \sum_{\tau=t}^T y_i(\tau) = \sum_{i=t_k+1}^t \sum_{\tau=t}^{t_{k+1}} y_i(\tau)$. Furthermore, we assume that during the interval $[t, t_{k+1}]$, all packets arriving between $[t_k + 1, t]$ (packet $t_k + 1$ to packet t) make m big updates and n small updates (some zero updates can also be made but we ignore them since they cannot increase the dual variables). In other words, we have $\sum_{i=t_k+1}^t \sum_{\tau=t}^{t_{k+1}} y_i(\tau) = 1 \cdot m + \lambda \cdot n = m + \lambda n$. We claim that if $m + \lambda n \geq c$, then the increment of ACK marker M due to those m big and n small updates (denoted by $M(m, n)$) will be larger than or equal to 1. This is true since we have $M(m, n) = m \cdot (1/\lambda c) + n \cdot (\lambda/c) = m/\lambda c + \lambda n/c \geq m/c + \lambda n/c = (m + \lambda n)/c \geq c/c = 1$. This claim, in turn, implies that $\sum_{i=t_k+1}^t \sum_{\tau=t}^{t_{k+1}} y_i(\tau) = m + \lambda n < c + 1$. To see this, consider the edge case where there are m' big updates and n' small updates made by all packets arriving between $[t_k + 1, t]$ since slot t , and they satisfy: (i) their sum of dual variables is smaller than c , i.e., $m' + \lambda n' < c$; (ii) with one more update (either big or small), the sum of their dual variables is no less than c , i.e., either $(m' + 1) + \lambda n' \geq c$ or $m' + \lambda(n' + 1) \geq c$ holds. From condition (ii) and our claim we know that with the arrival of one more update, the ACK marker M will be larger than or equal to be 1 (because we have $M \geq M(m, n) \geq 1$) at some slot t' ($t' \leq t_{k+1}$). When this happens, the sum of dual variables is at most $\max\{(m' + 1) + \lambda n', m' + \lambda(n' + 1)\} = (m' + 1) + \lambda n' = (m' + \lambda n') + 1 < c + 1$, and those dual variables will never be updated after slot t' according to Observation 11. Therefore, we have $\sum_{i=1}^t \sum_{\tau=t}^T y_i(\tau) = \sum_{i=t_k+1}^t \sum_{\tau=t}^{t_{k+1}} y_i(\tau) < c + 1$. Now scaling down the dual solution $\mathbf{y}$ by a factor of $c/(c + 1)$, we obtain a feasible dual solution $(c/(c + 1))\mathbf{y}$. $\square$

Now with Lemma C.2, we are ready to show our main result that LAPDOA is $((c + 1)/c) \cdot 3/\lambda$ -competitive.

Proof of Lemma 5.6. In the following, we assume that $d(t)$ is updated to the ACK marker M ($M \geq 1$) rather than 1 in Line 13, which possibly makes LAPDOA perform worse (i.e., has a larger total cost since the one-time ACK cost now is $c \cdot M$, which is larger than or equal to $c \cdot 1$). We show that our CR analysis holds for this worse setting (i.e., $d(t) = M$), and thus our CR analysis also holds for LAPDOA. The benefit of considering this worse setting is that this allows us to allocate the ACK costs to large and small updates. Specifically, under the worse setting, suppose that LAPDOA makes an ACK at slot t_k , and after m ($m \geq 0$) big updates and n ($n \geq 0$) small updates, LAPDOA is ready to make another ACK at some slot t_{k+1} . At this point, the ACK marker is $M = m/(\lambda c) + n\lambda/c$, and the ACK cost is $c \cdot M = m/\lambda + n\lambda$. However, instead of calculating the ACK cost at slot t_{k+1} , we can distribute the ACK cost to the updates in $[t_k + 1, t_{k+1}]$, that is, each big update gets an ACK cost of $1/\lambda$ and each small update gets an ACK cost of λ . The total ACK cost of those m big updates and n small updates is still $m/\lambda + n\lambda$. Doing this does not change the ACK cost, but now every big update or small update has a contribution to the ACK cost, which helps our analysis in the following when we compute the primal increment (i.e., the sum of ACK cost and holding cost) of each update.

We assume that LAPDOA makes a sequence of ACKs $\pi = \{t_1, t_2, \dots, t_K\}$, where LAPDOA makes the i -th ACK at the ON slot t_i . Our goal is to show that for any k -th ($k \in [0, K]$) ACK interval $[t_k + 1, t_{k+1}]$ (where the first ACK interval is $[1, t_1]$ when $k = 0$ and the last ACK interval is $[t_K + 1, T]$ when $k = K$), the ratio between the primal objective value and the dual objective value in this k -th ACK interval is at most $3/\lambda$, i.e., $P(k)/D(k) \leq 3/\lambda$. According to Observation 12, when this ACK interval ends at slot t_{k+1} , $P(k)$ and $D(k)$ are never changed. This implies that LAPDOA also achieves $P/D \leq 3/\lambda$ on the entire instance s . This, along with Lemma C.2, concludes that PDOA is $((c + 1)/c) \cdot 3/\lambda$ -competitive based on the weak duality [91].

We first discuss the relation between $P(k)$ and $D(k)$ in the first K ACK interval $[t_k + 1, t_{k+1}]$ (i.e., $k \in [0, K-1]$, where there is always an ACK made at slot t_{k+1}), and then discuss the relation between $P(K)$ and $D(K)$ in the last ACK interval $[t_K + 1, T]$ (i.e., $k = K$, where it is possible that no ACK is made at slot T) in the end.

Consider the k -th ($k \in [0, K-1]$) ACK interval $[t_k + 1, t_{k+1}]$ (denoted by I_k), where LAPDOA makes two ACKs at the ON slots t_k and t_{k+1} (the first ACK interval is the 0-th ACK interval $[t_0 + 1, t_1]$, where $t_0 = 0$ and LAPDOA only makes one ACK at slot t_1), respectively. There are two cases when we make an ACK at slot t_{k+1} : 1) the ACK marker M is equal to or larger than 1 at t_{k+1} ; 2) the ACK marker M equals or is larger than 1 at some OFF slot t' ($t' < t_{k+1}$) and t_{k+1} is the very first ON slot after slot t' (in this case, the channels are OFF during $[t', t_{k+1} - 1]$). We analyze the performance of LAPDOA in these two cases of t_{k+1} .

Case 1): The ACK marker M is equal to or larger than 1 at t_{k+1} . In this case, we do not have zero updates in I_k . Let ΔP and ΔD denote the increment of the primal objective value and the increment of the dual objective value when we make an update, respectively. In the case of a small update, we have $\Delta P = \lambda + 1$ and $\Delta D = \lambda$, that is, $\Delta P / \Delta D = 1 + 1/\lambda$. In the case of a big update, $\Delta P = 1/\lambda + 1$ and $\Delta D = 1$, and we still have $\Delta P / \Delta D = 1 + 1/\lambda$. Obviously, for this k -th ACK interval, we have $P(k)/D(k) = 1 + 1/\lambda$.

Case 2): The ACK marker M equals or is larger than 1 at some OFF slot t' ($t' < t_{k+1}$) and t_{k+1} is the very first ON slot after slot t' . An illustration is shown in Fig. 5.2. We use $C_A(k)$ and $C_H(k)$ to denote the ACK costs and holding costs in I_k , respectively. Here $P(k) = C_A(k) + C_H(k)$. In addition, we assume that there are m ($m \geq 0$) big updates and n ($n \geq 0$) small updates in I_k (some zero updates can also be made but we ignore them since they cannot increase the ACK cost and the dual variable). Given that each big update has an ACK cost of $1/\lambda$ and increases the dual variable by 1, and each small update has

an ACK cost of λ and increases the dual variable by λ , we have $C_A(k) = m/\lambda + \lambda n$ and $D(k) \geq m + \lambda n$ (i.e., $D(k)$ can be larger than $m + \lambda n$ due to the additional updates of dual variables in Lines 22-30). Now we can compute

$$\begin{aligned}
& P(k)/D(k) \\
&= (C_A(k) + C_H(k))/D(k) \\
&= (m/\lambda + \lambda n + C_H(k))/D(k) \\
&\leq (m/\lambda + \lambda n)/(m + \lambda n) + C_H(k)/D(k) \tag{C.9} \\
&\leq 1/\lambda + C_H(k)/D(k) \\
&\stackrel{(a)}{\leq} 1/\lambda + 2/\lambda \\
&= 3/\lambda,
\end{aligned}$$

where (a) is proven in the following. Similar to the analysis of Case-(2) in the proof of Theorem 5.3, we can first compute the total holding cost in I_k as $C_H(k) = \sum_{\tau=t_k+1}^{t_{k+1}-1} \sum_{i=t_k+1}^{\tau} z_i(\tau) = \sum_{\tau=t_k+1}^{t_{k+1}-1} \sum_{i=t_k+1}^{\tau} 1 = (t_{k+1} - t_k - 1)(t_{k+1} - t_k)/2$, where $z_i(\tau) = 1$ for any $\tau \in [t_k + 1, t_{k+1} - 1]$ and $i \in [t_k + 1, \tau]$ is because the packets in I_k are not acked until slot t_{k+1} , and they need to pay a holding cost, i.e., $z_i(\tau) = 1$. Next, we split $C_H(k)$ into three parts under two different cases. Assuming that the ACK marker M is equal to or larger than 1 after the updating of j -th packet at slot t' . There are two cases for packet j : 1) packet j is not packet t' ($j \neq t'$), and 2) packet j is packet t' ($j = t'$). In the first case ($j \neq t'$), we can split $C_H(k)$ into $C_H^1(k) = \sum_{\tau=t_k+1}^{t'-1} \sum_{i=t_k+1}^{\tau} z_i(\tau)$, $C_H^2(k) = \sum_{\tau=t'}^{t_{k+1}-1} \sum_{i=t'}^{\tau} z_i(\tau)$, and $C_H^3(k) = \sum_{\tau=t'}^{t_{k+1}-1} \sum_{i=t_k+1}^{t'-1} z_i(\tau)$ (see an illustration in Fig. C.2). In addition, when $j \neq t'$, we know that the total number of big updates and small updates satisfies $m + n = \sum_{\tau=t_k+1}^{t'-1} \sum_{i=t_k+1}^{\tau} z_i(\tau) + \sum_{i=t_k+1}^j z_i(t') \geq \sum_{\tau=t_k+1}^{t'-1} \sum_{i=t_k+1}^{\tau} z_i(\tau) = C_H^1(k)$. In the second case ($j = t'$), we can split $C_H(k)$ into $C_H^1(k) = \sum_{\tau=t_k+1}^{t'} \sum_{i=t_k+1}^{\tau} z_i(\tau)$, $C_H^2(k) = \sum_{\tau=t'+1}^{t_{k+1}-1} \sum_{i=t'+1}^{\tau} z_i(\tau)$, and $C_H^3(k) = \sum_{\tau=t'+1}^{t_{k+1}-1} \sum_{i=t_k+1}^{t'} z_i(\tau)$ (see an illustration in Fig. C.3).

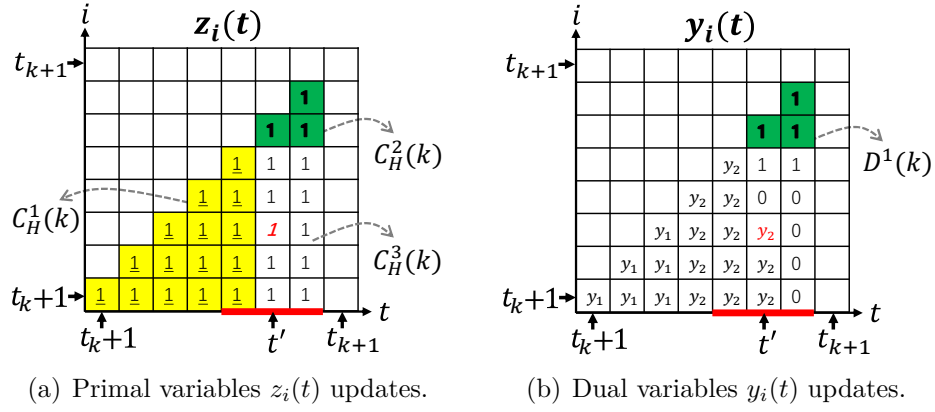


Figure C.2: An illustration of $C_H^1(k)$, $C_H^2(k)$, $C_H^3(k)$ and $D^1(k)$ when $j \neq t'$. $C_H^1(k)$ is an equilateral triangle made of 1 (the underlined 1's with yellow background in Fig. C.2(a)), $C_H^2(k)$ is an equilateral triangle made of 1 (the bold 1's with green background in Fig. C.2(a)), $C_H^3(k)$ is a rectangle made of 1 (the regular 1's without background in Fig. C.2(a)), and $D^1(k)$ is an equilateral triangle made of 1 (the bold 1's with green background in Fig. C.2(b)).

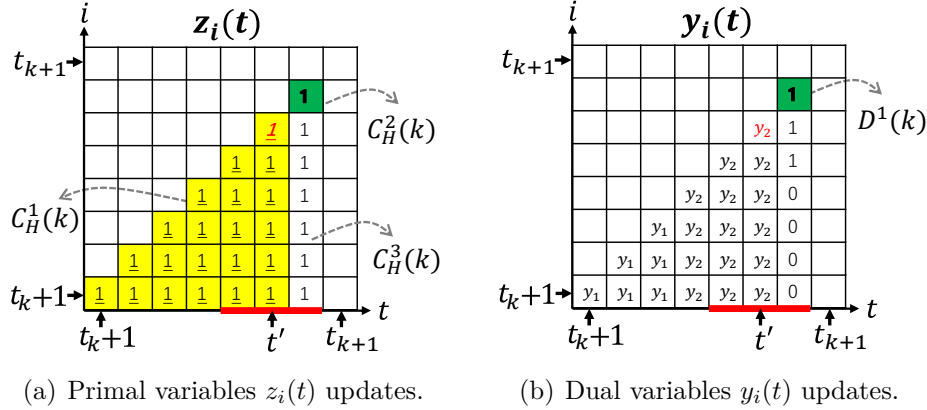


Figure C.3: An illustration of $C_H^1(k)$, $C_H^2(k)$, $C_H^3(k)$ and $D^1(k)$ when $j = t'$. $C_H^1(k)$ is an equilateral triangle made of 1 (the underlined 1's with yellow background in Fig. C.3(a)), $C_H^2(k)$ is an equilateral triangle made of 1 (the bold 1's with green background in Fig. C.3(a)), $C_H^3(k)$ is a rectangle made of 1 (the regular 1's without background in Fig. C.3(a)), and $D^1(k)$ is an equilateral triangle made of 1 (the bold 1's with green background in Fig. C.3(b)).

Furthermore, when $j = t'$, we know that the total number of big updates and small updates satisfies $m + n = \sum_{\tau=t_k+1}^{t'} \sum_{i=t_k+1}^{\tau} z_i(\tau) = C_H^1(k)$. In the following, we only focus on the analysis of $D(k)$ in the first case since the analysis in the second case is very similar. According to Lines 22-30, for the dual variables, we have an equilateral triangle made of 1, which

has the same shape as $C_H^2(k)$, we denote it by $D^1(k)$ (see an illustration in Fig. C.2). We can calculate that $D^1(k) = \sum_{\tau=t'}^{t_{k+1}-1} \sum_{i=t'}^{\tau} y_i(\tau) = \sum_{\tau=t'}^{t_{k+1}-1} \sum_{i=t'}^{\tau} 1 = C_H^2(k)$. Now we can compute

$$\begin{aligned}
C_H(k) &= C_H^1(k) + C_H^2(k) + C_H^3(k) \\
&\stackrel{(a)}{\leq} 2(C_H^1(k) + C_H^2(k)) \\
&\stackrel{(b)}{\leq} 2((m+n) + C_H^2(k)) \\
&= 2((m+n) + D^1(k)) \\
&\leq 2((m+\lambda n)/\lambda + (D^1(k))/\lambda) \\
&= 2/\lambda \cdot (m + \lambda n + D^1(k)) \\
&\stackrel{(c)}{\leq} 2/\lambda \cdot D(k),
\end{aligned} \tag{C.10}$$

where (a) can be proven using the same techniques in Eq. (C.7), (b) is because $C_H^1(k)$ is at least $m+n$ as we analyzed above, and (c) is because $D(k)$ is least $m + \lambda n + D^1(k)$ (i.e., $D(k)$ can be larger than $m + \lambda n + D^1(k)$ due to Lines 22-30). This completes (a) in Eq. (C.9).

In the end, we consider the last time interval $[t_K + 1, T]$ (denoted by I_K). If the last slot is an ON slot and LAPDOA makes the last ACK exactly at the last slot, i.e., $t_K = T$, then our previous analysis in Cases 1 and 2 still holds. Next, we consider the scenario where $t_K < T$. There are two cases at slot T : 1) T is the slot before the ACK marker M equals or is larger than 1; 2) T is the slot when or after the ACK marker M equals or is larger than 1. In both cases, there is no ACK made during I_K . Let $P(K)$ and $D(K)$ be the primal and dual objective value in I_K , respectively.

Case 1): T is the slot before the ACK marker M equals or is larger than 1. In this case, we do not have zero updates in I_K . Let ΔP and ΔD denote the increment of the primal objective value and the increment of the dual objective value when we make an update, respectively. In the case of a small update, we have $\Delta P = c \cdot 0 + 1 = 1$ and $\Delta D = \lambda$, that

is, $\Delta P/\Delta D = 1/\lambda$. In the case of a big update, $\Delta P = c \cdot 0 + 1 = 1$ and $\Delta D = 1$, so we have $\Delta P/\Delta D = 1$. Obviously, for this K -th ACK interval, we have $P(K)/D(K) \leq 1/\lambda$.

Case 2): T is the slot when or after the ACK marker M equals or is larger than 1. We assume that the ACK marker M equals or is larger than 1 at slot t' ($t' > t_K$). According to the definition of Case 2, we have $T \geq t'$. We claim that the channels are OFF during the interval $[t', T]$. Otherwise, an ACK will be made during $[t', T]$, which contradicts the fact that there is no ACK during I_K . We use $C_A(K)$ to denote the total ACK cost and use $C_H(K)$ to denote the total holding cost in I_K . Here $P(K) = C_A(K) + C_H(K) = 0 + C_H(K)$, where $C_A(K) = 0$ because there is no ACK made during I_K . We have

$$\begin{aligned} P(K)/D(K) &= (C_A(K) + C_H(K))/D(K) \\ &= 0 + C_H(K)/D(K) \\ &\stackrel{(a)}{\leq} 2/\lambda, \end{aligned} \tag{C.11}$$

where the analysis in (a) is the same as the (a) in Eq. (C.9).

In summary, LAPDOA achieves $P(k)/D(k) \leq 3/\lambda$ for any ACK interval, and thus LAPDOA also achieves $P/D \leq 3/\lambda$ on the entire instance. $\square$

C.4 Proof of Lemma 5.7

Proof. In this proof, similar to the proof of Lemma 5.6, we still assume that $d(t)$ is updated to the ACK marker M ($M \geq 1$) rather than 1 in Line 13 (except the analysis of big updates in the case of $\lambda \in (0, 1/c]$, where we still update $d(t)$ to be 1). Therefore, for the big updates and small updates in any ACK interval, each big update can be charged with an ACK cost of $1/\lambda$, and each small update can be charged with an ACK cost of λ . In particular, for the

big updates and small updates in the last time interval $[t_K, T]$ (we assume that LAPDOA makes the last ACK at slot t_K), though there is no ACK made during $[t_K, T]$, we still charge each big update and each small update with an ACK cost of $1/\lambda$ and λ , respectively. Doing this can possibly increase the total cost of LAPDOA, but we can show that the upper bound we derived still holds in this case.

We first consider $\lambda \in (0, 1/c]$. In this case, LAPDOA has three types of updates: big updates, small updates, and zero updates. Consider the total cost of big updates first. Once the prediction $\mathcal{P}$ makes an ACK at some ON slot t , LAPDOA will make a big update immediately, the ACK marker becomes $M = 1/\lambda c = 1/c \cdot 1/\lambda \geq 1/c \cdot c = 1$, and thus LAPDOA will also make an ACK at the beginning of slot t . Since the prediction $\mathcal{P}$ has a total number of $C_A(\mathbf{s}, \mathcal{P})/c$ ACKs, then LAPDOA has also $C_A(\mathbf{s}, \mathcal{P})/c$ big updates. Each big update leads to an ACK, which results in an ACK cost of c . Therefore, the total cost of the big updates in LAPDOA is $C_A(\mathbf{s}, \mathcal{P})/c \cdot c = C_A(\mathbf{s}, \mathcal{P})$. By the definition of small update and zero update, each small update or zero update in LAPDOA corresponds to one packet in the prediction $\mathcal{P}$ that has not been acked yet, which requires the prediction $\mathcal{P}$ to pay a holding cost of 1, so the total number of small updates and zero updates is at most $C_H(\mathbf{s}, \mathcal{P})/1 = C_H(\mathbf{s}, \mathcal{P})$. For each of the small updates, the increase in the primal is $\Delta P = \lambda + 1$, and for each of the zero updates, the increase in the primal is $\Delta P = 0 + 1 = 1$, so the total cost of small updates and zero updates is at most $(1 + \lambda)C_H(\mathbf{s}, \mathcal{P})$. In summary, the total cost of big updates, small updates, and zero updates is $C_A(\mathbf{s}, \mathcal{P}) + (1 + \lambda)C_H(\mathbf{s}, \mathcal{P})$. This concludes Eq. (5.12).

Next, we analyze $\lambda \in (1/c, 1]$. We consider two cases: 1) the channels are ON all the time, and 2) there are some OFF channels. We show that Eq. (5.13) holds for both cases.

Case 1): The channels are ON all the time. In this case, LAPDOA will generate only two types of updates: small updates and big updates. By the definition of small updates,

for each of them, there is one corresponding packet in the prediction $\mathcal{P}$ that has not been acked yet, which requires the prediction $\mathcal{P}$ to pay a holding cost of 1 for this packet, so the total number of the small updates is at most $C_H(\mathbf{s}, \mathcal{P})/1 = C_H(\mathbf{s}, \mathcal{P})$. Each small update contributes $(\lambda + 1)$ to the primal objective value, thus the total cost of small updates is at most $(1 + \lambda)C_H(\mathbf{s}, \mathcal{P})$. Next, we analyze the total cost of big updates. We claim that for any ACK made by the prediction $\mathcal{P}$, LAPDOA makes at most $\lceil \lambda c \rceil$ big updates for this ACK. To see this, assuming that prediction $\mathcal{P}$ makes an ACK at slot t , and consider all the big updates due to this ACK (i.e., those big updates produced by the packets in LAPDOA that have not been acked yet and arrives before or at slot t). After at most $\lceil \lambda c \rceil$ such big updates, the ACK marker will become $M = \lceil \lambda c \rceil \cdot 1/\lambda c \geq 1$ at some slot $t' \geq t$. Once the ACK marker M equals or is larger than 1, no more big updates will be made for the ACK made by prediction $\mathcal{P}$ at slot t . Given that prediction $\mathcal{P}$ makes $C_A(\mathbf{s}, \mathcal{P})/c$ ACKs, then LAPDOA makes at most $\lceil \lambda c \rceil \cdot C_A(\mathbf{s}, \mathcal{P})/c$ big updates. For each big update, the increment in the primal objective value is $\Delta P = 1/\lambda + 1$. Therefore, the total cost of big updates is at most $(1/\lambda + 1) \cdot \lceil \lambda c \rceil \cdot C_A(\mathbf{s}, \mathcal{P})/c$. In summary, when the channels are always ON, the total cost of LAPDOA is upper bounded by $C(\mathbf{s}, \mathcal{P}, \lambda) \leq (\lambda + 1)C_H(\mathbf{s}, \mathcal{P}) + (1/\lambda + 1) \lceil \lambda c \rceil C_A(\mathbf{s}, \mathcal{P})/c$, which is smaller than the bound in Eq. (5.13).

Case 2): There are some OFF channels. In this case, LAPDOA will generate three types of updates: small updates, big updates, and zero updates. Note that these zero updates only increase the holding costs. We use $C_{A,s}$ and $C_{H,s}$ to denote the ACK cost and the holding cost of all the small updates, respectively; use $C_{A,b}$ and $C_{H,b}$ to denote the ACK cost and the holding cost of all the big updates, respectively; and $C_{H,z}$ to denote the holding cost of all the zero updates. Obviously, we have $C(\mathbf{s}, \mathcal{P}, \lambda) = C_{A,b} + C_{H,b} + C_{A,s} + C_{H,s} + C_{H,z}$. Furthermore, similar to the analysis in Case 1, for the big updates, we can obtain $C_{A,b} + C_{H,b} \leq (1/\lambda + 1) \cdot \lceil \lambda c \rceil \cdot C_A(\mathbf{s}, \mathcal{P})/c$; and for the small updates, we have $C_{A,s} + C_{H,s} \leq (\lambda + 1)C_H(\mathbf{s}, \mathcal{P})$. To

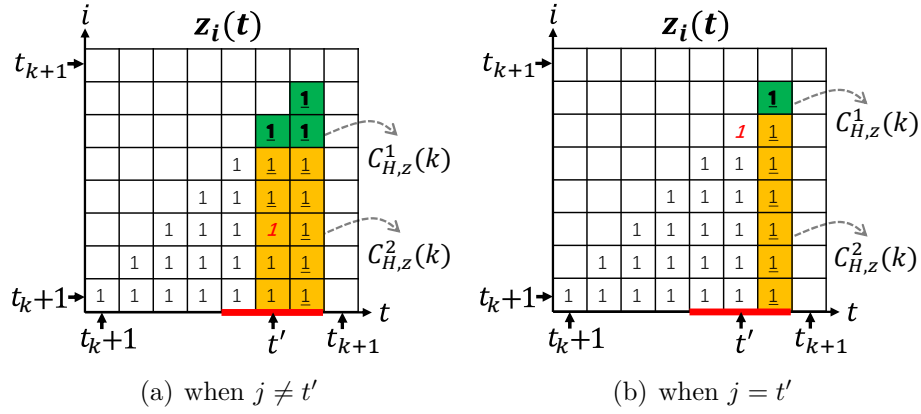


Figure C.4: An illustration of $C_{H,z}(k)$, $C_{H,z}^1(k)$, and $C_{H,z}^2(k)$ in two different cases ($j \neq t'$ and $j = t'$). $C_{H,z}(k)$ is the sum of the underlined 1's, $C_{H,z}^1(k)$ is an equilateral triangle made of 1 (the bold underlined 1's with green background), and $C_{H,z}^2(k)$ is a rectangle made of 1 (the sum of some regular 1's and some underlined 1's with orange background).

analyze the holding costs of zero updates $C_{H,z}$, our idea is to bound it by the holding cost of small updates and big updates, which can be further bounded by the total cost of prediction. To this end, we assume that LAPDOA makes a sequence of ACKs $\pi = \{t_1, t_2, \dots, t_K\}$, where LAPDOA makes the i -th ACK at the ON slot t_i (i.e., $s(t_i)d(t_i) = 1$). Our goal is to show that in any k -th ($k \in [0, K]$) ACK interval $[t_k + 1, t_{k+1}]$ (where the first ACK interval is $[1, t_1]$ when $k = 0$ and the last ACK interval is $[t_K + 1, T]$ when $k = K$), the holding costs of zero updates can be bounded by the holding cost of small updates and big updates.

We consider the k -th ($k \in [0, K - 1]$) ACK interval $[t_k + 1, t_{k+1}]$ (denoted by I_k), where the LAPDOA makes two ACKs at the ON slots t_k and t_{k+1} , respectively. Still, there are two cases when we make an ACK at t_{k+1} : 1) the ACK marker M is equal to or larger than 1 at t_{k+1} and t_{k+1} is an ON slot; 2) the ACK marker M equals or is larger than 1 at some OFF slot t' ($t' < t_{k+1}$) and t_{k+1} is the very first ON slot after slot t' . Note that though the following analysis is for the general ACK interval $[t_k + 1, t_{k+1}]$ ($k \in [0, K - 1]$), they can be easily extended the last ACK interval $[t_K + 1, T]$.

1. The ACK marker M is equal to or larger than 1 at the ON slot t_{k+1} . In this case,

there is no zero update, and the holding cost of zero updates is 0.

2. The ACK marker M equals or is larger than 1 at some OFF slot t' ($t' < t_{k+1}$) and t_{k+1} is the very first ON slot after slot t' (i.e., the channels are OFF during $[t', t_{k+1} - 1]$). Assuming that the ACK marker M is equal to or larger than 1 after the updating of the j -th packet at slot t' . In this case, for the holding costs of zero updates in I_k (denoted by $C_{H,z}(k)$), we can compute it under two different cases based on packet j : 1) packet j is not packet t' ($j \neq t'$), and 2) packet j is packet t' ($j = t'$). In the first case ($j \neq t'$), we can denote $C_{H,z}(k)$ as $C_{H,z}(k) = \sum_{i=j+1}^{t'} z_i(t') + \sum_{\tau=t'+1}^{t_{k+1}-1} \sum_{i=t_k+1}^{\tau} z_i(\tau)$ (see an illustration in Fig. C.4(a)); and in the second case ($j = t'$), we can denote $C_{H,z}(k)$ as $C_{H,z}(k) = \sum_{\tau=t'+1}^{t_{k+1}-1} \sum_{i=t_k+1}^{\tau} z_i(\tau)$ (see an illustration in Fig. C.4(b)). In the following, we only focus on the analysis of $C_{H,z}(k)$ in the first case since the analysis in the second case is very similar. Next, we bound $C_{H,z}(k)$ by two areas: $C_{H,z}^1(k) \triangleq \sum_{\tau=t'}^{t_{k+1}-1} \sum_{i=t'}^{\tau} z_i(\tau)$ and $C_{H,z}^2(k) \triangleq \sum_{\tau=t'}^{t_{k+1}-1} \sum_{i=t_k+1}^{t'-1} z_i(\tau)$ (see an illustration in Fig. C.4(a)). Clearly, we have $C_{H,z}(k) = \sum_{i=j+1}^{t'} z_i(t') + \sum_{\tau=t'+1}^{t_{k+1}-1} \sum_{i=t_k+1}^{\tau} z_i(\tau) \leq \sum_{\tau=t'}^{t_{k+1}-1} \sum_{i=t'}^{\tau} z_i(\tau) + \sum_{\tau=t'}^{t_{k+1}-1} \sum_{i=t_k+1}^{t'-1} z_i(\tau) = C_{H,z}^1(k) + C_{H,z}^2(k)$. We use $C_{H,z}^1$ to denote the sum of $C_{H,z}^1(k)$ over all the ACK intervals I_k ($k \in [0, K]$), i.e., $C_{H,z}^1 \triangleq \sum_{k=0}^K C_{H,z}^1(k)$. For any of the zero updates in $C_{H,z}^1$, there is one corresponding packet in the prediction $\mathcal{P}$ that has not been acked yet since the channels are OFF during some $[t', t_{k+1} - 1]$, which requires the prediction $\mathcal{P}$ to pay a holding cost of 1 for this packet. Similarly, for any of the small updates, by the definition of small updates, there is one corresponding packet in the prediction $\mathcal{P}$ that has not been acked yet, which requires the prediction $\mathcal{P}$ to pay a holding cost of 1 for this packet. Therefore, the total number of the zero updates in $C_{H,z}^1$ and the small updates is at most $C_H(\mathbf{s}, \mathcal{P})/1 = C_H(\mathbf{s}, \mathcal{P})$, and each of such update has a total cost at most $(1 + \lambda)$, which indicates that the total cost of the

zero updates in $C_{H,z}^1$ and the small updates is at most $(1 + \lambda)C_H(\mathbf{s}, \mathcal{P})$, i.e.,

$$C_{A,s} + C_{H,s} + C_{H,z}^1 \leq (1 + \lambda)C_H(\mathbf{s}, \mathcal{P}). \quad (\text{C.12})$$

For the holding cost $C_{H,z}^2(k)$, same as the analysis in (a) of Eq. (C.10), it is upper bounded by the sum of the holding cost of big updates and small updates in I_k and the holding cost of the zero updates in $C_{H,z}^1(k)$, i.e., $C_{H,z}^2(k) \leq C_{H,s}(k) + C_{H,b}(k) + C_{H,z}^1(k)$. More generally, let $C_{H,z}^2$ denote the sum of $C_{H,z}^2(k)$ over all the ACK intervals I_k ($k \in [0, K]$), i.e., $C_{H,z}^2 \triangleq \sum_{k=0}^K C_{H,z}^2(k)$. Then we have

$$\begin{aligned} C_{H,z}^2 &\leq C_{H,s} + C_{H,b} + C_{H,z}^1 \\ &= (C_{H,s} + C_{H,z}^1) + C_{H,b} \\ &\stackrel{(a)}{\leq} C_H(\mathbf{s}, \mathcal{P}) + C_{H,b} \\ &\stackrel{(b)}{\leq} C_H(\mathbf{s}, \mathcal{P}) + \lceil \lambda c \rceil \times C_A(\mathbf{s}, \mathcal{P})/c, \end{aligned}$$

where (a) is because as we showed before, the total number of the zero updates in $C_{H,z}^1$ and the small updates is at most $C_H(\mathbf{s}, \mathcal{P})$, and each of small updates or zero updates increases the holding costs of LAPDOA by 1, so the total holding cost of them is at most $C_H(\mathbf{s}, \mathcal{P})$; (b) is due to the total number of big updates is at most $\lceil \lambda c \rceil \times C_A(\mathbf{s}, \mathcal{P})/c$, and each of big updates increases the holding costs by 1, so we have $C_{H,b} \leq \lceil \lambda c \rceil \times C_A(\mathbf{s}, \mathcal{P})/c$.

In summary, the total cost of LAPDOA in Case 2 is

$$\begin{aligned} C(\mathbf{s}, \mathcal{P}, \lambda) &= C_{A,b} + C_{H,b} + C_{A,s} + C_{H,s} + C_{H,z} \end{aligned}$$

$$\begin{aligned}
&\leq C_{A,b} + C_{H,b} + C_{A,s} + C_{H,s} + C_{H,z}^1 + C_{H,z}^2 \\
&= (C_{A,b} + C_{H,b}) + (C_{A,s} + C_{H,s} + C_{H,z}^1) + C_{H,z}^2 \\
&\leq (1/\lambda + 1) \times \lceil \lambda c \rceil \times C_A(\mathbf{s}, \mathcal{P})/c + (1 + \lambda)C_H(\mathbf{s}, \mathcal{P}) \\
&\quad + C_H(\mathbf{s}, \mathcal{P}) + \lceil \lambda c \rceil \times C_A(\mathbf{s}, \mathcal{P})/c \\
&= (1/\lambda + 2) \times \lceil \lambda c \rceil \times C_A(\mathbf{s}, \mathcal{P})/c + (2 + \lambda)C_H(\mathbf{s}, \mathcal{P}).
\end{aligned}$$

Finally, combining the results in Case 1 and Case 2, we see that Eq. (5.13) holds. $\square$

Bibliography

- [1] S. Kaul, M. Gruteser, V. Rai, and J. Kenney, “Minimizing age of information in vehicular networks,” in *2011 8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, 2011, pp. 350–358.
- [2] S. Kaul, R. Yates, and M. Gruteser, “On piggybacking in vehicular networks,” in *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*, 2011, pp. 1–5.
- [3] S. Kaul, R. Yates, and M. Gruteser, “Real-time status: How often should one update?” in *2012 IEEE INFOCOM*, 2012, pp. 2731–2735.
- [4] A. Kosta, N. Pappas, V. Angelakis *et al.*, “Age of information: A new concept, metric, and tool,” *Foundations and Trends® in Networking*, vol. 12, no. 3, pp. 162–259, 2017.
- [5] Y. Sun, I. Kadota, R. Talak, and E. Modiano, *Age of information: A new metric for information freshness*. Springer Nature, 2022.
- [6] R. D. Yates, Y. Sun, D. R. Brown, S. K. Kaul, E. Modiano, and S. Ulukus, “Age of information: An introduction and survey,” *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 5, pp. 1183–1210, 2021.
- [7] M. Costa, M. Codreanu, and A. Ephremides, “Age of information with packet management,” in *Information Theory (ISIT), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 1583–1587.
- [8] E. Najm and R. Nasser, “Age of information: The gamma awakening,” in *Information Theory (ISIT), 2016 IEEE International Symposium on*. Ieee, 2016, pp. 2574–2578.

- [9] S. K. Kaul, R. D. Yates, and M. Gruteser, “Status updates through queues,” in *Information Sciences and Systems (CISS), 2012 46th Annual Conference on*. IEEE, 2012, pp. 1–6.
- [10] Y. Sang, B. Li, and B. Ji, “The power of waiting for more than one response in minimizing the age-of-information,” in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–6.
- [11] N. Pappas, J. Gunnarsson, L. Kratz, M. Kountouris, and V. Angelakis, “Age of information of multiple sources with queue management,” in *2015 IEEE International Conference on Communications (ICC)*, 2015, pp. 5935–5940.
- [12] R. D. Yates and S. Kaul, “Real-time status updating: Multiple sources,” in *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*. IEEE, 2012, pp. 2666–2670.
- [13] L. Huang and E. Modiano, “Optimizing age-of-information in a multi-class queueing system,” *arXiv preprint arXiv:1504.05103*, 2015.
- [14] 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.
- [15] I. Kadota, A. Sinha, and E. Modiano, “Optimizing age of information in wireless networks with throughput constraints,” in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1844–1852.
- [16] R. Talak, S. Karaman, and E. Modiano, “Optimizing age of information in wireless networks with perfect channel state information,” in *2018 16th International Symposium*

- on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*.
IEEE, 2018, pp. 1–8.
- [17] S. Wu, X. Ren, S. Dey, and L. Shi, “Optimal scheduling of multiple sensors with packet length constraint,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 14 430–14 435, 2017.
- [18] M. Harchol-Balter, *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.
- [19] A. M. Bedewy, Y. Sun, and N. B. Shroff, “Optimizing data freshness, throughput, and delay in multi-server information-update systems,” in *2016 IEEE International Symposium on Information Theory (ISIT)*, 2016, pp. 2569–2573.
- [20] M. E. Crovella, R. Frangioso, and M. Harchol-Balter, “Connection scheduling in web servers,” Boston University Computer Science Department, Tech. Rep., 1999.
- [21] L. Schrage, “A proof of the optimality of the shortest remaining processing time discipline,” *Operations Research*, vol. 16, no. 3, pp. 687–690, 1968.
- [22] 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.
- [23] M. Harchol-Balter, “Queueing disciplines,” *Wiley Encyclopedia of Operations Research and Management Science*, 2010.
- [24] 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.
- [25] M. Moltafet, M. Leinonen, and M. Codreanu, “On the age of information in multi-source queueing models,” *IEEE Transactions on Communications*, vol. 68, no. 8, pp. 5003–5017, 2020.

- [26] C. Kam, S. Kompella, and A. Ephremides, “Effect of message transmission diversity on status age,” in *2014 IEEE International Symposium on Information Theory*, 2014, pp. 2411–2415.
- [27] 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)*, 2018, pp. 124–129.
- [28] 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,” *arXiv preprint arXiv:1804.06139*, 2018.
- [29] R. Talak and E. Modiano, “Age-delay tradeoffs in single server systems,” *arXiv preprint arXiv:1901.04167*, 2019.
- [30] R. Devassy, G. Durisi, G. C. Ferrante, O. Simeone, and E. Uysal-Biyikoglu, “Delay and peak-age violation probability in short-packet transmissions,” in *2018 IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 2471–2475.
- [31] Z. Liu, L. Huang, B. Li, and B. Ji, “Anti-Aging Scheduling in Single-Server Queues: A Systematic and Comparative Study,” *arXiv e-prints*, p. arXiv:2003.04271, Oct. 2020.
- [32] 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.
- [33] C. Kam, S. Kompella, and A. Ephremides, “Age of information under random updates,” in *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 66–70.

- [34] M. Shaked and J. G. Shanthikumar, *Stochastic orders*. Springer Science & Business Media, 2007.
- [35] “Apache storm,” <http://storm.apache.org>.
- [36] A. Labrinidis and N. Roussopoulos, “Exploring the tradeoff between performance and data freshness in database-driven web servers,” *The VLDB Journal—The International Journal on Very Large Data Bases*, vol. 13, no. 3, pp. 240–255, 2004.
- [37] H. Qu, A. Labrinidis, and D. Mosse, “Unit: User-centric transaction management in web-database systems,” in *Data Engineering, 2006. ICDE’06. Proceedings of the 22nd International Conference on*. IEEE, 2006, pp. 33–33.
- [38] H. Qu and A. Labrinidis, “Preference-aware query and update scheduling in web-databases,” in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. IEEE, 2007, pp. 356–365.
- [39] M. Harchol-Balter, M. Crovella, and C. Murta, “To queue or not to queue?: When fcfs is better than ps in a distributed system,” MIT, Tech. Rep., 1997.
- [40] M. Harchol-Balter, M. Crovella, and S. Park, “The case for srpt scheduling in web servers,” MIT, Tech. Rep., 1998.
- [41] M. Harchol-Balter, B. Schroeder, N. Bansal, and M. Agrawal, “Size-based scheduling to improve web performance,” *ACM Transactions on Computer Systems (TOCS)*, vol. 21, no. 2, pp. 207–233, 2003.
- [42] T. Osogami, M. Harchol-Balter, and A. Scheller-Wolf, “Robustness and performance of threshold-based resource allocation policies,” Working paper, Tech. Rep., 2005.

- [43] I. Kadota, A. Sinha, E. Uysal-Biyikoglu, R. Singh, and E. Modiano, “Scheduling policies for minimizing age of information in broadcast wireless networks,” *arXiv preprint arXiv:1801.01803*, 2018.
- [44] N. Lu, B. Ji, and B. Li, “Age-based scheduling: Improving data freshness for wireless real-time traffic,” in *Proceedings of the Eighteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*. ACM, 2018, pp. 191–200.
- [45] R. D. Yates, “Lazy is timely: Status updates by an energy harvesting source,” in *Information Theory (ISIT), 2015 IEEE International Symposium on*. IEEE, 2015, pp. 3008–3012.
- [46] Y. Sun, E. Uysal-Biyikoglu, R. D. Yates, C. E. Koksal, and N. B. Shroff, “Update or wait: How to keep your data fresh,” *IEEE Transactions on Information Theory*, vol. 63, no. 11, pp. 7492–7508, 2017.
- [47] M. Patra, A. Sengupta, and C. S. R. Murthy, “On minimizing the system information age in vehicular ad-hoc networks via efficient scheduling and piggybacking,” *Wireless Networks*, vol. 22, no. 5, pp. 1625–1639, 2016.
- [48] X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. Bowers *et al.*, “Practical lessons from predicting clicks on ads at facebook,” in *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*. ACM, 2014, pp. 1–9.
- [49] O. J. Boxma, G. Koole, and I. Mitrani, “A two-queue polling model with a threshold service policy,” in *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 1995. MASCOTS’95., Proceedings of the Third International Workshop on*. IEEE, 1995, pp. 84–88.

- [50] E. Gelenbe and I. Mitrani, *Analysis and Synthesis of Computer Systems*, 2nd ed. London, UK: Imperial College Press, 2010.
- [51] L. Takács, “Two queues attended by a single server,” *Operations Research*, vol. 16, no. 3, pp. 639–650, 1968.
- [52] Waze mobile app. [Online]. Available: <https://www.waze.com/>
- [53] Gasbuddy mobile app. [Online]. Available: <https://www.gasbuddy.com/>
- [54] B. Li and J. Liu, “Achieving information freshness with selfish and rational users in mobile crowd-learning,” *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 5, pp. 1266–1276, 2021.
- [55] F. Li, Y. Sang, Z. Liu, B. Li, H. Wu, and B. Ji, “Waiting but not aging: Optimizing information freshness under the pull model,” *IEEE/ACM Transactions on Networking*, vol. 29, no. 1, pp. 465–478, 2021.
- [56] Z. Liu and B. Ji, “Towards the tradeoff between service performance and information freshness,” in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.
- [57] Y. Ling and J. Mi, “An optimal trade-off between content freshness and refresh cost,” *Journal of Applied Probability*, vol. 41, no. 3, p. 721–734, 2004.
- [58] Market data from morgan stanley. [Online]. Available: <https://us.etrade.com/l/f/disclosure-library/market-data>
- [59] K. Chen and L. Huang, “Age-of-information in the presence of error,” in *2016 IEEE International Symposium on Information Theory (ISIT)*, 2016, pp. 2579–2583.

- [60] I. Kadota, E. Uysal-Biyikoglu, R. Singh, and E. Modiano, “Minimizing the age of information in broadcast wireless networks,” in *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2016, pp. 844–851.
- [61] J. Yun, C. Joo, and A. Eryilmaz, “Optimal real-time monitoring of an information source under communication costs,” in *2018 IEEE Conference on Decision and Control (CDC)*, 2018, pp. 4767–4772.
- [62] V. Tripathi and E. Modiano, “An online learning approach to optimizing time-varying costs of aoi,” in *Proceedings of the Twenty-second International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, 2021, pp. 241–250.
- [63] K. Saurav and R. Vaze, “Minimizing the sum of age of information and transmission cost under stochastic arrival model,” in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.
- [64] R. D. Yates, P. Ciblat, A. Yener, and M. Wigger, “Age-optimal constrained cache updating,” in *2017 IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 141–145.
- [65] J. Zhong, R. D. Yates, and E. Soljanin, “Two freshness metrics for local cache refresh,” in *2018 IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 1924–1928.
- [66] M. Bastopcu and S. Ulukus, “Cache freshness in information updating systems,” in *2021 55th Annual Conference on Information Sciences and Systems (CISS)*, 2021, pp. 01–06.
- [67] H. Tang, P. Ciblat, J. Wang, M. Wigger, and R. D. Yates, “Cache updating strategy

- minimizing the age of information with time-varying files' popularities," in *2020 IEEE Information Theory Workshop (ITW)*, 2021, pp. 1–5.
- [68] A. Arafa, J. Yang, and S. Ulukus, "Age-minimal online policies for energy harvesting sensors with random battery recharges," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.
- [69] X. Wu, J. Yang, and J. Wu, "Optimal status update for age of information minimization with an energy harvesting source," *IEEE Transactions on Green Communications and Networking*, vol. 2, no. 1, pp. 193–204, 2018.
- [70] B. T. Bacinoglu, Y. Sun, E. Uysal-Bivikoglu, and V. Mutlu, "Achieving the age-energy tradeoff with a finite-battery energy harvesting source," in *2018 IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 876–880.
- [71] Y. Dong, P. Fan, and K. B. Letaief, "Energy harvesting powered sensing in iot: Timeliness versus distortion," *IEEE Internet of Things Journal*, vol. 7, no. 11, pp. 10 897–10 911, 2020.
- [72] V. Tripathi, R. Talak, and E. Modiano, "Age of information for discrete time queues," *arXiv preprint arXiv:1901.10463*, 2019.
- [73] S. M. Ross, *Introduction to stochastic dynamic programming*. Academic press, 2014.
- [74] J. Yang, Y. Yue, and K. V. Rashmi, "A large scale analysis of hundreds of in-memory cache clusters at twitter," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 191–208.
- [75] S. Li, L. D. Xu, and S. Zhao, "The internet of things: a survey," *Information systems frontiers*, vol. 17, pp. 243–259, 2015.

- [76] F. Wu, C. Rüdiger, and M. R. Yuce, “Real-time performance of a self-powered environmental iot sensor network system,” *Sensors*, vol. 17, no. 2, p. 282, 2017.
- [77] X. Cao, J. Wang, Y. Cheng, and J. Jin, “Optimal sleep scheduling for energy-efficient aoi optimization in industrial internet of things,” *IEEE Internet of Things Journal*, vol. 10, no. 11, pp. 9662–9674, 2023.
- [78] B. Yu, Y. Cai, X. Diao, and K. Cheng, “Adaptive packet length adjustment for minimizing age of information over fading channels,” *IEEE Transactions on Wireless Communications*, pp. 1–1, 2023.
- [79] Y.-H. Tseng and Y.-P. Hsu, “Online energy-efficient scheduling for timely information downloads in mobile networks,” in *2019 ISIT*, 2019, pp. 1022–1026.
- [80] A. R. Karlin, C. Kenyon, and D. Randall, “Dynamic tcp acknowledgement and other stories about $e/(e-1)$,” in *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, ser. STOC ’01. New York, NY, USA: Association for Computing Machinery, 2001, p. 502–509.
- [81] E. Fountoulakis, N. Pappas, M. Codreanu, and A. Ephremides, “Optimal sampling cost in wireless networks with age of information constraints,” in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2020, pp. 918–923.
- [82] Z. Liu, B. Li, Z. Zheng, Y. T. Hou, and B. Ji, “Towards optimal tradeoff between data freshness and update cost in information-update systems,” *IEEE Internet of Things Journal*, pp. 1–1, 2023.
- [83] A. M. Bedewy, Y. Sun, S. Kompella, and N. B. Shroff, “Optimal sampling and scheduling

- for timely status updates in multi-source networks,” *IEEE TIT*, vol. 67, no. 6, pp. 4019–4034, 2021.
- [84] A. Sinha and R. Bhattacharjee, “Optimizing age-of-information in adversarial and stochastic environments,” *IEEE Transactions on Information Theory*, vol. 68, no. 10, pp. 6860–6880, 2022.
- [85] S. Banerjee and S. Ulukus, “Age of information in the presence of an adversary,” in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2022, pp. 1–8.
- [86] S. Li, C. Li, Y. Huang, B. A. Jalaian, Y. T. Hou, and W. Lou, “Enhancing resilience in mobile edge computing under processing uncertainty,” *IEEE JSAC*, vol. 41, no. 3, pp. 659–674, 2023.
- [87] T. Lykouris and S. Vassilvitskii, “Competitive caching with machine learned advice,” *J. ACM*, vol. 68, no. 4, jul 2021.
- [88] M. Purohit, Z. Svitkina, and R. Kumar, “Improving online algorithms via ml predictions,” *Advances in Neural Information Processing Systems*, vol. 31, 2018.
- [89] E. Bamas, A. Maggiori, and O. Svensson, “The primal-dual method for learning augmented algorithms,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 20 083–20 094, 2020.
- [90] D. Rutten, N. Christianson, D. Mukherjee, and A. Wierman, “Smoothed online optimization with unreliable predictions,” *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 7, no. 1, mar 2023.
- [91] N. Buchbinder, J. S. Naor *et al.*, “The design of competitive online algorithms via a

- primal–dual approach,” *Foundations and Trends® in Theoretical Computer Science*, vol. 3, no. 2–3, pp. 93–263, 2009.
- [92] S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.
- [93] A. Narayanan, E. Ramadan, R. Mehta, X. Hu, Q. Liu, R. A. Fezeu, U. K. Dayalan, S. Verma, P. Ji, T. Li *et al.*, “Lumos5g: Mapping and predicting commercial mmwave 5g throughput,” in *Proceedings of the ACM Internet Measurement Conference*, 2020, pp. 176–193.
- [94] H. L. Royden and P. Fitzpatrick, *Real analysis*. Macmillan New York, 1988, vol. 32.
- [95] S. M. Ross, *Introduction to probability models*. Academic press, 2014.