Optimizing Information Freshness in Wireless Networks

Chengzhang Li

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 Engineering

Y. Thomas Hou, Chair
Atilla Eryilmaz
Bo Ji
Wenjing Lou
Jeffrey H. Reed

December 8, 2022
Blacksburg, Virginia

Keywords: Information freshness, latency, age of information, wireless communications, data collection, 5G/NextG, Internet of Things, optimization, algorithm design

Optimizing Information Freshness in Wireless Networks

Chengzhang Li

(ABSTRACT)

Age of Information (AoI) is a performance metric that can be used to measure the freshness of information. Since its inception, it has captured the attention of the research community and is now an area of active research. By its definition, AoI measures the elapsed time period between the present time and the generation time of the information. AoI is fundamentally different from traditional metrics such as delay or latency as the latter only considers the transit time for a packet to traverse the network.

Among the state-of-the-art in the literature, we identify two limitations that deserve further investigation. First, many existing efforts on AoI have been limited to information-theoretic exploration by considering extremely simple models and unrealistic assumptions, which are far from real-world communication systems. Second, among most existing work on scheduling algorithms to optimize AoI, there is a lack of research on guaranteeing AoI deadlines. The goal of this dissertation is to address these two limitations in the state-of-the-art. First, we design schedulers to minimize AoI under more practical settings, including varying sampling periods, varying sample sizes, cellular transmission models, dynamic channel conditions, etc. Second, we design schedulers to guarantee hard or soft AoI deadlines for each information source. More important, inspired by our results from guaranteeing AoI deadlines, we develop a general design framework that can be applied to construct high-performance schedulers for AoI-related problems.

This dissertation is organized into three parts. In the first part, we study two problems on AoI minimization under general settings. (i) We consider general and heterogeneous sampling behaviors among source nodes, varying sample size, and a cellular-based transmis-

sion model. We develop a near-optimal low-complexity scheduler—code-named Juventas—to minimize AoI. (ii) We study the AoI minimization problem under a 5G network with dynamic channels. To meet the stringent real-time requirement for 5G, we develop a GPU-based near-optimal algorithm—code-named Kronos—and implement it on commercial off-the-shelf (COTS) GPUs.

In the second part, we investigate three problems on guaranteeing AoI deadlines. (i) We study the problem to guarantee a *hard* AoI deadline for information from each source. We present a novel low-complexity procedure, called *Fictitious Polynomial Mapping (FPM)*, and prove that FPM can find a feasible scheduler for any hard deadline vector when the system load is under $\ln 2$. (ii) For *soft* AoI deadlines, i.e., occasional violations can be tolerated, we present a novel procedure called *Unstable Tolerant Scheduler* (UTS). UTS hinges upon the notions of *Almost Uniform Schedulers* (AUSs) and *step-down* rate vectors. We show that UTS has strong performance guarantees under different settings. (iii) We investigate a 5G scheduling problem to minimize the proportion of time when the AoI exceeds a soft deadline. We derive a property called *uniform fairness* and use it as a guideline to develop a 5G scheduler—Aequitas. To meet the real-time requirement in 5G, we implement Aequitas on a COTS GPU.

In the third part, we present Eywa—a general design framework that can be applied to construct high-performance schedulers for AoI-related optimization and decision problems. The design of Eywa is inspired by the notions of AUS schedulers and step-down rate vectors when we develop UTS in the second part. To validate the efficacy of the proposed Eywa framework, we apply it to solve a number of problems, such as minimizing the sum of AoIs, minimizing bandwidth requirement under AoI constraints, and determining the existence of feasible schedulers to satisfy AoI constraints. We find that for each problem, Eywa can either offer a stronger performance guarantee than the state-of-the-art algorithms, or provide

new/general results that are not available in the literature.

Optimizing Information Freshness in Wireless Networks

Chengzhang Li

(GENERAL AUDIENCE ABSTRACT)

Age of Information (AoI) is a performance metric that can be used to measure the freshness of information. It measures the elapsed time period between the present time and the generation time of the information. Through a literature review, we have identified two limitations: (i) many existing efforts on AoI have employed extremely simple models and unrealistic assumptions, and (ii) most existing work focuses on optimizing AoI, while overlooking AoI deadline requirements in some applications.

The goal of this dissertation is to address these two limitations. For the first limitation, we study the problem to minimize the average AoI in general and practical settings, such as dynamic channels and 5G NR networks. For the second limitation, we design schedulers to guarantee hard or soft AoI deadlines for information from each source. Finally, we develop a general design framework that can be applied to construct high-performance schedulers for AoI-related problems.

Dedication

To my beloved parents, Hongbo Yu and Jun Li.

Acknowledgments

First of all, I would like to express my most sincere gratitude to my Ph.D. advisor, Prof. Tom Hou, the Bradley Distinguished Professor of Electrical and Computer Engineering at Virginia Tech (VT), for his support, confidence in me, and hands-on guidance throughout my Ph.D. I still remember in my first year, when I was struggling to find a research topic, Prof. Hou introduced me to the field of age of information (AoI), which I have worked on for five years and has led to this dissertation. Throughout my five years in the Complex Networks and Security Research (CNSR) Lab, Prof. Hou spent numerous hours with me to develop new research problems, design algorithms, review my proposed solutions, and edit my draft papers word-by-word. During this process, he helped me strengthen my perspective on research career and publish high-quality papers. Prof. Hou also supported me to attend a number of academic conferences to meet with top researchers. He encouraged me to have a summer internship at NVIDIA Corp. to gain hands-on experience in industry. To better prepare me for an academic position, he strongly recommended me for a postdoc position with Prof. Ness Shroff at Ohio State University (OSU). During this pandemic, he created a safe workspace for all his students and kept reminding us to take precautions to stay safe. As a result, I was never infected with COVID while I was working on my research from 2020 to 2022. His devotion to research excellence and his genuine care for his students serve as a role model for my future work and life.

My deepest gratitude also extends to my Ph.D. committee members: Prof. Atilla Eryilmaz (OSU), Prof. Bo Ji, Prof. Wenjing Lou, and Prof. Jeff Reed, for their valuable time and feedback on this dissertation. Their advice helped me improve the quality of this dissertation in many ways.

I would also like to thank all my current and former colleagues in the CNSR Lab: Dr. Yan Huang, Dr. Yongce Chen, Dr. Shaoran Li, Yubo Wu, Dr. Qingyu Liu, Lei Li, Naru Jai, Shiva Acharya, Shabnam Wahed, and Heng Jin. Our group's alumni, Yan and Yongce, gave me much guidance and help since I joined VT, for which I am very grateful. I shared the same apartment with Shaoran and Yubo during the pandemic. We lived together, worked together, and played together. It was wonderful to have them on my side. I worked closely with Qingyu since he joined our lab as a postdoc. His talent and devotion to research made our collaboration fruitful and enjoyable. I also treasure the friendship with Lei, Naru, Shiva, Shabnam and Heng. It was a pleasure working with them.

Finally, and most importantly, I want to thank my parents, Hongbo Yu and Jun Li, for their unconditional love and support throughout my life. They gave birth to me, raised me up, and constantly supported me to pursue my dreams. Whenever I talk with them, I can always feel their encouragements, which renew my strength. Without them, I wouldn't become the person I am now. I hope this dissertation will make them proud.

Funding Acknowledgments

This research was supported in part by ONR MURI Grant N00014-19-1-2621, Virginia Commonwealth Cyber Initiative (CCI), and Virginia Tech Institute for Critical Technology and Applied Science (ICTAS).

Contents

List of Figures xviii

List of Tables xxi

1 Introduction 1

1.1 Motivation 1

1.2 Main Contributions 3

2 Minimizing Age of Information under General Models for IoT Data Collection 7

2.1 Introduction 7

2.2 Related Work 10

2.3 System Model 11

2.4 AoI Modeling and Problem Statement 14

2.5 Properties for An Optimal Scheduling Algorithm 17

2.5.1 An Order-based Scheduling 17

2.5.2 Cyclic Transmission 19

2.5.3 Complexity Analysis 21

2.6 Performance Bounds 21

2.6.1 The Case of Per Time Slot Sampling Under Finite Link Capacity . . 22
2.6.2 The Case of Arbitrary Sampling Under Infinite link Capacity 25
2.6.3 The Case of Arbitrary Sampling Under Finite Link Capacity 26
2.6.4 The Case of Periodic Sampling Under Finite Link Capacity 27
2.7 Juventas: A Near-Optimal Scheduler 32
2.8 Simulation Results 35
2.8.1 Per Time Slot Sampling 35
2.8.2 Periodic Sampling 37
2.8.3 Random Sampling 40
2.8.4 Synchronization for Periodic Sampling 45
2.9 Chapter Summary 46

3 Minimizing AoI in a 5G-based IoT Network under Varying Channel Conditions 47

3.1 Introduction 47
3.2 A 5G-based IoT Architecture 49
3.3 Modeling and Problem Statement 51
3.3.1 AoI Notation 51
3.3.2 Uplink Transmission 53
3.3.3 Problem Statement and Technical Challenges 56
3.4 Performance Bound 57

3.4.1 A Lower Bound For Known $\overline{R}_i$'s . . . 57
3.4.2 Finding A Lower Bound of $\bar{A}^{\mathrm{B}}$ . . . 62
3.5 Kronos: A Real-Time Scheduler . . . 68
3.5.1 Basic Idea . . . 68
3.5.2 Algorithm Details: Design of Scheduling Metric . . . 70
3.5.3 Running Time Speedup: A GPU-based Implementation . . . 74
3.6 Performance Evaluation . . . 78
3.6.1 Experiment Setup and Parameter Settings . . . 78
3.6.2 Results . . . 79
3.7 Chapter Summary . . . 90

4 AoI Scheduling with Hard Deadlines 92

4.1 Introduction . . . 92
4.2 System Model . . . 94
4.3 Problem Statement . . . 96
4.4 Schedulability Check with A Cyclic Scheduler . . . 97
4.4.1 Existence of A Feasible Cyclic Scheduler . . . 97
4.4.2 Detection of Feasible Cyclic Scheduler . . . 100
4.5 The Special Case of Polynomial Deadline Vectors . . . 102
4.5.1 Polynomial Deadline Vectors and System Load . . . 103

4.5.2 Scheduling for Polynomial Deadline Vectors . . . 104
4.6 Scheduling for General Deadline Vectors . . . 107
4.6.1 Basic Idea . . . 107
4.6.2 Scheduling for Fictitious Polynomial deadline vectors . . . 109
4.6.3 Mapping a General deadline vector to a Fictitious Polynomial Vector 113
4.7 System Load vs. Schedulability . . . 116
4.8 Numerical Results . . . 120
4.8.1 Feasibility Check . . . 120
4.8.2 Compare to EDF: Small N . . . 123
4.8.3 Compare to EDF: Large N . . . 125
4.8.4 FPM under different N . . . 126
4.8.5 FPM under different deadline distribution . . . 127
4.9 Chapter Summary . . . 128

5 AoI Scheduling with Soft Deadlines 130

5.1 Introduction . . . 130
5.2 System Model and Problem Statement . . . 131
5.3 System Load . . . 135
5.4 The Stable Tolerant Case . . . 136
5.4.1 Almost Uniform Scheduler . . . 136

5.4.2 Finding AUS for a Special Case . . . 138
5.4.3 Finding AUS for General Case . . . 143
5.5 From Stable Tolerant to Unstable Tolerant . . . 152
5.6 Numerical Results . . . 158
5.6.1 Case Study . . . 158
5.6.2 Impact of Tolerance Rate . . . 161
5.6.3 Impact of Packet Loss Rate . . . 162
5.7 Chapter Summary . . . 163

6 Scheduling with Soft AoI Deadlines in 5G Networks 165

6.1 Introduction . . . 165
6.2 System Model and Problem Statement . . . 167
6.2.1 System Model . . . 167
6.2.2 AoI Notation . . . 170
6.2.3 Problem Statement . . . 171
6.3 Performance Bound . . . 173
6.3.1 A New Objective Function for Lower Bound . . . 173
6.3.2 Reformulation and Relaxation . . . 175
6.3.3 Solving OPT-LB-α . . . 176
6.4 Algorithm Design . . . 178

6.4.1 Uniform Fairness . . . 179
6.4.2 Main Ideas . . . 181
6.4.3 Design Details . . . 182
6.5 Performance Evaluation . . . 188
6.5.1 Experiment Setup . . . 189
6.5.2 A Case Study . . . 189
6.5.3 Varying Parameters . . . 190
6.6 Chapter Summary . . . 195

7 Eywa: A General Framework for Scheduler Design and Its Applications to A Family of AoI-Related Problems 196

7.1 Introduction . . . 196
7.2 Eywa: A General Approach . . . 198
7.2.1 System Model . . . 200
7.2.2 Main Idea: Almost Uniform Scheduler . . . 202
7.2.3 Eywa: Complete Procedure . . . 203
7.3 Application to the Min-Sum Problem . . . 211
7.3.1 Problem Statement . . . 211
7.3.2 A Lower Bound . . . 212
7.3.3 Eywa: Step 1—Transform Objective Function and Constraints . . . 213
7.3.4 Eywa: Step 2—Find Optimal Transmission Rates . . . 215

7.3.5 Eywa: Step 3—Construct AUS 216
7.3.6 Performance and Comparison with State-of-the-Art 216
7.4 Application to the Min-BW Problem 218
7.4.1 Min-BW Under Peak AoI Constraints 218
7.4.2 Min-BW under Average AoI Constraints 221
7.5 Application to Decision Problems 223
7.5.1 A Decision Problem with Hard AoI Deadlines 224
7.5.2 A Decision Problem with Soft AoI Deadlines 226
7.6 Chapter Summary 228

8 Summary and Future Work **230**
8.1 Summary 230
8.2 Future Work 232

Appendices **235**

Appendix A A Proof of Theorem 2.1 **236**

Appendix B A Proof of Theorem 5.1 **240**

Appendix C A Proof of Lemma 5.2 **242**

Appendix D An Algorithm to Solve OPT-β **244**

Appendix E A Proof of Theorem 7.1 **247**

Appendix F An Algorithm to Solve OPT-SD (Min-BW) **252**

Appendix G A Proof of Theorem 7.2 **253**

Appendix H A Performance Guarantee for Aion in [102] **254**

Appendix I A Proof of Theorem 7.3 **256**

Appendix J An Algorithm to Solve DEC-SD **258**

Bibliography **259**

List of Figures

1.1 An IoT data collection model used throughout this dissertation. 3

1.2 A structure of this dissertation, where a solid arrow line between two chapters represents that the latter chapter extends the previous one, a dashed arrow line represents that the latter chapter is a generalization of the previous one, and a dotted line arrow represents that the latter block is an application of the previous one. 4

2.1 An example showing the evolution of $U_i^{\mathrm{s}}(t)$ and $A_i^{\mathrm{s}}(t)$ at source node i versus $U_i^{\mathrm{B}}(t)$ and $A_i^{\mathrm{B}}(t)$ at the BS during different time instances. 16

2.2 A cycle with N_i samples. Each sample has its time interval since the last sample. The first interval is formed by connecting two partial intervals in the beginning and end of this cycle. 23

2.3 The graph of function f. 29

2.4 Per time slot sampling: AoI for varying M 36

2.5 Per time slot sampling: AoI for varying N 36

2.6 Periodic sampling: AoI for varying M 38

2.7 Periodic sampling: AoI for varying T 38

2.8 Periodic sampling: AoI for varying L 39

2.9 Periodic sampling: AoI for varying N 39

2.10 Random sampling: AoI for varying M 41

2.11 Random sampling: AoI for varying p 41

2.12 Random sampling: AoI for varying L 43

2.13 Random sampling: AoI for varying N 43

2.14 Periodic sampling: weak synchronization 44

2.15 Periodic sampling: strong synchronization 44

3.1 An example illustrating the trade-off between MCS selection and the number of RBs that can contribute to total data rate. 55

3.2 Convergence time of our proposed approximate solution to OPT-LB when $\beta = 0.01$. 66

3.3 AoI performance of the scheduler in Algorithm 3.1, Kronos, and the lower bound. 67

3.4 A flowchart for a GPU-based implementation of Kronos. 75

3.5 $\bar{A}^{\mathrm{B}}$ under different numbers of source nodes. 80

3.6 Running time under different numbers of source nodes. 81

3.7 $\bar{A}^{\mathrm{B}}$ under different Rician factors. 83

3.8 Running time under different Rician factors. 84

3.9 $\bar{A}^{\mathrm{B}}$ under different coherence bandwidth. 86

3.10 Running time under different coherence bandwidth. 87

3.11 $\bar{A}^{\mathrm{B}}$ under different coherence time. 88

3.12 Running time under different coherence time. 89

4.1 Recursion in the example. 110

4.2 Success rates for FPM, CSD, and EDF under different $l(\boldsymbol{d})$ when $N = 5$. . . 124

4.3 Success rate for FPM and EDF under different $l(\boldsymbol{d})$ when $N = 20$. 125

4.4 Success rate for FPM and EDF under different $l(\boldsymbol{d})$ when $N = 50$. 126

4.5 Success rate for FPM and EDF under different $l(\boldsymbol{d})$ when $N = 100$. 127

4.6 Success rate for FPM under different N. 128

4.7 Success rate for FPM under different $\boldsymbol{d}$ distribution. 129

5.1 An example for using DP to solve OPT-k-left. 146

5.2 Success percentage for STS/UTS to find a feasible scheduler when $p = 0.1$. . 162

5.3 Success percentage for STS/UTS to find a feasible scheduler when $\epsilon = 0.1$. . 163

6.1 An illustration of convergence behavior of Algorithm 6.1 when $\alpha = 10$ and $\beta = 0.01$. 179

6.2 Results for a case study with $N = 100$, $B = 100$. 191

6.3 Aequitas under varying N when $B = 100$ 192

6.4 Aequitas under varying B when $N = 50$ 194

7.1 An example for Zigzag packing 211

7.2 Applications of Eywa to a family of AoI problems and a comparison of performance guarantees. NA indicates results are not available in the literature. 228

List of Tables

2.1 Notation 12

2.2 Simulation Parameters 35

3.1 Notation 49

3.2 Weights and sampling parameters for different types of source nodes. 78

4.1 Notation 95

4.2 Case study: $N = 5$ 120

4.3 Case study: $N = 100$ 120

4.4 Scheduling behavior of FPM, CSD and EDF. 122

4.5 Feasible schedulers found by FPM for different $\boldsymbol{d}$'s. 122

5.1 Notation 132

5.2 Stable tolerant case, $N = 10$ 158

5.3 Stable tolerant case, $N = 100$ 158

5.4 Unstable tolerant case, $N = 10$ 160

5.5 Unstable tolerant case, $N = 100$ 160

6.1 Notations 169

6.2 AoI deadlines and sample sizes for different types of source nodes. 189

6.3 Aequitas under different C_2 and C_3 190

7.1 Notations 199

7.2 Comparison between Eywa and state-of-the-art for the Min-Sum problem. . 216

Chapter 1

Introduction

1.1 Motivation

The availability of fresh information is of utmost importance to many IoT applications, including autonomous vehicles [1], UAV [2], industrial automation [3], and smart grids [4]. To quantify the level of *freshness,* the concept of "Age of Information" (AoI) was conceived [5, 6]. AoI is defined as the elapsed time for a sample (stored at a particular location, e.g., edge or cloud) between the current time (now) and the time when the sample was first generated (collected) at its source. AoI is an application-layer metric and is fundamentally different from delay or latency at transport/network/link layer, which only focuses on the lapsed time for moving information between two points inside the network. For the latter, once the journey is completed, the information's delay (or latency) will no longer change. In contrast, AoI measures the accrued time since the generation of a sample until the present, which includes transit delay (or latency) as only one of its components (in its accounting of total elapsed time).

Since its inception, the AoI metric has captured the attention of the research community and is now under intensive investigation (see a survey on AoI in [7] and an online bibliography in [8]). Despite active research, there are many limitations with the state-of-the-art AoI research. First, many existing efforts on AoI have been limited to information-theoretic exploration by considering extremely simple models and unrealistic assumptions. For exam-

ple, some common assumptions in these studies are: (i) each source node takes a sample in every time slot, (ii) each sample is of one unit data size, and (iii) at most one unit of data can be transmitted to the base station (BS) in a time slot. These simple assumptions have been used in AoI data collection models [9, 10, 11, 12, 13, 14], AoI queuing models [6, 15, 16, 17, 18, 19, 20, 21, 22, 23], multi-link AoI network models [25, 26, 27, 28, 29, 30], multi-hop AoI network models [32, 33], and so forth. Besides, few existing efforts on AoI modeling and analysis have considered characteristics or capabilities of the state-of-the-art transmission technologies such as 5G NR [31]. There has also been limited research on AoI scheduling that addresses the impact of varying channel conditions, such as joint dynamics in both time and frequency. But in reality, channel condition can change rapidly (e.g., for each transmission time interval (TTI) in 5G) and must be taken into consideration in real-time scheduling.

Second, most existing works on designing scheduling algorithms to optimize AoI are concerned with optimizing AoI (see [8]). Although a clever design of a scheduler to optimize AoI is important, it cannot offer performance guarantees on AoI metric. Imagine some AoI-critical applications such as autonomous vehicles and unmanned aerial vehicles, where the applications require certain freshness guarantee with respect to some sources. Although existing schedulers designed for AoI minimization has some relevance to AoI performance guarantee, they are fundamentally different problems. Simply put, existing schedulers designed for AoI minimization cannot offer any guarantee on AoI requirements. A close scanning of the literature [8] shows that there is a serious lack of research in this area.

In this dissertation, we will address these two limitations in the state-of-the-art by investigating a number of research problems:

- First, we will study how to minimize the average AoI in general settings that are more aligned to IoT applications in the real world, with the consideration of varying sam-

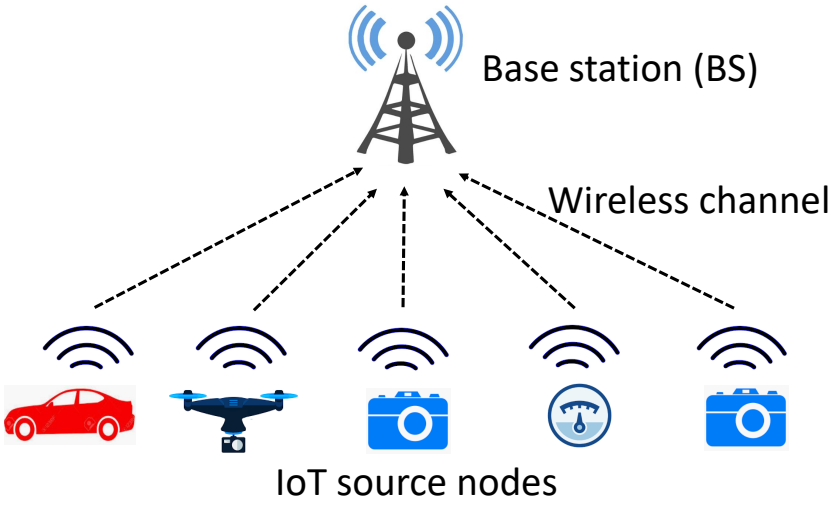


Figure 1.1: An IoT data collection model used throughout this dissertation.

pling periods, varying sample sizes, cellular transmission models, and dynamic channel conditions. A 5G-compliant scheduling problem will be studied with the consideration of the sub-millisecond real-time requirement in 5G.

- Second, we will study how to perform AoI scheduling where each source has an AoI deadline. Specifically, the following problems are addressed: (i) For a given set of AoI deadlines, does there exist a feasible scheduler that can satisfy this deadline set? (ii) If a feasible scheduler exists, then find such a scheduler. The deadlines can be hard (no violation is allowed) or soft (occasional violation is allowed), and we will study schedulers under both types of deadline.

1.2 Main Contributions

The goal of this dissertation is to make a concrete step to extend the state-of-the-art AoI research in wireless networks. We mainly focus on two key aspects of AoI research: minimizing average AoI and guaranteeing AoI deadlines. There are six chapters covering different

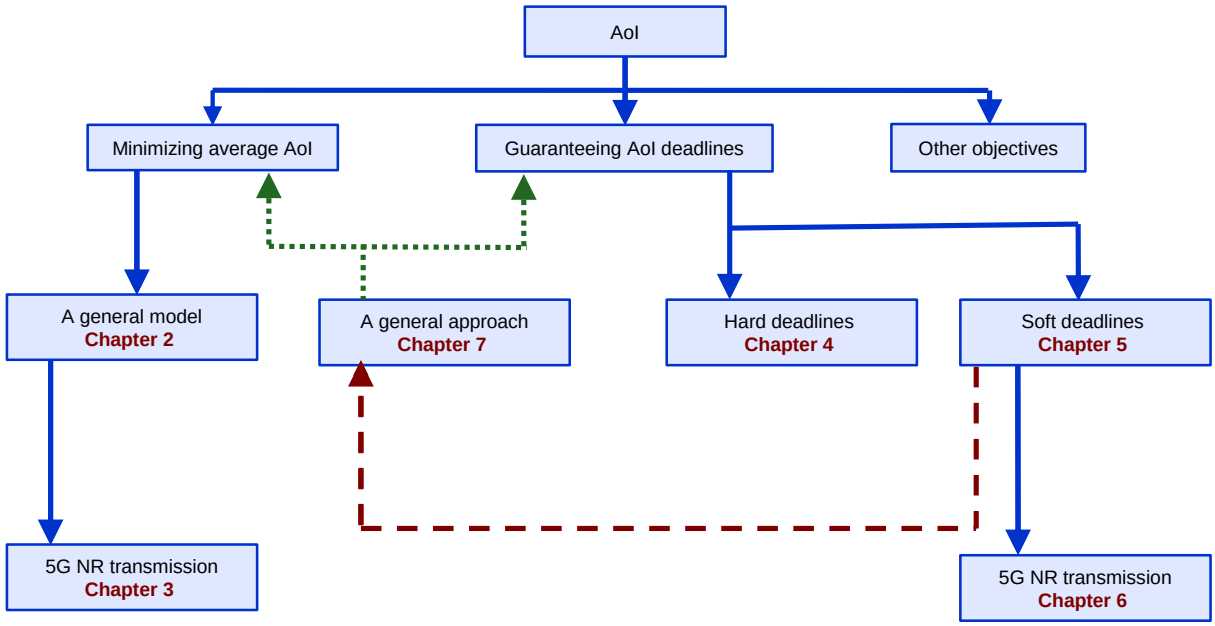


Figure 1.2: A structure of this dissertation, where a solid arrow line between two chapters represents that the latter chapter extends the previous one, a dashed arrow line represents that the latter chapter is a generalization of the previous one, and a dotted line arrow represents that the latter block is an application of the previous one.

research problems (Chapters 2 to 7). All these six chapters are under the same IoT data collection model (see Fig. 1.1), where multiple IoT source nodes are sending and updating their collected information to a single edge base station (BS) through a shared wireless channel. The relationship and contribution areas of these six chapters are shown in Fig. 1.2.

In Chapter 2, we study an average AoI minimization problem under more a general and practical system model, which includes varying sampling periods, varying sample sizes, and a transmission model with multiple transmission units in each time slot. Based on these generalizations, we develop new theoretical results (in terms of fundamental properties and performance bounds) and a near-optimal low-complexity scheduling algorithm—code-named Juventas— to minimize average AoI.

Chapter 3 is an extension of Chapter 2. In Chapter 3, we study the average AoI minimization problem under a 5G network with dynamic channel conditions, which was not considered in Chapter 2. We design a 5G-compliant scheduler—code-named Kronos— to minimize the average AoI. To meet the stringent real-time requirement for 5G, we develop a GPU-based implementation of Kronos on COTS GPUs. Through extensive experimentation, we show that Kronos can find near-optimal solutions under sub-millisecond time scale.

In Chapter 4, we study how to guarantee a *hard* AoI deadline for each source node. Specifically, we want to determine whether or not a vector of hard AoI deadlines for the source nodes is schedulable, and if so, find a feasible scheduler for it. For a small network, we present an optimal procedure called *Cyclic Scheduler Detection* (CSD) that can determine the schedulability with absolute certainty. For a large network where CSD is not applicable, we present a novel low-complexity procedure, called *Fictitious Polynomial Mapping* (FPM), and prove that FPM can find a feasible scheduler for any hard deadline vector when the load is under $\ln 2$.

In Chapter 5, we study how to guarantee a *soft* AoI deadline for each source node, where occasional deadline violations can be tolerated. The problem is to determine whether a set of users with given AoI deadlines, tolerance rates, and packet loss rates (due to each source's channel condition) is schedulable, and if so find a feasible scheduler. We study two cases: (i) the stable tolerant case where the tolerance rate is greater than the packet loss rate for all sources; (ii) the unstable tolerant case where the tolerance rate is less than the packet loss rate for at least one source. For the stable tolerant case, we design an algorithm called *stable tolerant scheduler* (STS), which can find a feasible scheduler for any network when the system load is no greater than $\ln 2$. For the unstable tolerance case, we develop an *unstable tolerant scheduler* (UTS) and identify a schedulability condition for it.

Chapter 6 is an extension of Chapter 5. In Chapter 6, we study a scheduling problem

in a 5G network where there is a soft AoI deadline for each source node. Our goal is to design a 5G scheduler to minimize the proportion of time when the AoI is beyond its soft deadline, i.e., to minimize the violation rate. We derive a property called *uniform fairness* for an offline optimal scheduler and use this property to develop an online 5G scheduler—Aequitas. To meet the sub-millisecond real-time requirement in 5G, we implement Aequitas on a COTS GPU. Through extensive experiments, we show that the objective achieved by Aequitas is close to the lower bound and its running time is under 1 ms.

Inspired by the design ideas in Chapter 5, in Chapter 7 we develop Eywa—a general design framework that can be applied to construct high-performance schedulers for AoI-related optimization and decision problems. The core of Eywa hinges upon the notions of AUS schedulers and step-down rate vectors. The framework of Eywa consists of three steps: (i) transform the (AoI-based) objective function and constraints to rate-based ones, (ii) find the optimal step-down rate vector by solving an optimization problem (OPT-SD), and (iii) construct an AUS-based scheduler using the optimal step-down rate vector. To validate the efficacy of the proposed Eywa framework, we apply it to solve a number of problems, such as minimizing the sum of AoIs, minimizing the bandwidth requirement under AoI constraints, and determining the existence of feasible schedulers to satisfy AoI constraints. We find that for each problem, Eywa can either offer a stronger performance guarantee than the state-of-the-art algorithms, or provide new/general results that are not available in the literature.

Chapter 2

Minimizing Age of Information under General Models for IoT Data Collection

2.1 Introduction

Data collection is a critical component in modern network systems. In a typical scenario, at the network edge, multiple source nodes for different applications collect samples of information from the physical environment and forward the sampled information to the edge server. The collected information can then be either processed and stored locally (edge computing) and/or forwarded to the cloud. Since many applications on the upper layer depend on the timeliness of the sampled information, it is critical to forward the freshest sample to the edge as soon as possible.

As we presented in Chapter 1, many existing efforts on AoI have been largely limited to information-theoretic exploration by considering extremely simple models and unrealistic assumptions which are far from real-world communication systems. These simple models and unrealistic assumptions include:

- **Sampling Behavior.** Most existing AoI research assumes time is slotted and all

source nodes collect a sample in each time slot. Although simple, such a sampling model can hardly capture what is happening in reality. For example, a thermometer may take temperature measurement every a few seconds while a video camera may take 30 samples (frames) per second. In other words, there is a wide range of sampling behavior that a source node may follow, depending on its application.

- **Sample Size.** In addition to unrealistically sampling behavior, most existing AoI research also assumes that the sample size from each source node is identical (one unit of data). Again such a simple model is disconnected from the real world, where the sample size varies from each source node and is determined by its underlying IoT application.

- **Transmission Capacity.** Many existing efforts consider that transmission capacity is one unit of data in each time slot, which conveniently matches their simple sampling behavior and sample size. Such a simplified model, however, does not reflect the capability of state-of-the-art transmission technologies such as 4G LTE [34] and 5G NR [35]). For example, under 4G LTE or 5G cellular, transmission resource occupies both temporal and spectral domains, and there are a large number of transmission units available to the source nodes for transmission in each time slot. Such transmission capability offers a much greater scheduling space than existing AoI transmission models.

In this chapter, we study AoI in a general setting that is more aligned to IoT applications in the real world. Specifically, we consider the following general models. (i) We consider various sampling periods at each source node, such as arbitrary sampling, periodic sampling, and per time slot sampling. Note that per time slot sampling, the simplest sampling behavior, is what has been mostly studied in the literature. (ii) We allow sample size collected at

each source node to vary, depending on the underlying application. (iii) We generalize the transmission capacity with multiple data units in each time slot to model a cellular environment (e.g., 4G LTE or 5G).

The main contributions of this chapter are the following:

- We study AoI with a much more general model than those used in the state-of-the-art in terms of sampling period at the source nodes, sample size collected from each source, and transmission capacity. Such generalizations offer a much better characterization of source heterogeneity and transmission behavior in a heterogeneous IoT world. As a result, findings based on this general model not only have more significance from theoretical perspective, but also have greater impacts on IoT applications in the field.
- Under this general model, it becomes much more challenging to design an AoI minimization scheduler, due to a much larger search space than those considered in the literature. As a first step, we develop two fundamental properties for an optimal scheduling solution. We show how these properties can help reduce the search space for the optimal and near-optimal solution. These properties also serve as an aid for the development of optimal AoI scheduling algorithms.
- In the reduced search space, we further develop theoretical lower bounds for AoI under different sampling periods (arbitrary, periodic and per time slot). These lower bounds serve as performance benchmarks to assess the quality of a scheduling algorithm under different sampling behaviors.
- Finally, we design a low-complexity scheduling algorithm (Juventas). Through theoretical analysis, we find that Juventas can guarantee a factor of 3 from the objective value. Through a simulation study, we find that Juventas is near-optimal when there is no synchronization among the source nodes during sampling.

2.2 Related Work

There is a body of work on modeling, analysis, and simulation of the AoI metric. In [6, 15, 16, 17, 18, 19, 20, 21, 22, 23], the authors considered different information-update queueing models (such as M/M/1, M/G/1/, D/M/1) and analyzed the AoI metric under these models. In [36, 37], the authors analyzed AoI under priority-based queuing systems. In [20], Kosta *et al.* analyzed AoI under a non-linear aging model for queuing system. In [23], Yates analyzed AoI in a network where a source updates information to a monitor through multiple servers, with each server modeled as a queue. In [38], Kam *et at.* analyzed AoI under random updates and in [39], they investigated AoI under different buffer (queue) sizes and deadlines. In [40], Kaul *et al.* analyzed AoI under both scheduled and random access (slotted-ALOHA). In [41, 42], Farazi *et al.* analyzed AoI in energy harvesting status update systems under an M/M/1 queuing model. Most of these analyses were based on overly simplified models in terms of sampling periods, sample size, transmission rate and channel conditions. Therefore, results from these analyses are only of interest from an information-theoretic perspective and are not applicable to address AoI problems under practical IoT settings.

There is an active body of work on AoI scheduling. The goal is to develop scheduling algorithms that can minimize AoI (either weighted average or otherwise) for all information sources. In [9, 10, 11, 12, 13, 14], the authors considered an infrastructure-based model where information sources share a common channel and only one packet (from at most one source) can be transmitted to the BS in one time slot. Specifically, in [9], Hsu *et al.* considered the Bernoulli packet arrival model. In [10, 11], Kadota *et al.* considered an unreliable channel, and in [11], they considered some throughput constraints. In [14], Zhong *et al.* explored synchronization together with AoI. In [43, 44], the authors considered a multi-channel system where information sources share multiple independent wireless channels and one packet can be transmitted on each channel to the BS in a time slot. In [25, 26, 27, 28, 29, 30], the

authors considered a multi-link network environment where a subset of links can transmit simultaneously in a time slot when they are not interfering with each other. Specifically, in [28] and [29], Talak *et al.* developed a centralized and a distributed AoI scheduling algorithm, respectively. In [26], Joo and Eryilmaz considered both AoI and information synchronization. In [27], Lu *et al.* developed scheduling algorithms to satisfy a per-flow throughput requirement. Finally, AoI scheduling has also been studied in multi-hop networks [32, 33]. Again, most of these works on AoI scheduling algorithms only considered extremely simple sampling behaviors, sample size, and transmission technologies.

2.3 System Model

Consider a network consisting of N IoT source nodes and one BS as shown in Fig. 1.1. Each source node samples information from its environment and attempts to forward it to the BS. Such IoT data collection corresponds to uplink data transmission in cellular terminology. For multiplexing, time is divided into time slots and each time slot can accommodate a certain number of data transmission units. Denote M as the number of data transmission units per time slot over the uplink bandwidth. Then, in each time slot, the BS will allocate M data transmission units to one more more source nodes for uplink data transmission. For simplicity, with respect to each source node, we assume each transmission unit carries the same amount of information over all time slots.[1]

At each source node, information is collected (or generated) with a specific sampling period at this source. Denote L_i (in number of transmission units) as the amount of information in each sample (sample size) for source node i. Due to the heterogeneity of IoT source nodes, the sampling periods and sample sizes generally differ among the source nodes.

[1]The more general case that considers channel diversity in time and frequency domains will be explored in a future work.

Table 2.1: Notation

Symbol	Definition
$\bar{A}^{\mathrm{B}}$	Weighted long-term average AoI over all source nodes at the BS
$\bar{A}_i^{\mathrm{B}}$	Long-term average AoI from source node i at the BS
$\bar{A}_i^{\mathrm{s}}$	Long-term average AoI at source node i
$A_i^{\mathrm{B}}(t)$	AoI from source node i at the BS at time slot t
$A_i^{\mathrm{s}}(t)$	AoI at source node i at time slot t
$\alpha_{(\mathrm{PTS},M)}$	A lower bound of $\bar{A}$ under per time slot sampling and finite link capacity
$\alpha_{(\mathrm{ARB},\infty)}$	A lower bound of $\bar{A}$ under arbitrary time slot sampling and infinite link capacity
$\alpha_{(\mathrm{ARB},M)}$	A lower bound of $\bar{A}$ under arbitrary time slot sampling and finite link capacity
$\alpha_{(\mathrm{PRD},M)}$	A lower bound of $\bar{A}$ under periodic sampling and finite link capacity
$b_i(k)$	Beginning time slot of k-th transmission from node i
$\Delta_i(t)$	AoI decrease for potential transmission from node i in time slot t
$e_i(k)$	Ending time slot of the k-th transmission from node i
L_i	Size of information sampled at node i
M	Uplink link capacity (the number of transmission units) at each time slot
N	Number of source nodes in the network
$\boldsymbol{S}(t)$	State of the entire network at time slot t
T_i	Sampling cycle (in number of time slots) at node i
$U_i^{\mathrm{B}}(t)$	Generation time of the freshest (and complete) sample at the BS from source node i at time slot t
$U_i^{\mathrm{s}}(t)$	Generation time of the freshest sample at source node i at time slot t
w_i	Weight for node i
$x_i(t)$	Number of data units allocated to user i at time slot t
$\boldsymbol{X}(t)$	Scheduling decision for time slot t

A wide range of sampling behavior are possible among the N sources, such as:

- **Arbitrary Sampling.** Each node perform its own sampling (either following a random or deterministic pattern) and is independent of other sources. This sampling behavior is the most general one among all sampling behaviors.
- **Periodic Sampling.** Source node i performs sampling at every T_i time slots. The sampling intervals (T_i's) are generally different among different source nodes. This sampling behavior is likely the most prevailing sampling behavior among real-world IoT applications. For instance, temperature sensors usually sample information with a lower rate, while accelerometers sample with a higher rate.
- **Per Time Slot Sampling.** Each source node samples information in every time slot. This is the special case for periodic sampling with $T_i = 1$ for every node i. It is a model used by most works in AoI research (see, e.g., [11, 13, 30]). Although simple, this model may not be an accurate characterization of real world sampling behavior as each source node usually samples at different rate, due to difference in applications.

When the BS allocates transmission units to a source node, the source node will always transmit its freshest sample (the most recently generated sample). Recall each sample from each node i consists of L_i units of information. Due to the size of L_i, it may take multiple time slots to complete the transmission of this sample. Only after all L_i units of the sample from source i are transmitted to the BS, we say the BS has received this sample. Once the transmission of a sample begins, the remaining unfinished units from this sample must be transmitted (over multiple time slots if needed) before any new sample is considered (even if the new sample is fresher than the one currently under transmission).

The BS maintains the sample that it has most recently received from each source node and considers it the freshest information that it possesses from that source. Again, a sample

from a source is not considered received until the sample (consisting of multiple units) is received in its entirety (possibly requiring multiple time slots). Upon receiving a sample from source i completely, the BS replaces the previous sample from source i with this newly received sample.

2.4 AoI Modeling and Problem Statement

At each source node i, denote $U_i^{\mathrm{s}}(t)$ as the generation time of the most recent sample at time slot t. Denote $A_i^{\mathrm{s}}(t)$ as the AoI at source node i at time slot t. Recall that AoI is defined as the elapsed time for a sample between current time (now) and its generation time, we have

$$A_i^{\mathrm{s}}(t) = t - U_i^{\mathrm{s}}(t). \tag{2.1}$$

Note that $A_i^{\mathrm{s}}(t)$ is a zigzag-like function with a slope of 1 between sampling intervals and is reset to 0 in each time slot when a new sample is generated. Clearly, $A_i^{\mathrm{s}}(t) = 0$ for all t under the per time slot sampling case, and $0 \leq A_i^{\mathrm{s}}(t) < T_i$ under the periodic sampling case.

At the BS, it maintains the most recent (complete) sample that it has received from each of the N source nodes. Note that this sample maintained at the BS from source node i may be different from (older than) the freshest sample currently at source node i. Denote $U_i^{\mathrm{B}}(t)$ as the generation time of the sample from source node i that is currently maintained by the BS at time slot t. Denote $A_i^{\mathrm{B}}(t)$ as the AoI for this sample at the BS at time slot t. Then we have

$$A_i^{\mathrm{B}}(t) = t - U_i^{\mathrm{B}}(t). \tag{2.2}$$

Since $U_i^{\mathrm{B}}(t) \leq U_i^{\mathrm{s}}(t)$, we have $A_i^{\mathrm{B}}(t) \geq A_i^{\mathrm{s}}(t)$, i.e., the AoI for source node i as perceived (maintained) by the BS is older (larger) than or equal to that at the source node, which is intuitive. Note that $A_i^{\mathrm{B}}(t)$ is also a zigzag-like function with a slope of 1 between time

instances when a sample is received and is reset at the end of each time slot when a new sample is completely received at the BS.

We now make a connection between $A_i^{\mathrm{B}}(t)$ and $A_i^{\mathrm{s}}(t)$. From source node i, for the k-th sample that is actually selected for transmission,[2] denote its beginning (starting) transmission time slot as $b_i(k)$, and ending (finishing) transmission time slot as $e_i(k)$, where $e_i(k) \geq b_i(k)$. Since this k-th sample is selected for transmission at time $b_i(k)$, it must be the freshest sample at source node i at that time, with a generation time of $U_i^{\mathrm{s}}\big(b_i(k)\big)$. After this k-th sample is completely sent to the BS at the end of time slot $e_i(k)$, in the beginning of the next time slot $(e_i(k)+1)$, we have

$$U_i^{\mathrm{B}}(e_i(k)+1) = U_i^{\mathrm{s}}\big(b_i(k)\big).$$

From (2.2) and (2.1), we have

$$\begin{aligned} A_i^{\mathrm{B}}(e_i(k)+1) &= e_i(k)+1-U_i^{\mathrm{B}}(e_i(k)+1) \\ &= e_i(k)+1-U_i^{\mathrm{s}}(b_i(k)) \\ &= e_i(k)+1-\Big(b_i(k)-A_i^{\mathrm{s}}\big(b_i(k)\big)\Big) \\ &= A_i^{\mathrm{s}}\big(b_i(k)\big)+e_i(k)-b_i(k)+1. \end{aligned}$$

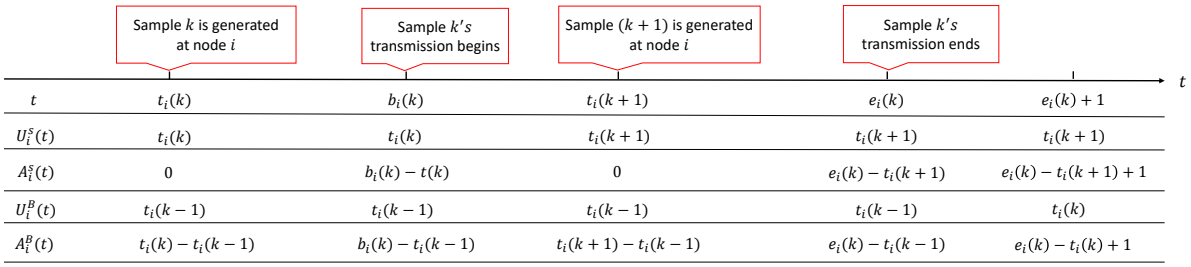
Therefore, over all t, we have

$$A_i^{\mathrm{B}}(t+1) = \begin{cases} A_i^{\mathrm{s}}\big(b_i(k)\big)+e_i(k)-b_i(k)+1, & \text{if } t=e_i(k), \\ A_i^{\mathrm{B}}(t)+1, & \text{otherwise.} \end{cases} \tag{2.3}$$

Note that $A_i^{\mathrm{s}}\big(b_i(k)\big)+e_i(k)-b_i(k) \geq A_i^{\mathrm{s}}\big(e_i(k)\big)$ at the source node. Considering (2.3), clearly, we have

$$A_i^{\mathrm{B}}(t) \geq A_i^{\mathrm{s}}(t-1)+1, \quad \forall t. \tag{2.4}$$

[2]Recall that not every sample generated at source node i will be transmitted to the BS.



t	$t_i(k)$	$b_i(k)$	$t_i(k+1)$	$e_i(k)$	$e_i(k)+1$
$U_i^s(t)$	$t_i(k)$	$t_i(k)$	$t_i(k+1)$	$t_i(k+1)$	$t_i(k+1)$
$A_i^s(t)$	0	$b_i(k)-t(k)$	0	$e_i(k)-t_i(k+1)$	$e_i(k)-t_i(k+1)+1$
$U_i^B(t)$	$t_i(k-1)$	$t_i(k-1)$	$t_i(k-1)$	$t_i(k-1)$	$t_i(k)$
$A_i^B(t)$	$t_i(k)-t_i(k-1)$	$b_i(k)-t_i(k-1)$	$t_i(k+1)-t_i(k-1)$	$e_i(k)-t_i(k-1)$	$e_i(k)-t_i(k)+1$

Figure 2.1: An example showing the evolution of $U_i^{\mathrm{s}}(t)$ and $A_i^{\mathrm{s}}(t)$ at source node i versus $U_i^{\mathrm{B}}(t)$ and $A_i^{\mathrm{B}}(t)$ at the BS during different time instances.

This implies that it takes at least one time slot to transmit a sample from a source node to the BS, and $A_i^{\mathrm{B}}(t)$ is always larger than $A_i^{\mathrm{s}}(t-1)+1$.

An example of AoI evolution is given in Fig. 2.1.

Based on (2.3), the long-term average of source node i's AoI at the BS can be written as:

$$\bar{A}_i^{\mathrm{B}} = \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} A_i^{\mathrm{B}}(t). \tag{2.5}$$

Denote w_i as the weight of source node i's information, which can be used to reflect the priority of node i. Then the AoI over all source nodes at the BS can be written as

$$\bar{A}^{\mathrm{B}} = \sum_{i=1}^{N} w_i \bar{A}_i^{\mathrm{B}}. \tag{2.6}$$

Since there are only M data units available for transmission in each time slot, a scheduling algorithm is needed to decide how to allocate the M data units to a subset of source nodes in each time slot. Denote $\boldsymbol{X}(t)$ as the scheduling decision for time slot t, where $\boldsymbol{X}(t)$ is an $N \times 1$ vector with its i-th element $x_i(t)$ being the number of data units that is allocated to user i at time slot t. Since each transmission data unit can be allocated to at most one source node, we have

$$\sum_{i=1}^{N} x_i(t) \leq M. \tag{2.7}$$

Clearly, each different scheduling algorithm will yield a very different performance of $\bar{A}^{\mathrm{B}}$ in (2.6). Our goal is to find an optimal scheduling algorithm under which $\bar{A}^{\mathrm{B}}$ is minimized.

Based on (2.5), minimizing $\bar{A}_i^{\mathrm{B}}$ requires the design of a scheduling algorithm over an infinite number of time slots, which makes the search space for $\boldsymbol{X}(t)$ infinite. To address this problem, we show, in the next section, how to reduce the search space for an optimal scheduling solution.

2.5 Properties for An Optimal Scheduling Algorithm

Given that an optimal solution to our scheduling problem may not be unique, an efficient approach to reduce the search space is to find some properties associated with a particular optimal scheduling solution. Based on these properties, it becomes more tractable to find an optimal solution or design a near optimal solution.

2.5.1 An Order-based Scheduling

At each time slot t, it's intuitive to perform an order-based scheduling, that is, to assign an order for all source nodes and allocate data transmission units to the source node currently with the highest order before allocating to the source node with the second highest order and so forth. The order can be designed based on $A_i^{\mathrm{s}}(t)$, $A_i^{\mathrm{B}}(t)$, w_i, L_i, and transmission status (i.e, how many units of data that have been transmitted before the current time slot) for each source node.

The following lemma states that for each time slot t, there exists an optimal order-based scheduling algorithm that minimizes $\bar{A}^{\mathrm{B}}$.

Lemma 2.1. *Under arbitrary sampling, there exists an order-based scheduling algorithm that*

achieves the optimal objective.

Proof. We prove this lemma by construction. Suppose $\boldsymbol{X}^*(t)$ is an optimal scheduling algorithm that minimizes $\bar{A}^{\mathrm{B}}$. For any time slot t, suppose the scheduled transmission samples are from source nodes $i_1, i_2, \cdots, i_P$ with ending time slots $e_1, e_2, \cdots, e_P$ such that $t \leq e_1 \leq e_2 \leq \cdots \leq e_P$. Denote $\mathcal{S}$ as the set of transmission units that are allocated to source nodes $i_1, i_2, \cdots, i_P$ to complete their current samples from time t (inclusive) under $\boldsymbol{X}^*(t)$. We define the order of those source nodes as $i_1 > i_2 > \cdots > i_P$. Based on this order, we can re-allocate the transmission units in $\mathcal{S}$ as follows. We first allocate transmission units to finish i_1's sample in its entirety and then move to i_2's sample, and so on. By doing this, the ending time slot of each node will not increase and thus the new $\bar{A}^{\mathrm{B}}$ is either equal or smaller than the previous objective. Since the previous objective is optimal, then only equality is possible and such order-based scheduling is optimal. By performing this transmission-unit-reallocation for each time slot t, we can construct a new optimal scheduling algorithm that follows the order-based pattern. □

Using this property, we only need to find or design an order-based scheduling algorithm. This property allows us to work in a much smaller search space. Also note that since this property is for arbitrary sampling, it applies to periodic and per time slot sampling policies as well.

2.5.2 Cyclic Transmission

Another property that we want to explore is whether an optimal scheduling algorithm exhibits a cyclic (periodic) transmission pattern, i.e., with the same scheduling decision for every, say T_c, time slots. Intuitively, if the sampling behaviors are not periodic, it makes no sense to perform periodic scheduling decision, so this property is hard to establish under arbitrary sampling policy. We will focus on periodic sampling policy, which also includes per time slot sampling policy.

More formally, we say a scheduling algorithm is *cyclic* if it repeats its scheduling decision for a fixed number of time slots. Denote $\boldsymbol{X}_c(t)$ as a cyclic scheduling algorithm and T_c as its cycle (in number of time slots). Then there exists a t_0 such that for any $t > t_0$, we have

$$\boldsymbol{X}_c(t) = \boldsymbol{X}_c(t + T_c).$$

The following lemma states the existence of such an optimal cyclic scheduler under periodic sampling policy.

Lemma 2.2. *When each source is sampled periodically (even with different periods), there exists a cyclic scheduling algorithm that achieves optimal objective.*

Proof. We prove this lemma by construction. We first define the *state* of the network in a time slot as the complete information of current AoI at the source nodes, current AoI at the BS, the number of remaining transmission units that are still needed for each source node, and the generation time of the unfinished sample (if there is) for each source node. Denote $\boldsymbol{S}(t)$ as the state at time slot t. Under periodic sampling, for two different time slots, if the states of the system are identical, then under the same scheduling decision, the states for

the following two respective time slots are also identical. That is, if $\boldsymbol{S}(t_1) = \boldsymbol{S}(t_2)$, then if $\boldsymbol{X}(t_1) = \boldsymbol{X}(t_2)$, we have $\boldsymbol{S}(t_1+1) = \boldsymbol{S}(t_2+1)$.

Denote $W\big(\boldsymbol{S}(t)\big)$ as the corresponding weighted-sum AoI at the BS for state $\boldsymbol{S}(t)$. Then the objective can be written as

$$\bar{A}^{\mathrm{B}} = \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} W\big(\boldsymbol{S}(t)\big). \tag{2.8}$$

Suppose $\boldsymbol{X}^*(t)$ is an optimal algorithm with states $\boldsymbol{S}^*(t)$ and average AoI $\bar{A}^*$ (at the BS). Among all the time slots (from 1 to infinity), it's easy to see that those states with $W\big(\boldsymbol{S}^*(t)\big) \leq 2\bar{A}^*$ should be visited for infinite times (actually, $\boldsymbol{X}^*(t)$ visits these states in more than a half of the time slots). Notice that the number of states with $W\big(\boldsymbol{S}(t)\big) \leq 2\bar{A}^*$ is finite because the number of the possible values for each component in a state is finite when $W\big(\boldsymbol{S}(t)\big)$ is bounded. From the pigeonhole principle, we know there must be one state $\boldsymbol{S}_1$ that is visited for infinite times under $\boldsymbol{X}^*(t)$.

We then divide the time domain into infinite number of segments based on the appearance of this state, with this state appearing in the first time slot of each segment. Obviously, there must be a segment with average AoI at the BS smaller than or equal to $\bar{A}^*$. Since $X^*(t)$ is optimal, only equality is possible. Then we can construct an optimal cyclic scheduling algorithm by repeating the states within this segment. This completes our proof. □

Since Lemma 2.2 applies to periodic sampling policy, it also applies to per time slot sampling policy.

2.5.3 Complexity Analysis

Under arbitrary sampling, Lemma 2.1 helps greatly reduce the search space since we only need to assign orders to the source nodes at each time slot. If sampling is periodic, the search space can be further reduced by Lemma 2.2. However, even under periodic sampling, the reduced search space for an optimal scheduling algorithm is still infinite since T_c can be any number. To the best of our knowledge, there is no efficient way to find the optimal scheduling algorithm, even under a very simple and special case (per time sampling, $L_i = 1$, $M = 1$) [13]. Therefore, we have to pursue an efficient heuristic algorithm to achieve near-optimal performance.

2.6 Performance Bounds

Before we design a scheduling algorithm, we first develop some lower bounds for our objective, $\bar{A}^{\mathrm{B}}$, under different cases (sampling behaviors and link capacities). These results are not only important to serve as a performance benchmark to assess the scheduling algorithm that we will develop (in Section 2.7), they are also of significant theoretical value on their own as they generalize a number of results (developed for special or simple cases) in the existing literature.

We will develop lower bounds of our objective function, denoted as $\alpha_{(*,*)}$ for the following four cases: (i) per time slot sampling under finite link capacity: $\alpha_{(\mathrm{PTS},M)}$, (ii) arbitrary sampling under infinite link capacity: $\alpha_{(\mathrm{ARB},\infty)}$, (iii) arbitrary sampling under finite link capacity: $\alpha_{(\mathrm{ARB},M)}$, and (iv) periodic sampling under finite link capacity: $\alpha_{(\mathrm{PRD},M)}$.

2.6.1 The Case of Per Time Slot Sampling Under Finite Link Capacity

In this case, each source node takes a sample at every time slot, i.e., $T_i = 1$ and $A_i^{\mathrm{s}}(t) = 0$ for all i. Here $\bar{A}_i^{\mathrm{B}}$ is purely limited by the link capacity, M. In the literature (see, e.g., [11, 13]), lower bounds for the same objective function have been developed for the simple case where $M = 1$ and $L_i = 1$ for each source node i. Our development here is for a general value of $M \geq 1$ and different values of L_i for each different source node, which is what happens in practice.

Since per time slot sampling is a special case of periodic sampling, by Lemma 2.2, there exists an optimal cyclic algorithm $\boldsymbol{X}_c^*(t)$ with a cycle T_c. Denote N_i as the number of fully transmitted samples from node i over a cycle of T_c time slots. In a cycle, the number of transmission units allocated among the source nodes cannot be more than the total number of available transmission units. We have

$$\sum_{i=1}^{N} N_i L_i \leq M \cdot T_c. \tag{2.9}$$

Define r_i as the transmission rate for source node i, i.e.,

$$r_i = \frac{N_i}{T_c}. \tag{2.10}$$

Under per time slot sampling, since at most one sample from each source node can be sent to the BS at each time slot, we have

$$0 < r_i \leq 1. \tag{2.11}$$

Intuitively, r_i shows the percentage of a sample from source node i that can be transmitted over a long term.

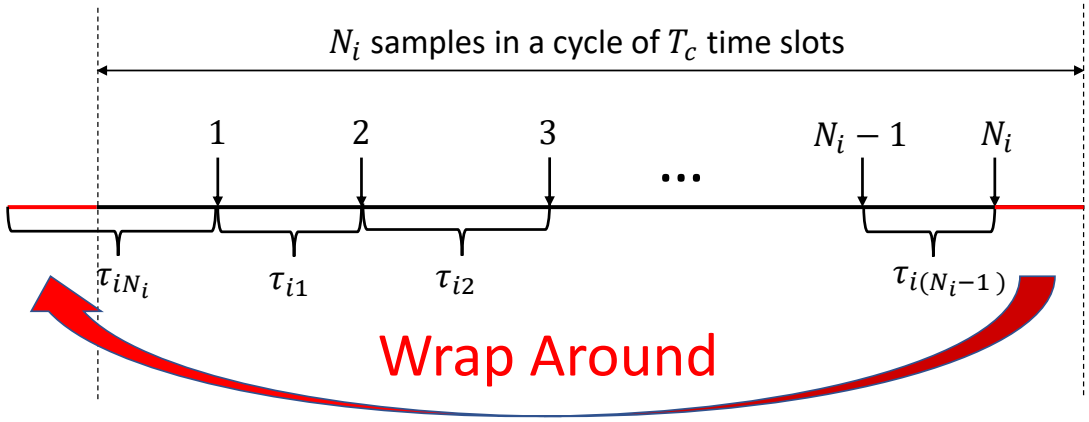


Figure 2.2: A cycle with N_i samples. Each sample has its time interval since the last sample. The first interval is formed by connecting two partial intervals in the beginning and end of this cycle.

Dividing (2.9) by T_c and using (2.10), we have

$$\sum_{i=1}^{N} r_i L_i \leq M. \tag{2.12}$$

For source node i, there are N_i samples transmitted in a cycle. Consider the time interval between two successive transmitted samples. Then we have $(N_i - 1)$ intervals. By wrapping around a transmission cycle and viewing it cyclically (see Fig. 2.2), the time from the beginning of the cycle until the first transmitted sample and the time from the N_i-th transmitted sample to the end of the cycle together can be considered as the interval time for the first transmitted sample. We have, therefore, a total of N_i intervals for source node i in T_c. Denote these N_i intervals as $\tau_{i1}, \tau_{i2}, ...\tau_{iN_i}$ (see Fig. 2.2), we have

$$\sum_{j=1}^{N_i} \tau_{ij} = T_c.$$

Since it takes at least one time slot to transmit a sample from source node i to the BS, we have $A_i^{\mathrm{B}}(t) \geq 1$. Then, during time interval τ_{ij}, the sum of AoI at the BS (for source

node i) is at least

$$1 + 2 + ... + \tau_{ij} = \frac{\tau_{ij}^2 + \tau_{ij}}{2}.$$

So in a cycle with T_c time slots, a lower bound for $\bar{A}_i^{\mathrm{B}}$ can be found by taking time average (over T_c time slots) of $\bar{A}_i^{\mathrm{B}}(t)$ for N_i time intervals. We have

$$\bar{A}_i^{\mathrm{B}} \geq \frac{1}{T_c} \sum_{j=1}^{N_i} \frac{\tau_{ij}^2 + \tau_{ij}}{2} = \frac{1}{T_c} \sum_{j=1}^{N_i} \frac{\tau_{ij}^2}{2} + \frac{1}{2}. \tag{2.13}$$

Applying the Cauchy-Schwarz inequality, we have

$$N_i \sum_{j=1}^{N_i} \tau_{ij}^2 \geq (\sum_{j=1}^{N_i} \tau_{ij})^2 = T_c^2. \tag{2.14}$$

Applying (2.14) to (2.13) we have

$$\bar{A}_i^{\mathrm{B}} \geq \frac{T_c}{2N_i} + \frac{1}{2} \quad \text{or} \quad \bar{A}_i^{\mathrm{B}} \geq \frac{1}{2r_i} + \frac{1}{2}.$$

Based on (2.6), we have

$$\bar{A}^{\mathrm{B}} \geq \sum_{i=1}^{N} w_i (\frac{1}{r_i} + \frac{1}{2}). \tag{2.15}$$

To find a lower bound for $\bar{A}^{\mathrm{B}}$, we can use a lower bound for $\sum_{i=1}^{N} w_i (\frac{1}{2r_i} + \frac{1}{2})$, which means we need to find the minimum of $\sum_{i=1}^{N} \frac{w_i}{r_i}$. We have the following optimization problem:

$$\begin{aligned} \min_{r_i} \quad & \sum_{i=1}^{N} \frac{w_i}{r_i} \\ \text{s.t.} \quad & \text{Constraints (2.11) and (2.12)} \end{aligned} \tag{2.16}$$

The above optimization problem is convex and can be easily solved. In the optimal solution, suppose there are K nodes ($0 \leq K \leq N$) with their $r_i^* = 1$ and the remaining $N - K$ nodes with their $r_i^* < 1$. Without loss of generality, we assume $r_i^* = 1$ for $i \leq K$ and $r_i^* < 1$ for

$i > K$. Define

$$M_K = M - \sum_{i=1}^{K} L_i. \tag{2.17}$$

In solving the convex optimization, the KKT conditions require, for $i \leq K$,

$$\sqrt{\frac{w_i}{L_i}} \geq \frac{\sum_{j=K+1}^{N} \sqrt{w_j L_j}}{M_K}, \tag{2.18}$$

and for $i > K$, r_i^* is given as:

$$r_i^* = \frac{M_K \sqrt{\frac{w_i}{L_i}}}{\sum_{j=K+1}^{N} \sqrt{w_j L_j}} < 1. \tag{2.19}$$

With the optimal solution to (2.16), a lower bound of $\bar{A}^{\mathrm{B}}$, denoted by $\alpha_{(\mathrm{PTS},M)}$, is given by

$$\alpha_{(\mathrm{PTS},M)} = \sum_{i=1}^{N} w_i (\frac{1}{2r_i^*} + \frac{1}{2}) = \frac{1}{2M_K} \quad (\sum_{i=K+1}^{N} \sqrt{w_i L_i})^2 + \frac{1}{2} \sum_{i=K+1}^{N} w_i + \sum_{i=1}^{K} w_i. \tag{2.20}$$

In the special case of $M = 1$ and $L_i = 1$, we have $K = 0$ and $M_K = 0$. Therefore,

$$r_i^* = \frac{\sqrt{w_i}}{\sum_{j=1}^{N} \sqrt{w_j}} \quad (1 \leq i \leq N)\ , \tag{2.21}$$

and the lower bound

$$\alpha_{(\mathrm{PTS},1)} = \frac{1}{2} (\sum_{i=1}^{N} \sqrt{w_i})^2 + \frac{1}{2} \sum_{i=1}^{N} w_i\ , \tag{2.22}$$

which are the main results reported in [13].

2.6.2 The Case of Arbitrary Sampling Under Infinite link Capacity

In this case the link capacity is infinite, i.e., $M \to \infty$. Here the BS can update information for all nodes in every time slot. $\bar{A}^{\mathrm{B}}$ is purely limited by the source sampling, $A_i^{\mathrm{s}}(t)$.

We define the average AoI at the source node i as

$$\bar{A}_i^{\mathrm{s}} = \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} A_i^{\mathrm{s}}(t). \tag{2.23}$$

From the evolution of AoI (2.3), under infinite link capacity, we have

$$A_i^{\mathrm{B}}(t) = A_i^{\mathrm{s}}(t-1) + 1, \quad \text{for } t > 0.$$

So in this case $\bar{A}^{\mathrm{B}}$ equals to

$$\alpha_{(\mathrm{ARB},\infty)} = \sum_{i=1}^{N} \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} w_i(A_i^{\mathrm{s}}(t)+1) = \sum_{i=1}^{N} w_i(\bar{A}_i^{\mathrm{s}}+1). \tag{2.24}$$

This means the average AoI at the BS side is the average AoI at the source side plus 1. Here "1" is the transport delay of the network, meaning that the fresh information at the source needs one time slot to be sent to the BS. Note that under infinite link capacity, $\alpha_{(\mathrm{ARB},\infty)}$ is actually a constant rather than a theoretical bound.

It appears that none of the existing works considered the limitation of source sampling. $\alpha_{(\mathrm{ARB},\infty)}$ reveals an important fact that infinitely increasing link capacity cannot decrease $\bar{A}^{\mathrm{B}}$ to 0. Instead, $\bar{A}^{\mathrm{B}}$ will converge to $\alpha_{(\mathrm{ARB},\infty)}$.

2.6.3 The Case of Arbitrary Sampling Under Finite Link Capacity

In Section 2.6.1 and 2.6.2, we have already derived two lower bounds, $\alpha_{(\mathrm{PTS},M)}$ and $\alpha_{(\mathrm{ARB},\infty)}$, respectively from the limitation of link capacity and source sampling. In the general case of arbitrary sampling and finite link capacity, both $\alpha_{(\mathrm{PTS},M)}$ and $\alpha_{(\mathrm{ARB},\infty)}$ will apply and we can choose the tighter of the two as the lower bound. That is,

$$\alpha_{(\mathrm{ARB},M)} = \max(\alpha_{(\mathrm{PTS},M)}, \alpha_{(\mathrm{ARB},\infty)}). \tag{2.25}$$

In the previous works (see, e.g., [11, 13]), the authors did not consider the impact of

source sampling when developing a lower bound. Thus, under arbitrary sampling and finite link capacity where source sampling is the major limiting for $\bar{A}^{\mathrm{B}}$ (e.g., when M is relatively large), $\alpha_{(\mathrm{PTS},M)}$ (a generalization of the lower bounds in [11, 13]) can be much looser than $\alpha_{(\mathrm{ARB},M)}$ developed in this chapter.

2.6.4 The Case of Periodic Sampling Under Finite Link Capacity

Under the periodic sampling case (under finite link capacity), we use a new relaxation technique to develop a tighter lower bound, $\alpha_{(\mathrm{PRD},M)}$.

By Lemma 2.2, there exists an optimal cyclic algorithm $\boldsymbol{X}_c^*(t)$ with a cycle T_c. It's easy to see T_c should be a multiple number of each node's sampling cycle, T_i. Just as in Section 2.6.1, denote N_i as the number of fully transmitted samples from source node i over a cycle. Eq. (2.9) still holds. Other than the transmission rate r_i, for the periodic sampling case, we define p_i as the transmission percentage for source node i, i.e.,

$$p_i = \frac{N_i T_i}{T_c}. \tag{2.26}$$

Intuitively, p_i represents the percentage of fully transmitted samples over all generated samples in a cycle of T_c time slots. Clearly, we have

$$0 < p_i \leq 1. \tag{2.27}$$

Dividing (2.9) by T_c and using (2.26), we have

$$\sum_{i=1}^{N} \frac{p_i L_i}{T_i} \leq M. \tag{2.28}$$

Under periodic sampling we can also find N_i time intervals for each source node i in one cycle, $\tau_{i1}, \tau_{i2}, ...\tau_{iN_i}$, as we did in Section 2.6.1. To obtain a lower bound of $\bar{A}^{\mathrm{B}}$, we assume

that transmission of each sample can be finished in one time slot. Consider the following problem: If N_i is given, when should these N_i transmissions occur in order to minimize $\bar{A}^{\mathrm{B}}$ in a cycle? Under the optimal strategy (to achieve the smallest $\bar{A}^{\mathrm{B}}$), transmission of a sample should occur in the time slot immediately following the time instance when the sample is taken. Further, under the optimal transmission strategy, the lengths of transmission intervals should be similar. That is, the difference between any two transmission intervals is at most one T_i. Otherwise, we can use the average of their intervals for transmission and obtain a smaller $\bar{A}^{\mathrm{B}}$.

Therefore, if we define $H_i = \lfloor \frac{1}{p_i} \rfloor$ (where $\lfloor \cdot \rfloor$ is the floor function), to minimize $\bar{A}^{\mathrm{B}}$ in one cycle, each transmission interval τ_{ij} should be equal to either $H_i T_i$ or $(H_i + 1)T_i$. Suppose in one cycle, there are n_1 intervals with length $H_i T_i$ and n_2 intervals with length $(H_i + 1)T_i$. We have

$$\begin{cases} n_1 + n_2 = N_i \\ n_1 H_i T_i + n_2 (H_i + 1) T_i = T_c. \end{cases} \tag{2.29}$$

Solving n_1 and n_2, we have

$$\begin{cases} n_1 = (H_i + 1)N_i - \frac{T_c}{T_i} \\ n_2 = \frac{T_c}{T_i} - H_i N_i. \end{cases} \tag{2.30}$$

Since (2.13) still holds in this case, we can substitute (2.30) into (2.13) and we have

$$\begin{aligned} \bar{A}_i^B &\geq \Big(\big((H_i + 1)N_i - \frac{T_c}{T_i}\big)(H_i T_i)^2 + (\frac{T_c}{T_i} - u_i N_i)\big((H_i + 1)T_i\big)^2 \Big) \frac{1}{2T_c} + \frac{1}{2} \\ &= \frac{T_i}{2}(2H_i + 1 - (H_i^2 + H_i)p_i) + \frac{1}{2} \\ &= \frac{T_i}{2} f(p_i) + \frac{1}{2}, \end{aligned}$$

where $f(p_i)$ is defined by

$$f(p_i) = 2\lfloor \frac{1}{p_i} \rfloor + 1 - \big(\lfloor \frac{1}{p_i} \rfloor^2 + \lfloor \frac{1}{p_i} \rfloor \big) p_i. \tag{2.31}$$

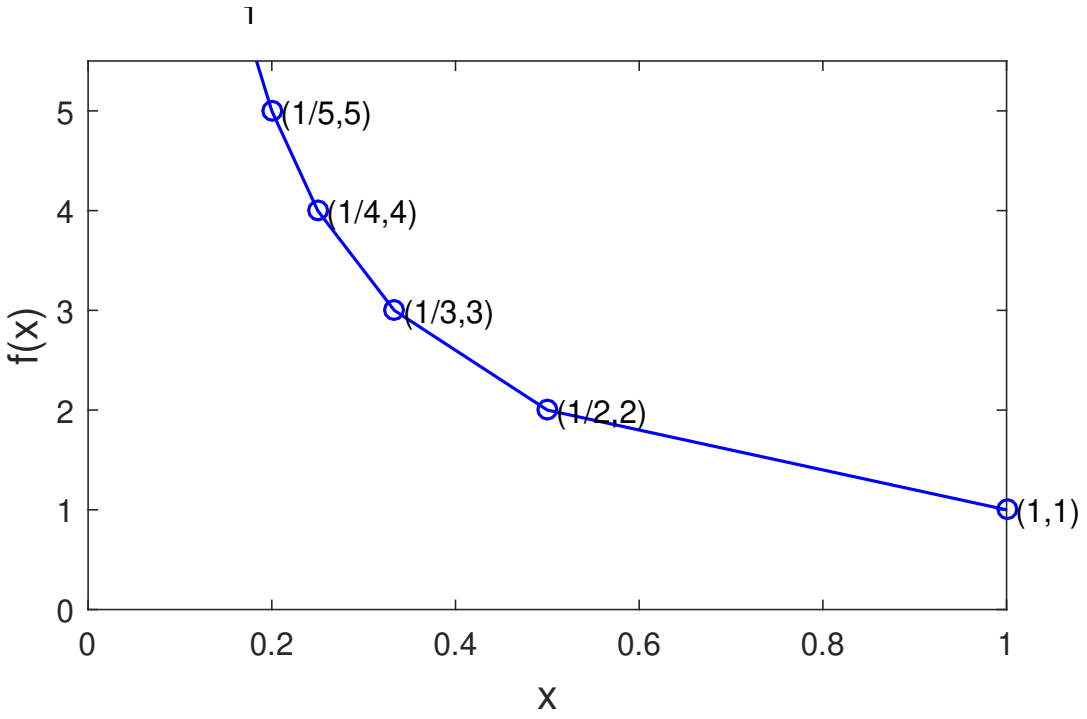


Figure 2.3: The graph of function f.

The graph of function f is shown in Fig. 2.3. We can see function f is a piecewise linear function. In particular, it connects each pair of adjacent points $(1/n, n)$ and $(1/(n+1), n+1)$, $n \in \mathbb{N}$.

To find a lower bound for $\bar{A}^{\mathrm{B}}$, we can use a lower bound for $\sum_{i=1}^{N} w_i(\frac{T_i}{2} f(p_i) + \frac{1}{2})$, which means we need to find the minimum of $\sum_{i=1}^{N} w_i T_i f(p_i)$. We have the following optimization problem:

$$\begin{aligned} \min_{p_i} \quad & \sum_{i=1}^{N} w_i T_i f(p_i) \\ \text{s.t.} \quad & \text{Constraints (2.27) and (2.28).} \end{aligned} \tag{2.32}$$

Look at the graph of function f, Fig. 2.3. We denote $c_i = f(p_i)$, then we have

$$c_i \geq 2u_i + 1 - (u_i^2 + u_i)p_i, \quad \forall u_i \in \mathbb{N}. \tag{2.33}$$

And the optimization problem (2.32) can be rewritten as

$$\begin{aligned} \min_{c_i,p_i} \quad & \sum_{i=1}^{N} w_i T_i c_i \\ \text{s.t.} \quad & \text{Constraints (2.27), (2.28) and (2.33).} \end{aligned} \tag{2.34}$$

Optimization problem (2.34) is a linear programming (LP) problem. However, we can not directly solve it by commercial LP solvers because there are infinite constraints lying in (2.33) (u_i can be any positive integer). To address this problem, we use the following procedure to solve optimization problem (2.34).

For each i, we reduce constraints (2.33) to a finite number of constraints:

$$c_i \geq 2u_i + 1 - (u_i^2 + u_i)p_i, \quad \forall u_i \in \mathbb{U}_i. \tag{2.35}$$

Here $\mathbb{U}_i$ is a finite subset of the set of natural numbers, $\mathbb{N}$. Compared to (2.33), in (2.35) we ignore some constraints to make the number of constraints finite. Then we construct a new optimization problem

$$\begin{aligned} \min_{c_i,p_i} \quad & \sum_{i=1}^{N} w_i T_i c_i \\ \text{s.t.} \quad & \text{Constraints (2.27), (2.28) and (2.35).} \end{aligned} \tag{2.36}$$

The new optimization problem (2.36) is a LP problem and can be easily solved by commercial solvers to get its optimal solution p_i^*. We have the following lemma.

Lemma 2.3. *If $\lfloor 1/p_i^* \rfloor \in \mathbb{U}_i$ for each i in the optimal solution to (2.36), then p_i^* is also an optimal solution to (2.34).*

Proof. Denote the optimal objective of (2.34) as J_1^*, and the optimal objective of (2.36) as J_2^*. Since the set of constraints of (2.36) is a subset of the set of constraints of (2.34), we

Algorithm 2.1 Key steps to solve optimization problem (2.34)

1: Decide the initial $\mathbb{U}_i$.

2: Solve (2.36), and get the optimal solution p_i^*

3: Check whether each $\lfloor 1/p_i^* \rfloor \in \mathbb{U}_i$ or not.

4: If Yes, stop and output p_i^* as the optimal solution to (2.34).

5: If No, expand those $\mathbb{U}_i$ to let each $\lfloor 1/p_i^* \rfloor \in \mathbb{U}_i$. Go to Step 2.

have $J_1^* \geq J_2^*$. By observing the graph of function f (see Fig. 2.3), we know that for any c_i and p_i, if $c_i \geq 2\lfloor \frac{1}{p_i} \rfloor + 1 - \left(\lfloor \frac{1}{p_i} \rfloor^2 + \lfloor \frac{1}{p_i} \rfloor \right) p_i$, then we have $c_i \geq 2u + 1 - (u^2 + u)p_i$ for any $u \in \mathbb{N}$. Therefore, if $\lfloor 1/p_i^* \rfloor \in \mathbb{U}_i$ for each i in the solution to (2.36), then p_i^* is also a feasible solution to (2.34) with the objective J_2^*. Recall $J_1^* \geq J_2^*$. So J_2^* is the optimal objective of (2.34) , and p_i^* is the optimal solution to (2.34). This completes our proof. □

We then propose a computation procedure (shown in Algorithm 2.1) to solve the optimization problem (2.34), which iteratively expands $\mathbb{U}_i$ until $\lfloor 1/p_i^* \rfloor \in \mathbb{U}_i$ for each i. At the end of this procedure we have $\lfloor 1/p_i^* \rfloor \in \mathbb{U}_i$ for each i. By Lemma 2.3, p_i^* is the optimal solution to (2.34).

With the optimal solution to (2.34), a lower bound of $\bar{A}^{\mathrm{B}}$, denoted by $\alpha_{(\mathrm{PRD},M)}$, is given by

$$\alpha_{(\mathrm{PRD},M)} = \sum_{i=1}^{N} w_i \left(\frac{T_i}{2} f(p_i^*) + \frac{1}{2} \right). \tag{2.37}$$

In the above derivation for $\alpha_{(\mathrm{PRD},M)}$, we consider the impacts of both link capacity and source sampling. With consideration of the fact that $f(p_i) \geq 1/p_i$ and $f(p_i) \geq 1$, we can find $\alpha_{(\mathrm{PRD},M)}$ is always tighter than both $\alpha_{(\mathrm{PTS},M)}$ and $\alpha_{(\mathrm{ARB},\infty)}$. Therefore, $\alpha_{(\mathrm{PRD},M)}$ is always tighter than the lower bound for arbitrary sampling, $\alpha_{(\mathrm{ARB},M)}$.

2.7 Juventas: A Near-Optimal Scheduler

In this section, we propose a low-complexity scheduling algorithm, code named Juventas[3], in the reduced search space derived in Section 2.5. Under arbitrary sampling, we want to develop an order-based scheduling algorithm, which requires us to assign each node an order based on $A_i^{\mathrm{s}}(t)$, $A_i^{\mathrm{B}}(t)$, w_i, L_i in each time slot. In the following we will design this order.

For source node i, suppose transmission of a sample begins at t_1 and ends at t_2 ($t_2 \geq t_1$). Then, at time slot (t_2+1), based on (2.3), we have

$$A_i^{\mathrm{B}}(t_2+1) = A_i^{\mathrm{s}}(t_1) + t_2 - t_1 + 1. \tag{2.38}$$

On the other hand, if during the same time interval $[t_1, t_2]$, source node i is not scheduled for any transmission, then based on (2.3), we have

$$A_i^{\mathrm{B}}(t_2+1) = A_i^{\mathrm{B}}(t_1) + t_2 - t_1 + 1. \tag{2.39}$$

Note that $A_i^{\mathrm{B}}(t_2+1)$ in (2.39) is greater than $A_i^{\mathrm{B}}(t_2+1)$ in (2.38) if the sample does not complete its transmission by time t_2. So the benefit of completing transmission of this sample by t_2 (in terms of decrease of $A_i^{\mathrm{B}}(t_2+1)$) is the difference on the RHS in (2.39) and (2.38), i.e.,

$$A_i^{\mathrm{B}}(t_1) - A_i^{\mathrm{s}}(t_1).$$

Note that this decrease of $A_i^{\mathrm{B}}(t)$ after t_2 is dependent on AoI difference between the BS and source node i at t_1. So the amount of AoI decrease at the BS w.r.t. source node i when a sample completes its transmission has already been determined by AoI status at an *earlier* time slot, i.e., the time slot when the sample starts its transmission.

Suppose the transmission of a sample at source node i starts at time slot t. Denote $\Delta_i(t)$

[3]Juventas is the ancient Roman goddess for youth and rejuvenation.

as the AoI *outage* which is given by

$$\Delta_i(t) = A_i^{\mathrm{B}}(t) - A_i^{\mathrm{s}}(t). \tag{2.40}$$

At each time slot t, we will use $\Delta_i(t)$ to make a scheduling decision to transmit new samples.[4] The motivation is intuitive: serving the node with largest $\Delta_i(t)$ will offer the greatest relieve in reducing its AoI at the BS. However, $\Delta_i(t)$ alone is not sufficient to be the scheduling metric. Both the weight w_i and packet size L_i must also be taken into considerations, as shown in (2.6). Therefore, we propose to use $\sqrt{w_i/L_i}\cdot\Delta_i(t)$ as the scheduling metric deciding the order for source node i. The source node with the largest value of $\sqrt{w_i/L_i}\Delta_i(t)$ will first be selected for transmission and the BS will allocate as many transmission units as available to transmit this sample before considering others.[5] The key steps of Juventas are shown in Algorithm 2.2.

Besides, under periodic sampling, Juventas always makes the same scheduling scheduling decision under the same network state (recall the definition of state in Section 2.5.2). It's obvious that for the periodic sampling case under Juventas $A_i^{\mathrm{B}}(t)$ for each source node cannot go to infinity. Therefore, following the same way in the proof of Lemma 2, we can show Juventas must visit a single state for at least 2 times. Once Juventas visits the same state twice, since its scheduling decisions at the two time slot are identical, their following states (states at the next time slots) are also identical. Then clearly, we can see Juventas follows a cyclic pattern, in which the cycle is the time interval between the two identical states.

[4]For a sample that is not finished in the previous time slot, Juventas will use as many transmission units as needed in the current time slot to complete it (before allocating transmission units to start new samples), as shown in Algorithm 2.2.

[5]Our idea is corroborated by the scheduling algorithm in [13] under per time slot scheduling with L_i and $M = 1$. The authors developed a near-optimal scheduling algorithm that allocates transmission rate in proportional to $\sqrt{w_i}$. Incidentally, Juventas performs better than this scheduling algorithm even in the same simple case with $M = 1$, $L_i = 1$ and per time slot sampling. This is because we make scheduling decision in each time slot while the one in [13] makes global scheduling decision at $t = 0$.

Algorithm 2.2 Key steps of Juventas algorithm.

For each time slot t, do the following:

1: Complete transmission of the un-completed sample from the previous time slot (if there is any).

2: Among all other source nodes, find node i with the largest $\sqrt{w_i/L_i} \cdot \Delta_i(t)$.

3: If L_i is less than or equal to the number of remaining transmission units, complete transmission of this sample. Go to Step 3.

4: If L_i is greater than the number of remaining transmission units, transmit this sample incompletely with all remaining transmission units.

Therefore, Juventas is an order-based algorithm under arbitrary sampling and a cyclic algorithm under periodic sampling. So we can say Juventas lies in the reduced search space that we have derived in Section 2.5.

The following theorem offers a performance guarantee of Juventas (with a factor 3) when $L_i \leq M$.[6]

Theorem 2.1. *Under arbitrary sampling, if $L_i \leq M$ for each source node i, $\bar{A}^B$ under Juventas scheduling algorithm is upper bounded by*

$$\bar{A}^B \leq 3\bar{A}^* + \sum_{i=1}^{N} w_i \tag{2.41}$$

where $\bar{A}^*$ is the optimal objective at the BS.

A proof of Theorem 2.1 is given in Appendix A.

[6]The condition $L_i \leq M$ can be easily justified in the real world where the sample taken from a source node (e.g., sensor) is almost always smaller than the cellular transmission rate in a TTI (M).

Table 2.2: Simulation Parameters

Type	1	2	3	4	5	6	7	8	9	10
w_i	6	33	25	39	36	46	35	17	35	10
L_i	1	15	11	10	19	13	13	18	17	12
T_i	10	12	45	2	25	9	49	36	26	24
p_i	0.03	0.05	0.10	0.24	0.19	0.18	0.20	0.13	0.06	0.11

2.8 Simulation Results

In this section, we evaluate the performance of Juventas. In our simulation, we assume the source nodes can be classified into 10 groups, with each group having the same number of source nodes. The weight, sampling rate, and sample size for the source nodes within the same group are identical but differ from those in a different group. The weight of each source node is normalized w.r.t. $\sum_{i=1}^{N} w_i$ before each simulation (i.e., after normalizing we have $\sum_{i=1}^{N} w_i = 1$). For each simulation, we start at time slot $t = 1$, and terminate when the BS has received at least 100 samples from each source node at time slot T_{term}. Then we calculate $\bar{A}^{\text{B}} = (1/T_{\text{term}}) \cdot \sum_{t=1}^{T_{\text{term}}} \sum_{i=1}^{N} w_i A_i^{\text{B}}(t)$.

We consider the three sampling behaviors: per time slot sampling, periodic sampling and random sampling respectively, and evaluate the performance of Juventas. Under periodic sampling, we further explore the impact of synchronization among sources on the performance of Juventas.

2.8.1 Per Time Slot Sampling

In this section we evaluate the performance of Juventas under per time slot sampling (i.e., each source node samples information at every time slot). We also show the lower bound for per time slot sampling, $\alpha_{(\text{PTS},M)}$, in the figures as a benchmark.

(i) With the parameter settings w_i and L_i given in Table 2.2, $N = 100$, and $T_i = 1$ for

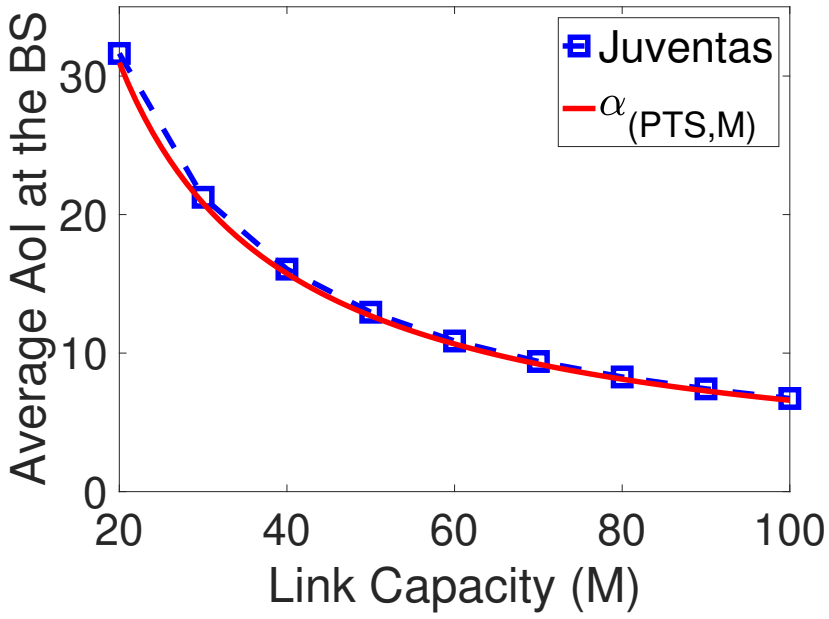


Figure 2.4: Per time slot sampling: AoI for varying M

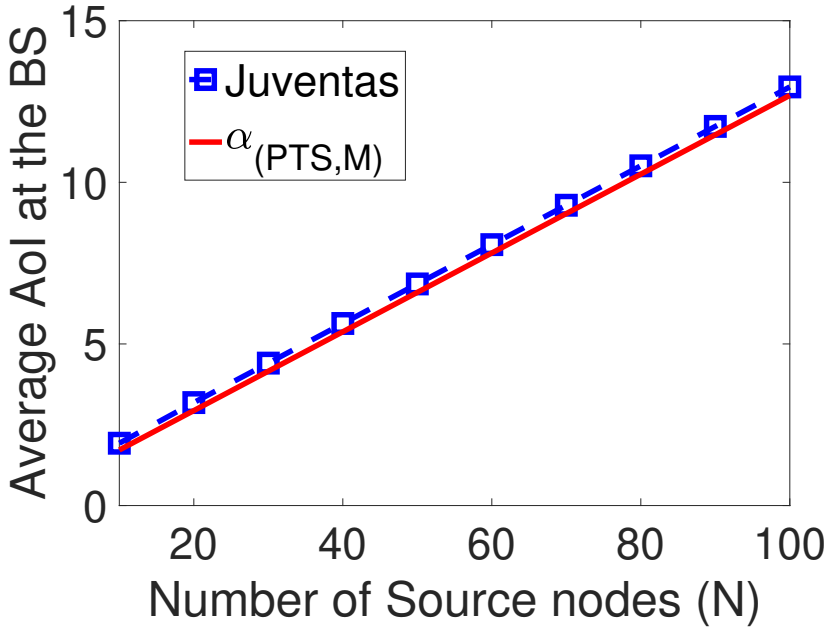


Figure 2.5: Per time slot sampling: AoI for varying N

each i, Fig. 2.4 shows the objective value, $\bar{A}^{\mathrm{B}}$, as a function of increasing link capacity M. We see that $\bar{A}^{\mathrm{B}}$ for Juventas decreases monotonically as M increases. Also shown in this figure is the lower bound for periodic sampling $\alpha_{(\mathrm{PTS},M)}$ derived in (2.20). Clearly, we see that Juventas can achieve near-optimal performance.

(ii) With the parameter settings w_i, L_i, given in Table 2.2, $M = 50$, and $T_i = 1$ for each i, Fig. 2.5 shows the objective value, $\bar{A}^{\mathrm{B}}$, as a function of increasing number of source nodes N. We see that $\bar{A}^{\mathrm{B}}$ for Juventas increases monotonically as N increases. The lower bound $\alpha_{(\mathrm{PTS},M)}$ is also shown in this figure, and we see that Juventas can achieve near-optimal performance.

If we compare per time slot sampling with other sampling behaviors (periodic sampling and random sampling), we can find the gap between the performance of Juventas and the lower bound is smallest under per time slot sampling (we can hardly distinguish the gaps in Fig. 2.4 and 2.5). Therefore, in terms of the gap between the performance and the lower bound, Juventas performs best under per time slot sampling.

2.8.2 Periodic Sampling

In this section we evaluate the performance of Juventas under periodic sampling (i.e., each source node i has a sampling cycle T_i, and it samples information at every T_i time slots). We also show the lower bound for periodic sampling, $\alpha_{(\mathrm{PRD},M)}$, in the figures as a benchmark.

(i) With the parameter settings w_i, L_i, and T_i given in Table 2.2 and $M = 50$, Fig. 2.6 shows the objective value, $\bar{A}^{\mathrm{B}}$, as a function of increasing link capacity M. We see that $\bar{A}^{\mathrm{B}}$ for Juventas decreases monotonically as M increase. Also shown in this figure is the lower bound for periodic sampling $\alpha_{(\mathrm{PRD},M)}$ derived in (2.37). we see that Juventas can achieve near-optimal performance.

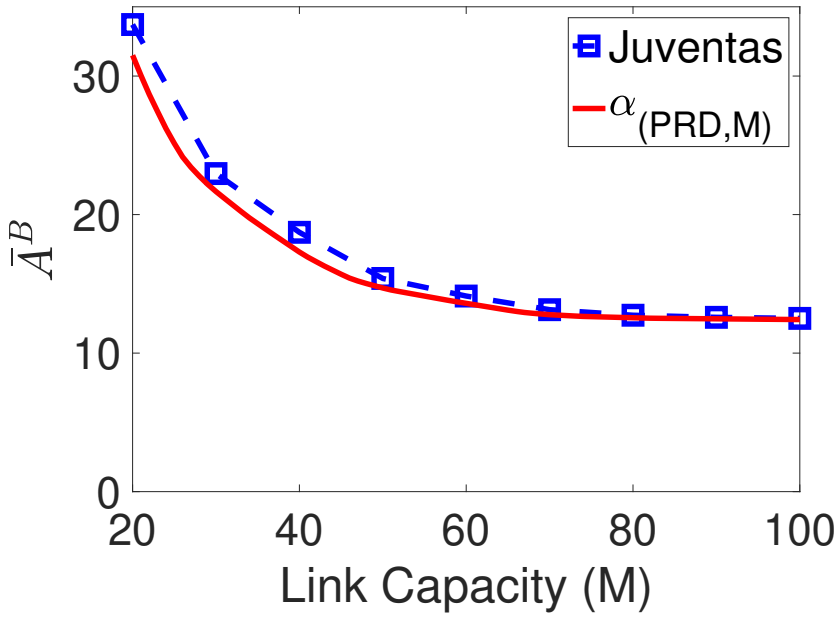


Figure 2.6: Periodic sampling: AoI for varying M

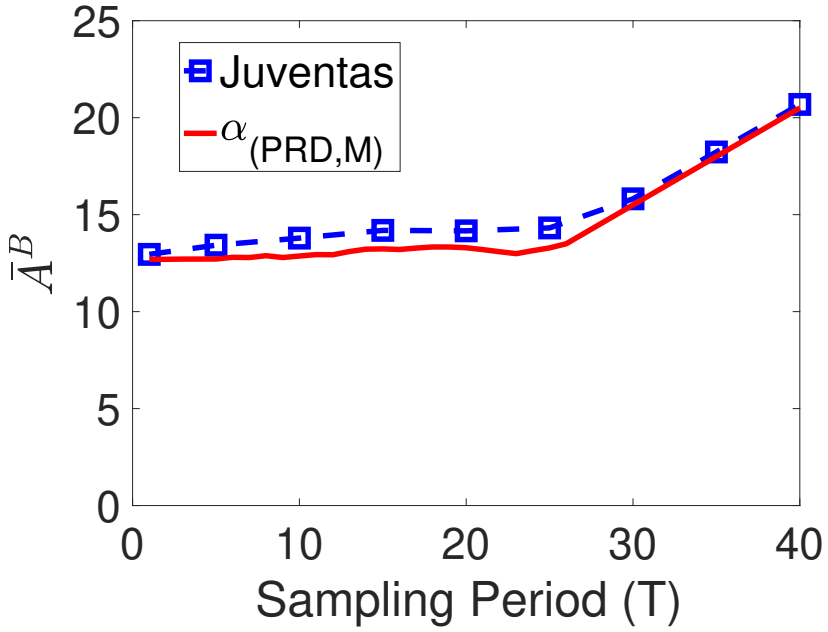


Figure 2.7: Periodic sampling: AoI for varying T

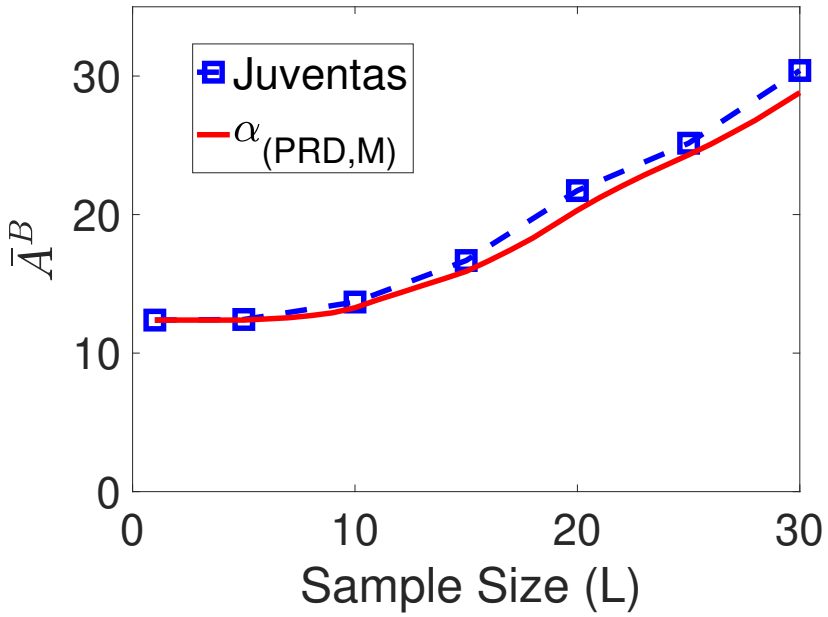


Figure 2.8: Periodic sampling: AoI for varying L

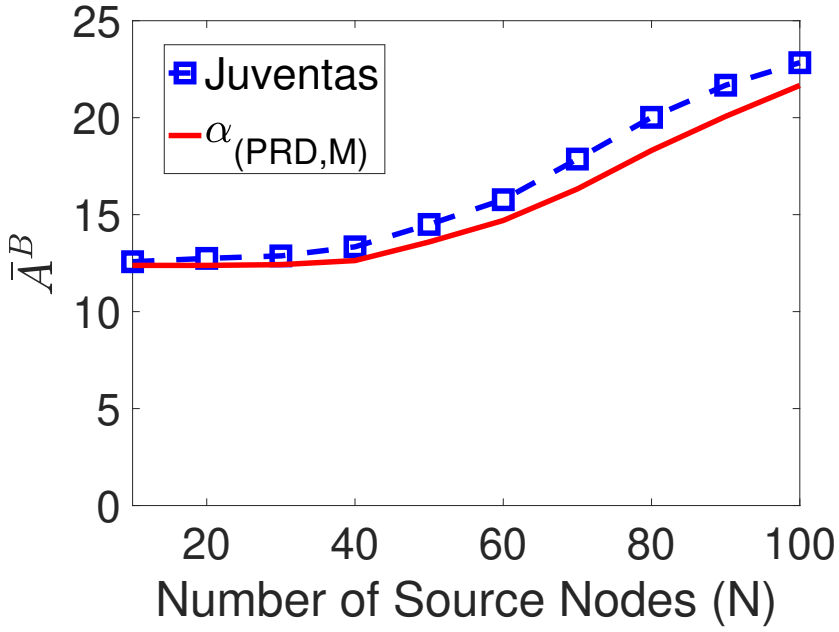


Figure 2.9: Periodic sampling: AoI for varying N

(ii) With the parameter settings w_i and L_i given in Table 2.2, $M = 50$ and $N = 100$, Fig. 2.7 shows the objective value, $\bar{A}^{\mathrm{B}}$, as a function of increasing sampling cycle T. Here, all source nodes use the same sampling cycle T. The lower bound $\alpha_{(\mathrm{PRD},M)}$ is also shown in this figure, and we see that Juventas can achieve near-optimal performance. Note that $f(p_i)$ in (2.31) is a piecewise function so $\alpha_{(\mathrm{PRD},M)}$ doesn't increase monotonically as T increases.

(iii) With the parameter settings w_i and T_i given in Table 2.2, $M = 50$ and $N = 100$, Fig. 2.8 shows the objective value, $\bar{A}^{\mathrm{B}}$, as a function of increasing sample size L. Here, all source nodes have the same sample size L. We see that $\bar{A}^{\mathrm{B}}$ for Juventas increases monotonically as L increases. The lower bound $\alpha_{(\mathrm{PRD},M)}$ is also shown in this figure, and we see that Juventas can achieve near-optimal performance.

(iv) With the parameter settings w_i, L_i, and T_i given in Table 2.2 and $M = 50$ Fig. 2.9 shows the objective value, $\bar{A}^{\mathrm{B}}$, as a function of increasing number of source nodes N. We see that $\bar{A}^{\mathrm{B}}$ for Juventas increases monotonically as N increases. The lower bound $\alpha_{(\mathrm{PRD},M)}$ is also shown in this figure, and we see that Juventas can achieve near-optimal performance.

2.8.3 Random Sampling

In this section we evaluate the performance of Juventas under random sampling. The sampling at each source node is modeled as an independent Bernoulli process. Specifically, each source node i samples information in probability p_i at each time slot independently. Since we haven't developed a specific lower bound for the Bernoulli sampling (random sampling) case, we have to use the most general one, $\alpha_{(\mathrm{ARB},M)}$, which is valid for arbitrary sampling, as the benchmark.

(i) With the parameter settings w_i, L_i, and p_i given in Table 2.2 and $M = 50$, Fig. 2.10 shows the objective value, $\bar{A}^{\mathrm{B}}$, as a function of increasing link capacity M. We see that $\bar{A}^{\mathrm{B}}$

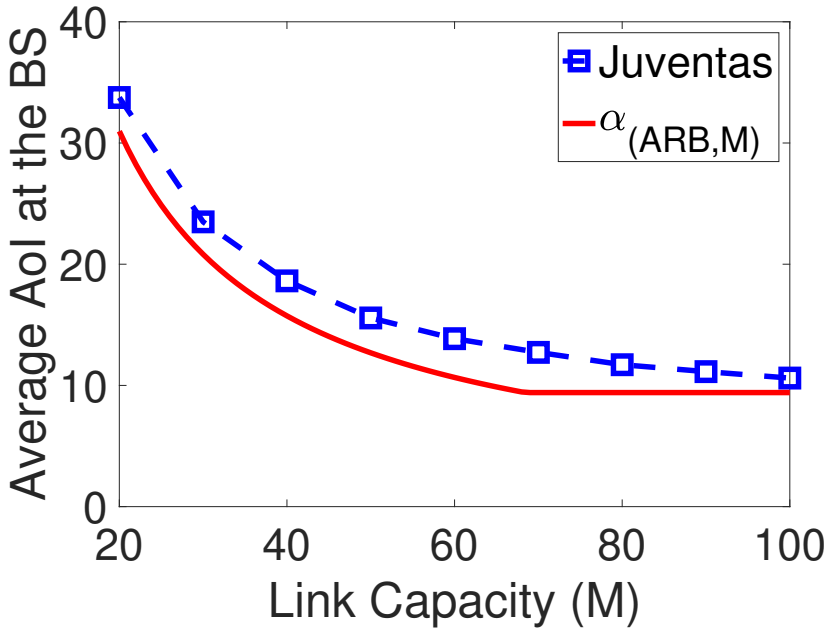


Figure 2.10: Random sampling: AoI for varying M

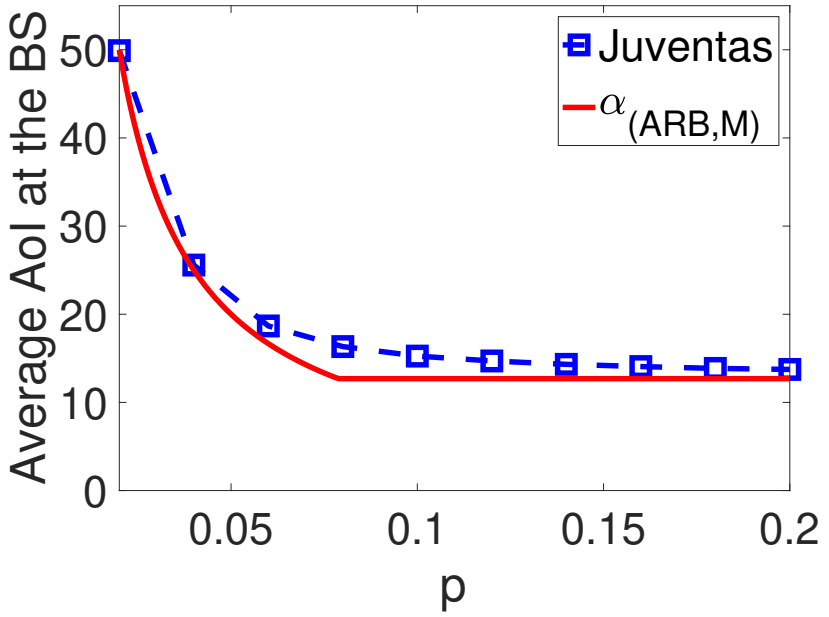


Figure 2.11: Random sampling: AoI for varying p

for Juventas decreases monotonically as M increase. Also shown in this figure is the lower bound for periodic sampling $\alpha_{(\mathrm{ARB},M)}$ derived in (2.25). we see that Juventas can achieve near-optimal performance.

(ii) With the parameter settings w_i, and L_i given in Table 2.2, $M = 50$ and $N = 100$, Fig. 2.11 shows the objective value, $\bar{A}^{\mathrm{B}}$, as a function of increasing sampling cycle p. Here, all source nodes use the same sampling cycle p. The lower bound $\alpha_{(\mathrm{ARB},M)}$ is also shown in this figure, and we see that Juventas can achieve near-optimal performance.

(iii) With the parameter settings w_i, and p_i given in Table 2.2, $M = 50$ and $N = 100$, Fig. 2.12 shows the objective value, $\bar{A}^{\mathrm{B}}$, as a function of increasing sample size L. Here, all source nodes have the same sample size L. We see that $\bar{A}^{\mathrm{B}}$ for Juventas increases monotonically as L increases. The lower bound $\alpha_{(\mathrm{ARB},M)}$ is also shown in this figure, and we see that Juventas can achieve near-optimal performance.

(iv) With the parameter settings w_i, L_i, and p_i given in Table 2.2 and $M = 50$, Fig. 2.13 shows the objective value, $\bar{A}^{\mathrm{B}}$, as a function of increasing number of source nodes N. We see that $\bar{A}^{\mathrm{B}}$ for Juventas increases monotonically as N increases. The lower bound $\alpha_{(\mathrm{ARB},M)}$ is also shown in this figure, and we see that Juventas can achieve near-optimal performance.

Compared to the cases of per time slot sampling and periodic sampling, in the case of random sampling the gap between the performance of Juventas and the lower bound is larger. The reason is that for per time slot sampling and periodic sampling, we have specifically derived tight lower bounds $\alpha_{(\mathrm{PTS},M)}$ and $\alpha_{(\mathrm{PRD},M)}$. However, we don't have specific lower bound for random sampling, and we use the lower bound for arbitrary sampling $\alpha_{(\mathrm{ARB},M)}$ as the benchmark, which is not very tight.

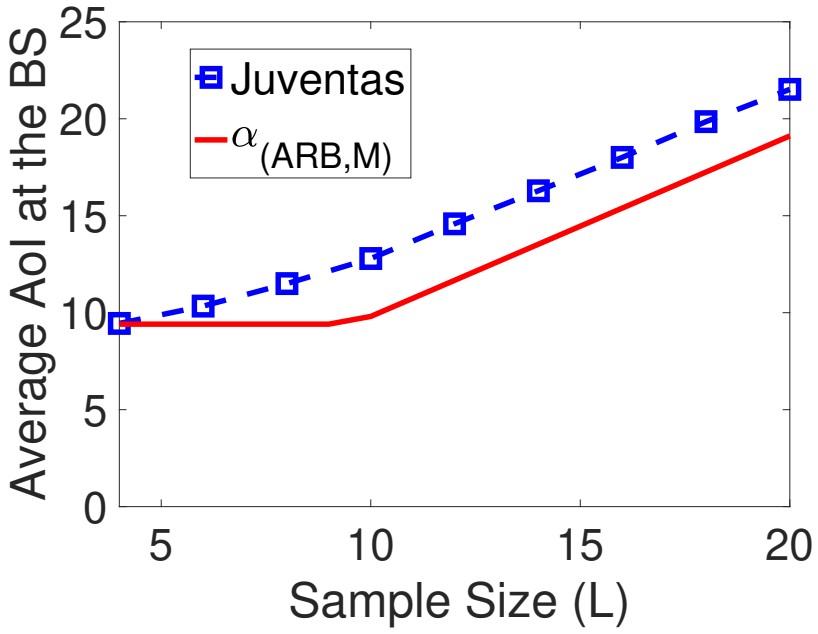


Figure 2.12: Random sampling: AoI for varying L

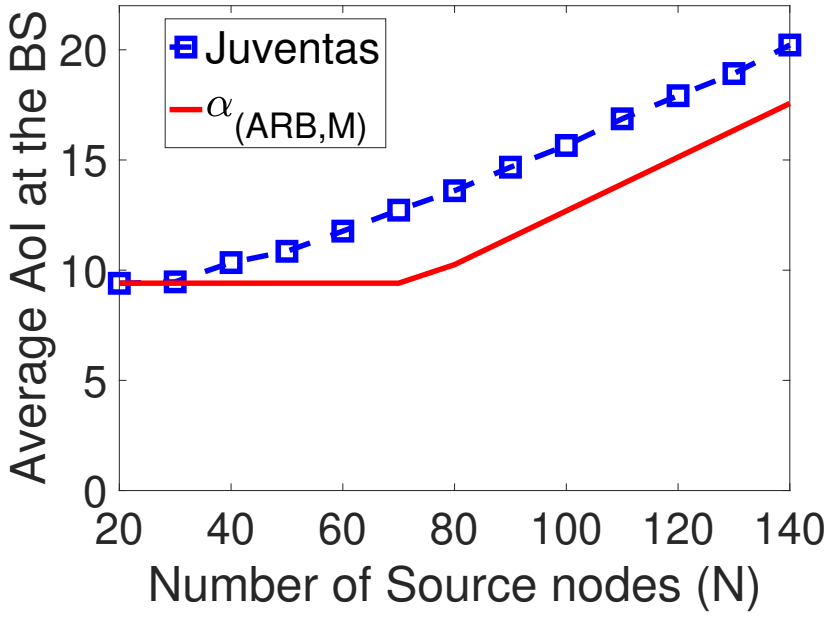


Figure 2.13: Random sampling: AoI for varying N

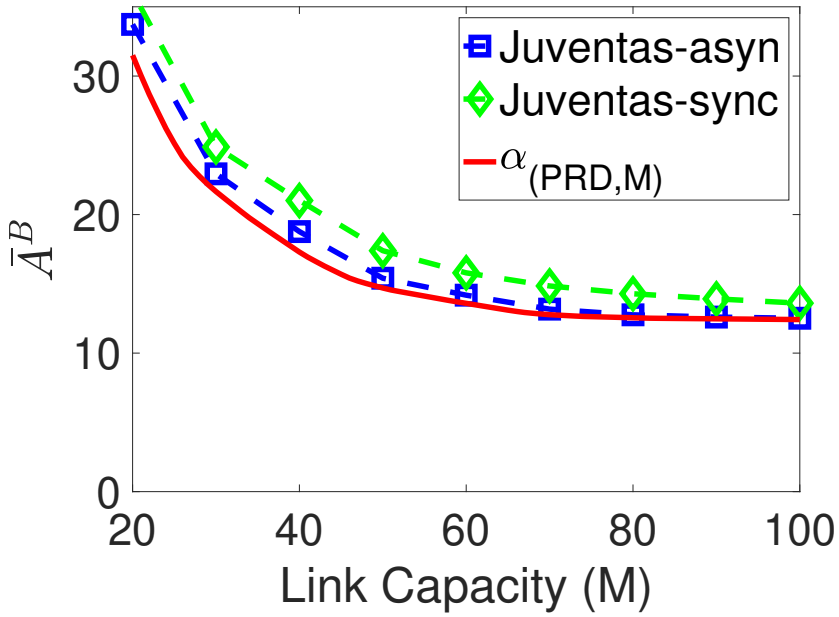


Figure 2.14: Periodic sampling: weak synchronization

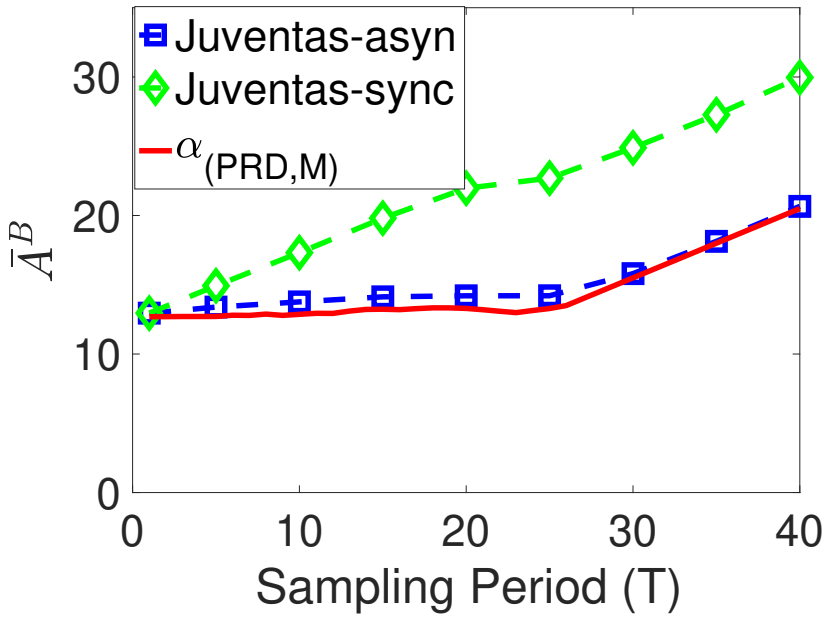


Figure 2.15: Periodic sampling: strong synchronization

2.8.4 Synchronization for Periodic Sampling

Finally, we explore the impact of synchronization in sampling on the performance of Juventas. If two source nodes have the same sampling cycle T_i and the same initial state, $A_i^{\mathrm{s}}(0)$, we say they are synchronized. In all the simulations in Section 2.8.2, the source nodes are not synchronized, either with different sampling rates or different initial states. We now study the impact of synchronization. In the first scenario, we consider synchronization only within each type of nodes (weak synchronization). In the second scenario, we consider synchronization among all source nodes (strong synchronization).

Fig. 2.14 shows the results under weak synchronization (with the same parameter settings as in Fig. 2.6). We see that the objective $\bar{A}^{\mathrm{B}}$ under weak synchronization is slightly larger than that under no synchronization. Fig. 2.15 shows the results under strong synchronization (with the same parameter settings as in Fig. 2.7). We see that the objective $\bar{A}^{\mathrm{B}}$ under strong synchronization is considerably larger than that under no synchronization. Based on the results in Fig. 2.14 and Fig. 2.15, we conclude that synchronization is harmful to AoI performance and should be avoided or minimized when we initialize the source nodes.

Note that synchronization only happens under periodic sampling when there are multiple source nodes having an identical sampling cycle. Under per time slot sampling, each source node samples information at every time slot, so we don't need to consider the initial states of the source nodes. Under random sampling, since each source node samples information independently, we can safely say after a few time slots all source nodes are not synchronized in general, even if some source nodes have an identical p_i.

2.9 Chapter Summary

Most of existing research on minimizing the AoI was based on overly simplified models that are hardly useful for real world IoT applications. In this chapter, we addressed this important issue by generalizing three key aspects in AoI research: sampling behavior, sample size, and transmission capacity. Under these three generalizations, we developed two fundamental properties to reduce the search space and derived tight lower bounds for an optimal solution. Further, we developed a low-complexity scheduling algorithm called Juventas, that was shown to offer near-optimal performance when there is no synchronization among the source nodes and have a guaranteed performance (within a factor of 3). The results in this chapter made significant advance in bridging the gap between AoI theory and IoT data collection in practice.

Chapter 3

Minimizing AoI in a 5G-based IoT Network under Varying Channel Conditions

3.1 Introduction

In Chapter 2, we study the average AoI minimization problem under more general settings than those in the state-of-the-art. In this chapter, we extend the results in Chapter 2 by focusing on the design of 5G-compliant AoI scheduler and address the impact of both time and frequency-varying channel conditions, which are not considered in Chapter 2. We also want to ensure that our AoI scheduler can strictly meet the stringent timing requirement (i.e., sub-millisecond running time for computing scheduling solution) as specified in 5G standard.

There are a number of technical challenges in this research. First, in a 5G setting, the AoI scheduling problem entails allocation of resource blocks (RBs) and selection of modulation and coding scheme (MCS) for each source node in each TTI based on the channel conditions. This presents a much larger search space for an optimal solution than any of those problems considered to date in the AoI research literature. Second, the stringent timing requirement for real-world 5G systems (i.e., sub-millisecond time scale) sets a hard performance benchmark against any design of AoI schedulers. As we shall see, it is extremely challenging to find a near-optimal solution for a problem of such size and complexity in sub-millisecond time scale.

This chapter addresses the above technical challenges with the following contributions:

- This chapter studies AoI scheduling in a 5G setting with considerations of varying channel conditions. Specifically, the uplink transmission resource is divided as grids of RBs that span both time and frequency domains with different channel conditions. The scheduling problem under 5G entails RB allocation to each source node and selection of MCS for each source node based on the channel condition on each RB, with the goal of minimizing long-term average AoI.

- Since the channel condition for the future is unknown, we purse to design an online AoI scheduling algorithm. For performance benchmark, We propose a novel computational procedure to find an asymptotic lower bound for the objective value. Specifically, we first relax the original AoI minimization problem to a data rate minimization problem. Then we employ a gradient scheduling algorithm to find an asymptotic lower bound for the latter problem. The gradient scheduling minimizes an empirical data rate for each TTI, which can be formulated into an integer quadratic programming (IQP) problem and solved by the CPLEX solver.

- For our AoI scheduling problem, we present Kronos—an online algorithm that conforms to 5G transmission standard and can cope varying channel conditions. The essence of Kronos is to iteratively select a source node for RB allocation until all RBs in a TTI are allocated. We propose a novel metric that takes into consideration of AoI outage and the channel conditions for the source node. Based on this metric, we can identify the next source node for RB allocation and determine its MCS.

- To ensure Kronos can meet the stringent timing requirement in 5G, we propose to employ commercial off-the-shelf (COTS) GPUs to speed up Kronos' running time through parallel computation. Specifically, we exploit the massive number of GPU processing

Table 3.1: Notation

Symbol	Definition
$A_i^{\mathrm{s}}(t)$	AoI of the most recent sample at source node i at TTI t
$A_i^{\mathrm{B}}(t)$	AoI of the sample from source node i at the BS at TTI t
$\bar{A}_i^{\mathrm{B}}$	Long term average of $A_i^{\mathrm{B}}(t)$
$\bar{A}^{\mathrm{B}}$	Weighted sum of all $\bar{A}_i^{\mathrm{B}}$'s
$\mathcal{B}$	A set of RBs available for scheduling at the 5G BS
c^m	Modulation and coding rate under MCS level m
$r_i^{b,m}(t)$	Achievable data rate by RB b w.r.t. source node i under MCS m at TTI t
L_i	Sample size at source node i
$\mathcal{M}$	A set of MCS levels that a source node can use
$\mathcal{N}$	A set of source nodes in the 5G-based IoT network
$q_i^b(t)$	The highest MCS level that source node i's channel can support on RB b at TTI t
$R_i(t)$	Amount of information transmitted by source node i in TTI t across all RBs allocated to it
T_i	Sampling period at source node i (in unit of TTIs)
$U_i^{\mathrm{B}}(t)$	Generation time of the most recent sample from source node i at the BS at TTI t
$U_i^{\mathrm{s}}(t)$	Generation time of the most recent sample at source node i at TTI t
w_i	An assigned weight for source node i at the BS

cores to compute and compare the scheduling metric for all possible combinations of source nodes and MCSs. For proof-of-concept, we implement Kronos on an Nvidia Tesla V100 GPU using the CUDA programming model. Through extensive performance evaluation, we find that Kronos can achieve near-optimal performance (when compared to our lower bound) in sub-millisecond time scale.

3.2 A 5G-based IoT Architecture

Consider a 5G-based IoT network where a set $\mathcal{N}$ of source nodes collect information and forward it to a base station (BS) (see Fig. 1.1). Each source node periodically takes a sample of information from its environment. Denote T_i as the sampling period (in unit of time slots)

at source node i. Due to the heterogeneity of IoT applications, the sampling periods are generally different among different source nodes. For source node i, denote L_i as the sample size (in unit of bits), which is the amount of information carried in a sample. Again, due to the heterogeneity of IoT devices, sample sizes are generally different among different source nodes.

Once a sample is produced at a source node i, it is stored in a local memory, which is of finite size. Due to limited channel bandwidth, not every sample from each source node will be transmitted to the BS. When a source node starts a new transmission, it always selects the freshest sample (i.e., the most recently generated sample) for transmission. This is to ensure the cellular BS can receive the freshest information. As a result, only a fraction of samples generated at each source node will be transmitted while the rest will be eventually discarded at the source nodes (due to limited node memory). Once a source starts to transmit a sample, it may take multiple consecutive time slots (if necessary) to complete the transmission. In this case, any newly generated sample afterward cannot preempt an ongoing transmission of an older sample.

Since the uplink transmission from IoT source nodes to the BS conforms to 5G standard [31], transmission resource is organized into grids of *resource blocks* (RBs) that span both time and frequency domains. In the time domain, time is equally slotted into *transmission time intervals* (TTIs), while in the frequency domain, bandwidth is equally slotted among a large number of sub-carriers, with 12 sub-carriers over an TTI being called an RB. That is, for each TTI, there is a large number of RBs that can be allocated to the source nodes for uplink transmission.

Due to varying channel conditions in time (across different TTIs, i.e., time-selective fading) and frequency (across different RBs, i.e., frequency-selective fading), channel feedback from each source node is necessary for optimal scheduling of RBs at the BS. Given such

channel variation over time and frequency, it is desirable to perform scheduling for each TTI, the smallest time resolution for 5G transmission.

At the BS, the collected information can be either processed and stored locally (edge computing) and/or be forwarded to a cloud, where the information can be further processed and accessed broadly by users from any location. Since many time-sensitive IoT applications need to access the latest sampled information from each source, it is desirable to maintain the freshest sample (from each source) at the edge BS. So it is necessary to design a specialized scheduler to minimize AoI for the maintained samples at the BS. This is the problem we are going to study in this chapter.

3.3 Modeling and Problem Statement

Table 3.1 lists key notations in this chapter. When there is no ambiguity, we use the term "TTI" and "time slot" interchangeably throughout this chapter. Also, wherever appropriate, we use the term "at TTI t" to refer to "at the beginning of TTI t" and use the term "in TTI t" to refer to the underlying action is completed "at the end of TTI t".

3.3.1 AoI Notation

Recall that for each source node i, it takes a sample every T_i time slots. Denote $U_i^{\mathrm{s}}(t)$ as the generation time of the most recent sample at source node i. Clearly, $U_i^{\mathrm{s}}(t) \leq t$. Then the AoI of the most recent sample at source node i at TT t, denoted as $A_i^{\mathrm{s}}(t)$ is defined as:

$$A_i^{\mathrm{s}}(t) = t - U_i^{\mathrm{s}}(t). \tag{3.1}$$

Since sampling at source node i has a period T_i, function $A_i^{\mathrm{s}}(t)$ exhibits a zigzag shape with slope 1 and period T_i. Clearly, we have $0 \leq A_i^{\mathrm{s}}(t) \leq T_i - 1$.

On the other hand, the sample maintained at the BS may be older than the sample just produced at the source node. Denote $U_i^{\mathrm{B}}(t)$ is the generation time of the most recently received sample from source node i at the BS. Then the AoI for source i at the BS at time slot t, denoted as $A_i^{\mathrm{B}}(t)$, is defined as

$$A_i^{\mathrm{B}}(t) = t - U_i^{\mathrm{B}}(t). \tag{3.2}$$

Function $A_i^{\mathrm{B}}(t)$ also exhibits a zigzag shape with slope 1, and we have $A_i^{\mathrm{B}}(t) \geq A_i^{\mathrm{s}}(t)$.

It is instructive to make a connection between $A_i^{\mathrm{B}}(t)$ (AoI at edge BS) and $A_i^{\mathrm{s}}(t)$ (AoI at a source node). At source node i, for the k-th sample that is successfully transmitted to the BS (excluding those that are not transmitted), denote its beginning transmission TTI as $b_i(k)$ and ending transmission TTI as $e_i(k)$. By the definition of $U_i^{\mathrm{s}}(t)$, the generation time of this k-th sample is $U_i^{\mathrm{s}}\big(b_i(k)\big)$. After the last unit of data of this sample is completely sent to the BS in TTI $e_i(k)$, in the next TTI $(e_i(k)+1)$, $U_i^{\mathrm{B}}(t)$ is updated and we have $U_i^{\mathrm{B}}(e_i(k)+1) = U_i^{\mathrm{s}}\big(b_i(k)\big)$. By the definitions of $A_i^{\mathrm{B}}(t)$ and $A_i^{\mathrm{s}}(t)$, it can be shown that AoI evolution at the BS follows the following expression:

$$A_i^{\mathrm{B}}(t+1) = \begin{cases} A_i^{\mathrm{s}}\big(b_i(k)\big) + e_i(k) - b_i(k) + 1, & \text{if } t = e_i(k), \\ A_i^{\mathrm{B}}(t) + 1, & \text{otherwise.} \end{cases} \tag{3.3}$$

The long term average of $A_i^{\mathrm{B}}(t)$ for source node i at the BS is defined as:

$$\bar{A}_i^{\mathrm{B}} = \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} A_i^{\mathrm{B}}(t). \tag{3.4}$$

The BS assigns a weight for each source i, which is denoted by w_i. Then the weighted sum of long term average of $A_i^{\mathrm{B}}(t)$ over all source nodes $i \in \mathcal{N}$, denoted as $\bar{A}^{\mathrm{B}}$, is:

$$\bar{A}^{\mathrm{B}} = \sum_{i\in\mathcal{N}} w_i \cdot \bar{A}_i^{\mathrm{B}}. \tag{3.5}$$

Our objective is to minimize $\bar{A}^{\mathrm{B}}$.

3.3.2 Uplink Transmission

As shown in Fig. 1.1, the set of IoT source nodes (users) $\mathcal{N}$ share an uplink channel to transmit to the BS. Denote $\mathcal{B}$ as the set of RBs in one TTI for uplink transmission. The scheduler at the BS must allocate this set $\mathcal{B}$ of RBs to a subset of source nodes at each TTI to minimize $\bar{A}^{\mathrm{B}}$.

Denote $x_i^b(t)$ as a binary variable indicating whether RB $b \in \mathcal{B}$ is allocated to source node i at TTI t, i.e.,

$$x_i^b(t) = \begin{cases} 1 & \text{if RB } b \text{ is allocated to node } i \text{ at TTI } t, \\ 0 & \text{otherwise.} \end{cases}$$

We assume each RB can only be allocated to at most one source node.[1] We have:

$$\sum_{i \in \mathcal{N}} x_i^b(t) \leq 1 \quad (b \in \mathcal{B}). \tag{3.6}$$

Besides RB allocation, for each TTI, the scheduler also needs to choose a modulation and coding scheme (MCS) for each source node [31]. The MCS of each source node determines the modulation and coding rate – how much information (in unit of bits) is modulated and coded in each RB for this source node. The higher the MCS is, the higher the modulation and coding rate is. On the other hand, the maximum amount of information that can be successfully transmitted by an RB also depends on the channel condition. If the channel condition for this RB is poor and the source uses a high MCS, information carried in the RB cannot be successfully received and decoded by the BS. Therefore, the achievable data rate

[1] The case of multi-user MIMO where an RB can be allocated to multiple sources is much more complex and will be explored in future work.

by an RB $b \in \mathcal{B}$ depends on both the MCS used by the source node as well as the channel condition for this RB.

Under 5G, there are 29 levels of MCSs for transmission [31]. Denote $\mathcal{M}$ as the set of these available MCSs (i.e., $\mathcal{M} = \{1, 2, ...29\}$), where we have $m = 1$ correspond to the lowest MCS and $m = 29$ correspond to the highest MCS. Denote $q_i^b(t)$ as the maximum MCS level that source node i's channel can support on RB b at TTI t. We have:

$$0 \leq q_i^b(t) \leq |\mathcal{M}|.$$

In practice, $q_i^b(t)$ is determined by the channel quality indicator (CQI) report carried in the feedback from source node i at TTI $(t-1)$. Denote c^m as the modulation and coding rate under MCS level m, which can be found in Table 5.1.3.1-1 in [31]. Denote $r_i^{b,m}(t)$ as the achievable data rate by RB b w.r.t. source node i under MCS m. If $m \leq q_i^b(t)$, the transmission is successful and the achievable data rate is c^m. Otherwise, i.e., $m > q_i^b(t)$, the transmission is unsuccessful and the achievable data rate is 0. We have:

$$r_i^{b,m}(t) = \begin{cases} c^m & \text{if } m \leq q_i^b(t), \\ 0 & \text{otherwise.} \end{cases} \tag{3.7}$$

Note that although each RB can only be allocated to at most one source node in a TTI, a source node may be allocated with multiple RBs. For a source node allocated with multiple RBs, it must choose and use one MCS $m \in \mathcal{M}$ for all its allocated RBs [31]. Denote $y_i^m(t)$ as a binary variable indicating whether MCS $m \in \mathcal{M}$ is chosen by source node i at TTI t, i.e.,

$$y_i^m(t) = \begin{cases} 1 & \text{if MCS } m \text{ is chosen for source } \ i \text{ at TTI } t, \\ 0 & \text{otherwise.} \end{cases}$$

Figure 3.1: An example illustrating the trade-off between MCS selection and the number of RBs that can contribute to total data rate.

We have

$$\sum_{m\in\mathcal{M}} y_i^m(t) \leq 1 \quad (i \in \mathcal{N}). \tag{3.8}$$

Denote $R_i(t)$ as the amount of information transmitted by source node i in TTI t across all RBs allocated to it. We have

$$R_i(t) = \sum_{b\in\mathcal{B}} \sum_{m\in\mathcal{M}} x_i^b(t) y_i^m(t) r_i^{b,m}(t) \quad (i \in \mathcal{N}). \tag{3.9}$$

Based on (3.7) and (3.8), there is a clear trade-off between the choice of m and number of RBs allocated to source node i that can contribute to $R_i(t)$, due to the differences in channel conditions on each RB allocated to the same source node. That is, the higher the MCS m is chosen, the fewer the number of RBs can help contribute to $R_i(t)$. In Fig. 3.1, we use an example to illustrate this trade-off. For a source node, suppose it is allocated with 4 RBs. Suppose under the current channel conditions on the 4 RBs, the maximum allowed MCS on the 4 RBs are 2, 4, 3 and 4, respectively. If we choose MCS 4 for transmission, then only RB 2 and 4 can contribute to $R_i(t)$, each with an MCS level 4. If we choose MCS 3 for transmission, then RB 2, 3 and 4 can contribute to $R_i(t)$, with each contributing to

$R_i(t)$ with MCS level 3. Clearly, choosing a higher MCS level may not translate to the total aggregate (effective) data rate among the RBs. From this example, we can see that judicious choice of MCS is necessary to balance the achievable bit rates from each RB and the number of RBs that can actually contribute to achievable bit rates.

3.3.3 Problem Statement and Technical Challenges

In this chapter, we want to design a real-time 5G-compliant scheduler to minimize $\bar{A}^{\mathrm{B}}$ at the BS. The scheduling algorithms entails to allocate $|\mathcal{B}|$ RBs to $|\mathcal{N}|$ source nodes in each TTI, and to choose an MCS for each user. That is, to determine the decision variables $x_i^b(t)$ and $y_i^m(t)$ for each TTI t so that $\bar{A}^{\mathrm{B}}$ is minimized.

There are a number of challenges associated with this problem. First, the search space of the scheduling problem is enormous. Within each TTI, the BS needs to allocate $|\mathcal{B}|$ RBs (e.g., 100) among $|\mathcal{N}|$ source nodes (e.g., 100), and assign each source node an optimal MCS (among 29 possible levels). The solution space consists of $|\mathcal{B}|^{|\mathcal{N}|} \cdot |\mathcal{M}|^{|\mathcal{N}|}$ possibilities. None of the existing AoI research (see [8]) has studied problems of such size and complexity.

Second, the scheduling algorithm that we need should be an online algorithm. This is because the scheduler can only makes a scheduling decision for the next TTI based on most recent channel feedback information. It does not have any knowledge of channel conditions for the future. Since we are minimizing a long term average AoI, it is not possible for a schedule to make an optimal decision without knowledge of the future. Therefore, one can at best design a near-optimal scheduler.

Finally, the timing requirement to run our scheduler is very stringent. Under 5G [31], our scheduler must find its scheduling decision within a TTI, which is in sub-millisecond time scale. To date, none of the existing AoI research has considered such real-time requirement

in the design of a scheduling solution.

3.4 Performance Bound

In this section, we develop a lower bound for the objective $\bar{A}^{\mathrm{B}}$. This lower bound (if tight) can be used as a benchmark to measure the performance of a scheduling algorithm that we will design later in Section 3.5.

Denote $\overline{R}_i$ as the long term average data rate for source node $i \in \mathcal{N}$, i.e.,

$$\overline{R}_i = \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} R_i(t). \tag{3.10}$$

In Section 3.4.1, we develop a lower bound for $\bar{A}^{\mathrm{B}}$ (applicable to all scheduling algorithms) assuming that $\overline{R}_i$'s are given *a priori*. In Section 3.4.2, we remove this assumption (i.e., knowledge of $\overline{R}_i$'s) and present a novel computational procedure to find a lower bound for $\bar{A}^{\mathrm{B}}$.

3.4.1 A Lower Bound For Known $\overline{R}_i$'s

For given $\overline{R}_i$'s, there may exist different scheduling algorithms. Among these different scheduling algorithms, how do we find a scheduling algorithm that offers the minimum $\bar{A}^{\mathrm{B}}$? This is the question we want to answer in this section.

Note that in (3.5), $\bar{A}^{\mathrm{B}}$ is a weighted sum of all $\bar{A}_i^{\mathrm{B}}$'s. A lower bound of $\bar{A}^{\mathrm{B}}$ under the given $\overline{R}_i$'s can be found by the following relaxation. If we can find a lower bound for each $\bar{A}_i^{\mathrm{B}}$ (for $i \in \mathcal{N}$) under the given $\overline{R}_i$ that is independent of the other $\bar{A}_j^{\mathrm{B}}$'s or $\overline{R}_j$'s (for $j \neq i$), then we can multiple each with its corresponding weight and sum them up. This sum of weighted lower bound is clearly a lower bound of $\bar{A}^{\mathrm{B}}$.

Following this idea, we now show how to find a lower bound for $\bar{A}_i^{\mathrm{B}}$ under a given $\overline{R}_i$ that is independent of other $\overline{R}_j$ (for $j \neq i$). For ease of exposition, denote p_i as the fraction (in percentage) of successfully transmitted samples over all generated samples in the long term. Clearly, $p_i \leq 1$. With p_i, we can rewrite $\overline{R}_i$ as:

$$\overline{R}_i = \frac{p_i L_i}{T_i}. \tag{3.11}$$

Note that $\overline{R}_i$ is proportional to p_i via a constant factor. Therefore, minimizing $\bar{A}_i^{\mathrm{B}}$ under a given $\overline{R}_i$ is equivalent to minimizing $\bar{A}_i^{\mathrm{B}}$ under a given p_i.

Since we want to find a lower bound of $\bar{A}_i^{\mathrm{B}}$ under a given p_i, let's artificially move ahead the update time for $A_i^{\mathrm{B}}(t)$ as follows. Instead of updating $A_i^{\mathrm{B}}(t)$ at the end of TTI $e_i(k)$, let's move the update time to the end of TTI $b_i(k)$. Although this is infeasible in reality, it serves our purpose of finding a lower bound. Clearly, $\bar{A}_i^{\mathrm{B}}$ at the BS under such a fictitious updating mechanism is smaller than that when the update is made at the end of TTI $e_i(k)$ (since the update is performed earlier than it should be). Therefore, we will use $\bar{A}_i^{\mathrm{B}}$ obtained under such a fictitious update mechanism as a lower bound.

Ideally, to minimize this new lower bound of $\bar{A}_i^{\mathrm{B}}$ under a given p_i, we would like to have each of those samples be transmitted *immediately* after they are generated. Clearly, such a hypothesized (ideal) scheduler would offer a new lower bound for $\bar{A}_i^{\mathrm{B}}$ under a given p_i.

Denote $T_i^{\mathrm{BTT}}(k)$ as the "begin-to-transmit" (BTT) interval for the k-th and $(k+1)$-th sample at source node i. That is, $T_i^{\mathrm{BTT}}(k)$ is the interval between the starting (beginning) time of transmitting the k-th sample and the starting time of transmitting the $(k+1)$-th sample at source node i. Based on this definition, we have

$$T_i^{\mathrm{BTT}}(k) = b_i(k+1) - b_i(k). \tag{3.12}$$

Then, under a hypothesized scheduler, $T_i^{\mathrm{BTT}}(k)$ is an integral multiple of the sampling period

T_i. Clearly, such a hypothesized scheduler is not unique and many (each with different behavior of $T_i^{\mathrm{BTT}}(k)$'s) may offer the same p_i. Among this group of hypothesized schedulers, we want to identify a scheduler that minimizes $\bar{A}_i^{\mathrm{B}}$.

More formally, we define *Almost Uniform Scheduler* (AUS) as a hypothesized scheduler with:

$$T_i^{\mathrm{BTT}}(k) = \text{ either } h_i T_i \text{ or } (h_i + 1)T_i \text{ for } k > 0,$$

where h_i is defined as

$$h_i = \lfloor \frac{1}{p_i} \rfloor, \tag{3.13}$$

and $\lfloor \cdot \rfloor$ is the floor function. Then we have the following lemma for the scheduler that minimize $\bar{A}_i^{\mathrm{B}}$ among the hypothesized schedulers.

Lemma 3.1. *AUS minimizes $\bar{A}_i^B$ among all hypothesized schedulers with*

$$\bar{A}_i^B = \frac{T_i}{2} f(p_i) + \frac{1}{2}, \tag{3.14}$$

where

$$f(p_i) = 2\lfloor \frac{1}{p_i} \rfloor + 1 - (\lfloor \frac{1}{p_i} \rfloor^2 + \lfloor \frac{1}{p_i} \rfloor) \cdot p_i. \tag{3.15}$$

Lemma 3.1 says that the hypothesized scheduler that employs almost uniform BTT intervals (or exactly uniform in the case when $1/p_i$ is an integer) minimizes $\bar{A}_i^{\mathrm{B}}$. This result is very intuitive. It can be shown (as in the proof below) that for any other scheduler with a larger variance in BTT intervals, one can always find a scheduler with a smaller variance in BTT intervals that reduces $\bar{A}_i^{\mathrm{B}}$.

Proof. To prove Lemma 1, we need to prove (i) AUS is the hypothesized scheduler that

minimizes $\bar{A}_i^{\mathrm{B}}$, and (ii) under AUS, $\bar{A}_i^{\mathrm{B}}$ is given in (3.14).

We first prove AUS is the hypothesized scheduler that minimizes $\bar{A}_i^{\mathrm{B}}$. Our proof is based on contradiction. Suppose AUS is not the hypothesized scheduler that minimizes $\bar{A}_i^{\mathrm{B}}$. Then denote π^* (which is not AUS) as the hypothesized scheduler that minimizes $\bar{A}_i^{\mathrm{B}}$. Recall the length of any BTT interval under a hypothesized scheduler is an integral multiple of T_i. Therefore, we can find two BTT intervals under π^* with length n_1T_i and n_2T_i such that $n_1 \geq n_2 + 2$, with n_1 and n_2 being integers. We can change the lengths of the two BTT intervals to $(n_1 - 1)T_i$ and $(n_2 + 1)T_i$, and get a new hypothesized scheduler π_0. Clearly, π_0 and π^* has the same p_i. However, the average AoI within the two BTT intervals in π^* is

$$A^* = \frac{(n_1T_i)^2 + (n_2T_i)^2 + (n_1 + n_2)T_i}{2(n_1 + n_2)T_i}, \tag{3.16}$$

while the average AoI within the two BTT intervals in π_0 is

$$A_0 = \frac{\Big((n_1 - 1)T_i\Big)^2 + \Big((n_2 + 1)T_i\Big)^2 + (n_1 + n_2)T_i}{2(n_1 + n_2)T_i}. \tag{3.17}$$

We have

$$A^* - A_0 = \frac{(n_1 - n_2 - 1)T_i}{n_1 + n_2} > 0, \tag{3.18}$$

which means π_0 has a smaller $\bar{A}_i^{\mathrm{B}}$ than what π^* has. This contradicts to that π^* is the hypothesized scheduler that minimizes $\bar{A}_i^{\mathrm{B}}$.

Now we have proved that AUS is the hypothesized scheduler that minimizes $\bar{A}_i^{\mathrm{B}}$. We want

to find $\bar{A}_i^{\mathrm{B}}$ under AUS. Recall p_i is the fraction of all generated samples that are successfully transmitted. So the average of all BTT intervals at the BS is T_i/p_i. Clearly, the length of BTT interval for source node i under AUS is either h_iT_i or $(h_i+1)T_i$. Denote a as the percentage of those BTT intervals with length h_iT_i, then $(1-a)$ is the percentage of those BTT intervals with length $(h_i+1)T_i$. When the time interval $[0,T)$ is large, i.e., $T\to\infty$, the fraction of successfully transmitted samples approaches p_i, and the occurrence rates (the number of occurrences over the number of TTIs) of those two types of BTT intervals are $\frac{p_ia}{T_i}$ and $\frac{p_i(1-a)}{T_i}$, respectively. We have

$$\lim_{T\to\infty}\frac{T\cdot\frac{p_ia}{T_i}\cdot h_iT_i+T\cdot\frac{p_i(1-a)}{T_i}\cdot(h_i+1)T_i}{T}=1\ , \tag{3.19}$$

which gives us

$$h_ia+(h_i+1)(1-a)=\frac{1}{p_i}. \tag{3.20}$$

We have

$$a=h_i+1-\frac{1}{p_i}. \tag{3.21}$$

Based on a, we can calculate $\bar{A}_i^{\mathrm{B}}$ under AUS:

$$\begin{aligned}\bar{A}_i^{\mathrm{B}}&=\frac{a\cdot\frac{h_iT_i(1+h_iT_i)}{2}+(1-a)\cdot\frac{(h_i+1)T_i\left(1+(h_i+1)T_i\right)}{2}}{ah_iT_i+(1-a)(h_i+1)T_i}\\&=\frac{T_i}{2}\big(2h_i+1-(h_i^2+h_i)p_i\big)+\frac{1}{2}\\&=\frac{T_i}{2}f(p_i)+\frac{1}{2}.\end{aligned}$$

This completes our proof. □

Combining (3.5), (3.11) and (3.14), we have the following lower bound for $\bar{A}^{\mathrm{B}}$ as a functions of $\overline{R}_i$:

$$\bar{A}^{\mathrm{B}} \geq \sum_{i\in\mathcal{N}} w_i \cdot \big(\frac{T_i}{2} f\big(\frac{\overline{R}_i T_i}{L_i}\big) + \frac{1}{2}\big). \tag{3.22}$$

In the next subsection, we will remove the assumption of prior knowledge of $\overline{R}_i$.

3.4.2 Finding A Lower Bound of $\bar{A}^{\mathbf{B}}$

Based on (3.22), a lower bound for $\bar{A}^{\mathrm{B}}$ can be found by minimizing the RHS of (3.22), i.e.,

$$\begin{aligned} \text{OPT-LB: } \min \quad & \sum_{i\in\mathcal{N}} w_i\big(\frac{T_i}{2} f\big(\frac{\overline{R}_i T_i}{L_i}\big) + \frac{1}{2}\big) \\ \text{s.t.} \quad & \text{Constraints (3.6), (3.7), (3.8), (3.9), (3.10),} \end{aligned}$$

where the optimization variables are $x_i^b(t)$'s and $y_i^m(t)$'s. In the rest of this section, we show how to solve OPT-LB. We emphasize that the optimal solution to OPT-LB only serves as a lower bound of $\bar{A}^{\mathrm{B}}$, while its own AoI performance is not of our concern (and is likely much higher than this lower bound).

For the ease of exposition, we define

$$J_i(\overline{R}_i) = w_i\big(\frac{T_i}{2} f\big(\frac{\overline{R}_i T_i}{L_i}\big) + \frac{1}{2}\big). \tag{3.23}$$

Then the objective of OPT-LB becomes:

$$\min \quad \sum_{i\in\mathcal{N}} J_i(\overline{R}_i). \tag{3.24}$$

We will design an optimal scheduling algorithm to OPT-LB and obtain a lower bound for $\bar{A}^{\mathrm{B}}$.

OPT-LB is a scheduling problem to minimize a function of $\overline{R}_i$. Similar problems have appeared in the information theory community (see, e.g., [60, 61]), where it has been shown that a gradient scheduling algorithm can achieve the same optimal objective value asymptotically (when the number of TTIs goes to infinity). Specifically, in a gradient scheduling algorithm, we define an empirical data rate $R_i^e(t)$ for each TTI t and it is updated as a moving average as follows:

$$R_i^e(t+1) = (1-\beta)R_i^e(t) + \beta R_i(t), \tag{3.25}$$

where β is a small positive constant (e.g., 0.01) and $R_i(t)$ is the instant data rate at TTI t. It can be easily shown that under the moving average updating algorithm in (3.25), when $\beta \to 0$,

$$\lim_{t\to\infty} R_i^e(t) = \overline{R}_i \ . \tag{3.26}$$

That is, $R_i^e(t)$ asymptotically approaches $\overline{R}_i$ when $t \to \infty$. In practice, t does not need to be very large to achieve this approximation.

Based on (3.26), OPT-LB becomes

$$\begin{aligned} \text{OPT-1:} \quad \min \quad & \lim_{t\to\infty} \sum_{i\in\mathcal{N}} J_i\big(R_i^e(t)\big) \\ \text{s.t.} \quad & \text{Constraints (3.6), (3.7), (3.8), (3.9), (3.25),} \end{aligned}$$

where the optimization variables are $x_i^b(t)$'s and $y_i^m(t)$'s.

The idea of the gradient scheduling algorithm is to minimize $\sum_{i\in\mathcal{N}} J_i\big(R_i^e(t+1)\big)$ at every t. It has been shown that by performing such a minimization for every TTI, $\lim_{t\to\infty} \sum_{i\in\mathcal{N}} J_i\big(R_i^e(t)\big)$ is also minimized when $\beta \to 0$ [60, 61].

We now show how to minimize $\sum_{i\in\mathcal{N}} J_i\big(R_i^e(t+1)\big)$ at TTI t. When $\beta \to 0$, using (3.25),

we have

$$\begin{aligned}&\sum_{i\in\mathcal{N}} J_i\big(R_i^e(t+1)\big) = \sum_{i\in\mathcal{N}} J_i\big((1-\beta)R_i^e(t)+\beta R_i(t)\big)\\ =&\sum_{i\in\mathcal{N}} J_i\Big(R_i^e(t)+\beta\big(R_i(t)-R_i^e(t)\big)\Big)\\ =&\sum_{i\in\mathcal{N}} J_i\big(R_i^e(t)\big)+\sum_{i\in\mathcal{N}}\frac{dJ_i(R)}{dR}\Big|_{R=R_i^e(t)}\beta\big(R_i(t)-R_i^e(t)\big),\end{aligned} \tag{3.27}$$

where the last equality follows from the definition of derivative of $J_i(\cdot)$. Since $R_i^e(t)$ can be computed at TTI t, $J(R_i^e(t))$ and $\frac{dJ_i(R)}{dR}\Big|_{R=R_i^e(t)} R_i^e(t)$ can also be computed. Therefore, to minimize $\sum_{i\in\mathcal{N}} J_i(R_i^e(t+1))$ at TTI t, we only need to solve the following problem:

$$\begin{aligned}\text{OPT-2:}\ \min\quad &\sum_{i\in\mathcal{N}}\frac{dJ_i(R)}{dR}\Big|_{R=R_i^e(t)} R_i(t)\\ \text{s.t.}\quad &\text{Constraints (3.6), (3.7), (3.8), (3.9), (3.28),}\end{aligned}$$

where the derivative of $J_i(\cdot)$ is computed as

$$\begin{aligned}\frac{dJ_i(R)}{dR}\Big|_{R=R_i^e(t)} =&\frac{w_iT_i}{2}\frac{df(\frac{RT_i}{L_i})}{dR}\Big|_{R=R_i^e(t)}\\ =&-\frac{w_iT_i^2}{2L_i}\big(\lfloor\frac{L_i}{R_i^e(t)T_i}\rfloor^2+\lfloor\frac{L_i}{R_i^e(t)T_i}\rfloor\big).\end{aligned} \tag{3.28}$$

To ensure $f(\cdot)$ is continuously differentiable at every point, we need to define how to perform derivative at certain points. Note that $f(\cdot)$ is a piecewise linear function. When $\frac{L_i}{R_i^e(t)T_i}$ is exactly an integer, the function $f(\frac{RT_i}{L_i})$ is continuous but not differentiable at $R=R_i^e(t)$ (i.e., the left derivative doesn't equal to the right derivative). For these points, we use the right derivative as the derivative at $R=R_i^e(t)$, as shown in the RHS of (3.28).

Since each term in the RHS of (3.28) is either a constant or a known value at TTI t, let's denote the RHS of (3.28) as $W_i(t)$. Using (3.9), OPT-3 can be written as:

$$\begin{aligned}\text{OPT-}t\text{:}\ \min\quad &\sum_{i\in\mathcal{N}}\sum_{b\in\mathcal{B}}\sum_{m\in\mathcal{M}} W_i(t)r_i^{b,m}(t)x_i^b(t)y_i^m(t)\\ \text{s.t.}\quad &\text{Constraints (3.6), (3.7), (3.8).}\end{aligned}$$

Algorithm 3.1 A procedure to find a lower bound for $\bar{A}^{\mathrm{B}}$.

An Approximate Solution to OPT-LB Input: β, T.

1: Initialization: $R_i^e(0) = 0$ for each $i \in \mathcal{N}$.

2: **for** $t = 1, 2, \cdots, T$ **do**

3: Solve OPT-t and get the optimal solution $x_i^b(t)$'s and $y_i^m(t)$'s.

4: Use $x_i^b(t)$'s and $y_i^m(t)$'s to perform scheduling at TTI t and get the instant data rate $R_i(t)$.

5: Use (3.25) to update $R_i^e(t)$.

6: **end for**

7: Let $\overline{R}_i = R_i^e(t)$ for each $i \in \mathcal{N}$.

8: Substitute $\overline{R}_i$'s into the objective function of OPT-LB and use this value as a lower bound for $\bar{A}^{\mathrm{B}}$.

where the optimization variables are $x_i^b(t)$'s and $y_i^m(t)$'s. The minimization problem OPT-t is an integer quadratic program (IQP), which can be solved by a commercial solver such as CPLEX [62].

Based on OPT-t, we propose a procedure to solve OPT-LB and use it as a lower bound for $\bar{A}^{\mathrm{B}}$, as shown in Fig. 3.1. Although in theory it requires $\beta \to 0$ and $T \to \infty$ for the procedure to converge asymptotically, in practice we can get an excellent approximation even for a small T.

We use the following example to illustrate this point.

Example 3.1. Consider a sizable IoT-network with $|\mathcal{N}| = 100$. Suppose that we have 10 types of source nodes as specified in Table 3.2, we assume that our 100 nodes consist of all

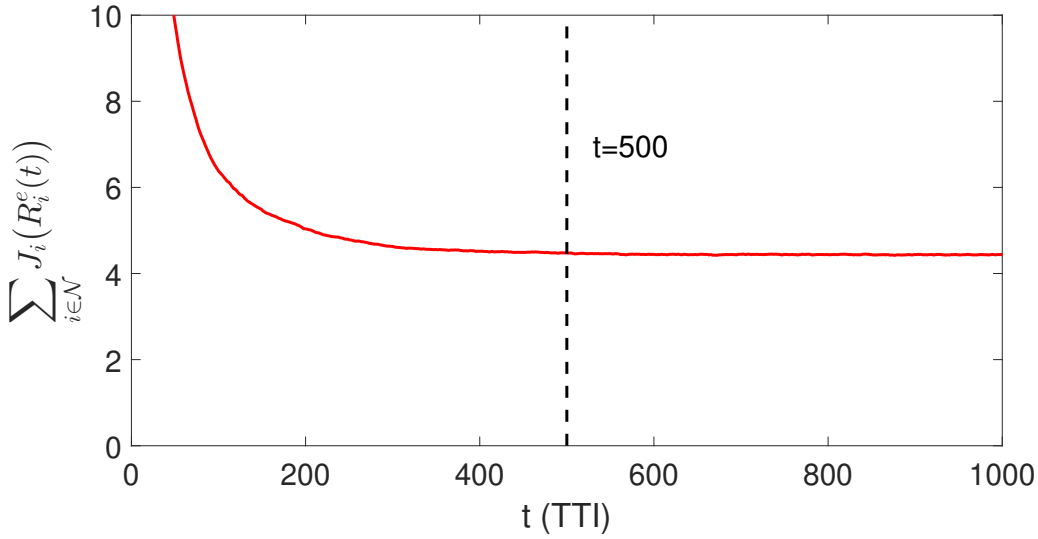


Figure 3.2: Convergence time of our proposed approximate solution to OPT-LB when $\beta = 0.01$.

10 types, with 10 nodes in each type. More details on our experimental setup and parameter settings are given in Section 3.6.1. We assume a Rayleigh fading channel with no frequency nor time correlation. For $\beta = 0.01$, we run the approximate solution for 1000 TTIs. The relationship between $\sum_{i\in\mathcal{N}} J_i\left(R_i^e(t)\right)$ (which is used as the lower bound for $\bar{A}^{\mathrm{B}}$) and TTI t is shown in Fig. 3.2. When $t = 500$, we have $\sum_{i\in\mathcal{N}} J_i\left(R_i^e(t)\right) = 4.48$ while for $t = 1,000$, we have $\sum_{i\in\mathcal{N}} J_i\left(R_i^e(t)\right) = 4.44$. Given the negligible difference between the two, we can choose $t = 500$ as our terminating time.

It turns out for all of our experiments in Section 3.6.1, $t = 500$ is sufficient for our procedure (in Algorithm 3.1) to terminate and obtain an excellent approximate solution.

□

Note that although the scheduler in Algorithm 3.1 can find an approximate objective value of OPT-LB and use it as a lower bound for $\bar{A}^{\mathrm{B}}$, its own AoI performance is much

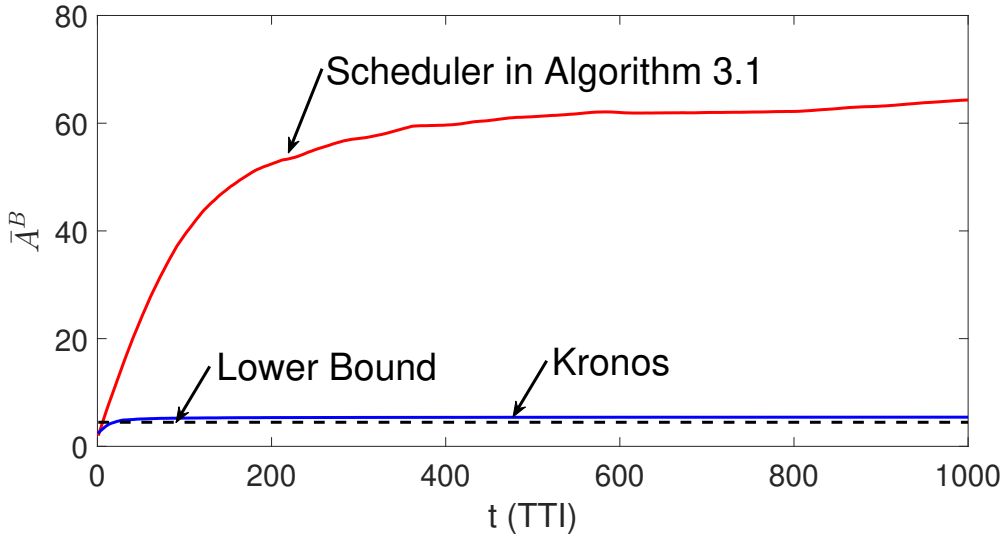


Figure 3.3: AoI performance of the scheduler in Algorithm 3.1, Kronos, and the lower bound.

higher than this lower bound. This is because under this scheduler, the RBs allocated to a particular source node tend to be uniformly distributed across all TTIs (e.g., under a random channel with small coherence time) and thus leading to large AoI. On the other hand, an obviously better scheduler would allocate RBs to a source node right after a new sample is generated and use as few TTIs as possible. We use the following example to illustrate this point.

Example 3.2. Following the same network settings as in Example 3.1, Fig. 3.3 shows the AoI performance of the scheduler in Algorithm 3.1 across 1000 TTIs. Also shown in the same figure are the lower bound that we developed and AoI performance of Kronos (the scheduler that we will design in the next section). As we can see, the AoI performance of the scheduler in Algorithm 3.1 is much worse (larger) than Kronos, which confirms our argument that it cannot be directly used as a scheduler to minimize AoI. □

3.5 Kronos: A Real-Time Scheduler

The goal of this section is two-fold. First, we want to design a scheduler that minimize $\bar{A}^{\mathrm{B}}$. Second, we want to ensure the scheduler can meet the stringent timing requirement in 5G.

We present Kronos[2], a 5G-compliant real time AoI scheduler that offers near-optimal performance. We organize this section as follows. In Section 3.5.1, we present the basic design ideas of Kronos. In Section 3.5.2. we elaborate the details of a key step of Kronos. In Section 3.5.3, we present a GPU-based implementation for Kronos that can meet the stringent timing requirement in 5G.

3.5.1 Basic Idea

The design of Kronos is based on the following key ideas.

1. For the objective function in (3.5), it is obvious that we need to minimize $\bar{A}_i^{\mathrm{B}}$ from each source node $i \in \mathcal{N}$. For A_i^{B}, its value at the BS is not reduced until a new sample is received by the BS in its entirety. That is, a partially transmitted sample will not reduce (update) A_i^{B} at the BS. Based on this observation, we should minimize the number of partially (incomplete) transmission of samples at the end of each TTI. As an extreme, we can limit the number of samples that are partially (incompletely) transmitted at the end of a TTI to at most one. This can be done by devoting all the remaining RBs to one sample, rather than spreading out to multiple samples.

2. Following the last idea, at the beginning of a new (the next) TTI, we will inherit at most one partially (incomplete) transmission of a sample from the previous TTI. Recall that we cannot preempt a sample once it starts transmission, even if there is a

[2]Kronos is the god of time in Greek mythology.

newly generated sample from the same source node. Further, for our IoT applications, a sample size is relatively small. So the remaining portion of the partially transmitted sample is not large (in most cases) and it makes sense to complete its transmission before starting to transmit any other samples.

3. After we complete the transmission of the remaining (incomplete) sample (carried from the last TTI), we need to decide which sample to transmit next in the current TTI. To do this, we need a metric to compare among the samples from different source nodes and decide which sub-set of samples that we will allocate the remaining RBs. Clearly, this metric should consist of the weight and the "outage" (difference between AoI at the BS and the source, i.e., $A_i^{\mathrm{B}}(t) - A_i^{\mathrm{s}}(t)$) for each source node $i \in \mathcal{N}$. In our previous work [24], in the absence of considering channel conditions, we use the metric $w_i \Delta_i^2(t)$ for scheduling, where $\Delta_i(t)$ is defined as

$$\Delta_i(t) = A_i^{\mathrm{B}}(t) - A_i^{\mathrm{s}}(t). \tag{3.29}$$

It was shown in [24] that a scheduler based on this metric can offer near-optimal performance (under simplified channel conditions). Therefore, it would be wise to have Kronos to inherit this basic trait before we add additional features to cope with varying channel conditions.

4. To incorporate channel conditions into the scheduling decision metric, we must consider the impact of MCS setting on RBs. As shown in the example in Fig. 3.1, the higher the MCS m is chosen, the fewer number of RBs (with a higher rate) can be used for transmission. Intuitively, we prefer to use as few RBs as possible to transmit a sample. Therefore, the scheduling metric should also include the number of RBs required to transmit a sample, i.e., the more RBs required, the lower the priority (or smaller the metric) associated with a source node. We will elaborate the details of how to

incorporating channel conditions into the scheduling metric in the next section.

5. Once we have a scheduling metric (see next section), we can compare samples and perform scheduling, i.e. RB allocation. Clearly, RB allocation is an iterative process, where in each iteration, we will consider how to allocate a subset of RBs among the *remaining* unallocated RBs to a sample in the *remaining* unscheduled samples. Eventually (after a number of iterations), all RBs are allocated and the algorithm terminates.

3.5.2 Algorithm Details: Design of Scheduling Metric

We devote this section to the discussion of how channel conditions are incorporated into the scheduling metric, which is at the heart of our design. Recall that the choice of MCS value m at a source node will set the corresponding coding rate c^m, which will in turn determine two parameters:

- the set of RBs in the remaining un-allocated RBs that can contribute at this bit rate c^m. We denote the number in this set as $n_i^m(t)$.

- the number of RBs that is needed to transmit a sample for source node i, which we denote as s_i^m.

That is,

$$n_i^m(t) = \sum_{\text{un-allocated } b} [q_i^b(t) \geq m], \tag{3.30}$$

where $q_i^b(t)$ (see Section 3.3) is the maximum MCS that can be used for RB b and source node i for transmission (which is determined by the channel condition on RB b), and "$[\cdot]$" is the notation for Iverson bracket, returning 1 if the inside statement is true and 0 otherwise

[63]. We have:

$$s_i^m = \lceil \frac{L_i}{c^m} \rceil, \tag{3.31}$$

where "$\lceil \cdot \rceil$" is the ceiling function.

Clearly, the scheduling metric for a sample is dependent on m and is a function of $n_i^m(t)$ and s_i^m, in addition to $w_i\Delta^2(t)$ (as discussed in the last section). As a start, denote $V_i^m(t)$ as the scheduling metric under MCS m with the following general form:

$$V_i^m(t) = g\big(w_i\Delta_i^2(t), n_i^m(t), s_i^m\big), \tag{3.32}$$

where "g" denotes a function of $w_i\Delta^2(t)$, $n_i^m(t)$ and s_i^m.

For each sample from source node $i \in \mathcal{N}$, we have the pair $(n_i^m(t), s_i^m)$ under each $m \in \mathcal{M}$. If $n_i^m(t) \geq s_i^m$, it means that this sample can possibly be transmitted in its entirety in this TTI. Otherwise (i.e., $n_i^m(t) < s_i^m$), this sample can only be partially transmitted even if we allocate all the remaining RBs to it. Now we have a dilemma: shall we transmit a partial sample (while holding back one or more other samples that can otherwise be transmitted in their entirety) or shall we transmit one or more complete samples first?

Since our goal is to minimize (3.5), based on the the *shortest-job-first* principle from queuing theory [64], we should first schedule one or more samples that can be fully transmitted. Therefore, we purposely design the function $g(w_i\Delta_i^2(t), n_i^m(t), s_i^m) > 0$ when $n_i^m(t) \geq s_i^m$ and $g(w_i\Delta_i^2(t), n_i^m(t), s_i^m) < 0$ when $n_i^m(t) < s_i^m$. Under such definition, the priority for a sample that can be fully transmitted within this TTI is always higher than that for a sample that can only be transmitted partially. After RBs have been allocated to those samples that can be fully transmitted, we move on to consider how to allocate the remaining RBs to those samples that cannot be fully transmitted (i.e., samples with $V_i^m(t) < 0$). Recall that in each TTI we only schedule at most one partially transmitted sample. So when $V_i^m(t) < 0$ for all

remaining source nodes i and MCS m, we will choose one with the largest value of $V_i^m(t) < 0$ for transmission.

Based on the above discussion, we show how to design function $g(w_i\Delta_i^2(t), n_i^m(t), s_i^m)$ as follows.

- When $n_i^m(t) \geq s_i^m$, sample i can be fully transmitted with un-allocated RBs under MCS m in this TTI. In this case, the fewer RBs required for transmission (i.e., s_i^m), the higher the priority it should have. Therefore, we define function g as

$$g(w_i\Delta_i^2(t), n_i^m(t), s_i^m) = w_i\Delta_i^2(t) \cdot \frac{1}{s_i^m}. \tag{3.33}$$

- When $n_i^m(t) < s_i^m$, sample i cannot be fully transmitted under MCS m. In this case, the greater the fraction of the sample that can be transmitted (i.e., $n_i^m(t)/s_i^m$), the higher the priority it should have. Based on this idea, the function g should be proportional to the term $n_i^m(t)/s_i^m$. On the other hand, as discussed earlier, $g(w_i\Delta_i^2(t), n_i^m(t), s_i^m)$ should be negative when $n_i^m(t) < s_i^m$. To ensure this is the case, we can add a negative offset constant and define function g as

$$g(w_i\Delta_i^2(t), n_i^m(t), s_i^m) = w_i\Delta_i^2(t) \cdot \frac{n_i^m(t)}{s_i^m} - C, \tag{3.34}$$

  where C is a large (offset) constant that can ensure $g(n_i^m(t), s_i^m) < 0$ for all i and m when $n_i^m(t) < s_i^m$. For example, we can set $C = \sum_{i\in\mathcal{N}} w_i\Delta_i^2(t)$ or $C = \max_{i\in\mathcal{N}} w_i\Delta_i^2(t)$.

Combining (3.32), (3.33) and (3.34), the scheduling metric $V_i^m(t)$ is given as

$$V_i^m(t) = \begin{cases} \frac{w_i\Delta_i^2(t)}{s_i^m}, & \text{if } n_i^m(t) \geq s_i^m, \\ \frac{w_i\Delta_i^2(t)n_i^m(t)}{s_i^m} - C, & \text{otherwise.} \end{cases} \tag{3.35}$$

Based on the previous discussions, a pseudocode for Kronos is given in Algorithm 3.2.

Algorithm 3.2 A pseudocode for Kronos.

At TTI t:

1: If there is a partially transmitted sample carried from the last TTI, allocate minimum number of RBs (based on channel conditions) to complete its transmission.

2: Compute $n_i^m(t)$ by (3.30). Then compute $V_i^m(t)$ by (3.35) for all $i \in \mathcal{N}$ that have not been scheduled at t and for all $m \in \mathcal{M}$.

3: Choose i^* and m^* with the largest $V_{i^*}^{m^*}(t)$ among all $V_i^m(t)$'s.

4: **if** $V_{i^*}^{m^*}(t) > 0$ **then**

5: Allocates RBs to source i^* (using MCS m^*) to complete its transmission. Goto Step 2.

6: **else**

7: Allocate all remaining RBs to source i^* (using MCS m^*).

8: **end if**

We now discuss the complexity of Kronos. To allocate RBs to complete the transmission of the incomplete sample from the last TTI, the time complexity is $O(|\mathcal{B}||\mathcal{M}|)$. After that, if Kronos is implemented sequentially (e.g., in a CPU), then in each iteration, Kronos needs to compute $|\mathcal{N}||\mathcal{M}|$ different $V_i^m(t)$'s, which has a time complexity $O(|\mathcal{N}||\mathcal{M}||\mathcal{B}|)$. After that, Kronos selects i^* and m^* with the largest $V_{i^*}^{m^*}(t)$, which has a time complexity $O(|\mathcal{N}||\mathcal{M}|)$. Then Kronos allocates RBs to the selected source node i^*, which has a time complexity of $O(|\mathcal{B}|)$. Therefore, the time complexity for each iteration is $O(|\mathcal{N}||\mathcal{M}||\mathcal{B}|) + O(|\mathcal{N}||\mathcal{M}|) + O(|\mathcal{B}|) = O(|\mathcal{N}||\mathcal{M}||\mathcal{B}|)$. Since there are at most $|\mathcal{N}|$ iterations in each TTI, the time complexity for scheduling new samples is $O(|\mathcal{N}|^2|\mathcal{M}||\mathcal{B}|)$. Thus, the total time complexity in each TTI is $O(|\mathcal{B}||\mathcal{M}|) + O(|\mathcal{N}|^2|\mathcal{M}||\mathcal{B}|) = O(|\mathcal{N}|^2|\mathcal{M}||\mathcal{B}|)$.

In one of our experiments in Section 3.6, we find that for a typical 5G-based IoT network with $|\mathcal{N}| = 100$, $|\mathcal{B}| = 100$ and $|\mathcal{M}| = 29$, the average running time for Kronos is about ∼10 ms, which can't meet the 5G timing requirement (sub-millisecond time scale). To address this real-time problem in 5G, we will incorporate parallel computation to speed up its running timing. This will be discussed in the next section.

3.5.3 Running Time Speedup: A GPU-based Implementation

Motivation and Basic Idea We observe that in each iteration of Kronos, the computation of $V_i^m(t)$'s for each i and m is independent from each other. This motivates us to compute them in parallel rather than in sequence. We propose to employ a commercial off-the-shelf (COTS) GPU to compute $V_i^m(t)$'s in parallel. Today's COTS GPUs consist of a large number (1,000s) of processing cores and are highly optimized for massive parallel computations. However, unlike a CPU core, each GPU core processor has very limited computational capability and is designed to handle very simple computations (and thus has a low cost).

To best utilize a GPU's capability, it is utmost important to ensure that each sub-problem handled by a GPU core processing is of extremely low complexity and requires very few iterations to find a solution. Specifically, to calculate $V_i^m(t)$'s for all i's and m's in an iteration, we can decompose this problem into $|\mathcal{N}||\mathcal{M}|$ independent sub-problems, each of which is to calculate $V_i^m(t)$ under a specific value of i and m. Recall that the computational complexity of this sub-problem is only $O(|\mathcal{B}|)$, which can be done very quickly by GPU cores.

Implementation Details In our implementation, we employ an NVIDIA Tesla V100 GPU and the CUDA programming platform. This GPU consists of 80 streaming multiprocessors (SMs), with each SM consisting of 64 small processing cores (CUDA cores), i.e., 5,120 cores in total. These cores are capable of performing concurrent computation tasks

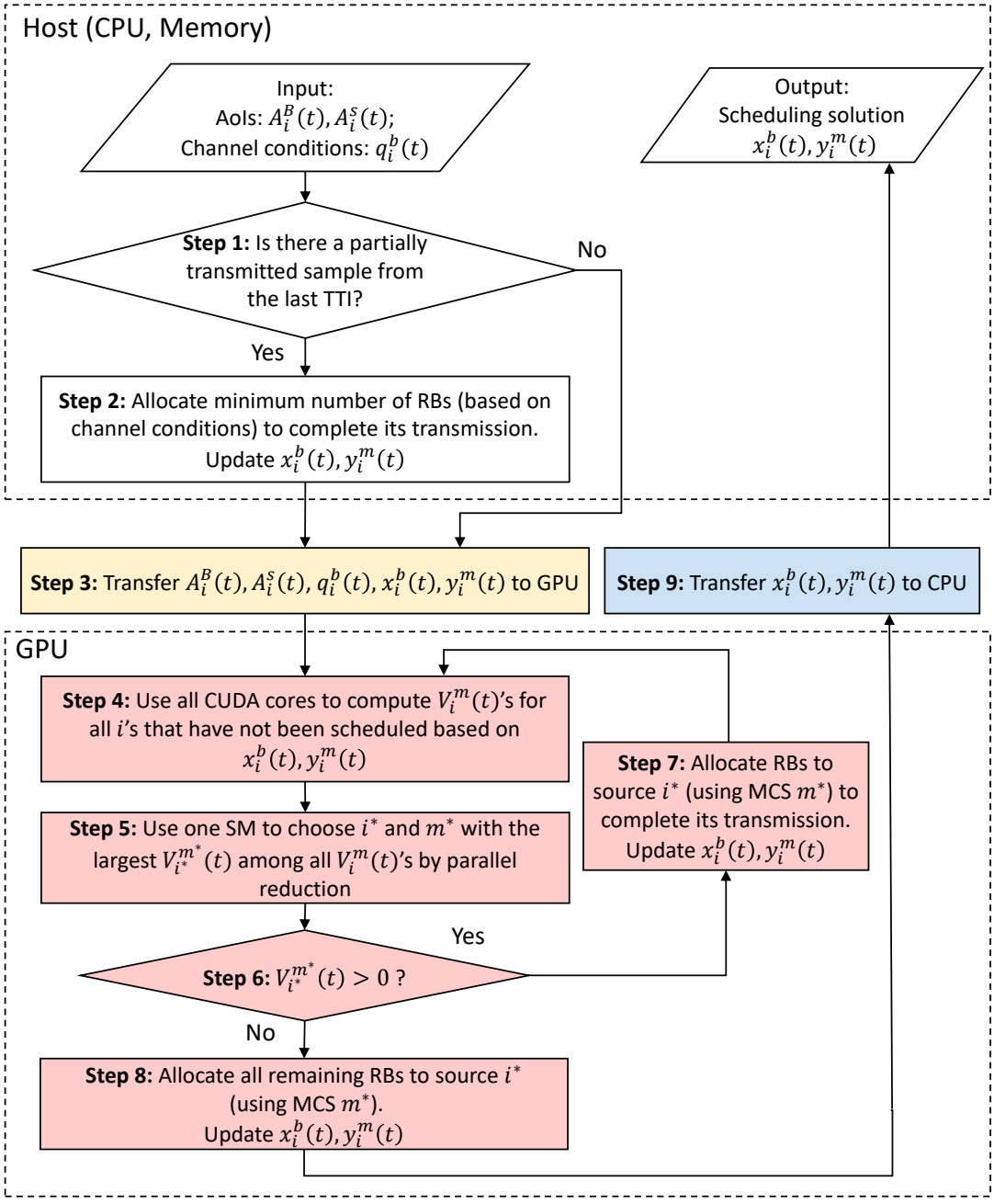


Figure 3.4: A flowchart for a GPU-based implementation of Kronos.

involving arithmetic and logic operations.

Figure 3.4 shows the flow chart of our GPU-based implementation of Kronos. Note that Steps 1 and 2 are still performed in CPU because GPU cannot help much in these steps.

After Step 2, we transfer the AoI information (i.e., $A_i^{\mathrm{B}}(t)$'s and $A_i^{\mathrm{s}}(t)$'s), the channel conditions information (i.e., $q_i^b(t)$'s), and the scheduling variables (i.e., $x_i^b(t)$'s and $y_i^m(t)$'s) from CPU memory to GPU. This is shown as Step 3 in Fig. 3.4. In our CUDA implementation, to save time, we use the asynchronous mode for this data transfer.

In GPU, we need to perform the following steps:

- In Step 4, we compute $V_i^m(t)$ for all $i \in \mathcal{N}$ that have not been scheduled at t and for all $m \in \mathcal{M}$. In this step, we can use all CUDA cores to compute $V_i^m(t)$'s in parallel. Under CUDA, the sub-tasks to compute $V_i^m(t)$'s are handle by a grid of thread blocks, each with a certain number of threads. We limit each SM to handle at most one thread block to avoid sequential execution of thread blocks on the same SM.

- In Step 5, we choose i^* and m^* based on the largest $V_{i^*}^{m^*}(t)$. This involves a comparison of $|\mathcal{N}||\mathcal{M}|$ values. We can use parallel reduction [65] to reduce complexity. For example, when $|\mathcal{N}||\mathcal{M}|$ is a power of 2, we can construct an elimination tournament, where only half of $V_i^m(t)$'s survive after each round. After $\log_2(|\mathcal{N}||\mathcal{M}|)$ rounds, the champion (with the largest $V_{i^*}^{m^*}(t)$ among all $|\mathcal{N}||\mathcal{M}|$ $V_i^m(t)$'s) will be found. This parallel reduction technique helps reduce the time complexity to $\log_2(|\mathcal{N}||\mathcal{M}|)$. When $|\mathcal{N}||\mathcal{M}|$ is not a power of 2, we can add some dummy elements at the beginning to increase the number of elements to a power of 2 and then apply parallel reduction. Note that parallel reduction is done in the same (one) SM in our implementation.

- Step 6, 7, and 8 follow the design of Kronos, which have been discussed in detail in

the previous section.

Finally, in Step 9, we transfer the scheduling solutions for $x_i^b(t)$'s and $y_i^m(t)$'s from GPU back to CPU memory. Again, we use the asynchronous mode in CUDA.

Complexity Analysis We now examine the computational complexity of our GPU-based implementation. As discussed in Section 3.5.2, the time complexity for Steps 1 and 2 is $O(|\mathcal{B}||\mathcal{M}|)$. Note that these two steps are executed in CPU as they do not involve any parallelism. Time (latency) for data transfer (i.e., Step 3 and Step 9) between CPU memory and GPU mainly depends on hardware.

The time complexity for Step 4, i.e., computing $V_i^m(t)$'s, is $O(|\mathcal{B}|)$. The time complexity for Step 5, i.e., choosing the largest $V_i^m(t)$ (with parallel reduction), is $O\big(\log(|\mathcal{N}||\mathcal{M}|)\big)$. The time complexity of Step 7, i.e., RB allocation, is $O(|\mathcal{B}|)$. Therefore, the time complexity of each iteration of Step 4, 5, 6, and 7 is $O(|\mathcal{B}|)+O\big(\log(|\mathcal{N}||\mathcal{M}|)\big)$. Recall there are at most $|\mathcal{N}|$ iterations, so the time complexity in GPU before Step 8 is $O(|\mathcal{B}||N|)+O\big(|N|\cdot\log(|\mathcal{N}||\mathcal{M}|)\big)$. As the last step, the time complexity of Step 8 is $O(|\mathcal{B}|)$. Then the total computation complexity in each TTI is

$$\begin{aligned}&O(|\mathcal{B}||\mathcal{M}|)+O(|\mathcal{B}||\mathcal{N}|)+O\big(|N|\cdot\log(|\mathcal{N}||\mathcal{M}|)\big)+O(|\mathcal{B}|)\\&=O\big(|\mathcal{B}|\cdot(|\mathcal{N}|+|\mathcal{M}|)\big)+O\big(|\mathcal{N}|\cdot(\log|\mathcal{N}|+\log|\mathcal{M}|)\big).\end{aligned}$$

We emphasize that the above $O(\cdot)$ analysis is only of theoretical interest. What we are really interested in is the actual "wall clock" computational time of our Kronos implementation. In Section 3.6, we use experiments to show that for a typical 5G IoT network with $|\mathcal{N}|=100$, $|\mathcal{B}|=100$ and $|\mathcal{M}|=29$, the average running time for GPU-based implementation of Kronos is about $\sim$0.3 ms, which meets 5G requirement (e.g., numerology 0 and 1). This timing performance is more than an order of magnitude faster than that when GPU is not used.

Table 3.2: Weights and sampling parameters for different types of source nodes.

Type	w_i	L_i (bits)	T_i (TTIs)
1	8	5400	2
2	2	7200	5
3	10	6800	3
4	6	6200	6
5	5	7600	1
6	2	8200	11
7	9	6000	4
8	1	7100	5
9	4	9600	6
10	3	8400	3

3.6 Performance Evaluation

The objective of this section is twofold. First, we will examine the timing performance of Kronos and see if it can meet the real time requirement under 5G. Second, we will evaluate Kronos in terms of its ability to achieve our objective function. The primary benchmark for this purpose is the lower bound that we developed in Section 3.4.

3.6.1 Experiment Setup and Parameter Settings

Our GPU implementation is done on an NVIDIA DGX station with an Intel Xeon E5-2698 v4 CPU (2.20 GHz, 256GB memory) and an NVIDIA Tesla V100 GPU (32 GB memory). Data communication between CPU and GPU goes through a PCIe 3.0 X16 slot with default configuration. We use CUDA 10.2 to program Kronos in our GPU.

For the source nodes, we assume there are 10 different types, each with different weight, sample size, and sampling period, as shown in Table 3.2.

We will consider different channel fading models, e.g., Rayleigh fading and Rician fading with different time and frequency correlation. The specific channel fading model will be given

in each experiment. Under each fading model, the average channel condition for each source node depends on its physical location. In this chapter, to help readers reproduce the same results, we artificially assign an average channel condition for each type of source nodes. In particular, we set the MCS levels corresponding to the average channel conditions of Types 1–10 source nodes in Table 3.2 to 26, 28, 24, 23, 20, 22, 25, 18, 24, and 21, respectively. For each MCS m, we get the corresponding modulation and coding rate c^m from [31] (Table 5.1.3.1-1).

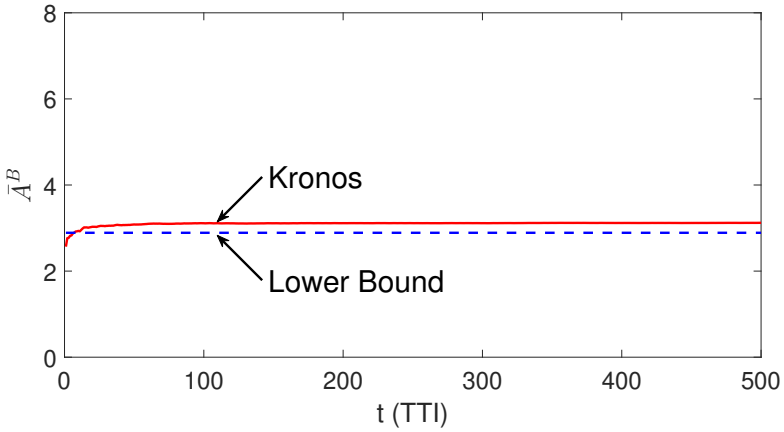
We assume the uplink transmission consists of 100 RBs, i.e., $|\mathcal{B}| = 100$. In each network setting, we run Kronos over 500 TTIs and then calculate the average AoI $\bar{A}^{\mathrm{B}}$. For initialization, $A_i^{\mathrm{s}}(0)$ for each i is set to a random number.

To compute the lower bound (used for benchmark), we use IBM ILOG CPLEX Optimizer (version 12.10.0).

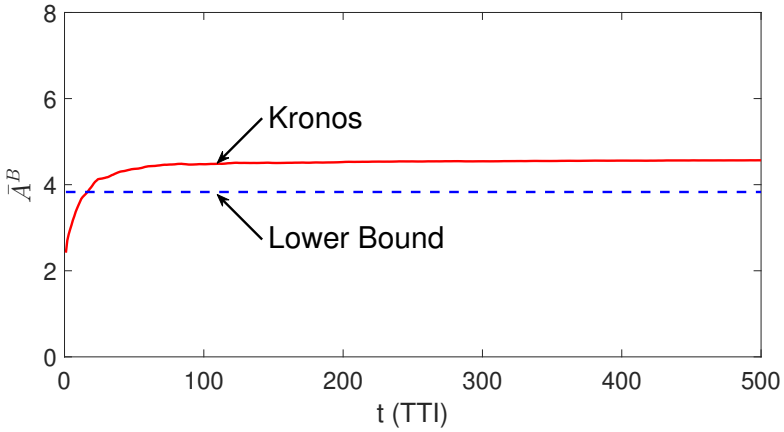
3.6.2 Results

Varying Numbers of Source Nodes We first evaluate Kronos under channels with different number of source nodes. We assume Rayleigh fading channel with no frequency or time correlation. We consider $|\mathcal{N}| = 50, 80, 100$ source nodes (equally distributed in each source type in Table 3.2). For ease of presentation, we normalize the weight of each source node w.r.t $\sum_{i\in\mathcal{U}} w_i$.

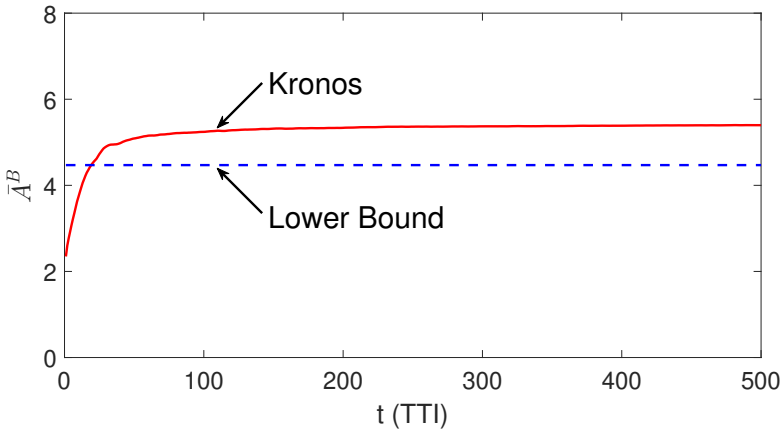
Figure 3.5 shows the evolution of $\bar{A}^{\mathrm{B}}$ under Kronos across 500 TTIs for different $|\mathcal{N}|$'s. In terms of objective value, both Kronos' implementations (with and without GPU) offer the same values. Also shown in each sub-figure is the lower bound by the procedure in Algorithm 3.1. Clearly, we see Kronos can achieve near-optimal performance. In particular, when $|\mathcal{N}| = 50$, 80 and 100, the objectives of Kronos are 8.0%, 19.2% and 20.8% within



(a) $|\mathcal{N}| = 50$

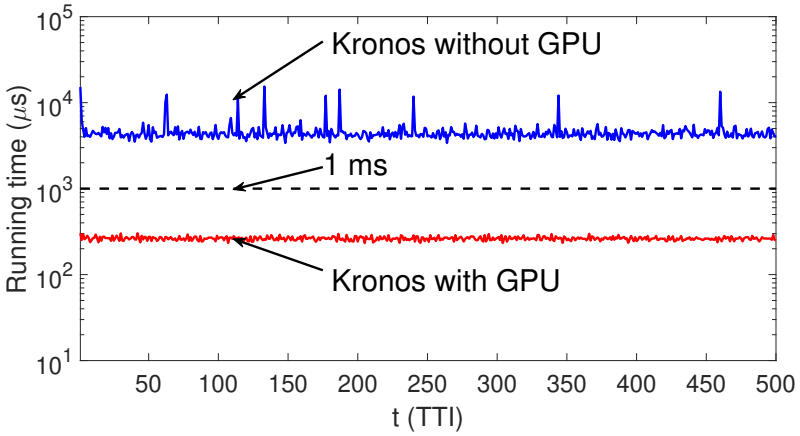


(b) $|\mathcal{N}| = 80$

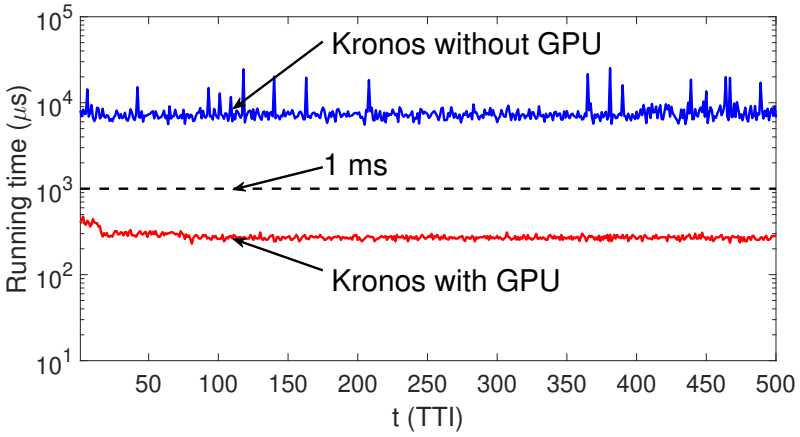


(c) $|\mathcal{N}| = 100$

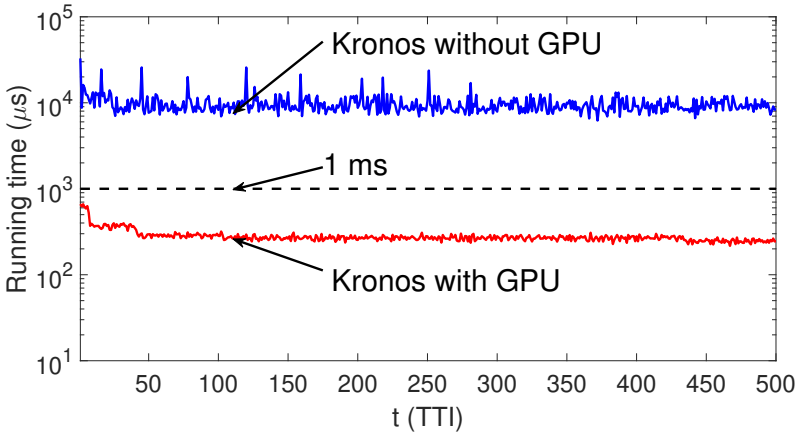
Figure 3.5: $\bar{A}^{\mathrm{B}}$ under different numbers of source nodes.



(a) $|\mathcal{N}| = 50$



(b) $|\mathcal{N}| = 80$



(c) $|\mathcal{N}| = 100$

Figure 3.6: Running time under different numbers of source nodes.

their respective lower bounds. Since the optimal objective values lie between Kronos and the lower bound, the gap between Kronos and the optimal is even closer.

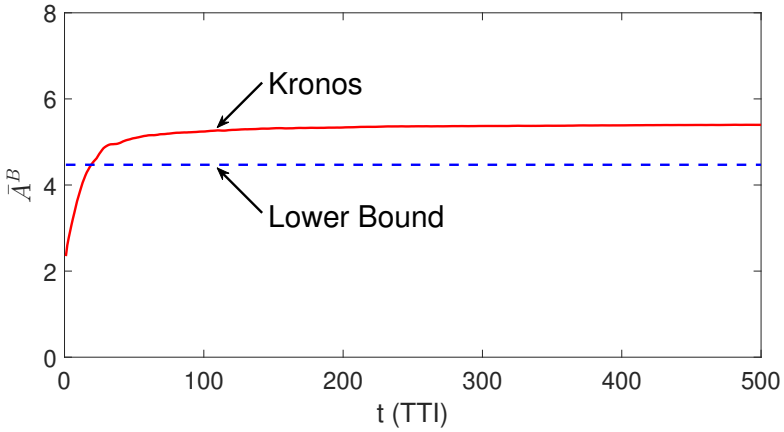
Figure 3.6 shows the running time for Kronos in each TTI with and without GPU implementation. As a benchmark, we also show the 5G timing requirement for numerology 0 (1 ms) in each sub-figure. We can see under all K, the running time of Kronos falls below 1 ms when it is implemented with GPU. When it is implemented only with CPU, it is about $\sim$10 ms. In particular, when $K = 0$, 2 and 10, the average running times of Kronos (with GPU implementation) are 263 μs, 277 μs and 279 μs respectively.

Varying Channel Propagation We now evaluate Kronos under channels with different LOS signal strength. We assume Rician fading channel with no frequency or time correlation. We consider 100 source nodes (10 from each type in Table 3.2). The weight of each source node is also normalized.

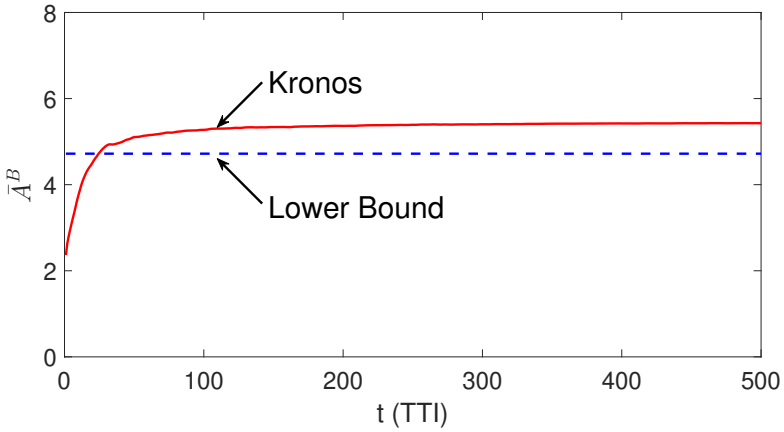
Figure 3.7 shows the evolution of $\bar{A}^{\mathrm{B}}$ under Kronos across 500 TTIs for different Rician factor K. Also shown in each sub-figure is the lower bound by the procedure in Algorithm 3.1. Clearly, we see Kronos can achieve near-optimal performance. In particular, when Rician factor $K = 0$ (i.e., Rayleigh fading), 2 and 10, the objectives of Kronos are 20.8%, 15.0% and 7.7% within their respective lower bounds.

Figure 3.8 shows the running time for Kronos in each TTI with and without GPU implementation. As a benchmark, we also show the 5G timing requirement for numerology 0 (1 ms) in each sub-figure. We can see under all K, the running time of Kronos falls below 1 ms when it is implemented with GPU. When it is implemented only with CPU, it is about $\sim$10 ms. In particular, when $K = 0$, 2 and 10, the average running times of Kronos (with GPU implementation) are 275 μs, 278 μs and 260 μs respectively.

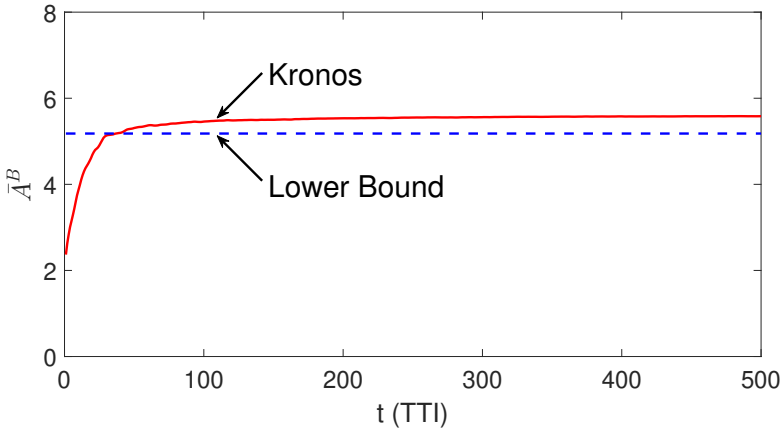
Varying Frequency Correlation We now evaluate Kronos under channels with different



(a) $K = 0$

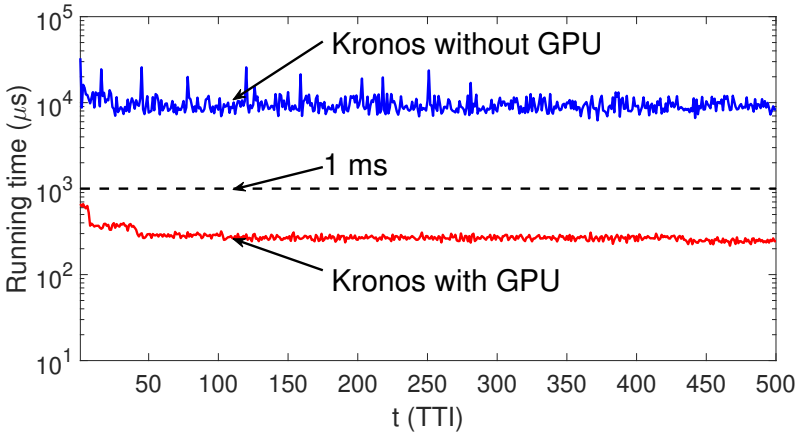


(b) $K = 2$

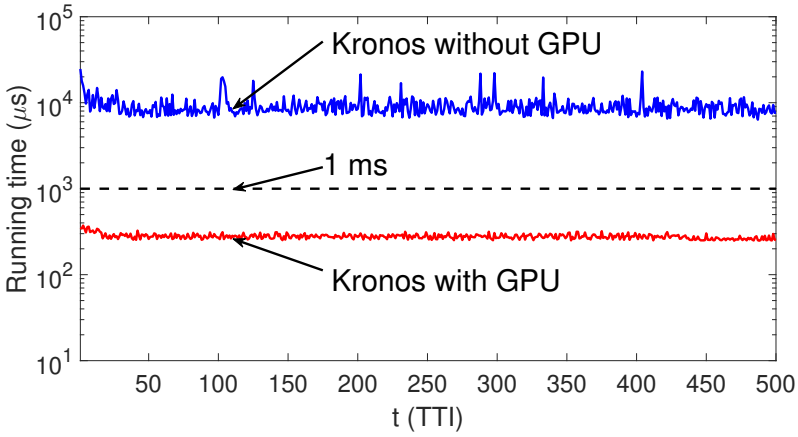


(c) $K = 10$

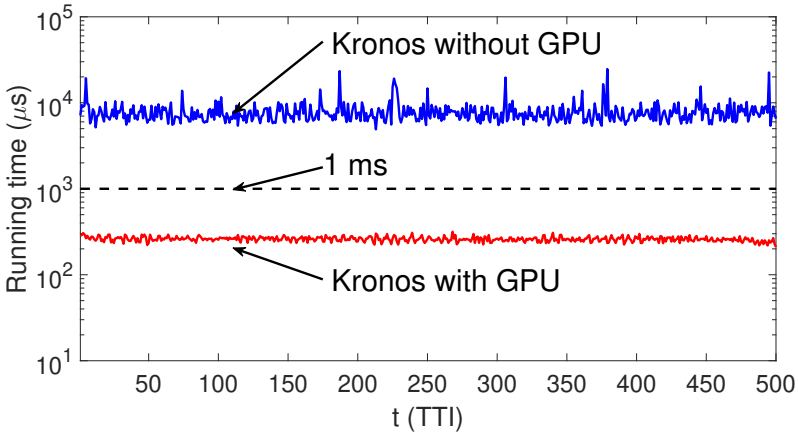
Figure 3.7: $\bar{A}^{\mathrm{B}}$ under different Rician factors.



(a) $K = 0$



(b) $K = 2$



(c) $K = 10$

Figure 3.8: Running time under different Rician factors.

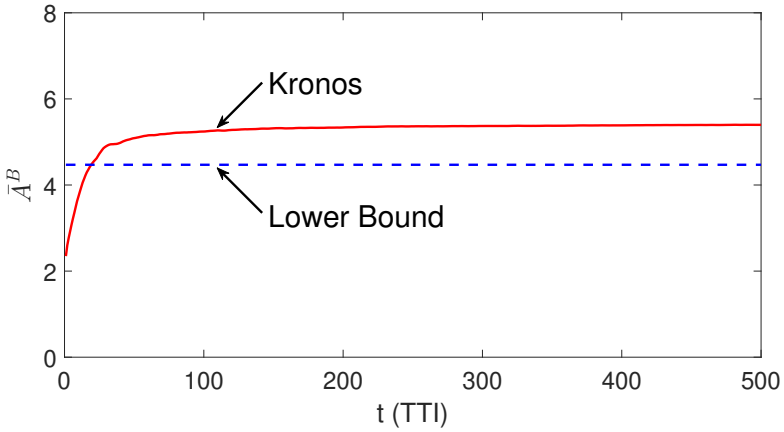
frequency correlation. We assume Rayleigh fading channels with no time correlation, and the coherence bandwidth is B_c, i.e., the channel conditions on adjacent B_c RBs are identical for each source node. We consider 100 source nodes (10 from each type in Table 3.2). The weight of each source node is also normalized.

Figure 3.9 shows the evolution of $\bar{A}^{\mathrm{B}}$ under Kronos across 500 TTIs for different coherence bandwidth B_c. Also shown in each sub-figure is the lower bound by the procedure in Algorithm 3.1. Clearly, we see Kronos can achieve near-optimal performance. In particular, when $B_c = 1$ (i.e, no frequency correlation), 4 and 10, the objectives of Kronos are respectively 20.8%, 17.7% and 15.8% within their respective lower bounds.

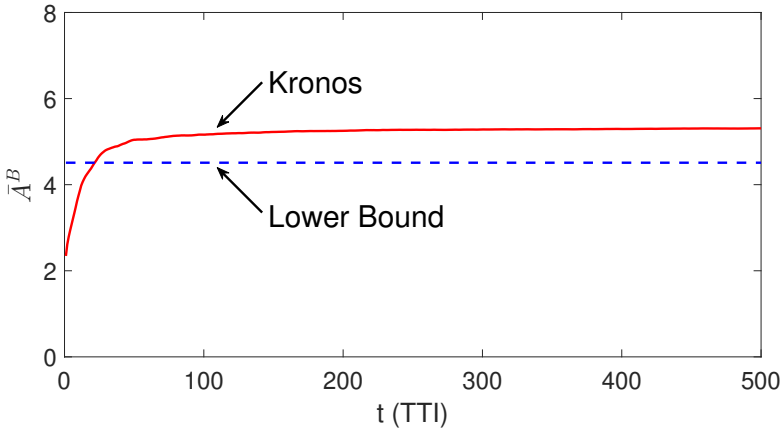
Figure 3.10 shows the running time for Kronos in each TTI with and without GPU implementation. As a benchmark, we also show the 5G timing requirement for numerology 0 (1 ms) in each sub-figure. We can see under all coherence bandwidth B_c, the running time of Kronos falls below 1 ms when it is implemented with GPU. When it is implemented only with CPU, it is about $\sim$10 ms. In particular, when $B_c = 1$, 4 and 10, the average running times of Kronos (with GPU implementation) are respectively 275 μs, 270 μs and 234 μs.

Varying Time Correlation Finally, we evaluate Kronos under channels with different time correlation. We assume Rayleigh fading channels with no frequency correlation, and the coherence bandwidth is T_c, i.e., the channel conditions on adjacent T_c TTIs are identical for each source node. We consider 100 source nodes (10 from each type in Table 3.2). The weight of each source node is also normalized.

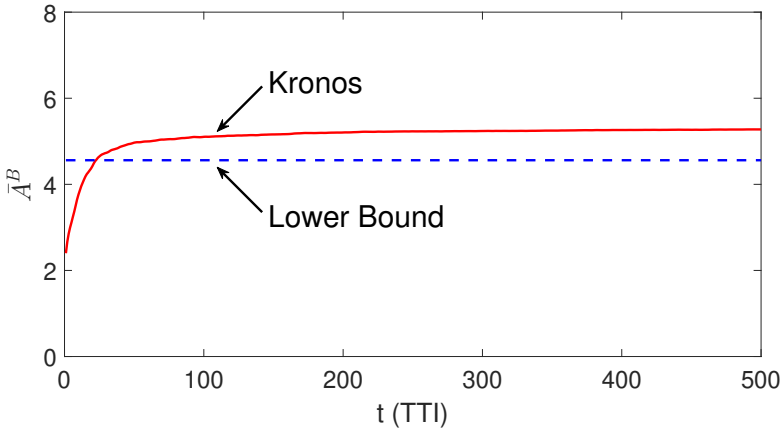
Figure 3.11 shows evolution of $\bar{A}^{\mathrm{B}}$ under Kronos across 500 TTIs for different coherence bandwidth T_c. Also shown in each sub-figure is the lower bound by the procedure in Algorithm 3.1. Clearly, we see Kronos can achieve near-optimal performance. In particular, when $T_c = 1$ (i.e, no time correlation), 2 and 5, the objectives of Kronos are respectively



(a) $B_c = 1$

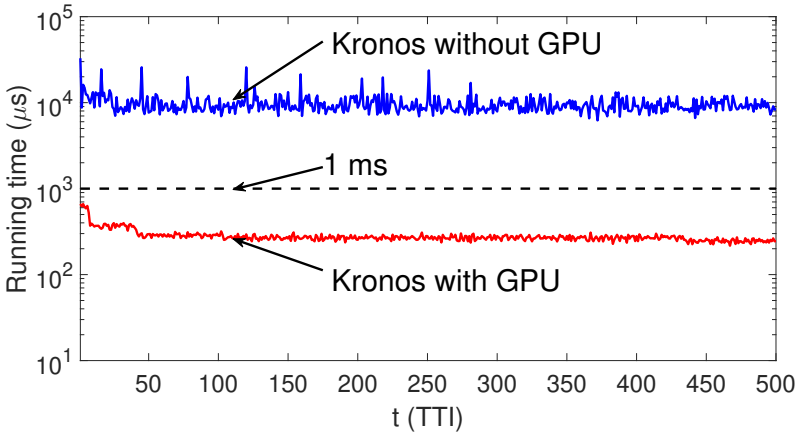


(b) $B_c = 4$

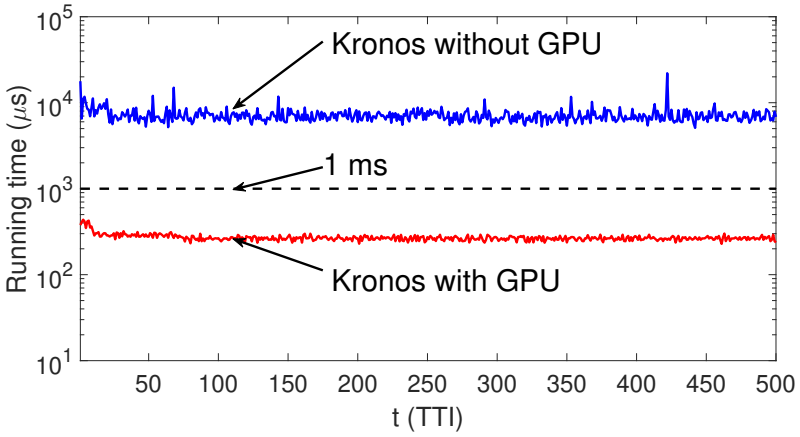


(c) $B_c = 10$

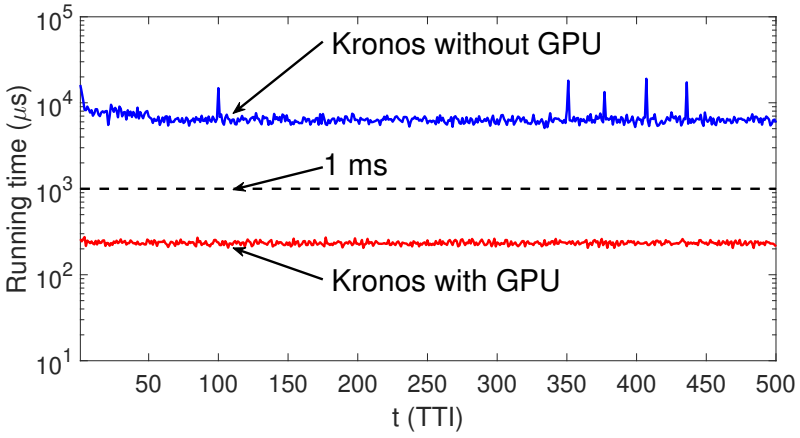
Figure 3.9: $\bar{A}^{\mathrm{B}}$ under different coherence bandwidth.



(a) $B_c = 1$

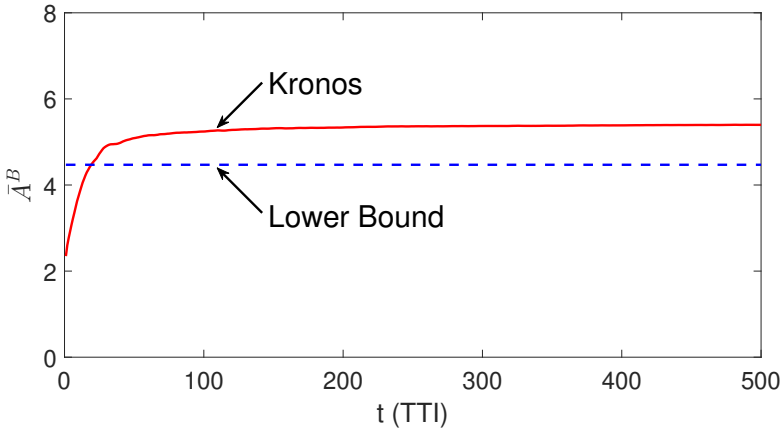


(b) $B_c = 4$

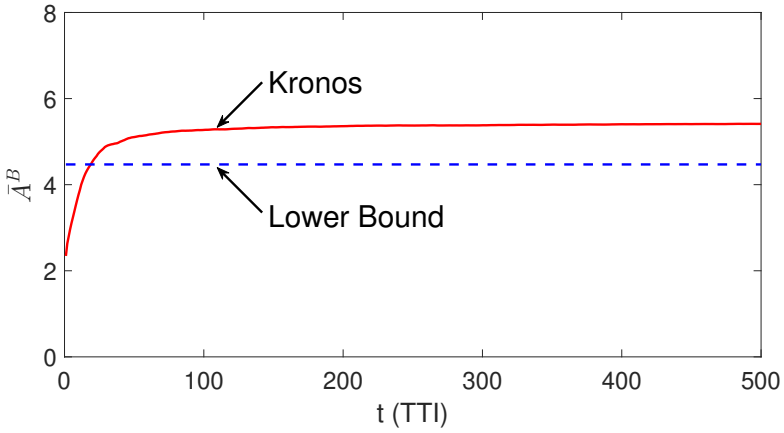


(c) $B_c = 10$

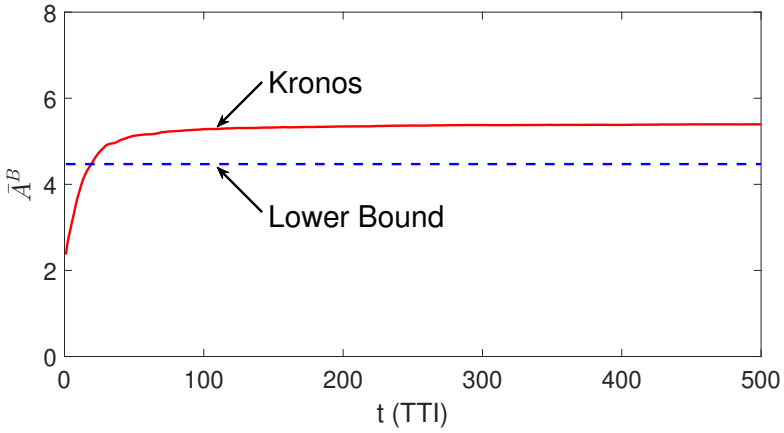
Figure 3.10: Running time under different coherence bandwidth.



(a) $T_c = 1$

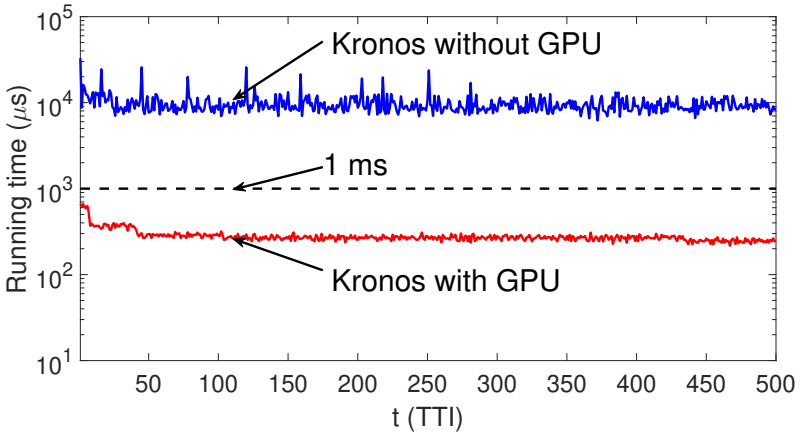


(b) $T_c = 2$

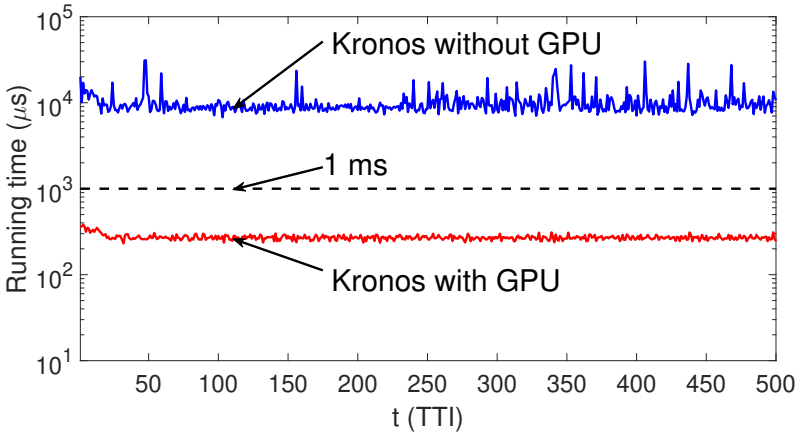


(c) $T_c = 5$

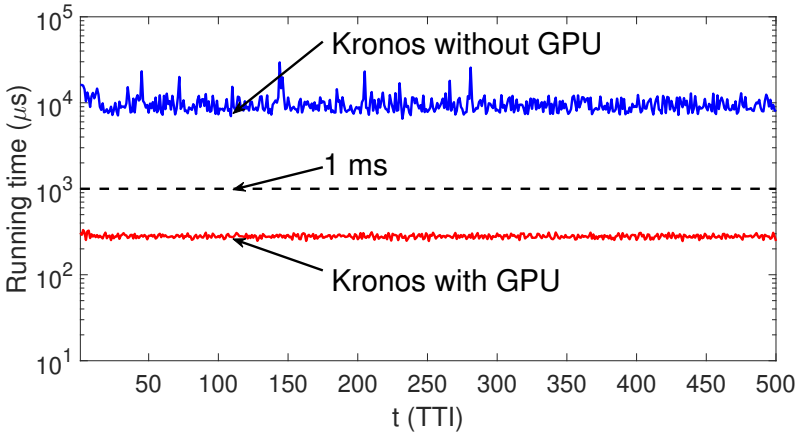
Figure 3.11: $\bar{A}^{\mathrm{B}}$ under different coherence time.



(a) $T_c = 1$



(b) $T_c = 2$



(c) $T_c = 5$

Figure 3.12: Running time under different coherence time.

20.8%, 20.6% and 21.0% within their respective lower bounds.

Figure 3.12 shows the running time for Kronos in each TTI with and without GPU implementation. As a benchmark, we also show the 5G timing requirement for numerology 0 (1 ms) in each sub-figure. We can see under all coherence bandwidth T_c, the running time of Kronos falls below 1 ms when it is implemented with GPU. When it is implemented only with CPU, it is about $\sim$10 ms. In particular, when $T_c = 1$, 2 and 5, the average running times of Kronos (with GPU implementation) are respectively 275 μs, 271 μs and 280 μs.

Summary of Results In summary, under all network settings (e.g., varying number of source nodes, channel propagation, time or frequency correlation), we find that the objective value achievement by our GPU-based implementation of Kronos is no more than 21% of the lower bounds. Since the optimal value of the objective (i.e., $\bar{A}^{\mathrm{B}}$) lies between Kronos and the lower bound, the gap between Kronos and the optimal value is even closer. Further, under all network settings, the average running time of Kronos (with GPU implementation) is under 0.3 ms, which meets the 5G timing requirement (e.g., numerology 0 and 1).

3.7 Chapter Summary

This chapter extended the results in Chapter 2, and presented Kronos—a 5G-compliant real-time scheduler to minimize average AoI. Kronos is capable of performing RB allocation and MCS selection for each source node based on channel conditions within each TTI (in sub-millisecond time scale). To cope with the enormous search space for optimal solution and the unknown nature of channel conditions, we developed a novel computation procedure to find an asymptotic lower bound for the objective as a performance benchmark. We further developed a novel metric, which is used by Kronos to select a source node, allocate RBs and determine MCS in each iteration. To meet the stringent timing requirement in 5G, we

proposed to implement Kronos on COTS GPU by exploiting its massive parallel computing capability. For demonstration, we implemented Kronos on an NVIDIA Tesla V100 GPU. Through extensive simulation experiments, we showed that Kronos can find a near-optimal AoI scheduling solution while complying 5G standard and its stringent timing requirement.

Chapter 4

AoI Scheduling with Hard Deadlines

4.1 Introduction

One of the most active lines of research on AoI is to design schedulers to minimize AoI (e.g., weighted average AoI) for all information sources at the network edge [9, 10, 11, 12, 13, 14, 24, 25, 26, 27, 28, 29, 30, 32, 33, 43, 45, 66, 67, 68, 69, 70, 71, 72, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91]. However, most of these efforts did not address some important application areas where there is a hard performance requirement on AoI metric. Although existing schedulers designed for AoI minimization has some remote relevance to AoI deadlines, it is easy to see that they are fundamentally different problems. Simply put, existing schedulers designed for AoI minimization cannot offer any guarantee on AoI deadlines. Further, by scanning the literature, there is a serious lack of current research in this area.

In this chapter, we address this issue by studying AoI scheduling under a *hard* AoI deadline for each source node. Specifically, this chapter addresses the following problems: (i) For a vector of deadline requirement for the source nodes, does there exist a feasible scheduler that can satisfy this requirement vector? (ii) If a feasible scheduler exists, then find such a scheduler. As we shall see, these two problems are intertwined with each other and are very different from the existing AoI minimization problems.

It is also instructive to see that these two problems are different from traditional task scheduling problems with deadlines [73, 74, 75, 76]. In particular, *Earliest Deadline First*

(EDF), the most well-known scheduler, was shown to be very efficient in the task scheduling problem [73, 74]. But we shall see that it performs poorly for our AoI problem in this chapter, due to the fundamental difference between the definitions of AoI and delay.

We summarize the main contributions of this chapter as follows:

- First, we prove that if there exists a feasible scheduler w.r.t. a deadline vector $\boldsymbol{d}$, then there must exist at least one feasible *cyclic* scheduler. This result allows us to narrow down the search space and to focus only on cyclic schedulers. Based on this result, we present an optimal solution called *Cyclic Scheduler Detection* (CSD) that can determine whether there exists a feasible scheduler with absolute certainty. The only limiting issue with CSD is its high complexity. So it is only useful for a small network.

- For a large network where CSD is not applicable, we pursue a fast (polynomial time complexity) procedure to solve our problem. We first give a definition for the so-called "system load" of a network, denoted as $l(\boldsymbol{d})$, and show that for any $\boldsymbol{d}$, if $l(\boldsymbol{d}) > 1$, then $\boldsymbol{d}$ is not schedulable. Then we identify a special type of deadline vector, called *polynomial* deadline vector, and present a low-complexity procedure called *Polynomial Scheduler Construction* (PSC) that can find a feasible scheduler for any polynomial deadline vector $\boldsymbol{d}$ with $l(\boldsymbol{d}) \leq 1$.

- For a general (non-polynomial) deadline vector $\boldsymbol{d}$, we propose to map it to a polynomial deadline vector $\boldsymbol{d}_1$ with $l(\boldsymbol{d}_1) \leq 1$ and subsequently construct a feasible scheduler for $\boldsymbol{d}$ based on this mapping. To better facilitate a successful mapping, we further generalize the definition of polynomial deadline vector to "fictitious polynomial" vector $\widetilde{\boldsymbol{d}}$, in which the elements are allowed to be fractions instead of just integers. Based on this generalization, we present a low-complexity procedure called *Fictitious Scheduler*

Construction (FSC), which can always find a feasible scheduler for $\boldsymbol{d}$ if it can be mapped to a $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \leq 1$. With FSC in hand, the only remaining issue to find a feasible scheduler for $\boldsymbol{d}$ is to find a mapping between $\boldsymbol{d}$ and a fictitious polynomial $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \leq 1$. To address this, we design a low-complexity procedure called *Fictitious Polynomial Mapping* (FPM), which is proved to be able to find a such mapping if there exists at least one such mapping.

- We prove that FPM is able to find a feasible scheduler for any $\boldsymbol{d}$ with $l(\boldsymbol{d}) < \ln 2$ ($\approx 69.3\%$). We also show that the cycle length of a feasible scheduler found by FPM is always no greater than the largest element in $\boldsymbol{d}$. Extensive numerical results are provided to validate the theoretical results and show that the performance of FPM is significantly better than a delay-optimized scheduler such as EDF.

4.2 System Model

Consider a network (see Fig. 1.1) consisting of N source nodes and one base station (BS). Each source node collects data (information sample) and forwards it to the BS through a shared wireless channel. Assume time is slotted and each source node takes a new sample at the beginning of each time slot. Due to limited channel capacity, not every sample collected at a source node can be sent to the BS. Upon a transmission opportunity, only the freshest (latest) sample at a source will be chosen for transmission. Similar to [10, 11, 12, 13], we assume the transmission of a sample takes one time slot. Therefore, at most one sample from a source node can be chosen for transmission in each time slot. Depending on the objective, a scheduler is needed to decide which sample will be chosen and transmitted in each time slot. Denote $\pi(t) \in \{0, 1, 2, \cdots, N\}$ as the scheduling decision for time slot t ($t = 0, 1, 2, \cdots$). Then when $\pi(t) = i$ and $i \geq 1$, the scheduler chooses source node i for transmission at t; when $\pi(t) = 0$, none of the source nodes is chosen for transmission at t.

Table 4.1: Notation

Symbol	Definition
$A_i(t)$	AoI for source node i at the BS in time slot t
c	Cycle length for a cyclic scheduler
$\boldsymbol{d}$	Hard deadline vector.
d_i	deadline for source node i
d_{max}	Biggest element in $\boldsymbol{d}$
$l(\boldsymbol{d})$	Network load for deadline set $\boldsymbol{d}$
N	Number of source nodes in the network
$\pi(t)$	Scheduling decision for time slot t
r_i	Data rate for source node i
$U_i(t)$	Generation time of the freshest (and complete) sample at the BS from source node i in time slot t

At the BS, it maintains the most recent (freshest) sample from each source that it has received. Once a new sample from a source node is received, the BS updates the current sample for this source node with this new one. For the sample from source node i that is currently maintained by the BS, denote $U_i(t)$ as its generation time at its source node. Then the AoI for source node i (as perceived by the BS), denoted as $A_i(t)$, can be defined as the elapsed time between now (t) and the generation time of the sample from this source node $U_i(t)$, i.e.,

$$A_i(t) = t - U_i(t). \tag{4.1}$$

Recall each source node generates a sample at each $t = 0, 1, 2, \cdots$. If the sample from source node i is chosen for transmission at t after it is generated (i.e. $\pi(t) = i$), then at time $(t+1)$, it will be received by the BS, i.e., $U_i(t+1) = t$ and

$$A_i(t+1) = t + 1 - U_i(t+1) = 1.$$

On the other hand, if the sample from source node i is not chosen for transmission at t (i.e., $\pi(t) \neq i$), then at time $(t+1)$, its AoI at the BS will increase by one. Combining both cases,

we have:

$$A_i(t+1) = \begin{cases} 1, & \text{if } \pi(t) = i, \\ A_i(t) + 1, & \text{otherwise.} \end{cases} \tag{4.2}$$

Initially, at time $t = 0$, we assume the system has just been turned on and there is no sample yet at the BS. So $A_i(0)$ for each i is "undefined". As time goes on, more and more samples from different source nodes will be received at the BS. Intuitively, an "undefined" AoI for a source node is "worse" than a very large AoI at the BS, as an undefined AoI does not offer any useful information, let alone to consider its "freshness". Therefore, whenever $A_i(t)$ remains undefined for source node i at the BS, our scheduler should consider a transmission of a sample from this source ASAP.

4.3 Problem Statement

In this chapter, we assume there is a hard AoI deadline, denoted by d_i, that is associated with each source node at the BS. d_i serves as an upper bound for $A_i(t)$ and our goal is to design a scheduler such that $A_i(t) \leq d_i$ for all $i = 1, 2, \cdots, N$. Note that at $t = 0$, $A_i(t)$'s are undefined for all i. So it only makes sense that we design a scheduler so that the above objective is achieved after some warm-up period.

Formally, we say a scheduler π is *feasible* if there exists a warm-up period t_0 such that for $t > t_0$, we have $A_i(t) \leq d_i$ for $i = 1, 2, \cdots, N$. Note that for practical purpose, t_0 should not be too large. We will address this issue in Section 4.4.1.

Denote $\boldsymbol{d} = [d_1 \ d_2 \ \cdots \ d_N]$ as the vector of deadlines for all source nodes. We say $\boldsymbol{d}$ is *schedulable* if there exists at least one feasible scheduler π. The problem that we want to address is to determine whether or not a given $\boldsymbol{d}$ is schedulable. If $\boldsymbol{d}$ is schedulable, we want to find at least one feasible scheduler to achieve it.

The above problem is very different from the existing research on minimizing AoI [9, 10, 11, 12, 13, 14, 24, 25, 26, 27, 28, 29, 30, 32, 33, 43, 45, 66, 67, 68, 69, 70, 71, 72, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91]. Specifically, most of these works attempted to minimize the weighted-sum long-term average AoI, $\bar{A} = \sum_{i=1}^{N} w_i \bar{A}_i$, where $\bar{A}_i$ is a long-term average AoI for source node i. Although $\bar{A}$ is minimized in the final solution, there is no concern of whether AoI for a source will exceed a deadline during the process. In other words, existing research on AoI minimization is conducted with no consideration of hard AoI requirement. But when such a requirement on AoI is present, it becomes a totally different problem, which we will address in this chapter.

4.4 Schedulability Check with A Cyclic Scheduler

In this section, we present an error-free procedure, named *Cyclic Scheduler Detection* (CSD), to determine the schedulability of $\boldsymbol{d}$. By "error-free", we mean that by executing CSD, we will be able to determine (with absolute certainty or no error) whether or not $\boldsymbol{d}$ is schedulable. The only issue with CSD is its high complexity (exponential w.r.t N), which will serve as the motivation of our work in Sections 4.5 and 4.6.

4.4.1 Existence of A Feasible Cyclic Scheduler

A scheduler may exhibit either cyclic or non-cyclic behavior. We say a scheduler is *cyclic* if its scheduling decision exhibits a periodic pattern over a finite number of time slots, i.e., $\pi_c(t) = \pi_c(t + c)$ for some constant c when $t \geq 0$. Here c is the cycle length of this cyclic scheduler. The following lemma helps us narrow down the search space for a feasible scheduler (w.r.t. $\boldsymbol{d}$) to only cyclic scheduler.

Lemma 4.1. *If a deadline vector $\boldsymbol{d}$ is schedulable, then there exists at least one cyclic*

scheduler that is feasible w.r.t. $\boldsymbol{d}$.

To prove Lemma 4.1, we define the *state* of AoI at the BS at time t (denoted as $\boldsymbol{s}(t)$) as a vector comprising of current values of AoI for all source nodes, i.e., $\boldsymbol{s}(t) = [A_1(t)\ A_2(t)\ \cdots\ A_N(t)]$. For two different time t_1 and t_2, if the current states and the scheduling decisions are both identical, i.e., $\boldsymbol{s}(t_1) = \boldsymbol{s}(t_2)$ and $\pi(t_1) = \pi(t_2)$, then $\boldsymbol{s}(t_1 + 1) = \boldsymbol{s}(t_2 + 1)$.

We now consider the possible state space under a feasible scheduler. After warm-up time t_0, for each source node i, we have $1 \leq A_i(t) \leq d_i$ (by definition of a feasible scheduler). We define the state space of feasible schedulers as a set $\mathcal{S}$ as

$$\mathcal{S} = \{\boldsymbol{s}(t) : |\ 1 \leq A_i(t) \leq d_i,\ i = 1, 2, \cdots, N\}. \tag{4.3}$$

Clearly, there is a total of $d_1 \cdot d_2 \cdots d_N$ unique states in $\mathcal{S}$.

Under a feasible cyclic scheduler π_c, for any consecutive d_i time slots, there must be at least one sample that is transmitted from each source node i (or the deadline d_i will not be satisfied and thus infeasible). Denote $d_{\max}$ as the largest deadline among all source nodes, i.e., $d_{\max} = \max_{i=1,2,\cdots,N}\{d_i\}$. Then by time $t = d_{\max}$, each source node should have been selected for transmission for at least once. Recall $U_i(t)$ is the generation time of the sample maintained at the BS at t for source node i. For any $t > d_{\max}$, the sample at the BS from source node i (with a generation time $U_i(t)$) must be chosen by the BS for transmission at time $U_i(t)$, i.e., $\pi_c\big(U_i(t)\big) = i$. Further, during the time interval $(U_i(t),\ t)$, no other sample(s) from source node i will be chosen for transmission (or the time stamp at the BS will not be $U_i(t)$). That is, for any τ s.t. $U_i(t) < \tau < t$, $\pi_c(\tau) \neq i$.

Since π_c is cyclic, we have $\pi_c(U_i(t) + c) = \pi_c\big(U_i(t)\big) = i$ and for $U_i(t) + c < \tau < t + c$, we have $\pi_c(\tau) \neq i$. This implies that the generation time of the sample maintained by the BS at $t + c$ is $U_i(t) + c$, i.e., $U_i(t + c) = U_i(t) + c$. Then for any $t > d_{\max}$, we have

$A_i(t+c) = t + c - U_i(t+c) = t - U_i(t) = A_i(t)$ for each source node i. That is, under a feasible cyclic scheduler, the evolution of state also exhibits a cyclic behavior, with a cycle length equaling the length of the scheduling cycle, i.e.,

$$\boldsymbol{s}(t) = \boldsymbol{s}(t+c), \text{ for } t > d_{\max}. \tag{4.4}$$

Based on the above analysis, we are now ready to prove Lemma 4.1.

Proof. Our proof is based on construction. If $\boldsymbol{d}$ is schedulable, then there exists a feasible scheduler $\pi(t)$ with a warm-up time t_0. Since there is a total of $d_1 \cdot d_2 \cdots d_N$ states that $\pi(t)$ can visit after t_0, there must exist a state that $\pi(t)$ visits at least twice over the time interval $[t_0 + 1, t_0 + d_1 \cdot d_2 \cdots d_N + 1]$. Suppose the two time instances that $\boldsymbol{s}(t)$ visit this state are t_1 and t_2, with $t_1 < t_2$. Then we have $\boldsymbol{s}(t_1) = \boldsymbol{s}(t_2)$. We can take the scheduling decisions within the time interval $[t_1,\ t_2 - 1]$ as one cycle, and repeat it to construct a feasible cyclic scheduler. □

By Lemma 4.1, to determine the schedulability of $\boldsymbol{d}$, we only need to check the existence of a feasible cyclic scheduler w.r.t. $\boldsymbol{d}$. If there exists one (through any construction), then $\boldsymbol{d}$ is schedulable; otherwise (i.e., there does not exist any feasible cyclic scheduler), then $\boldsymbol{d}$ is unschedulable.

Before we determine the existence of a feasible cyclic scheduler, we make a comment on the warm-up period t_0 for a feasible cyclic scheduler (if it exists). The following lemma shows one possible value for the warm-up period.

Lemma 4.2. *For any feasible cyclic scheduler, $t_0 = d_{\max}$ can be used as the warm-up period.*

Proof. To prove this lemma, it is sufficient to prove that for any feasible cyclic scheduler, when $t > d_{\max}$, $A_i(t) \leq d_i$ for $i = 1, 2, \cdots, N$.

Our proof is based on contradiction. Suppose under a feasible cyclic scheduler with a cycle length c, at time $t_1 > d_{\max}$, we have $A_i(t_1) > d_i$. Then from (4.4), we have $A_i(t_1 + nc) = A_i(t_1) > d_i$ for all $n \in \mathbb{N}$, which contradicts to the feasibility assumption of the cyclic scheduler (i.e., for any $t > t_0$, $A_i(t) \leq d_i$). This completes the proof. □

Based on Lemmas 4.1 and 4.2, we will focus on the design of a feasible cyclic scheduler w.r.t. $\boldsymbol{d}$. From this point on, we use "feasible scheduler" and "feasible cyclic scheduler" interchangeably in this chapter.

4.4.2 Detection of Feasible Cyclic Scheduler

In the last section, we showed that we can determine $\boldsymbol{d}$'s schedulability by determining the existence of a feasible cyclic scheduler w.r.t. $\boldsymbol{d}$. In this section, we show that we can reduce the latter problem to checking the existence of a cycle in a directed graph, which is a well-known problem with a known solution [77, 78].

To see how this is possible, we construct a state transition graph (STG) that consists of $d_1 \cdot d_2 \cdots d_N$ nodes (the maximum number of states for $\boldsymbol{s}(t)$). Each node in this STG represents a state in $\mathcal{S}$. For each node in this STG, there are N possible scheduling decisions. If a scheduling decision leads to a feasible state (i.e., another node in this STG), we draw a directed edge from the current node to the next node in the STG. Clearly, there is a one-to-one mapping between a feasible cyclic scheduler w.r.t. $\boldsymbol{d}$ and a cycle in STG. We have the following lemma.

Lemma 4.3. *The existence of a feasible cyclic scheduler w.r.t.* $\boldsymbol{d}$ *is equivalent to the existence*

of a cycle in STG.

The proof of Lemma 4.3 follows directly from the above discussion and is thus omitted here.

The following corollary follows from Lemma 4.3, which shows us a way to construct a feasible cyclic scheduler (if $\boldsymbol{d}$ is schedulable).

Corollary 4.1. *A cycle in STG constitutes a feasible cyclic scheduler w.r.t.* $\boldsymbol{d}$*, with each edge in the cycle corresponding to the scheduling decision for that state.*

Now we outline the CSD procedure in Algorithm 4.1. Based on Lemma 4.3, CSD is an error-free procedure.

There are some well-known solutions to check the existence of a cycle (and find one if there exists) in a directed graph, such as topological sorting [77] and Depth-First-Search (DFS) [78]. The time complexity of both algorithms is $O(|V| + |E|)$, where $|V|$ is the number of nodes and $|E|$ is the number of edges in the graph. In STG, $|V| = d_1 \cdot d_2 \cdots d_N$ and $|E| \leq N \cdot d_1 \cdot d_2 \cdots d_N$ (since there are at most N edges from each node). Therefore, the time complexity of CSD is $O(d_1 d_2 \cdots d_N) + O(N d_1 d_2 \cdots d_N) = O(N d_1 d_2 \cdots d_N)$. In practice, for $N > 1$, we have $d_i \geq 2$ for each source node i.[1] Therefore, the time complexity $O(N d_1 d_2 \cdots d_N)$ is no less than $O(N \cdot 2^N)$, which is exponential w.r.t. N. That is, although CSD can determine $\boldsymbol{d}$'s schedulability, its exponential time complexity poses a serious problem when N is large.

[1] If $d_i = 1$, source node i must transmit a sample in every time slot to achieve feasibility. This means other source nodes cannot transmit any samples and thus feasibility cannot be achieved.

Algorithm 4.1 CSD procedure

For a deadline vector $\boldsymbol{d}$:

1: Construct STG based on $\boldsymbol{d}$.

2: Detect whether there exists a cycle in STG. If there exists, use it to construct a feasible cyclic scheduler.

4.5 The Special Case of Polynomial Deadline Vectors

Before we start out to design any scheduling algorithm,[2] let us first clarify the main objectives of our scheduling algorithm, which we list as follows.

1. It should determine the schedulability of $\boldsymbol{d}$.
2. If $\boldsymbol{d}$ is determined to be schedulable, the procedure should be able to find a feasible scheduler w.r.t. $\boldsymbol{d}$.
3. The procedure should have a low time complexity.

Clearly, the procedure proposed in Section 4.4, CSD, meets the first two objectives but not the third one.

In our quest to find a procedure to achieve the above three objectives, we find that it is extremely difficult to have a procedure that meets all three objectives perfectly (unconditionally). In this section, as a first step to achieve our design objectives, we consider a special deadline vector, called *polynomial* deadline vector. We show that for this special deadline vector, we can design a procedure that meets all three objectives. In Section 4.6, we will use this procedure as a basis to design a procedure for the general (non-polynomial) deadline vectors.

[2]We use the term algorithm and procedure interchangeably in this chapter when there is no confusion.

4.5.1 Polynomial Deadline Vectors and System Load

We first give a definition of polynomial deadline vector.

Definition 4.1. *An deadline vector* $\boldsymbol{d}$ *is polynomial if* $d_i = b \cdot 2^{n_i}$ *for* $1 \leq i \leq N$*, where* b *is a positive integer and* n_i *is a non-negative integer.*

For example, $\boldsymbol{d} = [5\ 5\ 10\ 20\ 20\ 40]$ is a polynomial deadline vector with $b = 5$, $n_1 = n_2 = 0$, $n_3 = 1$, $n_4 = n_5 = 2$ and $n_6 = 3$. In Section 4.5.2, we will design a procedure that can find a feasible scheduler for a polynomial deadline vector $\boldsymbol{d}$ under a very general condition. To do this, we need a definition to tie traffic load and deadline.

First, we define the long-term average data rate for source node i under scheduler π as

$$r_i = \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} [\pi(t) = i], \tag{4.5}$$

where "$[\cdot]$" is Iverson bracket, returning 1 if the statement within is true and 0 otherwise [63]. The data rate r_i is a direct measure of the percentage of the time slots that are assigned to source node i for transmission. Since each time slot can be used for at most one such transmission, we have

$$\sum_{i=1}^{N} r_i \leq 1. \tag{4.6}$$

Eq. (4.6) gives an upper bound for the sum of rates. There is also a lower bound associated with each r_i. Specifically, to meet d_i for each source node i, there should be at least one transmission over consecutive d_i time slots. That is,

$$r_i \geq \frac{1}{d_i}, \quad i = 1, 2, \cdots, N. \tag{4.7}$$

Intuitively, $1/d_i$ represents the minimum guaranteed rate that a feasible scheduler should

provision to source node i. We define *system load* for a deadline vector $\boldsymbol{d}$ as

$$l(\boldsymbol{d}) = \sum_{i=1}^{N} \frac{1}{d_i}, \tag{4.8}$$

which represents the sum of minimum guaranteed rate that a feasible scheduler should provide to all source nodes. Clearly, by (4.6), any $\boldsymbol{d}$ with $l(\boldsymbol{d}) > 1$ is unschedulable, which is quite intuitive.

Naturally, we would like to use the system load as a metric in our design of a feasible scheduler. In the next section, we show that for $l(\boldsymbol{d}) = 1$ (maximum possible load), we can design a feasible scheduler when $\boldsymbol{d}$ is polynomial.

4.5.2 Scheduling for Polynomial Deadline Vectors

A Motivating Example. Consider six source nodes A, B, C, D, E, F and a polynomial deadline vector $\boldsymbol{d} = [3\ 6\ 6\ 6\ 12\ 12]$ corresponding to these six sources. It can be easily verified (based on our definitions in the last section) that $\boldsymbol{d}$ is polynomial and $l(\boldsymbol{d}) = 1$. We now show how to construct a feasible scheduler by exploiting the polynomial property.

Since the least common multiple (LCM) of the elements in $\boldsymbol{d}$ is 12, we set the cycle length to 12 time slots as follows:

$$(\square\square\square\square\square\square\square\square\square\square\square\square),$$

where each $\square$ inside the "()" represents a yet-to-be-determined scheduling decision in that particular time slot.

Since $l(\boldsymbol{d})$ is exactly 1, we must have $r_i = 1/d_i$ under a feasible scheduler. Thus, for each source node i, the length between two adjacent transmissions must be equal to d_i. Therefore, we can iteratively assign time slots to source node A, B, C, D, E, F, following the sequence $d_A \leq d_B \leq d_C \leq d_D \leq d_E \leq d_F$. In the first iteration, we assign the first and every $d_A = 3$

time slots to source node A. The cycle becomes

$$(A\square\square A\square\square A\square\square A\square\square).$$

In the second iteration, we assign the second time slot and every $d_B = 6$ time slots following it to source node B. Since d_B is an integral multiple of d_A, every d_B time slots after the second time slot is empty before this assignment. The cycle now becomes

$$(AB\square A\square\square AB\square A\square\square).$$

Following the same token, we make the assignment for the third, fourth, fifth and sixth iterations for source node C, D, E, and F, respectively. The final cycle is

$$(ABCADEABCADF),$$

and is a feasible scheduling cycle for $\boldsymbol{d}$. □

Based on the key ideas in the above motivating example, we outline a scheduling algorithm for polynomial deadline vectors, which we call *Polynomial Scheduler Construction* (PSC). Algorithm 4.2 shows the pseudocode of PSC.

For PSC, we have the following lemma.

Lemma 4.4. *For any polynomial deadline vector $\boldsymbol{d}$ with $l(\boldsymbol{d}) \leq 1$, PSC can always find a feasible scheduler w.r.t. $\boldsymbol{d}$.*

Proof. Our proof is based on contradiction. Suppose PSC cannot find a feasible scheduler for a polynomial $\boldsymbol{d}$ with $l(\boldsymbol{d}) \leq 1$. Then PSC must terminate at an iteration i_0 ($i_0 \leq N$), i.e., PSC failed to execute Step 4 or 5 at iteration i_0. Since $l(\boldsymbol{d}) \leq 1$ and in PSC we allocate $r_i = 1/d_i$ for each $i < i_0$, at the beginning of iteration i_0 there must be an empty time slot

Algorithm 4.2 A pseudocode for PSC

For a polynomial deadline vector $\boldsymbol{d}$ with $l(\boldsymbol{d}) \leq 1$:

1: Set the cycle length to $d_{\max}$ time slots.

2: Sort $\boldsymbol{d}$ such that $d_1 \leq d_2 \leq \cdots \leq d_N$.

3: **for** $i = 1, 2, \cdots, N$ **do**

4: Choose the first empty (unassigned) time slot in the cycle,

5: Assign this time slot and every d_i time slots following it (within the cycle) to source node i.

6: **end for**

in the cycle. Therefore, PSC cannot terminate on Step 4 at iteration i_0, so the reason of this termination is PSC cannot execute Step 5 at iteration i_0. Denote the first empty time slot at iteration i_0 as t_0. Then there exists an n such that time slot $(t_0 + nd_{i_0})$ is not empty, i.e., it has been allocated to another source node $i_1 < i_0$. However, since $\boldsymbol{d}$ is polynomial, d_{i_0} is an integer multiple of d_{i_1}. This means if time slot $(t_0 + nd_{i_0})$ has been allocated to source node i_1, then time slot t_0 must have been allocated to source node i_1 as well, which contradicts to the fact that t_0 is an empty time slot. This completes our proof. □

The significance of Lemma 4.4 is that for a polynomial deadline vector $\boldsymbol{d}$, PSC is guaranteed to find a feasible scheduler for system load $l(\boldsymbol{d})$ as high as 1.

To see PSC's time complexity, note that in each iteration, PSC will visit no more than $d_{\max}$ time slots in the cycle. So the time complexity for each iteration is $O(d_{\max})$. Since there are N iterations, the total time complexity of PSC is $O(Nd_{\max})$.

In summary, PSC meets all three design objectives when $\boldsymbol{d}$ is polynomial.

4.6 Scheduling for General Deadline Vectors

In this section, we consider the general case when $\boldsymbol{d}$ may not be polynomial. We present a novel algorithm called *Fictitious Polynomial Mapping* (FPM), which can meet all three objectives when $l(\boldsymbol{d}) < \ln 2$ ($\approx 69.3\%$) regardless of whether $\boldsymbol{d}$ is polynomial or not.

4.6.1 Basic Idea

The basic idea is to "map" a general (non-polynomial) deadline vector $\boldsymbol{d}$ that is under consideration to a polynomial deadline vector by "tightening" one or more elements in $\boldsymbol{d}$. In other words, we can always offer a source with a deadline that is smaller than its requirement. If we can do this mapping and the reference polynomial deadline vector has a load no greater than 1, then we can apply Lemma 4.4 and use PSC to find a feasible scheduler.

Example. For a general deadline vector $\boldsymbol{d} = [3\ 6\ 7\ 8\ 12\ 13]$, we can map (tighten) it to the polynomial deadline vector $\boldsymbol{d}_1 = [3\ 6\ 6\ 6\ 12\ 12]$. Since the polynomial deadline vector has a load $l(\boldsymbol{d}_1) = 1$, we can construct a feasible scheduler w.r.t. $\boldsymbol{d}_1$ and use the same scheduler to satisfy $\boldsymbol{d}$. □

Naturally, we ask the following question: Can we always map a schedulable deadline vector $\boldsymbol{d}$ to some polynomial deadline vector with a load no greater than 1? Unfortunately, the answer is no and can be illustrated in the following example. Consider four source nodes A, B, C, D and a non-polynomial deadline vector $\boldsymbol{d} = [3\ 5\ 5\ 5]$. For this $\boldsymbol{d}$, the smallest deadline (3, for source node A) can only be mapped to either 2 (tightening) or 3 (no change). If it is tightened to 2, then the other deadlines will be tightened to 4 (based on Definition 4.1), and thus $\boldsymbol{d}$ is mapped to the polynomial deadline vector $\boldsymbol{d}_1 = [2\ 4\ 4\ 4]$, with load $l(\boldsymbol{d}_1) = 1.25 > 1$, which is not helpful (we cannot use Lemma 4.4). If it is mapped to 3 (unchanged), then the other deadlines will be tightened to 3, and thus $\boldsymbol{d}$ is mapped to the

polynomial deadline vector $\boldsymbol{d}_2 = [3\ 3\ 3\ 3]$, with load $l(\boldsymbol{d}_2) = 1.33 > 1$, which is again not helpful. However, as we shall soon see, $\boldsymbol{d}$ is in fact schedulable, despite that it cannot be mapped to a polynomial deadline vector with a load no greater than 1.

To address the limitation of polynomial deadline vector, we introduce a concept called "fictitious polynomial" as follows.

Definition 4.2. *A vector* $\widetilde{\boldsymbol{d}} = [\widetilde{d}_1\ \widetilde{d}_2\ \cdots\ \widetilde{d}_N]$ *is fictitious polynomial if* $\widetilde{d}_i = b \cdot 2^{n_i}$ *for* $1 \leq i \leq N$, *where* b *is a positive integer and* n_i *is an integer.*

Note that the difference between Definition 4.2 and Definition 4.1 is only in n_i (positive integer or any integer value). But this difference is significant as $\widetilde{d}_i$ now can be a fraction instead of just being a positive integer as d_i. For example, $\widetilde{\boldsymbol{d}} = [\frac{7}{4}\ \frac{7}{2}\ 7\ 14\ 14]$ is not polynomial but is fictitious polynomial with $b = 7$, $n_1 = -2$, $n_2 = -1$, $n_3 = 0$, and $n_4 = n_5 = 1$. Note that $\widetilde{d}_1$ and $\widetilde{d}_2$ here are fractions.

The next question is: What is the benefit of allowing n_i to be negative (or $\widetilde{d}_i$ to be fractional) in this fictitious polynomial definition? The answer is that it will give us much bigger room for mapping. Let's go back to the previous example $\boldsymbol{d} = [3\ 5\ 5\ 5]$. Under Definition 4.1, the smallest deadline in $\boldsymbol{d}$ can only be mapped to 2 or 3, and both mapping will have an system load greater than 1, which is not helpful. However, under fictitious polynomial vector definition, we will have more options to map the smallest deadline (i.e., 3) onto, say $\frac{5}{2}$ (with $b = 5$) and $\frac{7}{4}$ (with b=7). Specifically, when 3 is mapped to $\frac{5}{2}$, the deadline vector $\boldsymbol{d}$ is mapped to the fictitious polynomial vector $\widetilde{\boldsymbol{d}} = [\frac{5}{2}\ 5\ 5\ 5]$, with a load $l(\widetilde{\boldsymbol{d}}) = 1$. In Section 4.6.2 we will present a low-complexity procedure to find a feasible scheduler w.r.t. any deadline vector $\boldsymbol{d}$ that can be mapped to a fictitious vector $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \leq 1$. In particular, for $\boldsymbol{d} = [3\ 5\ 5\ 5]$ with $\widetilde{\boldsymbol{d}} = [\frac{5}{2}\ 5\ 5\ 5]$, a feasible scheduler to $\boldsymbol{d}$ is $(ABACD)$. The readers can easily verify its feasibility.

4.6.2 Scheduling for Fictitious Polynomial deadline vectors

There are two results in this section. The first result is stated in the following theorem.

Theorem 4.1. *For any deadline vector* $\boldsymbol{d}$ *that can be mapped to a fictitious polynomial vector* $\widetilde{\boldsymbol{d}}$ *with* $l(\widetilde{\boldsymbol{d}}) \leq 1$*, there exist a feasible scheduler w.r.t.* $\boldsymbol{d}$*.*

A proof of Theorem 4.1 is based on constructing one such feasible scheduler, which must also be of low-complexity procedure. In this section, we present one such scheduler, called *Fictitious Scheduler Construction* (FSC). FSC is the second main result of this section.

FSC is best explained with an example.

Example. Consider six source nodes A, B, C, D, E, F with $\boldsymbol{d} = [3\ 5\ 9\ 11\ 19\ 21]$. During the initialization phase, we map $\boldsymbol{d}$ to a fictitious polynomial vector $\widetilde{\boldsymbol{d}} = [\frac{9}{4}\ \frac{9}{2}\ 9\ 9\ 18\ 18]$ with $b = 9$ and $l(\widetilde{\boldsymbol{d}}) = 1$ (we will show how to find this $\widetilde{\boldsymbol{d}}$ in Section 4.6.3). Now, we focus on showing how to construct a feasible scheduler w.r.t. $\boldsymbol{d}$ based on $\widetilde{\boldsymbol{d}}$.

In Section 4.5.2, we used LCM for d_i's as cycle length where d_i's are all integers. But $\widetilde{d}_i$'s in $\widetilde{\boldsymbol{d}}$ can be fractions and it's necessary to generalize the definition of LCM. We define *fictitious common multiple* (FCM) for $\widetilde{d}_i$'s in $\widetilde{\boldsymbol{d}}$ as the smallest integer m such that $m/\widetilde{d}_i$ is a positive integer for all $1 \leq i \leq N$.

Since the FCM of $\widetilde{d}_i$'s in $\widetilde{\boldsymbol{d}}$ is 18, we set the cycle length $c = 18$. Denote N_i as the number of time slots scheduled for source node i in a cycle c. As we did in Section 4.5.2, we reserve a long-term average data rate $r_i = 1/\widetilde{d}_i$ for each source node i. It's easy to see $\sum_{i=1}^{N} r_i \leq 1$ when $l(\widetilde{\boldsymbol{d}}) \leq 1$. Then we allocate $N_i = c \cdot r_i = c/\widetilde{d}_i$ for each node i, and we have $N_A = 8$, $N_B = 4$, $N_C = N_D = 2$, and $N_E = N_F = 1$.

We propose a recursive procedure to construct a feasible scheduler for $\boldsymbol{d}$, as shown in

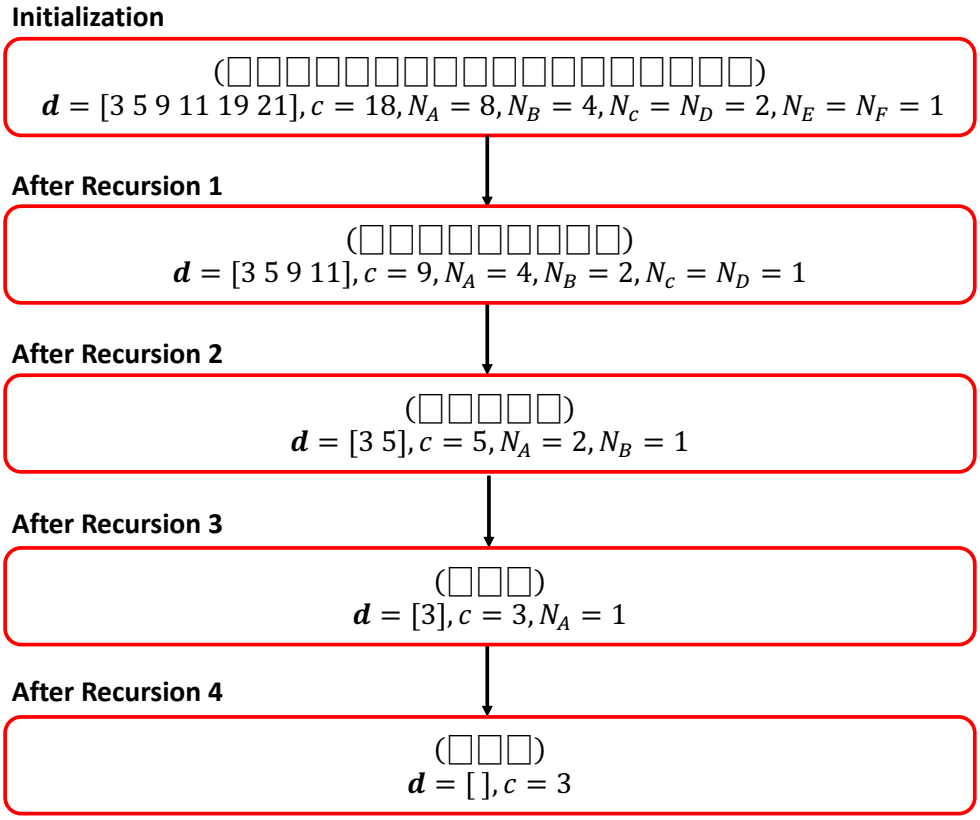


Figure 4.1: Recursion in the example.

Figure 4.1. The original problem (after Initialization) is reduced to a smaller problem after each recursion and degenerates into a trivial problem after the last recursion. We will show how this recursive procedure works.

After initialization (as we discussed above), we have $\boldsymbol{d} = [3\ 5\ 9\ 11\ 19\ 21]$, $\widetilde{\boldsymbol{d}} = [\frac{9}{4}\ \frac{9}{2}\ 9\ 9\ 18\ 18]$, $l(\widetilde{\boldsymbol{d}}) = 1$, $c = 18$ and $N_A = 8$, $N_B = 4$, $N_C = N_D = 2$, $N_E = N_F = 1$.

At the beginning of Recursion 1, $N_E = N_F = 1$, which means we need to assign one time slot to each source node E and F in a cycle. Since this one time slot assignment can be made anywhere in the cycle, we can defer this assignment later. For now, we can remove nodes E and F from $\boldsymbol{d}$ and $\widetilde{\boldsymbol{d}}$, and assign 16 time slots to other source node A, B, C, D in the cycle with $c = 18$ time slots (and later use any remaining two time slots to assign to nodes

E and F). Since $c = 18$, $N_A = 8$, $N_B = 4$, $N_C = 2$, and $N_D = 2$ are all even, it is sufficient to construct a feasible scheduler with $c \leftarrow c/2$ and $N_A \leftarrow N_A/2$, $N_B \leftarrow N_B/2$, $N_C \leftarrow N_C/2$, $N_D \leftarrow N_D/2$. Then we can combine the two identical cycles together and form a full cycle with length 18.

Therefore, after Recursion 1, we have $\boldsymbol{d} = [3\ 5\ 9\ 11]$, $\widetilde{\boldsymbol{d}} = [\frac{9}{4}\ \frac{9}{2}\ 9\ 9]$, $l(\widetilde{\boldsymbol{d}}) = \frac{8}{9} < 1$, and $c = 9$, $N_A = 4$, $N_B = 2$, $N_C = N_D = 1$.

At the beginning of Recursion 2, we have $N_C = N_D = 1$. Again, we will first remove nodes C and D from $\boldsymbol{d}$ and $\widetilde{\boldsymbol{d}}$, and then assign 6 time slots to source node A (with $N_A = 4$) and B (with $N_B = 2$) in the cycle with length $c = 9$. Since N_A and N_B are both even, we would like to divide the current cycle into two identical but smaller cycles. But since the current cycle length $c = 9$ is odd, we need to do some extra work here. Now we construct a feasible scheduler with $c \leftarrow \frac{c+1}{2}$ (which is 5) and $N_A \leftarrow N_A/2$, $N_B \leftarrow N_B/2$. Then we can combine the two identical cycles together and form a full cycle with length 10, and remove one empty time slot to get a length 9. Note that removing an empty time slot won't increase the AoI for any source node. With this cycle length reduction, we update each element in $\widetilde{\boldsymbol{d}}$ with a factor $\frac{c+1}{c}$, which is 10/9.

Therefore, after Recursion 2, we have $\boldsymbol{d} = [3\ 5]$, $\widetilde{\boldsymbol{d}} = [\frac{5}{2}\ 5]$, $l(\widetilde{\boldsymbol{d}}) = \frac{3}{5} < 1$, and $c = 5$, $N_A = 2$, $N_B = 1$.

Following the same idea in the previous recursive steps, the problem is reduced to a trivial problem after Recursion 4: to construct an empty cycle with $c = 3$. After constructing it, we go back step-by-step in the recursion and construct the following feasible scheduler w.r.t. the original $\boldsymbol{d}$: $(ABCADABEAABCADABFA)$. The readers can easily verify its feasibility. □

Based on the key recursive ideas in the above example (i.e., remove some nodes that

require only 1 time slot and cut down cycle length in half in each step), we give a pseudocode for FSC in Algorithm 4.3.

With FSC in hand, we now prove Theorem 4.1 by showing FSC can construct a feasible scheduler for any $\boldsymbol{d}$ that can be mapped to $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \leq 1$.

Proof. Suppose $\boldsymbol{d}$ can be mapped to $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \leq 1$. We are going to prove FSC can find a feasible scheduler for it.

At the beginning of each recursion of FSC, we have a $\boldsymbol{d}$ that can be mapped to $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \leq 1$ and cycle length c (the FCM of $\widetilde{d}_i$'s). We are going to show that if $\boldsymbol{d} \neq \emptyset$, FSC can reduce the problem to a smaller problem, $\boldsymbol{d}_1$, that can be mapped to $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \leq 1$, with $\dim(\boldsymbol{d}_1) < \dim(\boldsymbol{d})$.

We discuss c's parity.

- If c is even, FSC will execute Step 3 and 4. Since $\boldsymbol{d}$ can be mapped to $\widetilde{\boldsymbol{d}}$, $\boldsymbol{d}_1$ can also be mapped to $\widetilde{\boldsymbol{d}}_1$. Besides, $l(\widetilde{\boldsymbol{d}}) \leq l(\widetilde{\boldsymbol{d}}_1) \leq 1$. Therefore, FSC cam successfully reduce the problem (w.r.t. $\boldsymbol{d}$) to a smaller problem (w.r.t. $\boldsymbol{d}_1$).

- If c is odd, FSC will execute Step 6, 7 and 9. For each i such that $\widetilde{d}_i \in \widetilde{\boldsymbol{d}}_1$, we have $\widetilde{d}_i = c/2^{n_i}$, where n_i is positive (since $\widetilde{d}_i < c$). Besides, for each i such that $d_i \in \boldsymbol{d}_1$, we have $d_i \geq \widetilde{d}_i$ and d_i is an integer. Therefore, for each i such that $d_i \in \boldsymbol{d}_1$, we have $d_i \geq \lceil \widetilde{d}_i \rceil \geq \frac{c+1}{2^{n_i}} = \frac{c+1}{c}\widetilde{d}_i$, which means $\boldsymbol{d}_1$ can be mapped to $\frac{c+1}{c}\widetilde{\boldsymbol{d}}_1$. Besides, $l(\frac{c+1}{c}\widetilde{\boldsymbol{d}}_1) = \frac{c}{c+1} \cdot l(\widetilde{\boldsymbol{d}}_1) \leq \frac{c}{c+1} < 1$. Therefore, FSC can successfully reduce the problem (w.r.t. $\boldsymbol{d}$) to a smaller problem (w.r.t. $\boldsymbol{d}_1$). Note that since $l(\widetilde{\boldsymbol{d}}_3) \leq \frac{c}{c+1}$, there must be

at least one empty time slot in the cycle constructed in Step 6. So Step 7 can always find an empty time slot to remove and construct a feasible cycle with length c.

Therefore, for $\boldsymbol{d}_1 \neq \emptyset$, FSC can reduce the problem to another problem with a smaller c.

When $\boldsymbol{d}_1 \neq \emptyset$, we have $c > 0$, since in the previous steps neither $c \leftarrow c/2$ nor $c \leftarrow (c+1)/2$ can make $c = 0$. Therefore, Step 9 can successfully construct an empty cycle with length c. This completes our proof. □

4.6.3 Mapping a General deadline vector to a Fictitious Polynomial Vector

In this last section, we will show how to map $\boldsymbol{d}$ into $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \leq 1$. Recall this is a necessary step in the initialization phase of FSC in the last section.

Note that there are infinite $\widetilde{\boldsymbol{d}}$'s that $\boldsymbol{d}$ can be mapped into, and we only need to check whether one of them satisfies $l(\widetilde{\boldsymbol{d}}) \leq 1$. We introduce the following lemma to narrow down the search space of $\widetilde{\boldsymbol{d}}$.

Lemma 4.5. *To check whether $\boldsymbol{d}$ can be mapped to a $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \leq 1$, it's sufficient to check those $\widetilde{\boldsymbol{d}}$'s with $d_i = \widetilde{d}_i$ for at least one i.*

Proof. We prove this lemma by the construction of $\widetilde{\boldsymbol{d}}_0 = [\widetilde{d}_{01}\ \widetilde{d}_{02}\ \cdots\ \widetilde{d}_{0N}]$ which satisfies $d_i = \widetilde{d}_{0i}$ for at least one i. Specifically, if $\boldsymbol{d}$ can be mapped to $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \leq 1$, then we construct $\widetilde{\boldsymbol{d}}_0$ as $\widetilde{\boldsymbol{d}}_0 = \min_i\{d_i/\widetilde{d}_i\} \cdot \widetilde{\boldsymbol{d}}$. Clearly, $\widetilde{\boldsymbol{d}}_0$ is a fictitious polynomial vector that $\boldsymbol{d}$ can be mapped into, $l(\widetilde{\boldsymbol{d}}_0) \leq l(\widetilde{\boldsymbol{d}}) \leq 1$, and there exists at least one i such that $d_i = \widetilde{d}_{0i}$. This completes the proof. □

Algorithm 4.3 A pseudocode for FSC.

FSC: Find a feasible scheduler w.r.t. $\boldsymbol{d}$ that can be mapped to $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \leq 1$.

Initialization: Set cycle length c to the FCM of $\widetilde{d}_i$'s in $\widetilde{\boldsymbol{d}}$;

Set $N_i \leftarrow c/\widetilde{d}_i$ for $i = 1, 2, \cdots, N$.

Recursion:

1: Separate $\widetilde{\boldsymbol{d}}$ into two vectors $\widetilde{\boldsymbol{d}}_1$ and $\widetilde{\boldsymbol{d}}_2$, such that: (i) all $\widetilde{d}_i$ with $N_i \geq 2$ are the elements of $\widetilde{\boldsymbol{d}}_1$; (ii) all $\widetilde{d}_i$ with $N_i = 1$ are the elements in $\widetilde{\boldsymbol{d}}_2$.

Also, separate $\boldsymbol{d}$ into $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ accordingly.

2: **if** integer c is even and $\dim(\boldsymbol{d}_1) > 0$ **then**

3: Call **Recursion** to construct a feasible cycle for $\boldsymbol{d}_1$, with $\widetilde{\boldsymbol{d}}_1$, $c \leftarrow c/2$, $N_i \leftarrow N_i/2$.

4: Combine two identical cycles together and form one full cycle.

5: **else if** integer c is odd and $\dim(\boldsymbol{d}_1) > 0$ **then**

6: Call **Recursion** to construct a feasible cycle for $\boldsymbol{d}_1$, with $\widetilde{\boldsymbol{d}}_1 \leftarrow \frac{c+1}{c}\widetilde{\boldsymbol{d}}_1$, $c \leftarrow \frac{c+1}{2}$, $N_i \leftarrow N_i/2$.

7: Combine two identical cycles together and form one full cycle, and remove one empty time slot in the cycle.

8: **else**

9: Construct an empty cycle with length c.

10: **end if**

11: Arbitrarily assign one empty time slot in the cycle to each source node in $\boldsymbol{d}_2$.

Algorithm 4.4 A Pseudocode of FPM.

For a deadline vector $\boldsymbol{d}$:

1: **for** $i = 1, 2, \cdots, N$ **do**

2: Compute $\widetilde{d}_j$ for each $1 \leq j \leq N$ by (4.9). Compute $l(\widetilde{\boldsymbol{d}}) = \sum_{j=1}^{N} 1/\widetilde{d}_j$.

3: **if** $l(\widetilde{\boldsymbol{d}}) \leq 1$ **then**

4: Call FSC to construct a feasible scheduler w.r.t. $\boldsymbol{d}$ and **break**.

5: **end if**

6: **end for**

Based on Lemma 4.5, to map $\boldsymbol{d}$ into $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \leq 1$, it's sufficient to test N fictitious polynomial vectors with $\widetilde{d}_i = d_i$, where $i \in \{1, 2, \cdots, N\}$. More specifically, for each $i \in \{1, 2, \cdots, N\}$, we compute the N elements in $\widetilde{\boldsymbol{d}}$ by

$$\widetilde{d}_j = d_i \cdot 2^{\lfloor \log_2(\frac{d_j}{d_i}) \rfloor}, \text{ for each } j \in \{1, 2, \cdots, N\}. \tag{4.9}$$

Eq. (4.9) guarantees that $\widetilde{d}_j \leq d_j$ for each j, $\widetilde{\boldsymbol{d}}$ is fictitious polynomial, and $\widetilde{d}_i = d_i$. If for one i we have $l(\widetilde{\boldsymbol{d}}) \leq 1$, we can use FSC to construct a feasible scheduler. If for all i's we have $l(\widetilde{\boldsymbol{d}}) > 1$, then $\boldsymbol{d}$ cannot be mapped to any $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \leq 1$.

Now we can finalize the FPM procedure to find a feasible scheduler for non-polynomial $\boldsymbol{d}$, as shown in Algorithm 4.4.

The following corollary directly follows from Lemma 4.5.

Corollary 4.2. *If $\mathbf{d}$ can be mapped to any $\widetilde{\mathbf{d}}$ with $l(\widetilde{\mathbf{d}}) \leq 1$, FPM can always find a feasible scheduler w.r.t. $\mathbf{d}$.*

The following lemma shows the cycle length of the feasible scheduler found by FPM is at most $d_{\max}$, which is both interesting and significant for practical implementation.

Lemma 4.6. *The cycle length of the feasible scheduler found by FPM is no greater than* $d_{\max}$.

Proof. Suppose at iteration i, $\widetilde{d}_i = d_i$, $l(\widetilde{\boldsymbol{d}}) \leq 1$ and FPM calls FSC to construct a feasible scheduler. For the largest element in $\widetilde{\boldsymbol{d}}$, denoted by $\widetilde{d}_{\max}$, we have $\widetilde{d}_{\max} = d_i \cdot 2^{\lfloor \log_2(\frac{d_{\max}}{d_i}) \rfloor}$ is a positive integer. By definition, $\widetilde{d}_{\max}$ is the FCM of the elements in $\widetilde{\boldsymbol{d}}$ (as $\widetilde{d}_{\max}$ is an integer here). So we have the cycle length $c = \widetilde{d}_{\max} \leq d_{\max}$. □

We now analyze the time complexity of FPM. FPM computes at most N different $l(\widetilde{\boldsymbol{d}})$'s. Since the complexity of computing one $l(\widetilde{\boldsymbol{d}})$ is $O(N)$, the complexity for computing all $l(\widetilde{\boldsymbol{d}})$'s is $O(N^2)$. If there exists a $\widetilde{\boldsymbol{d}}$ such that $l(\widetilde{\boldsymbol{d}}) \leq 1$, then FPM will call FSC (at most once). By Lemma 4.6, we have $c \leq d_{\max}$ in FSC. Since after each recursion c is reduced to $\lceil \frac{c}{2} \rceil$, there are a total of $O(\log c)$ recursions. The complexity of each recursion[3] is $O(c)$ (since the cycle length is always no greater than c). Therefore, the time complexity of FSC (called by FPM) is $O(c \log c) = O(d_{\max} \cdot \log d_{\max})$. So the total time complexity of FPM is $O(N^2) + O(d_{\max} \cdot \log d_{\max})$.

4.7 System Load vs. Schedulability

In the last section, by introducing the notation of mapping from physical $\boldsymbol{d}$ to fictitious $\widetilde{\boldsymbol{d}}$, we showed that we can construct a feasible scheduler w.r.t. $\boldsymbol{d}$ as long as $l(\widetilde{\boldsymbol{d}}) \leq 1$. However, since $d_i \geq \widetilde{d}_i$, we have $l(\boldsymbol{d}) \leq l(\widetilde{\boldsymbol{d}})$. Even though $l(\widetilde{\boldsymbol{d}})$ may be close to 1, it is still not clear how large $l(\boldsymbol{d})$ can be. A natural question becomes: What is the maximum load $l(\boldsymbol{d})$ while still guaranteeing to find a feasible scheduler? We answer this question in this section.

[3]Here we assume $N \leq d_{\max}$, because any $\boldsymbol{d}$ with $N > d_{\max}$ is clearly unschedulable (with $l(\boldsymbol{d}) > 1$).

By the mapping in (4.9), it is easy to see $d_i < 2 \cdot \widetilde{d}_i$ and $l(\widetilde{\boldsymbol{d}}) < 2 \cdot l(\boldsymbol{d})$. So for all $\boldsymbol{d}$ with $l(\boldsymbol{d}) \leq 0.5$, we are guaranteed to find a feasible scheduler based on Theorem 4.1. The following proposition shows that we can in fact do better than this.

Proposition 4.1. *For any $\boldsymbol{d}$ with $l(\boldsymbol{d}) < \ln 2$, FPM can always construct a feasible scheduler w.r.t. $\boldsymbol{d}$.*

Proof. We prove this proposition by proving its contrapositive, i.e., if FPM cannot construct a feasible scheduler w.r.t. $\boldsymbol{d}$, then $l(\boldsymbol{d}) > \ln 2$.

Suppose FPM cannot construct a feasible scheduler w.r.t. $\boldsymbol{d}$. Then FPM cannot find a mapping to $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \leq 1$, so we have

$$l(\widetilde{\boldsymbol{d}}) = \sum_{j=1}^{N} \frac{1}{d_i \cdot 2^{\lfloor \log_2(\frac{d_j}{d_i}) \rfloor}} > 1, \text{ for } i = 1, 2, \cdots, N. \tag{4.10}$$

Define the following function $f(x)$:

$$f(x) = \sum_{j=1}^{N} \frac{1}{x \cdot 2^{\lfloor \log_2(\frac{d_j}{x}) \rfloor}} \quad \text{for } x \in \mathbb{R}_{>0}. \tag{4.11}$$

Clearly, we have $f(2x) = f(x)$ for any x. Denote x^* as the optimal value that minimizes $f(x)$. Then d_i/x^* must be a power of 2 for at least one i, otherwise we can slightly increase x^* to get a smaller $f(x)$. So we have $f(x^*) = f(d_i)$ for at least one i. From (4.10) and (4.11),

we have $f(d_i) > 1$ for $i = 1, 2, \cdots, N$. So we have

$$f(x) > 1 \quad \text{for } x \in \mathbb{R}_{>0}. \tag{4.12}$$

Define $y = 1/x$. Replacing x with $1/y$ in (4.12), along with (4.11), we have

$$\sum_{j=1}^{N} y \cdot 2^{-\lfloor \log_2(d_j y) \rfloor} > 1, \quad \text{for } y \in \mathbb{R}_{>0}. \tag{4.13}$$

Define $g(y) = \frac{1}{y \cdot \ln 2}$. We have

$$\int_{0.5}^{1} g(y) dy = 1 \tag{4.14}$$

Define u_j as

$$u_j = \frac{1}{d_j} \cdot 2^{\lfloor \log_2(d_j) \rfloor}, \ j = 1, 2, \cdots, N. \tag{4.15}$$

We have $u_j \in (0.5, 1]$ for each $j = 1, 2, \cdots, N$.

Multiplying the two sides of (4.13) and (4.14) respectively, we have

$$\int_{0.5}^{1} g(y) \cdot \sum_{j=1}^{N} y \cdot 2^{-\lfloor \log_2(d_j y) \rfloor} dy > 1. \tag{4.16}$$

Substituting $g(y) = \frac{1}{y \cdot \ln 2}$ into (4.16), we have

$$\frac{1}{\ln 2} \sum_{j=1}^{N} \int_{0.5}^{1} 2^{-\lfloor \log_2(d_j y) \rfloor} dy > 1. \tag{4.17}$$

Breaking the integral in (4.17) with u_j, we have

$$\frac{1}{\ln 2}\sum_{j=1}^{N}\Big(\int_{0.5}^{u_j} 2^{-\lfloor \log_2(d_j)\rfloor+1}dy + \int_{u_j}^{1} 2^{-\lfloor \log_2(d_j)\rfloor}dy\Big) > 1,$$

which gives us:

$$\frac{1}{\ln 2}\sum_{j=1}^{N}\Big(2^{-\lfloor \log_2(d_j)\rfloor}\cdot(2u_j-1) + 2^{-\lfloor \log_2(d_j)\rfloor}\cdot(1-u_j)\Big) > 1,$$

or equivalently,

$$\frac{1}{\ln 2}\sum_{j=1}^{N} u_j \cdot 2^{-\lfloor \log_2(d_j)\rfloor} > 1. \tag{4.18}$$

Substituting (4.15) into (4.18), we have

$$\frac{1}{\ln 2}\sum_{j=1}^{N}\frac{1}{d_j} > 1,$$

which gives us

$$l(\boldsymbol{d}) > \ln 2.$$

This completes our proof. □

Proposition 4.1 tells us $\ln 2$ is a valid guarantee for FPM. We will then show it is also the best guarantee that FPM can offer. Consider a special group of deadline vectors, denoted by $\boldsymbol{d}^n$, which is defined as $\boldsymbol{d}^n = [n\ \ n+1\ \ n+2\ \ \cdots\ \ 2n-1\ \ 2n]$. It can be shown that for any $n \in \mathbb{N}^+$, $\boldsymbol{d}^n$ cannot be mapped to any $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \le 1$, thus FPM cannot find a feasible scheduler for $\boldsymbol{d}^n$. Considering $\lim_{n\to\infty} l(\boldsymbol{d}^n) = \ln 2$, we can say $\ln 2$ is indeed the best

Table 4.2: Case study: $N = 5$

Source node i	1	2	3	4	5
d_i	3	5	7	10	12
CSD: $\max_t A_i^{\pi_1}(t)$	3	5	7	10	12
FPM: $\max_t A_i^{\pi_2}(t)$	3	5	5	10	10

Table 4.3: Case study: $N = 100$

Group j	$\mathcal{G}_1$	$\mathcal{G}_2$	$\mathcal{G}_3$	$\mathcal{G}_4$	$\mathcal{G}_5$	$\mathcal{G}_6$	$\mathcal{G}_7$	$\mathcal{G}_8$	$\mathcal{G}_9$	$\mathcal{G}_{10}$
Source i	1-10	11-20	21-30	31-40	41-50	51-60	61-70	71-80	81-90	91-100
d_i	60	80	90	120	140	160	180	200	250	300
FPM: $\max_{i\in\mathcal{G}_j} A_i(t)$	60	60	60	120	120	120	120	120	240	240

guarantee that FPM can offer.

4.8 Numerical Results

In this section, we use simulations to validate our theoretical results and evaluate our algorithms. We also compare them to EDF, a well-known scheduler. An EDF scheduler is given as

$$\pi^{\text{EDF}}(t) = \arg\min_i \big(d_i - A_i(t)\big), \tag{4.19}$$

under which the source with the earliest deadline is selected for transmission at each t.

4.8.1 Feasibility Check

In this section, we validate that the schedulers found by our algorithms (CSD and FPM) are feasible, i.e, the maximum AoI for each source i is no greater than d_i. Our validation is based on case studies.

Small N

First we consider $N = 5$ source nodes, with $\boldsymbol{d} = [3\ 5\ 7\ 10\ 12]$. The system load is $l(\boldsymbol{d}) = 0.860$. Since N is small, we are able to execute CSD even though the complexity is exponential.

In CSD, the first step is to construct STG, where there are 12,600 nodes and 29,076 edges. We use DFS to detect cycle in this STG, and find one feasible scheduler:

$$\pi_1 = (ABABACEABDAACBACBAEDABCAACBADE \\ ABCAABACDABEAACBAADABCAEABACD).$$

For our algorithm FPM, for each $i = 1, 2, \cdots, N$ we compute $\widetilde{\boldsymbol{d}}$ by (4.9) and try to find one $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \leq 1$. We find that when $i = 2$, $\widetilde{\boldsymbol{d}} = [0.4\ 0.2\ 0.2\ 0.1\ 0.1]$ with $l(\widetilde{\boldsymbol{d}}) = 1$. Then we call FSC to construct the scheduler

$$\pi_2 = (ABCADABCAE).$$

We list the largest $A_i(t)$ for each source node $i = 1, 2, 3, 4, 5$ under π_1 and π_2 in the last two rows of Table 4.2. We can see that $A_i(t) \leq d_i$ for each i under both π_1 and π_2, so they are indeed feasible schedulers. Also note that the scheduler constructed by FSC has a much smaller cycle than that by CSD (10 vs. 59).

Large N

We now consider $N = 100$ source nodes that are organized in 10 groups, each group having the same deadline, as shown in Table 4.3. The load is $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) = 0.658$. Since N is large, we are not able to execute CSD due to its exponential complexity. So we can only validate our FPM and FSC. In FPM, for each $i = 1, 2, \cdots, N$ we compute $\widetilde{\boldsymbol{d}}$ by (4.9) and try to find one $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \leq 1$. We find that when $i = 1$, $l(\widetilde{\boldsymbol{d}}) = 1$. We use FSC to construct a feasible

Table 4.4: Scheduling behavior of FPM, CSD and EDF.

$\boldsymbol{d}$	$l(\boldsymbol{d})$	FPM	CSD	EDF
[3 12 13 13]	0.571	✓$c = 12$	✓$c = 117$	✗
[5 8 10 12 13]	0.585	✓$c = 10$	✓$c = 429$	✓$c = 131$
[3 7 8]	0.601	✓$c = 6$	✓$c = 48$	✓$c = 6$
[2 13 14]	0.648	✓$c = 8$	✓$c = 13$	✗
[4 6 7 8]	0.685	✓$c = 8$	✓$c = 96$	✓$c = 18$
[3 7 9 11 13]	0.755	✓$c = 12$	✓$c = 130$	✗
[2 3 10000]	0.833	✗	✗	✗
[3 5 7 10 12]	0.860	✓$c = 10$	✓$c = 59$	✗
[3 5 8 9 10 13]	0.946	✗	✗	✗
[3 6 6 7 13 14]	0.958	✓$c = 12$	✓$c = 12$	✗
[4 6 7 8 9 12 12]	0.962	✗	✓$c = 24$	✗

Table 4.5: Feasible schedulers found by FPM for different $\boldsymbol{d}$'s.

$\boldsymbol{d}$	$l(\boldsymbol{d})$	Feasible Scheduler found by FPM
[3 12 13 13]	0.571	$(ABCAD\square A\square\square A\square\square)$
[5 8 10 12 13]	0.585	$(ABCDEAB\square\square\square)$
[3 7 8]	0.601	$(ABCA\square\square)$
[2 13 14]	0.648	$(ABACA\square A\square)$
[4 6 7 8]	0.685	$(ABCDABC\square)$
[3 7 9 11 13]	0.755	$(ABCADEABCAD\square)$
[3 5 7 10 12]	0.860	$(ABCADABCAE)$
[3 6 6 7 13 14]	0.958	$(ABCADEABCADF)$

scheduler π (with cycle length $c = 240$. For each group of 10 nodes, we list the largest $A_i(t)$ in the group (e.g., for the 10 source nodes in group $\mathcal{G}_1$, 60 is the largest among $A_1(t)$, $A_2(t)$, $\cdots$, $A_{10}(t)$ for all $t > 0$) in the last row of Table 4.3. We can see that $A_i(t) \leq d_i$ for all $i = 1, 2, \cdots, 100$. So our scheduler constructed by FPM is indeed feasible.

4.8.2 Compare to EDF: Small N

In this section we compere our schedulers to EDF when N is small. In Table 4.4, we present results of FPM, CSD, and EDF for various $\boldsymbol{d}$. In the table, if a feasible scheduler is found by the underlying algorithm, we mark it by ✓and show its cycle length c. Otherwise (i.e., a feasible scheduler cannot be found by the underlying algorithm), we mark it by ✗. Not surprisingly, we see that CSD has the best performance. However, FPM's performance is quite close: It only fails to find a feasible scheduler when $\boldsymbol{d} = [4\ 6\ 7\ 8\ 10\ 12]$. On the other hand, EDF has the worst performance. Further, for any $l(\boldsymbol{d}) \leq \ln 2$, FPM can find a feasible scheduler, which confirms the result in Proposition 4.1. Also, the cycle length of the feasible schedulers found by FPM is always no greater than $d_{\max}$, which confirms the result in Lemma 4.6.

For those $\boldsymbol{d}$'s in Table 4.4 for which FPM can find feasible schedulers, we present the feasible schedulers in Table 4.5. The readers can easily verify their feasibility. Note that we didn't remove the empty time slots after executing FPM. If they are removed, the scheduler is still feasible.

In Fig. 4.2, we show the performance of FPM, CSD, and EDF when $N = 5$. We assume $d_i \in \{2, 3, \cdots, 20\}$ for each source node $i = 1, 2, \cdots, 5$, and randomly generate 100 different $\boldsymbol{d}$'s for each load interval $l(\boldsymbol{d}) \in (0.3, 0.32], (0.32, 0.34], \cdots, (0.98, 1]$. For each $\boldsymbol{d}$, we run FPM, CSD and EDF and calculate the rate (percentage) of success for finding a feasible scheduler. In Fig. 4.2, we see that the performance of FPM is close to CSD (the optimal

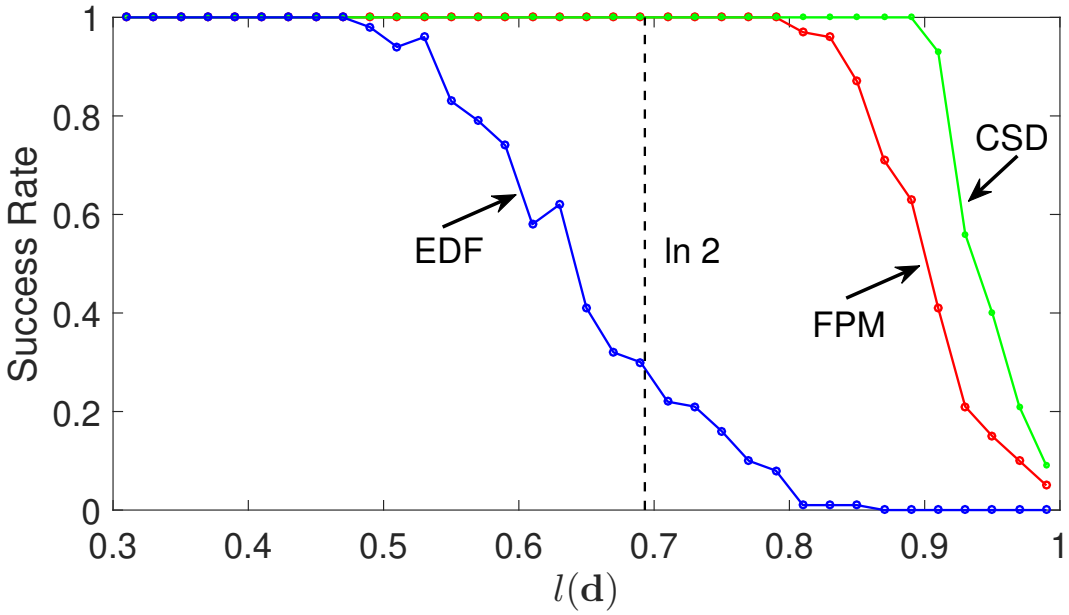


Figure 4.2: Success rates for FPM, CSD, and EDF under different $l(\boldsymbol{d})$ when $N = 5$.

algorithm). In particular, when $l(\boldsymbol{d}) \leq \ln 2$, the success rate of FPM is 100%, which confirms the result in Proposition 4.1. Also, EDF's performance is significantly inferior to FPM.

As we can see, in this AoI scheduling problem EDF doesn't perform very well. This is because EDF scheduler is designed to address a different problem. In that problem (for which EDF was designed), we are given a set of tasks—with each task having its arrival time and deadline being independent of the scheduler. In contrast, in the deadline guarantee problem in this chapter, the equivalent "arrival time" and "deadline" (as observed on wall-clock time) are dependent on the scheduler, i.e., an earlier transmission from a source node will lead to an earlier equivalent "arrival time" and "deadline" for the next transmission for the same source. With this subtle difference, the EDF scheduler is not designed to solve the deadline guarantee problem and our simulation results show that.

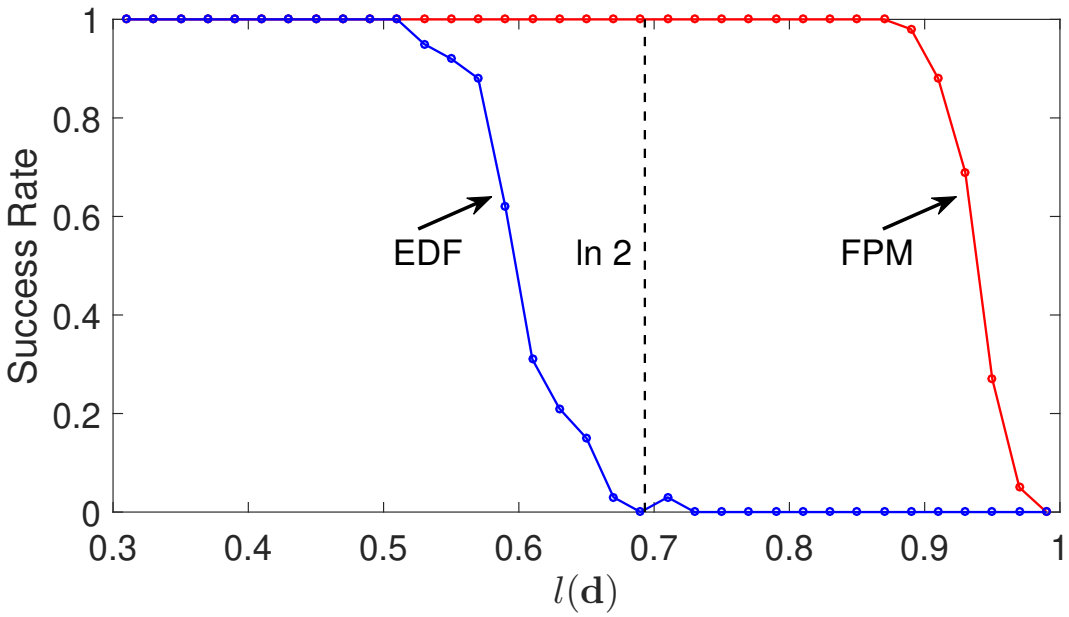


Figure 4.3: Success rate for FPM and EDF under different $l(\boldsymbol{d})$ when $N = 20$.

4.8.3 Compare to EDF: Large N

When N becomes sufficiently large, we will not be able to execute CSD due to its exponential time complexity. In this regime, we will only show results for FPM and EDF. Fig. 4.3 shows the performance of FPM and EDF when $N = 20$. We assume $d_i \in \{10, 20, 30, \cdots, 150\}$ for each source node $i = 1, 2, \cdots, 20$, and we randomly generate 100 different $\boldsymbol{d}$'s for each load interval $l(\boldsymbol{d}) \in (0.3, 0.32], (0.32, 0.34], \cdots, (0.98, 1]$. For each $\boldsymbol{d}$, we run FPM and EDF[4] and calculate the rate (percentage) of success for finding a feasible scheduler. In Fig. 4.3, we see that the performance of FPM is far better than EDF. Also, when $l(\boldsymbol{d}) \leq \ln 2$, the success rate of FPM is 100%, which confirms the result in Proposition 4.1.

Fig. 4.4 shows the performance of FPM and EDF when $N = 50$ (with $d_i \in \{10, 20, 30, \cdots, 400\}$ for each source node $i = 1, 2, \cdots, 50$) and Fig. 4.5 shows the performance of FPM and EDF

[4]For EDF, we simulate the first 100,000 time slots. If $\boldsymbol{d}$ is satisfied in this interval, we consider EDF feasible.

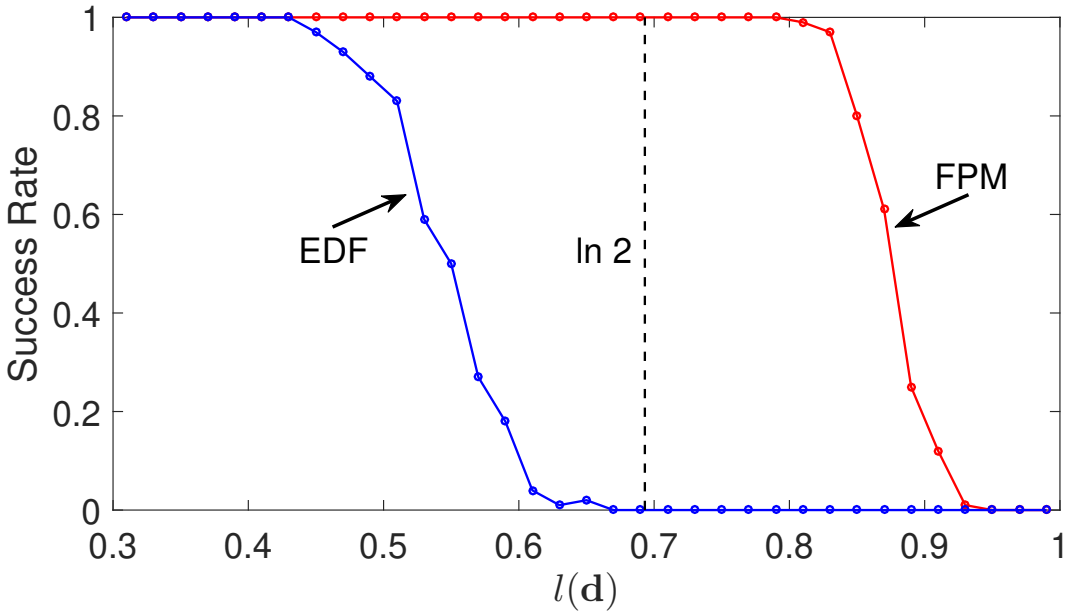


Figure 4.4: Success rate for FPM and EDF under different $l(\boldsymbol{d})$ when $N = 50$.

when $N = 100$ (with $d_i \in \{10, 20, 30, \cdots, 800\}$ for each source node $i = 1, 2, \cdots, 100$). We have similar results for both cases: FPM performs better than EDF, and when $l(\boldsymbol{d}) \leq \ln 2$, the success rate of FPM is 100%.

4.8.4 FPM under different N

We then investigate the impact of the number of source node, N, on FPM. We consider 5 different N's: N =20, 40, 60, 80, and 100. For each N, we consider four load intervals $l(\boldsymbol{d}) \in [0.68, 0.69], [0.79, 0.8], [0.84, 0.85]$, and $[0.89, 0.9]$. Note that $0.69 < \ln 2$. So for $l(\boldsymbol{d}) \in [0.68, 0.69]$, the success rate should be 1 under FPM, while for the other three load intervals, the success rate may be less than 1.

For $N = 20$, we assume $d_i \in \{10, 20, 30, \cdots, 150\}$. We randomly generate 100 different $\boldsymbol{d}$'s for each load interval, and calculate the success rate. For $N = 40$, we assume $d_i \in \{10, 20, 30, \cdots, 300\}$; for $N = 60$; we assume $d_i \in \{10, 20, 30, \cdots, 450\}$; for $N = 80$, we

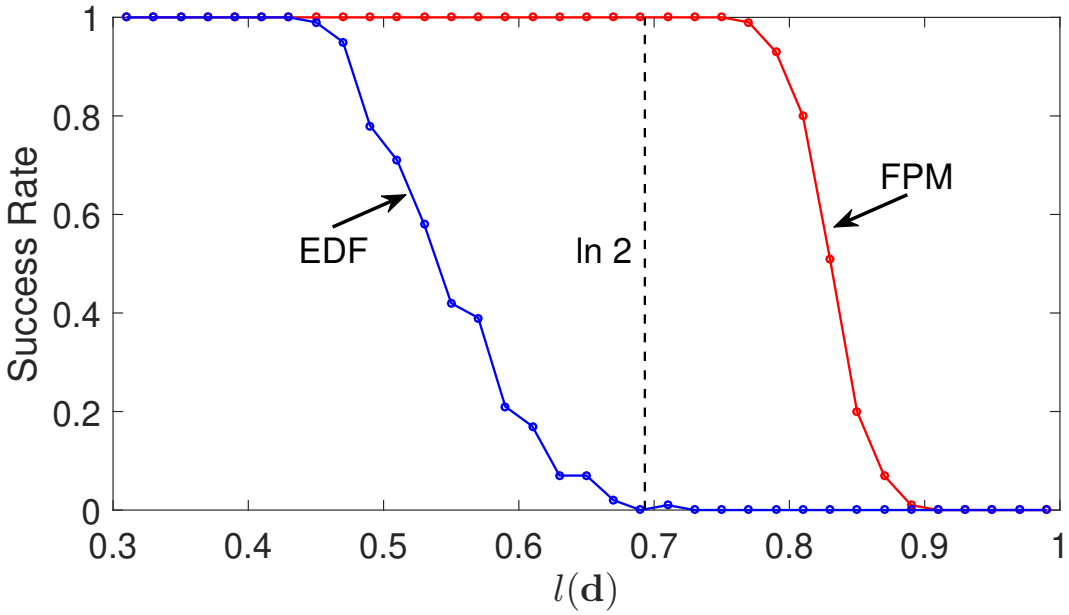


Figure 4.5: Success rate for FPM and EDF under different $l(\boldsymbol{d})$ when $N = 100$.

assume $d_i \in \{10, 20, 30, \cdots, 600\}$; for $N = 100$ we assume $d_i \in \{10, 20, 30, \cdots, 750\}$.

The results are shown in Fig. 4.6. For load interval $[0.68, 0.69]$ the success rates are indeed 1 for all N. This affirms that Proposition 4.1 is correct. For the other three load intervals with $l(\boldsymbol{d}) > \ln 2$, the success rate decreases as N increases. This is intuitive as the more the source nodes, the greater the challenge for the scheduler to find a feasible solution to meet the deadlines of all source nodes.

4.8.5 FPM under different deadline distribution

We now investigate the impact of $\boldsymbol{d}$'s distributions on FPM. We assume $N = 100$ source nodes. We assume three different set: $\mathcal{D}_1 = \{10, 11, 12, \cdots, 800\}$, $\mathcal{D}_2 = \{10, 20, 30, \cdots, 800\}$, and $\mathcal{D}_3 = \{10, 100, 200, 300, \cdots, 800\}$. For each set $\mathcal{D}_k$ $(k = 1, 2, 3)$, we randomly generate 100 different $\boldsymbol{d}$'s with $d_i \in \mathcal{D}_k$ for each load interval $l(\boldsymbol{d}) \in (0.4, 0.42], (0.42, 0.44], \cdots, (0.98, 1]$. Then we apply FPM and calculate the success rate for this interval. The results are shown

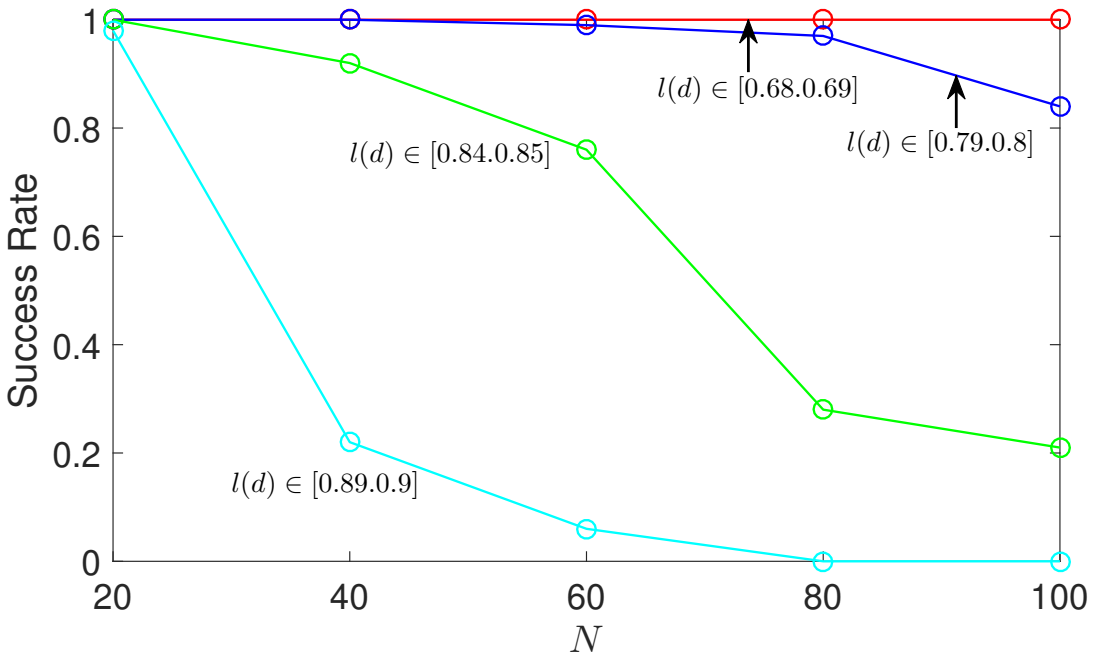


Figure 4.6: Success rate for FPM under different N.

in Fig. 4.7.

Again we see that under any distributions of $\boldsymbol{d}$, the success rate is 1 when $l(\boldsymbol{d}) \leq \ln 2$. But when $l(\boldsymbol{d}) > \ln 2$, the distribution of $\boldsymbol{d}$ does affect the performance of FPM. Since $\mathcal{D}_i$'s are all in the range of $[10, 800]$, the greater the granularity of deadlines, the worse the FPM's performance.

4.9 Chapter Summary

This chapter studied AoI scheduling subject to hard AoI deadlines. Specifically, we investigated the following two intertwined problems for AoI scheduling at network edge: (i) For a given hard deadline vector $\boldsymbol{d}$, determine whether it is schedulable; and (ii) If $\boldsymbol{d}$ is schedulable, find a feasible scheduler. To narrow down the search space, we first proved that if $\boldsymbol{d}$ is schedulable, then there must exist a feasible cyclic scheduler w.r.t. $\boldsymbol{d}$. Based on this result,

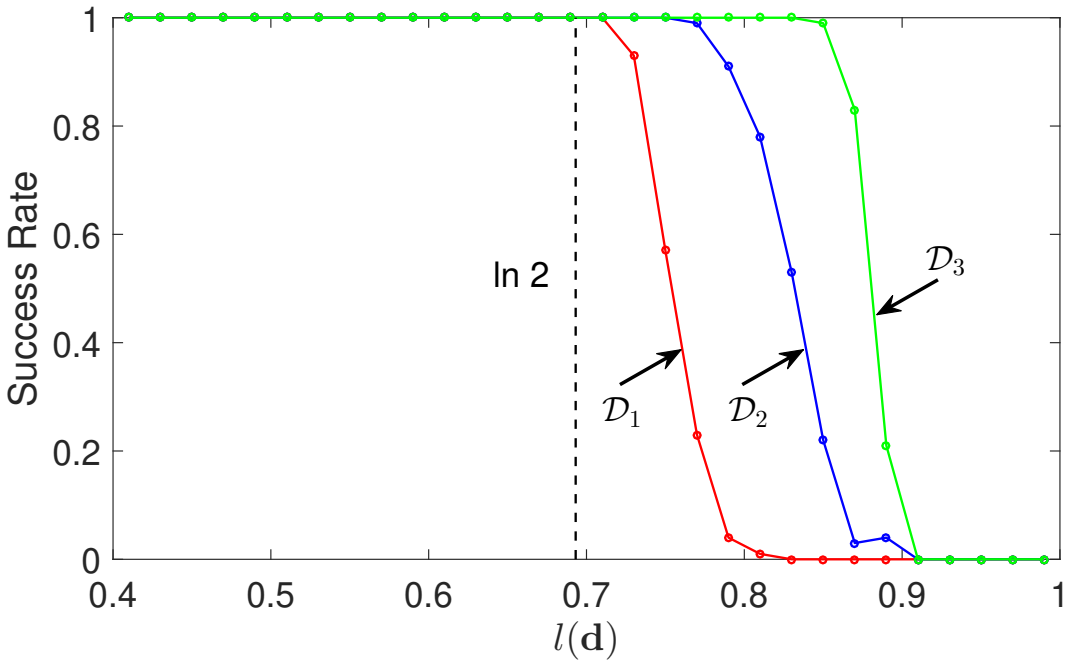


Figure 4.7: Success rate for FPM under different $\boldsymbol{d}$ distribution.

we proposed an error-free procedure CSD, which can be used to solve the two problems when the network size is small. For a large network, we introduced a load concept based on a given deadline vector and presented a low complexity procedure PSC that can find a feasible scheduler for any polynomial $\boldsymbol{d}$ with $l(\boldsymbol{d}) \leq 1$. For general (non-polynomial) $\boldsymbol{d}$'s, we presented FPM that can find a feasible scheduler for any $\boldsymbol{d}$ that can be mapped to a fictitious polynomial vector $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \leq 1$. We proved that FPM can find a feasible scheduler for any $\boldsymbol{d}$ with $l(\boldsymbol{d}) < \ln 2$, and the cycle length of the scheduler is no great than $d_{\max}$ (the largest element in $\boldsymbol{d}$). We used numerical results to validate our theoretical results. We also found that the performance of FPM is significantly better than EDF.

Chapter 5

AoI Scheduling with Soft Deadlines

5.1 Introduction

In Chapter 4, we studied the scheduling problem with a hard AoI deadline for each information source. In this chapter, we consider the case where the AoI deadlines are *soft*, i.e., occasional violations can be tolerated. In particular, we assume there is a soft AoI deadline for information from each source and a tolerance rate for violating this deadline at the BS. Further, we assume there is a packet loss rate on the wireless link between each source and the BS. Denote the vectors (for all source nodes) of soft AoI deadlines, tolerance rates and packet loss rates as $\boldsymbol{d}$, $\boldsymbol{\epsilon}$ and $\boldsymbol{p}$. We are interested in addressing the following two questions: (i) For a given triple $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$, does there exist a feasible scheduler? (ii) If a feasible scheduler exists, then find it.

The main contributions of this chapter are the following:

- We define a system load metric, denoted by $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$, that seamlessly fuses AoI deadline $\boldsymbol{d}$, channel condition $\boldsymbol{p}$ and violation tolerance $\boldsymbol{\epsilon}$ together. We show that $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ is schedulable only if $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) \leq 1$.
- We address the scheduling problem by exploring the relationship between $\boldsymbol{\epsilon}$ and $\boldsymbol{p}$. In the case where $\epsilon_i \geq p_i$ for all i's, which we call *stable tolerant* case, we present *stable tolerant scheduler* (STS), which can find a feasible scheduler as long as $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) \leq \ln 2$.
- STS is based on three key results. (i) We introduce the notion of *Almost Uniform*

Scheduler (AUS), which follows a uniform or nearly uniform transmission schedule for each source. AUS serves as a key construct in our scheduler design throughout this chapter. We also present a necessary and sufficient condition for an AUS to be feasible. (ii) We consider a special case of $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ with *step-down* rate vector, and present a procedure that can construct a feasible AUS for any $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ with a step-down rate vector as long as $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) \leq 1$. (iii) For a general $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ that does not have a step-down rate vector, we present a dynamic programming (DP) solution to map it to a step-down rate vector, for which we can construct a feasible AUS.

- In the case where $\epsilon_i < p_i$ for some i's, which we call *unstable tolerant* case, we follow the same design roadmap (with modifications along the way) of STS and present a low-complexity algorithm called *Unstable Tolerant Scheduler* (UTS). We also give a sufficient condition for UTS to find a feasible scheduler for unstable tolerant $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$.

5.2 System Model and Problem Statement

We consider a wireless network with N source nodes and one BS in Fig. 1.1. Assume time is equally slotted and each source takes a new sample at the beginning of each time slot. Due to potential interference, at most one sample from a source can be chosen for transmission within each time slot. For each transmission, the source will forward its freshest (latest generated) sample to the BS. A scheduler (denoted by π) is needed to choose one source for transmission at the beginning of each time slot. Denote $\pi_i(t) \in \{0, 1\}$ as the scheduling decision for source i $(i = 1, 2, \cdots, N)$ at time t $(t = 0, 1, 2, \cdots)$, i.e.,

$$\pi_i(t) = \begin{cases} 1, & \text{if source } i \text{ is scheduled at time } t, \\ 0, & \text{otherwise.} \end{cases} \tag{5.1}$$

Table 5.1: Notation

Symbol	Definition
$A_i(t)$	AoI for source node i at the BS at time t
c	Cycle length for a cyclic scheduler
$\boldsymbol{d}$	Vector of the AoI deadlines
d_i	AoI deadline for source node i
d_{max}	The largest element in $\boldsymbol{d}$
$\boldsymbol{\epsilon}$	Vector of the tolerance rates
ϵ_i	Tolerance rate for source node i
$l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$	System load for $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$
N	Number of source nodes in the network
$\boldsymbol{p}$	Vector of the packet loss rates
p_i	Packet loss rate for source node i
$\pi_i(t)$	Scheduling decision for source i at time t
$q_i(t)$	Channel indicator for source i at time t
r_i	Average scheduling rate for source node i
r_i^{lb}	The lower bound of r_i for any feasible scheduler
$U_i(t)$	Generation time of the most recent sample at the BS from source node i at time t

Clearly, the following must hold for any scheduler π:

$$\sum_{i=1}^{N} \pi_i(t) \leq 1, \ t = 0, 1, 2, \cdots . \tag{5.2}$$

The success (or failure) for transmitting the sample depends on the wireless channel condition. Denote $q_i(t)$ as an indicator function of the channel for node i at time t, i.e., $q_i(t) = 1$ if the channel is under good condition and the transmission from source i will be successful, and 0 otherwise. Denote p_i as the probability of $q_i(t) = 0$, i.e., $\mathbb{P}\{q_i(t) = 0\} = p_i$. Then $\mathbb{P}\{q_i(t) = 1\} = 1 - p_i$. Note that p_i is location dependent (on source node i). Also, we assume p_i is time-invariant.

The BS only stores the freshest sample it has received from each source. Once a new sample is received at the BS, the old sample from the same source stored at the BS will be replaced. Denote $U_i(t)$ as the generation time of the sample stored at the BS from source i

at time t. Then the AoI of source i (at the BS) at time t, denoted as $A_i(t)$, is defined as the elapsed time between the current time t and the sample generation time $U_i(t)$, i.e.,

$$A_i(t) = t - U_i(t),\ t = 0, 1, 2, \cdots . \tag{5.3}$$

If the source i is not scheduled for transmission at time t (i.e., $\pi_i(t) = 0$), by definition of AoI, at time $(t+1)$ we have $A_i(t+1) = A_i(t) + 1$. We assume then the generation time of a new sample occurs at the beginning of each time slot t. Then if source i is scheduled for transmission at time t (i.e., $\pi_i(t) = 1$), then: (i) if the sample is successfully received by the BS (i.e., $q_i(t) = 1$), then at time $(t+1)$, we have $U_i(t+1) = t$, and $A_i(t+1) = 1$; (ii) if the sample is not received (i.e., $q_i(t) = 0$), then at time $(t+1)$ we have $A_i(t+1) = A_i(t) + 1$. Combining the two cases, we have:

$$A_i(t+1) = \begin{cases} 1, & \text{if } \pi_i(t) \cdot q_i(t) = 1, \\ A_i(t) + 1, & \text{otherwise.} \end{cases} \tag{5.4}$$

Tables 5.1 lists key notations in this chapter. When there is no ambiguity, we use the term "at TTI t" to refer to *at the beginning of TTI* t and use the term "in TTI t" to refer to the underlying action is completed *at the end of TTI* t.

In this chapter we assume there is a *soft AoI deadline* d_i (d_i is a positive integer) and a *tolerance rate* $\epsilon_i \in [0, 1)$ for each source i. Due to the difference in types of application, d_i and ϵ_i may vary among different sources. Our goal is to design a scheduler such that for each source i the fraction of time slots when its AoI exceeds its deadline d_i is no greater than its tolerance rate ϵ_i. More formally, we say a scheduler π is *feasible* if it satisfies the following *violation tolerance* constraint:

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} [A_i(t) > d_i] \leq \epsilon_i,\ i = 1, 2, \cdots, N, \tag{5.5}$$

where "$[\cdot]$" is Iverson bracket, returning 1 if the inside statement is true and 0 otherwise [63].

Denote $\boldsymbol{d} = [d_1, d_2, \cdots, d_N]$ as the vector of AoI deadlines for all N sources, $\boldsymbol{\epsilon} = [\epsilon_1, \epsilon_2, \cdots, \epsilon_N]$ as the vector of all N tolerance rates, and $\boldsymbol{p} = [p_1, p_2, \cdots, p_N]$ as the vector of all N packet loss probabilities. We say an ordered triple $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ is *schedulable* if there exists at least one feasible scheduler for it. The problems we want to study in this chapter are:

1. Determine whether or not a given ordered triple $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ is schedulable.

2. If $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ is schedulable, find a feasible scheduler for it.

It turns out that our answers to the above two questions heavily depend on the relationship between $\boldsymbol{\epsilon}$ and $\boldsymbol{p}$. For now, let's define two cases: (i) if $\epsilon_i \geq p_i$ for *all* $i = 1, 2, \cdots, N$, then we say $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ is *stable tolerant*; (ii) if there exists at least some $i = 1, 2, \cdots, N$ such that $\epsilon_i < p_i$, the we say $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ is *unstable tolerant*. Note that the two cases are complimentary in the entire space for $(\boldsymbol{\epsilon}, \boldsymbol{p})$. Thus, it is sufficient to examine these two cases.

The major challenge of designing schedulers subject to AoI constraints is that the scheduler must have a provable feasibility guarantee, i.e., it should be theoretically proved to satisfy the constraints in (5.5). Clearly, this significantly differs from previous AoI minimization literature, e.g., [9, 10, 11, 24, 32, 43, 44, 45, 79, 81, 84, 88], where schedulers have no feasibility consideration and their performances are evaluated via simulations. In this chapter we focus on the design of schedulers with provable feasibility guarantee, which makes the problem different from and much harder than those in most existing work.

5.3 System Load

Recall that one of our goals is to determine schedulability of an ordered triple $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$. In this section we derive a necessary condition for $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ to be schedulable. We define a concept called *system load*, denoted by $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$, and show that $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ is schedulable only if $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) \leq 1$.

Under a scheduler π, we define the average scheduling rate r_i for each source i as

$$r_i = \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \pi_i(t), \tag{5.6}$$

which represents the fraction of time slots in which source i is scheduled for transmission. Clearly, based on (5.2), we have

$$\sum_{i=1}^{N} r_i \leq 1. \tag{5.7}$$

Since the success probability for each transmission from source i is $(1 - p_i)$, the fraction of time slots in which its sample is successfully received by the BS is $r_i(1 - p_i)$.

Recall a scheduler π is feasible if it can ensure $A_i(t) \leq d_i$ for at least a fraction of $(1 - \epsilon_i)$ of time slots. Note that when a new update arrives at the BS, it will bring the AoI (for that source) at the BS immediately down to 1. This effectively ensures that the AoI for that source will not exceed its deadline for up to d_i time slots. So for a feasible scheduler, the fraction of the time slots when a sample from source i is successfully received by the BS must be no less than $\frac{1-\epsilon_i}{d_i}$. That is,

$$r_i(1 - p_i) \geq \frac{1 - \epsilon_i}{d_i}, \ i = 1, 2, \cdots, N, \tag{5.8}$$

which is equivalent to

$$r_i \geq \frac{1 - \epsilon_i}{(1 - p_i)d_i}, \ i = 1, 2, \cdots, N. \tag{5.9}$$

Denote the RHS of (5.9) as r_i^{lb}, which is the lower bound of r_i. We have

$$r_i^{\text{lb}} = \frac{1-\epsilon_i}{(1-p_i)d_i}. \tag{5.10}$$

Clearly, under a feasible scheduler we must have $r_i \geq r_i^{\text{lb}}$ for each i. Considering (5.7), for a feasible scheduler we must have

$$\sum_{i=1}^{N} r_i^{\text{lb}} \leq 1. \tag{5.11}$$

We define the sum of r_i^{lb}'s as *system load* w.r.t. $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$, which is written as

$$l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) = \sum_{i=1}^{N} r_i^{\text{lb}} = \sum_{i=1}^{N} \frac{1-\epsilon_i}{(1-p_i)d_i}. \tag{5.12}$$

The following lemma shows a necessary condition for $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ to be schedulable.

Lemma 5.1. *An ordered triple* $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ *is schedulable only if* $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) \leq 1$.

The lemma follows directly from the discussions in this section.

5.4 The Stable Tolerant Case

In this section we will study scheduler design for the stable tolerance case, i.e., the case when $\epsilon_i \geq p_i$ for all $i = 1, 2, \cdots, N$. First, we introduce *Almost Uniform Scheduler* (AUS) and offer a necessary and sufficient condition for an AUS to be feasible for $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$. Then we consider a special case of $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ and show how to construct an AUS for this special case. Finally we extend the same procedure from the special case to the general case of $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$.

5.4.1 Almost Uniform Scheduler

We first define what we call a "cyclic" scheduler.

Definition 5.1. *We say a scheduler π is cyclic if there exists a cycle length c such that $\pi_i(t) = \pi_i(t+c)$ for all $i = 1, 2, \cdots, N$ and $t \geq 0$.*

In other words, a scheduler is cyclic if its scheduling decision over a frame of length c time slots repeats itself over time. Since a cyclic scheduler is most easy to understand and implementation friendly, we will focus our attention on this class of schedulers.

Denote τ_i^k as the time slot when the k-th sample from source i is scheduled for transmission under scheduler π. Clearly we have $\pi_i(\tau_i^k) = 1$ for each $k = 1, 2, \cdots$. Denote T_i^k as the time interval (in number of time slots) between the k-th and the $(k+1)$-th transmission for source i. Then we have:

$$T_i^k = \tau_i^{k+1} - \tau_i^k. \tag{5.13}$$

Clearly, when there is only one transmission for source i within a cycle c, we have $T_i^k = c$. When there are more than one transmissions within c, we have $T_i^k < c$.

Definition 5.2. *A cyclic scheduler π is an Almost Uniform Scheduler (AUS) if for each source i, there exists an integer b_i such that T_i^k is either b_i or $(b_i + 1)$ for any $k \geq 1$.*

By definition, if a scheduler π is an AUS, then the intervals between two consecutive transmissions of a source do not differ by more than 1 time slot. As an example, consider three sources A, B, and C. Denote "()" as a group of scheduling decisions for one cycle. Then for scheduler $(ABACB)$ (with $c = 5$), we have $T_A^1 = 2$, $T_A^2 = 3$, $\cdots$, $T_B^1 = 3$, $T_B^2 = 2$, $\cdots$, $T_C^1 = 5$. We have $b_A = 2$, $b_B = 2$ and $b_C = 5$, so this scheduler is AUS. In contrast, for scheduler $(AABACB)$ (with $c = 6$), we have $T_A^1 = 1$, $T_A^2 = 2$, $T_A^3 = 3$, $\cdots$, $T_B^1 = 3$, $T_B^2 = 3$, $\cdots$, $T_C^1 = 6$. Since $T_A^1 = 1$ and $T_A^3 = 3$, we cannot find a b_A per AUS definition. So this scheduler is not AUS.

The following theorem shows that if a scheduler π is AUS, then there exists a necessary

and sufficient condition for it to be feasible for $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$.

Theorem 5.1. *In the stable tolerance case, an AUS π is feasible for $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ if and only if $r_i \geq r_i^{lb}$ for each $i = 1, 2, \cdots, N$.*

A proof of Theorem 5.1 is given in Appendix B.

5.4.2 Finding AUS for a Special Case

In this section we study a special case of $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$'s, for which we can construct an AUS when the system load $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ is no greater than 1. We first introduce a concept called *step-down rate vector*. Denote $\boldsymbol{r}^{\text{lb}}$ as the lower bound rate vector, i.e., $\boldsymbol{r}^{\text{lb}} = [r_1^{\text{lb}}, r_2^{\text{lb}}, \cdots, r_N^{\text{lb}}]$, where r_i^{lb} is given in (5.10) for $i = 1, 2, \cdots, N$. Without loss of generality, we sort the elements in $\boldsymbol{r}^{\text{lb}}$ in non-increasing order, i.e., $r_1^{\text{lb}} \geq r_2^{\text{lb}} \geq \cdots \geq r_N^{\text{lb}}$.

Definition 5.3. *We say $\boldsymbol{r}^{\text{lb}}$ is a step-down rate vector if $r_i^{lb}/r_{i+1}^{lb} \in \mathbb{N}^*$ for $i = 1, 2, \cdots, N-1$.*

In other words, in a step-down rate vector $\boldsymbol{r}^{\text{lb}}$, the radio of an element over its succeeding element is a positive integer. For example, $\boldsymbol{r}^{\text{lb}} = [1/2, 1/6, 1/12, 1/12, 1/24]$ is a step-down vector.

Now we show how to construct an AUS that is feasible for a given $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ with a step-down rate vector $\boldsymbol{r}^{\text{lb}}$. We use an example to illustrate the main ideas. The complete pseudocode is given in Algorithm 5.1.

Example 5.1. Consider seven sources A, B, C, D, E, F and G with $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) = ([2, 4, 5, 15, 15, 16, 16], [0.4, 0.4, 0.25, 0.25, 0.25, 0.2, 0.2], [0.15, 0.15, 0.15, 0.15, 0.15, 0.15, 0.15])$. By (5.10), we have $\boldsymbol{r}^{\text{lb}} = [\frac{6}{17}, \frac{3}{17}, \frac{3}{17}, \frac{1}{17}, \frac{1}{17}, \frac{1}{17}, \frac{1}{17}]$, which is a step-down rate vector. Further, $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) = \frac{16}{17} < 1$.

By Theorem 5.1, an AUS is feasible if $r_i \geq r_i^{\text{lb}}$ for each i. Since $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) < 1$, we can set $r_i = r_i^{\text{lb}}/l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ for each i, which still guarantees the AUS is feasible. We have $r_A = \frac{6}{16}$, $r_B = \frac{3}{16}$, $r_C = \frac{3}{16}$, $r_D = \frac{1}{16}$, $r_E = \frac{1}{16}$, $r_F = \frac{1}{16}$, and $r_G = \frac{1}{16}$. Since the smallest rate among the r_i's is $1/16$, we set the cycle length of the AUS to $c^{\text{AUS}} = 16$. Denote N_i as the number of scheduled transmission for source i within one cycle. Since $N_i = r_i \cdot c^{\text{AUS}}$, we have $N_A = 6$, $N_B = 3$, $N_C = 3$, $N_D = 1$, $N_E = 1$, $N_F = 1$, and $N_G = 1$.

To construct an AUS, we first consider a special case, which we call *Exact Uniform Scheduler* (EUS). We say a scheduler an EUS if for each source i, the interval between any two adjacent transmissions is exactly b_i.

Now we continue with our example. Since $N_A = 6$, we propose to first construct an EUS with $c = \lceil \frac{c^{\text{AUS}}}{N_A} \rceil \cdot N_A = 3 \cdot 6 = 18$. This EUS has $18 - 16 = 2$ more time slots than our AUS. We will take care of these 2 empty slots in the last step.

First, we lay out an unassigned EUS cycle with $c = 18$. Under each slot in the cycle, we label its position with an index number.

π	(□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □)
t	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

Consider source A. Since $N_A = 6$, in the EUS we have $b_A = 18/6 = 3$. We allocate the first empty time slot to A, along with every 3 time slots after it.

π	(A □ □ A □ □ A □ □ A □ □ A □ □ A □ □)
t	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

Then we consider the second source B. Since $N_B = 3$, in the EUS we have $b_B = 18/3 = 6$. Among all the empty time slots, the smallest elapsed time slots following a transmission of A are: $t = 2, 5, 8, 11, 14, 17$. We allocate the first one among these possibilities, i.e., $t = 2$, to source B, along with every $b_B = 6$ slots after it.

π	(A	B	□	A	□	□	A	B	□	A	□	□	A	B	□	A	□	□)
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Then we consider source C. Since $N_C = 3$, in the EUS we have $b_C = 18/3 = 6$. Among all the empty time slots, the smallest elapsed time after a transmission of A is 1 time slot: i.e., $t = 5, 11, 17$ have this smallest elapsed time. Among time slots $t = 5, 11, 17$, the smallest elapsed time from a transmission of B is 3 time slot: i.e., $t = 5, 11, 17$. We allocate the first one among these possibilities, i.e., $t = 5$, to source C, along with every $b_C = 6$ slots after it.

π	(A	B	□	A	C	□	A	B	□	A	C	□	A	B	□	A	C	□)
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Then we consider source D. Recall $N_D = 1$. Among all the empty time slots, the smallest elapsed time from a transmission of A is 2 time slot: i.e., $t = 3, 6, 9, 12, 15, 18$. Among time slots $t = 3, 6, 9, 12, 15, 18$, the smallest elapsed time from a transmission of B is 1 time slot: i.e., $t = 3, 9, 15$. Among time slots $t = 3, 9, 15$, the smallest elapsed time from a transmission of C is 4 time slot: i.e., $t = 3, 9, 15$. Among these three possibilities, we allocate the first one, i.e., $t = 3$, to source D.

π	(A	B	D	A	C	□	A	B	□	A	C	□	A	B	□	A	C	□)
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Following the same token, we allocate time slots $t = 9$ to E, $t = 15$ to F, and $t = 6$ to G, respectively.

π	(A	B	D	A	C	G	A	B	E	A	C	□	A	B	F	A	C	□)
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

It is easy to understand that the above scheduler is an EUS. In the final step, we remove the two extra (empty) slots in the EUS and we have:

π	(A	B	D	A	C	G	A	B	E	A	C	A	B	F	A	C)
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Readers can easily verify that the final scheduler is AUS based on Definition 5.2. □

Algorithm 5.1 present a pseudocode based on the ideas in Example 5.1. The following lemma says Algorithm 5.1 can always construct an AUS when $\boldsymbol{r}^{\text{lb}}$ is step-down rate and $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) \leq 1$.

Lemma 5.2. *For any stable tolerant* $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ *with a step-down rate vector* $\boldsymbol{r}^{lb}$ *and* $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) \leq 1$*, the scheduler constructed by Algorithm 5.1 is an AUS.*

A proof of Lemma 5.2 is given in Appendix C.

The following theorem says that the scheduler by Algorithm 5.1 is not only AUS, but also feasible for the given $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$.

Theorem 5.2. *For any stable tolerant* $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ *with a step-down rate vector* $\boldsymbol{r}^{lb}$ *and* $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) \leq$

Algorithm 5.1 AUS Construction

Input: A step-down rate vector $\boldsymbol{r}^{\text{lb}}$ with $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) \leq 1$.

Output: A feasible AUS.

1: Set $r_i = r_i^{\text{lb}}/l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ for each $1 \leq i \leq N$. Set $c^{\text{AUS}} = 1/r_N$, and $N_i = r_i \cdot c^{\text{AUS}}$ for each $1 \leq i \leq N$.

2: Construct an unassigned EUS cycle with $c = \lceil \frac{c^{\text{AUS}}}{N_1} \rceil \cdot N_1$.

3: Allocate the first empty slot to source 1, along with every c/N_1 slots after it.

4: **for** $i = 2, 3, \cdots, N$ **do**

5: Let $\mathcal{S}_0$ be the set of empty time slots.

6: **for** $j = 1, 2, \cdots, i-1$ **do**

7: Let $\mathcal{S}_j$ be the subset of $\mathcal{S}_{j-1}$ containing time slots corresponding to the smallest elapsed time after a transmission of source j in the cycle.

8: **end for**

9: Allocate the first time slot in $\mathcal{S}_{i-1}$ to source i, along with every c/N_1 slots after it.

10: **end for**

11: Remove the $(c - c^{\text{AUS}})$ unassigned time slots in the EUS cycle and output the resulting schedule.

1, *the scheduler constructed by Algorithm 5.1 is feasible for* $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$.

Proof. Denote π as the scheduler constructed by Algorithm 5.1. By Lemma 5.2, π is an AUS. Besides, in Algorithm 5.1 we set $r_i = r_i^{\text{lb}}/l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ for π. Since $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) \leq 1$, we have $r_i \geq r_i^{\text{lb}}$. Then by Theorem 5.1, π is feasible for $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$. □

Complexity We now analyze the time complexity of Algorithm 5.1. For each iteration of step 7 in Algorithm 5.1, we need to visit all unassigned time slots in the EUS, which has a complexity of $O(c)$. Since there are $O(N^2)$ such iterations in Steps 4–9 in Algorithm 5.1, the time complexity of all iterations is $O(N^2 c)$. Considering $c < 2c^{\text{AUS}} \leq 2/r_N^{\text{lb}}$, so $O(c) = O(1/r_N^{\text{lb}}) = O(d_{\max})$ where $d_{\max}$ is the largest element in $\boldsymbol{d}$. Here we assume both ϵ_i and p_i are smaller than 0.5. Therefore, the total complexity of Algorithm 5.1 is $O(N^2 c) = O(N^2 d_{\max})$.

5.4.3 Finding AUS for General Case

In this section, we consider the general case where $\boldsymbol{r}^{\text{lb}}$ may not be a step-down rate vector. Recall that by Theorem 5.1, an AUS is feasible if $r_i \geq r_i^{\text{lb}}$ for each i. So if we could find a step-down rate vector $\hat{\boldsymbol{r}} = [\hat{r}_1, \hat{r}_2, \cdots, \hat{r}_N]$ with $\sum_{i=1}^{N} \hat{r}_i \leq 1$ and $\hat{r}_i \geq r_i^{\text{lb}}$ for each i, then we can use Algorithm 5.1 to find an AUS with $r_i \geq \hat{r}_i \geq r_i^{\text{lb}}$ that is feasible for $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$.

To find $\hat{r}_i$ for $i = 1, 2, \cdots, N$, we can solve the following optimization problem. Again, we assume the elements in $\boldsymbol{r}^{\text{lb}}$ are sorted in non-increasing order, i.e., $r_1^{\text{lb}} \geq r_2^{\text{lb}} \geq \cdots \geq r_N^{\text{lb}}$.

$$\begin{aligned} \text{OPT:} \min_{\hat{\boldsymbol{r}}} \quad & \sum_{i=1}^{N} \hat{r}_i \\ \text{s.t.} \quad & \frac{\hat{r}_i}{\hat{r}_{i+1}} \in \mathbb{N}^*, \ i = 1, 2, \cdots, N-1, \\ & r_i^{\text{lb}} \leq \hat{r}_i \leq 1, \ i = 1, 2, \cdots, N. \end{aligned}$$

Denote the optimal solution to OPT as $\hat{\boldsymbol{r}}^*$, which is a step-down rate vector. If $\sum_{i=1}^{N} \hat{r}_i^* \leq 1$, then we can use Algorithm 5.1 (by setting $\boldsymbol{r}^{\text{lb}} = \hat{\boldsymbol{r}}^*$ in Algorithm 5.1) to find a feasible AUS for $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$. Otherwise, if $\sum_{i=1}^{N} \hat{r}_i^* > 1$, we cannot use Algorithm 5.1 to find a feasible scheduler.

A key step to solve OPT is to show that there exists some k ($k = 1, 2, \cdots, N$) such that $\hat{r}_k^* = r_k^{\text{lb}}$ in the optimal solution $\hat{\boldsymbol{r}}^*$. Using this result, we can solve OPT by fixing $\hat{r}_k^* = r_k^{\text{lb}}$ for N times ($k = 1, 2, \cdots, N$). For each k, since $\hat{r}_k^* = r_k^{\text{lb}}$ is fixed, we can divide OPT into to parts: (i) optimizing $\hat{r}_1, \hat{r}_2, \cdots, \hat{r}_{k-1}$; and (ii) optimizing $\hat{r}_{k+1}, \hat{r}_{k+2}, \cdots, \hat{r}_N$. Each part can be solved by dynamic programming (DP). Due to page limit, we will not elaborate on how we develop our solution. Instead, we present our solution algorithm (in pseudocode) in Algorithm 5.4. We encourage readers to study the algorithm and convince themselves that it solves OPT. It can be shown the time complexity of Algorithm 5.4 is $O(N^2 d_{\max}^2)$.

To solve OPT, we first consider the following question: What will $\hat{\boldsymbol{r}}^*$ (the optimal solution to OPT) look like? The answer is given in the following lemma.

Lemma 5.3. *For $\hat{\boldsymbol{r}}^*$, there is at least one i such that $\hat{r}_i^* = r_i^{lb}$.*

Proof. Our proof is based on contradiction. Suppose there is no i such that $\hat{r}_i^* = r_i^{\text{lb}}$. Then let $\hat{\boldsymbol{r}}_1 = \max_{i=1,2,\cdots,N} \{\frac{r_i^{\text{lb}}}{\hat{r}_i^*}\} \cdot \hat{\boldsymbol{r}}^*$. Clearly, $\hat{\boldsymbol{r}}_1$ is feasible to OPT and has a smaller objective value than $\hat{\boldsymbol{r}}^*$, which contradicts to the fact that $\hat{\boldsymbol{r}}^*$ is optimal. □

With Lemma 5.3 in hands, in order to solve OPT we can solve N different problems OPT-k for $k = 1, 2, \cdots, N$, and choose the solution with the smallest objective.

$$\text{OPT-}k\text{:}\quad \min_{\hat{\boldsymbol{r}}} \quad \sum_{i=1}^{N} \hat{r}_i$$
$$\text{s.t.} \quad \frac{\hat{r}_i}{\hat{r}_{i+1}} \in \mathbb{N}^*,\ i = 1, 2, \cdots, N-1,$$
$$r_i^{\text{lb}} \le \hat{r}_i \le 1,\ i = 1, 2, \cdots, N,\ i \ne k,$$
$$\hat{r}_k = r_k^{\text{lb}}.$$

Problem OPT-k can be divided to two separate problems: OPT-k-left and OPT-k-right.

$$\text{OPT-}k\text{-left:}\quad \min_{\hat{r}_1, \hat{r}_2, \cdots, \hat{r}_{k-1}} \quad \sum_{i=1}^{k} \hat{r}_i$$
$$\text{s.t.} \quad \frac{\hat{r}_i}{\hat{r}_{i+1}} \in \mathbb{N}^*,\ i = 1, 2, \cdots, k-1,$$
$$r_i^{\text{lb}} \le \hat{r}_i \le 1,\ i = 1, 2, \cdots, k-1,$$
$$\hat{r}_k = r_k^{\text{lb}}.$$

$$\text{OPT-}k\text{-right:}\quad \min_{\hat{r}_{k+1}, \cdots, \hat{r}_N} \quad \sum_{i=k}^{N} \hat{r}_i$$
$$\text{s.t.} \quad \frac{\hat{r}_i}{\hat{r}_{i+1}} \in \mathbb{N}^*,\ i = k, \cdots, N-1,$$
$$r_i^{\text{lb}} \le \hat{r}_i \le 1,\ i = k+1, \cdots, N,$$
$$\hat{r}_k = r_k^{\text{lb}}.$$

We propose to solve OPT-k-left by Dynamic Programming (DP). We will use an example to show how it works.

Example 5.2. For OPT-k-left, consider $k = 4$, $[r_1^{\text{lb}}, r_2^{\text{lb}}, r_3^{\text{lb}}, r_4^{\text{lb}}] = [0.6, 0.5, 0.3, 0.2]$. Since $k = 4$, we have $\hat{r}_4 = r_4^{\text{lb}} = 0.2$. We will use this example to show how to solve OPT-k-left using DP.[1]

[1] In this example, $\sum_{i=1}^{k} r_i^{\text{lb}} > 1$, which trivially means there doesn't exist a feasible scheduler. The purpose of this example is purely to show how to use DP to solve the optimization problem OPT-k-left. Choosing

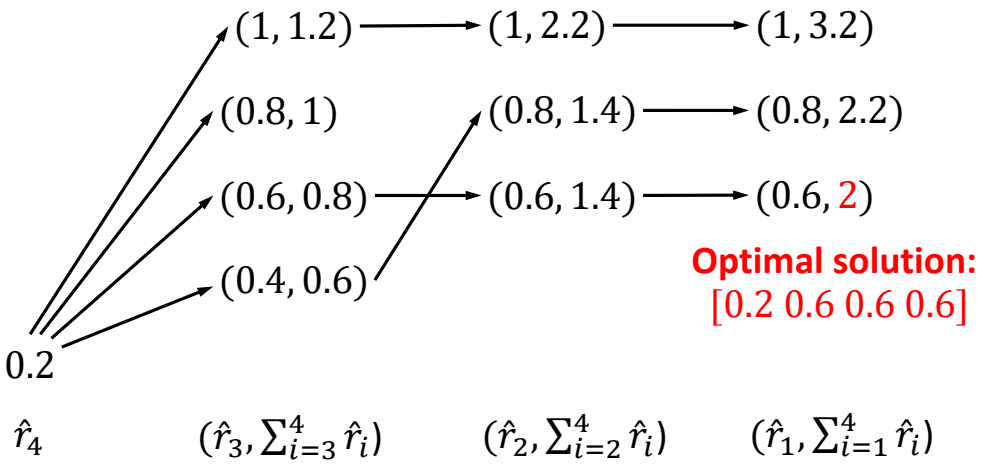


Figure 5.1: An example for using DP to solve OPT-k-left.

The approach to solve this problem is shown in Fig. 5.1. As initialization, we have $\hat{r}_4 = 0.2$. In the first step, we consider $\hat{r}_3$. Since $\hat{r}_3/\hat{r}_4$ is an integer and $1 \geq \hat{r}_3 > r_3^{\text{lb}} = 0.3$, $\hat{r}_3$ can be 0.4, 0.6, 0.8, or 1. For each possible value of $\hat{r}_3$, we compute $(\hat{r}_3 + \hat{r}_4)$ and draw an arrow from $\hat{r}_4$ to it.

Then we consider $\hat{r}_2$. Since $\hat{r}_2/\hat{r}_4$ is an integer and $1 \geq \hat{r}_2 > r_2^{\text{lb}} = 0.5$, $\hat{r}_2$ can be 0.6, 0.8, or 1. For each possible value of $\hat{r}_2$, among those $\hat{r}_3$ with $\hat{r}_2/\hat{r}_3 \in \mathbb{N}^*$, we pick the $\hat{r}_3$ with smallest $(\hat{r}_3 + \hat{r}_4)$, draw an arrow from it, and compute $(\hat{r}_2 + \hat{r}_3 + \hat{r}_4)$. This arrow represents the optimal decision of $\hat{r}_3$ that minimizes $(\hat{r}_2 + \hat{r}_3 + \hat{r}_4)$ when $\hat{r}_2$ is fixed.

Finally, we do the same for $\hat{r}_1$. Since $\hat{r}_1/\hat{r}_4$ is an integer and $1 \geq \hat{r}_1 > r_1^{\text{lb}} = 0.6$, $\hat{r}_1$ can be 0.6, 0.8, or 1. For each possible value of $\hat{r}_1$, among those $\hat{r}_2$ with $\hat{r}_1/\hat{r}_2 \in \mathbb{N}^*$, we pick the $\hat{r}_2$ with smallest $(\hat{r}_2 + \hat{r}_3 + \hat{r}_4)$, draw an arrow from it, and compute $(\hat{r}_1 + \hat{r}_2 + \hat{r}_3 + \hat{r}_4)$. Then we pick the $\hat{r}_1$ with the smallest $(\hat{r}_1 + \hat{r}_2 + \hat{r}_3 + \hat{r}_4)$ as the optimal solution. In the example,

larger r_i^{lb}'s will make the search space in Fig. 5.1 smaller, and thus improve readability.

Algorithm 5.2 The pseudocode to solve OPT-k-left.

1: Set $\hat{r}_k = r_k^{\text{lb}}$. Set $\mathcal{R}_k = \{\hat{r}_k\}$.

2: Compute $\mathcal{R}_i = \{n\hat{r}_k \mid r_i^{\text{lb}} \le n\hat{r}_k \le 1, n \in \mathbb{N}^*\}$ for each $i = 1, 2, \cdots, k-1$.

3: **for** $i = k-1, k-2, \cdots, 1$ **do**

4: For each $\hat{r}_i \in \mathcal{R}_i$, among $\hat{r}_{i+1} \in \mathcal{R}_{i+1}$ with $\hat{r}_i/\hat{r}_{i+1} \in \mathbb{N}^*$, pick the $\hat{r}_{i+1}$ with smallest $\sum_{m=i+1}^{k} \hat{r}_m$ and draw an arrow from it. Then compute $\sum_{m=i}^{k} \hat{r}_m$.

5: **end for**

6: For $\hat{r}_1 \in \mathcal{R}_1$, choose the one with smallest $\sum_{m=1}^{k} \hat{r}_m$ and get the optimal solution.

the smallest objective is 2, and $[\hat{r}_4^*, \hat{r}_3^*, \hat{r}_2^*, \hat{r}_1^*] = [0.2, 0.6, 0.6, 0.6]$. □

Based on the idea of the example, we formally give the pseudocode to solve OPT-k-left in Algorithm 5.2.

Now we analyze the time complexity of the algorithm in Algorithm 5.2. The number of elements in each $\mathcal{R}_i$ is no greater than $1/\hat{r}_k$, which is $O(1/\hat{r}_k)$. In each iteration of Step 3–5, we need to check whether there exists an arrow between each element in $\mathcal{R}_i$ and each element in $\mathcal{R}_{i+1}$, so the complexity is $O(1/\hat{r}_k^2)$. There are $(k-1)$ iterations, so the total complexity of the algorithm in Algorithm 5.2 is $O(k/\hat{r}_k^2)$.

Similar to solving OPT-k-left, we can use DP to solve OPT-k-right. The pseudocode to solve OPT-k-right is given in Algorithm 5.3. For time complexity, the number of elements in each $\mathcal{R}_i$ is $O(\hat{r}_k/\hat{r}_N^{\text{lb}})$. Similar to OPT-$k$-left, we can find the time complexity of the algorithm in Algorithm 5.3 is $O\big((n-k)\hat{r}_k^2/(r_N^{\text{lb}})^2\big)$.

By combining the solutions to OPT-k-left and OPT-k-right, we can get the solution to OPT-k. And by fixing $k = 1, 2, \cdots, N$, we can solve OPT. The pseudocode to solve OPT is given in Algorithm 5.4.

Algorithm 5.3 The pseudocode to solve OPT-k-right.

1: Set $\hat{r}_k = \hat{r}_k^{\text{lb}}$. Set $\mathcal{R}_k = \{\hat{r}_k\}$.

2: Compute $\mathcal{R}_i = \{\frac{\hat{r}_k}{n} \mid \frac{\hat{r}_k}{n} \geq r_i^{\text{lb}}, n \in \mathbb{N}^*\}$ for each $i = k+1, k+2, \cdots, N$.

3: **for** $i = k+1, k+2, \cdots, N$ **do**

4: For each $\hat{r}_i \in \mathcal{R}_i$, among $\hat{r}_{i-1} \in \mathcal{R}_{i-1}$ with $\hat{r}_{i-1}/\hat{r}_i \in \mathbb{N}^*$, pick the $\hat{r}_{i-1}$ with the smallest $\sum_{m=k}^{i-1} \hat{r}_m$ and draw an arrow from it. Then compute $\sum_{m=k}^{i} \hat{r}_m$.

5: **end for**

6: For $\hat{r}_N \in \mathcal{R}_N$, choose the one with smallest $\sum_{m=k}^{N} \hat{r}_m$ and get the optimal solution.

Algorithm 5.4 The pseudocode to solve OPT

Input: A general $\boldsymbol{r}^{\text{lb}}$ with $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) \leq 1$.

Output: An optimal solution to OPT, $\hat{\boldsymbol{r}}^*$

Set $\hat{r}_i^* = 1$ for each i. Set $J^* = N$.
for $k = 1, 2, \cdots, N$ **do**
  Set $\hat{r}_k = r_k^{\text{lb}}$ and $\mathcal{R}_k = \{\hat{r}_k\}$.
  if $k \geq 2$ **then**
    Use Algorithm 5.2 to find $\hat{r}_1, \hat{r}_2, \cdots \hat{r}_{k-1}$.
  end if
  if $k \leq N - 1$ **then**
    Use Algorithm 5.3 to find $\hat{r}_{k+1}, \hat{r}_{k+2}, \cdots \hat{r}_N$.
  end if
  if $\sum_{i=1}^N \hat{r}_i < J^*$ **then**
    Set $\hat{r}_i^* = \hat{r}_i$ for $i = 1, 2, \cdots, N$. Set $J^* = \sum_{i=1}^N \hat{r}_i$.
  end if
end for
return $\hat{\boldsymbol{r}}^* = [\hat{r}_1^*, \hat{r}_1^*, \cdots, \hat{r}_N^*]$

Algorithm 5.5 Stable Tolerant Scheduler (STS)

Input: $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ is stable tolerant, with $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) \leq 1$

Output: A feasible scheduler π

1: Compute $\boldsymbol{r}^{\text{lb}}$ by (5.10) and sort its elements in non-increasing order.

2: Solve OPT by Algorithm 5.4 and obtain an optimal solution $\boldsymbol{r}^*$.

3: If $\sum_{i=1}^{N} r_i^* \leq 1$, use Algorithm 5.1 (by letting $\boldsymbol{r}^{\text{lb}} = \boldsymbol{r}^*$) to find a feasible scheduler π.

Now we analyze the time complexity of Algorithm 5.4. The time complexity to solve OPT-k is $O(k/\hat{r}_k^2) + O\big((n-k)\hat{r}_k^2/(r_N^{\text{lb}})^2\big) = O\big(N/(r_N^{\text{lb}})^2\big) = O(Nd_{\max}^2)$. We need to solve N different OPT-k (for $k = 1, 2, \cdots, N$) in Algorithm 5.4, so the time complexity of Algorithm 5.4 is $O(N^2 d_{\max}^2)$.

Now we are ready to present our complete procedure, which we call *Stable Tolerance Scheduler* (STS) in Algorithm 5.5. Clearly, the time complexity of STS is $O(N^2 d_{\max}^2)$.

Note that Algorithm 5.5 isn't always able to find a feasible scheduler, even though $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) \leq 1$. The next question to ask is: *Under what condition is Algorithm 5.5 guaranteed to to find a feasible scheduler?* The answer is given in the following lemma.

Lemma 5.4. *For any stable tolerant* $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$*, STS can always find a feasible scheduler if* $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) \leq \ln 2$.

Proof. We prove Lemma 5.4 by proving its contrapositive, i.e., if STS cannot find a feasible scheduler for $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$, then $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) > \ln 2$.

Since STS cannot find a feasible scheduler for $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$, we must have $\sum_{i=1}^{N} \hat{r}_i^* > 1$. Since $\sum_{i=1}^{N} \hat{r}_i^*$ is the minimum, then for any $\hat{\boldsymbol{r}}$ that is feasible to OPT, we must have $\sum_{i=1}^{N} \hat{r}_i > 1$.

For any $x > 0$, we define a vector $\hat{\boldsymbol{r}}^x$ as

$$\hat{r}_i^x = x \cdot 2^{\lceil \log_2(\frac{r_i^{\text{lb}}}{x}) \rceil}, \ i = 1, 2, \cdots, N. \tag{5.14}$$

We have $\hat{r}_i^x \geq r_i^{\text{lb}}$ for each $i = 1, 2, \cdots, N$. We consider two cases.

- For all $i = 1, 2, \cdots N$, $\hat{r}_i^x \leq 1$. In this case, $\hat{\boldsymbol{r}}^x$ is feasible to OPT. So we have $\sum_{i=1}^N \hat{r}_i^x > 1$.
- For some i's, $\hat{r}_i^x > 1$. In this case, it's easy to see $\sum_{i=1}^N \hat{r}_i^x > 1$.

Therefore, we have

$$\sum_{i=1}^N \hat{r}_i^x > 1. \tag{5.15}$$

Define $g(x) = \frac{1}{x}$. We have

$$\int_{0.5}^1 g(x)dx = \ln 2. \tag{5.16}$$

Multiplying the two sides of (5.15) and (5.16) respectively, we have

$$\int_{0.5}^1 g(x)dx \cdot \sum_{i=1}^N \hat{r}_i^x > \ln 2. \tag{5.17}$$

Substituting $g(x) = \frac{1}{x}$ and (5.14) into (5.17), we have

$$\sum_{i=1}^N \int_{0.5}^1 2^{\lceil \log_2(\frac{r_i^{\text{lb}}}{x}) \rceil} dx > \ln 2. \tag{5.18}$$

It can be shown that the LHS of (5.18) equals to $\sum_{i=1}^{N} r_i^{\text{lb}} = l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$. So we have $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) > \ln 2$. □

5.5 From Stable Tolerant to Unstable Tolerant

In the last section, we studied the stable tolerant case (i.e., $\epsilon_i \geq p_i$ for all i's) and presented STS algorithm. In this section, we study the unstable tolerant case, i.e., $\epsilon_i < p_i$ for at least some i.

We will follow the same roadmap in the last section, with some necessary modifications along the way. We start with Theorem 5.1, which no longer holds in the unstable tolerance case. This is because $r_i \geq r_i^{\text{lb}}$ for each i is no longer a sufficient condition for an AUS to be feasible. As a fix to this problem, we propose *target rate* vector, defined as $\boldsymbol{r}^{\text{tar}} = [r_1^{\text{tar}}, r_2^{\text{tar}}, \cdots, r_N^{\text{tar}}]$. We wish to have an AUS be feasible if its $r_i \geq r_i^{\text{tar}}$.

In the unstable tolerant case, for those sources i's with $\epsilon_i \geq p_i$, we can just set $r_i^{\text{tar}} = r_i^{\text{lb}}$. So if $r_i \geq r_i^{\text{tar}}$ is satisfied, then (5.5) will be satisfied for these i's (from the proof of Theorem 5.1). Now we consider those i's with $\epsilon_i < p_i$. We want to find a r_i^{tar} such that: if $r_i \geq r_i^{\text{tar}}$ is satisfied, then (5.5) is also satisfied for these i's.

We consider one source i with $\epsilon_i < p_i$. Since it's difficult to analyze source i's AoI performance for a general AUS, we are going to consider an EUS, which guarantees a strict periodic transmission pattern for source i by relaxing certain transmission intervals with one extra slot. Let's consider an EUS with transmission rate r_i' for node i. Clearly, if (5.5) can be satisfied under this EUS (with r_i' for source i), then under any AUS with $r_i > r_i'$ (5.5) is also satisfied for source i. So all we need to do is to find the smallest r_i' for an EUS such that (5.5) is satisfied, and choose it as r_i^{tar} for source i.

Under the EUS, over all time windows $[t-d_i, t-1]$ (for all $t > d_i$), the average number of transmissions within a window is $d_i r_i'$. It is also easy to see that the number of transmissions within any window does not differ by more than 1. Therefore for any t, within a time window $[t-d_i, t-1]$, there are either $\lfloor d_i r_i' \rfloor$ or $(\lfloor d_i r_i' \rfloor + 1)$ transmissions for source i. Denote a as the fraction of t's with $\lfloor d_i r_i' \rfloor$ transmissions in $[t-d_i, t-1]$. Then the fraction of t's with $(\lfloor d_i r_i' \rfloor + 1)$ transmissions in $[t-d_i, t-1]$ is $(1-a)$. We have

$$a \cdot \lfloor d_i r_i' \rfloor + (1-a) \cdot (\lfloor d_i r_i' \rfloor + 1) = d_i r_i' \,. \tag{5.19}$$

Solving the above for a, we have:

$$a = \lfloor d_i r_i' \rfloor + 1 - d_i r_i' \,. \tag{5.20}$$

At each time t, suppose there are k transmissions for source i within the window $[t-d_i, t-1]$. Then only if all of these k transmissions fail, we will have $A_i(t) > d_i$. That is, $\mathbb{P}\{A_i(t) > d_i\} = p_i^k$. Since $k = \lfloor d_i r_i' \rfloor$ with probability a and $k = \lfloor d_i r_i' \rfloor + 1$ with probability $(1-a)$, we have

$$\begin{aligned} \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} [A_i(t) > d_i] &= a \cdot p_i^{\lfloor d_i r_i' \rfloor} + (1-a) \cdot p_i^{\lfloor d_i r_i' \rfloor + 1} \\ &= (\lfloor d_i r_i' \rfloor + 1 - d_i r_i') \cdot p_i^{\lfloor d_i r_i' \rfloor} + (d_i r_i - \lfloor d_i r_i' \rfloor) \cdot p_i^{\lfloor d_i r_i' \rfloor + 1}. \end{aligned} \tag{5.21}$$

To satisfy (5.5), we need to have

$$(\lfloor d_i r_i' \rfloor + 1 - d_i r_i') \cdot p_i^{\lfloor d_i r_i' \rfloor} + (d_i r_i' - \lfloor d_i r_i' \rfloor) \cdot p_i^{\lfloor d_i r_i' \rfloor + 1} \le \epsilon_i. \tag{5.22}$$

It can be shown the LHS of (5.22) is monotonically decreasing w.r.t. r_i'. To ensure (5.22) holds, we can solve the following equation:

$$(\lfloor d_i x \rfloor + 1 - d_i x) \cdot p_i^{\lfloor d_i x \rfloor} + (d_i x - \lfloor d_i x \rfloor) \cdot p_i^{\lfloor d_i x \rfloor + 1} = \epsilon_i \,. \tag{5.23}$$

We notice that for any x we have

$$p_i^{\lfloor d_i x \rfloor + 1} < \text{ LHS of (5.23)} \leq p_i^{\lfloor d_i x \rfloor}, \tag{5.24}$$

which means

$$p_i^{\lfloor d_i x \rfloor + 1} < \epsilon_i \leq p_i^{\lfloor d_i x \rfloor}. \tag{5.25}$$

Note that $p_i < 1$. By taking $\log_{p_i}(\cdot)$ on the three elements of (5.25), we have

$$\lfloor d_i x \rfloor + 1 > \log_{p_i} \epsilon_i \geq \lfloor d_i x \rfloor, \tag{5.26}$$

which indicates

$$\lfloor d_i x \rfloor = \lfloor \log_{p_i} \epsilon_i \rfloor. \tag{5.27}$$

Plugging (5.27) into (5.23), we can get the solution to (5.23):

$$x = \frac{(1-p_i)\lfloor \log_{p_i} \epsilon_i \rfloor + 1 - \epsilon_i p_i^{-\lfloor \log_{p_i} \epsilon_i \rfloor}}{(1-p_i)d_i}. \tag{5.28}$$

Then for any $r_i' \geq x$, (5.22) will hold. But under our EUS, $1/r_i'$ is the transmission interval length and must be an integer. So the smallest r_i' that ensures (5.22) holds is $r_i' = 1/\lfloor 1/x \rfloor$.

In summary, r_i^{tar} is given by

$$r_i^{\text{tar}} = \begin{cases} \frac{1}{\lfloor \frac{1}{x} \rfloor}, & \text{if } \epsilon_i < p_i, \\ r_i^{\text{lb}}, & \text{if } \epsilon_i \geq p_i, \end{cases} \tag{5.29}$$

where x is given by (5.28) and r_i^{lb} is given by (5.10).

Following the above discussions, we have the following lemma.

Lemma 5.5. *In the unstable tolerant case, an AUS π is feasible for $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ if $r_i \geq r_i^{tar}$ for each $i = 1, 2, \cdots, N$, where r_i^{tar} is given in (5.29).*

The proof of Lemma 5.5 is based on our discussion that once $r_i \geq r_i^{\text{tar}}$ for each $i = 1, 2, \cdots, N$, then (5.5) is satisfied for each i. Lemma 5.5 generalizes the results of the "if" part in Theorem 5.1. However, The "only if" part in Theorem 5.1 does not hold even if we generalize r_i^{lb} with r_i^{tar}. This is because we employed EUS relaxation when we define r_i^{tar} in Lemma 5.5 while there was no relaxation when we find r_i^{lb} in Theorem 5.1.

With Lemma 5.5 in hand, following the same roadmap in the last section, we then consider the special case of step-down rate vector $\boldsymbol{r}^{\text{tar}}$ (instead of $\boldsymbol{r}^{\text{lb}}$). We can use Algorithm 5.1 to construct an AUS for unstable tolerant $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ with step-down rate vector $\boldsymbol{r}^{\text{tar}}$ by merely replacing $\boldsymbol{r}^{\text{lb}}$ with $\boldsymbol{r}^{\text{tar}}$. We have the following lemma, which is similar to Theorem 5.2.

Lemma 5.6. *For any unstable tolerant $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ with a step-down rate vector $\boldsymbol{r}^{tar}$ and $\sum_{i=1}^{N} r_i^{tar} \leq 1$, the scheduler constructed by Algorithm 5.1 (with $\boldsymbol{r}^{lb}$ replaced by $\boldsymbol{r}^{tar}$) is feasible for $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$.*

Then we consider the general unstable tolerant $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$'s that may not have a step-down rate vector $\boldsymbol{r}^{\text{tar}}$. Similar to what we did in the last section, we propose to find a step-down rate vector $\hat{\boldsymbol{r}}$ with $\sum_{i=1}^{N} \hat{r}_i \leq 1$ and $\hat{r}_i \geq r_i^{\text{tar}}$ for each i. Again, we can solve an OPT, with r_i^{lb}'s replaced by r_i^{tar}'s.

Based on the above discussions, we present an algorithm, which we call *Unstable Tolerance Scheduler* (UTS), for the unstable tolerant case in Algorithm 5.6. The complexity of UTS is the same as the complexity of STS, which is $O(N^2 d_{\max}^2)$.

We have the following lemma that gives a sufficient condition for UTS to find a feasible scheduler.

Lemma 5.7. *For any unstable tolerant $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$, UTS can always find a feasible scheduler if $\sum_{i=1}^{N} r_i^{tar} \leq \ln 2$.*

Algorithm 5.6 Unstable Tolerant Scheduler (UTS)

Input: $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ is unstable tolerant, with $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) \leq 1$

Output: A feasible scheduler π

1: Compute $\boldsymbol{r}^{\text{tar}}$ by (5.29), and sort its elements in non-increasing order.

2: Replace r_i^{lb}'s by r_i^{tar}'s in OPT, solve OPT by Algorithm 5.4, and obtain an optimal solution $\boldsymbol{r}^*$.

3: If $\sum_{i=1}^{N} r_i^* \leq 1$, use Algorithm 5.1 (by letting $\boldsymbol{r}^{\text{lb}} = \boldsymbol{r}^*$) to find a feasible scheduler π.

The proof of Lemma 5.7 follows the same token as Lemma 5.4.

Denote $d_{\min}$ as the smallest element in $\boldsymbol{d}$, $\epsilon_{\min}$ as the smallest element in $\boldsymbol{\epsilon}$, and $p_{\max}$ as the smallest element in $\boldsymbol{p}$. Clearly, under the unstable tolerant case, we have $p_{\max} > \epsilon_{\min}$. The following lemma gives a sufficient condition in terms of system load for UTS to find a feasible scheduler.

Lemma 5.8. *For any unstable tolerant* $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$*, UTS can always find a feasible scheduler if*

$$l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) \leq \min_i \Big\{ \frac{d_i - \lfloor \log_{p_i} \epsilon_i \rfloor - 1}{(\lfloor \log_{p_i} \epsilon_i \rfloor + 1) \cdot d_i} \Big\} \cdot \ln 2. \tag{5.30}$$

Proof. Clearly, if $d_i - \lfloor \log_{p_i} \epsilon_i \rfloor - 1 \leq 0$ for some i's, then Lemma 5.8 holds trivially. So we will focus on the case when $d_i - \lfloor \log_{p_i} \epsilon_i \rfloor - 1 > 0$ for all i's.

Considering Lemma 5.7, to prove Lemma 5.8, it's sufficient to prove

$$\frac{d_i - \lfloor \log_{p_i} \epsilon_i \rfloor - 1}{(\lfloor \log_{p_i} \epsilon_i \rfloor + 1) \cdot d_i} \cdot r_i^{\text{tar}} \leq r_i^{\text{lb}} \tag{5.31}$$

for each $i = 1, 2, \cdots, N$. If $\epsilon_i \geq p_i$, we have $r_i^{\text{tar}} = r_i^{\text{lb}}$ so (5.31) holds trivially. Then we will focus on the case when $\epsilon_i < p_i$.

When $\epsilon_i < p_i$, we denote $k = \lfloor \log_{p_i} \epsilon_i \rfloor$. Then we have $k \in \mathbb{N}^*$ and $p_i^{k+1} < \epsilon_i \leq p_i^k$. Plugging $p_i^{k+1} < \epsilon_i$ into (5.28), we have

$$\begin{aligned} x &= \frac{(1-p_i)k + 1 - \epsilon_i p_i^{-k}}{(1-p_i)d_i} \\ &< \frac{(1-p_i)k + 1 - p_i^{k+1} \cdot p_i^{-k}}{(1-p_i)d_i} = \frac{k+1}{d_i}. \end{aligned} \tag{5.32}$$

Since $p_i > \epsilon_i$ and $r_i^{\text{lb}} = \frac{1-\epsilon_i}{(1-p_i)d_i}$, we have

$$x < (k+1) \cdot r_i^{\text{lb}}. \tag{5.33}$$

On the other hands,

$$r_i^{\text{tar}} = \frac{1}{\lfloor \frac{1}{x} \rfloor} < \frac{1}{\frac{1}{x} - 1} = \frac{d_i}{d_i - k - 1} \cdot x. \tag{5.34}$$

Combining (5.33) and (5.34), we can get (5.31). This completes our proof. □

Note that when d_i is relatively large, the guarantee factor given by Lemma 5.8 is $\frac{\ln 2}{\lfloor \log_{p_i} \epsilon_i \rfloor + 1}$. When $p_i \leq \epsilon_i$ for all i's (i.e., the stable tolerant case), the guarantee factor is $\ln 2$, which is the same as what Lemma 5.4 gives. In practice, generally the factor $\lfloor \log_{p_i} \epsilon_i \rfloor + 1$ wouldn't be too large. For example, LTE use link adaption algorithms to make sure the BLER (packet loss rate) is about 10% for each user [109]. In that case when $p_i = 0.1$, when $\epsilon_i \in (0.01, 0.1)$ the factor is 2, when $\epsilon_i \in (0.001, 0.01)$ the factor is 3, and so on.

Table 5.2: Stable tolerant case, $N = 10$

Source	1	2	3	4	5	6	7	8	9	10
d_i	6	10	10	12	15	16	18	20	25	30
p_i	0.1	0.05	0.15	0.05	0.1	0.05	0.1	0.05	0.2	0.1
ϵ_i	0.1	0.05	0.2	0.05	0.1	0.1	0.15	0.05	0.2	0.2
ρ_i	0.071	0.046	0.134	0.034	0.039	0.013	0.099	0.045	0.139	0.041

Table 5.3: Stable tolerant case, $N = 100$

Source	1-10	11-20	21-30	31-40	41-50	51-60	61-70	71-80	81-90	91-100
d_i	50	80	100	140	170	230	260	280	300	340
p_i	0.05	0.15	0.05	0.05	0.1	0.1	0.2	0.05	0.1	0.05
ϵ_i	0.1	0.2	0.15	0.05	0.1	0.2	0.25	0.15	0.15	0.1
$\rho_i \leq$	0.048	0.063	0.048	0.028	0.029	0.083	0.143	0.029	0.048	0.013

5.6 Numerical Results

In this section, we use simulations to validate our theoretical results and evaluate our algorithms.

5.6.1 Case Study

In this section we study four cases to validate that violation rate for each source is no greater than ϵ_i.

Stable Tolerant Case, $N = 10$

We consider $N = 10$ source nodes, with parameters d_i's, p_i's and ϵ_i's for $i = 1, 2, \cdots, 10$ given in Table 5.2. The load is $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) = 0.742$. As we can see, we have $\epsilon_i \geq p_i$ for each i, so $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ is stable tolerant and we will use STS to find a scheduler for it.

In STS, we first compute $\boldsymbol{r}^{\text{lb}} = [0.167\ 0.100\ 0.094\ 0.083\ \ 0.067\ 0.059\ 0.052\ 0.050\ 0.040\ 0.030]$. Then we solve OPT by Algorithm 5.4 and obtain an optimal solution $\boldsymbol{r}^* = [0.210\ \ 0.105\ 0.105$

0.105 0.105 0.105 0.052 0.052 0.052 0.052]. We have $\sum_{i=1}^{N} r_i^* = 0.944 \leq 1$, so we use Algorithm 5.1 to find a feasible scheduler π:

$$\pi = (ABDFJACEGABDFIACEH).$$

Readers can verify π is an AUS.

Under this AUS π, we simulate $T = 10,000,000$ time slots. Denote the measured (observed) violation rate for source i as $\rho_i = \frac{1}{T}\sum_{t=1}^{T}[A_i(t) > d_i]$ over these $10,000,000$ time slots. In Table 5.2 (last row), we show the measured ρ_i for each $i = 1, 2, \cdots, 10$. We find that $\rho_i < \epsilon_i$ for each i, so our AUS π is indeed feasible for the given $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$.

Stable Tolerant Case, $N = 100$

We now consider $N = 100$ source nodes as shown in Table 5.3. The load is $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) = 0.693$. As we can see, this $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ is unstable tolerant (e.g., $p_i \leq \epsilon_i$ for all sources). So we will use STS to find a scheduler for it.

In STS, we first solve OPT by Algorithm 5.4 and obtain an optimal solution $\boldsymbol{r}^*$. We have $\sum_{i=1}^{N} r_i^* = 0.900 < 1$, so we use Algorithm 5.1 to find a feasible scheduler π (with cycle length $c = 190$). Under π, we simulate $T = 10,000,000$ time slots, and calculate the violation rate $\rho_i = \frac{1}{T}\sum_{t=1}^{T}[A_i(t) > d_i]$ for all i's. For each group of 10 nodes, we list the largest measured violation rate ρ_i in the group (e.g., for $i = 1$–10, 0.048 is the largest measured violation rate among $\rho_1, \rho_2, \cdots, \rho_{10}$) in the last row of Table 5.3. We can see that $\rho_i < \epsilon_i$ for all $i = 1, 2, \cdots, 100$. So π is indeed feasible for the given $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$.

Unstable Tolerant Case, $N = 10$

We consider $N = 10$ source nodes, with parameters d_i's, p_i's and ϵ_i's for $i = 1, 2, \cdots, 10$ given in Table 5.4. The load is $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) = 0.639$. As we can see, we have $\epsilon_i > p_i$ for sources

Table 5.4: Unstable tolerant case, $N = 10$

Source	1	2	3	4	5	6	7	8	9	10
d_i	7	12	12	14	17	20	22	25	28	30
p_i	0.1	0.15	0.15	0.05	0.1	0.1	0.2	0.15	0.05	0.1
ϵ_i	0.1	0.1	0.2	0.15	0.05	0.1	0.15	0.2	0.1	0.1
$\rho_i \leq$	0.080	0.021	0.145	0.040	0.001	0.034	0.054	0.139	0.040	0.072

Table 5.5: Unstable tolerant case, $N = 100$

Source	1-10	11-20	21-30	31-40	41-50	51-60	61-70	71-80	81-90	91-100
d_i	60	100	120	150	190	240	270	300	330	360
p_i	0.2	0.1	0.05	0.1	0.15	0.2	0.1	0.05	0.2	0.15
ϵ_i	0.1	0.05	0.15	0.2	0.1	0.05	0.15	0.1	0.2	0.1
$\rho_i \leq$	0.041	0.041	0.051	0.079	0.078	0.040	0.090	0.039	0.144	0.087

$i = 2, 5, 7$, so $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ is unstable tolerant and we will use UTS to find a scheduler for it.

In UTS, we first compute $\boldsymbol{r}^{\text{tar}} = [0.143\ 0.125\ 0.100\ 0.078\ 0.064\ 0.063\ 0.050\ 0.038\ 0.034\ 0.033]$ (after sorting in non-increasing order). Then we solve OPT by Algorithm 5.4 and obtain an optimal solution $\boldsymbol{r}^* = [0.157\ 0.157\ 0.157\ 0.078\ 0.078\ 0.078\ 0.078\ 0.039\ 0.039\ 0.039]$. We have $\sum_{i=1}^{N} r_i^* = 0.902 \leq 1$, so we use Algorithm 5.1 to find a feasible AUS π:

$$\pi = (ABECGHABEDFJABECGIABEDF).$$

Under this AUS π, we simulate $T = 10,000,000$ time slots. In Table 5.2 (last row), we show the measured violation rate ρ_i for each $i = 1, 2, \cdots, 10$. We find that $\rho_i < \epsilon_i$ for each i, so our AUS π is indeed feasible for the given $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$.

Unstable Tolerant Case, $N = 100$

We now consider $N = 100$ source nodes, as shown in Table 5.5. The load is $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) = 0.658$. As we can see, this $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ is unstable tolerant (e.g., $p_i > \epsilon_i$ for sources 1–10, among others). So we will use UTS to find a scheduler for it. In UTS, we first compute $\boldsymbol{r}^{\text{tar}}$, then solve OPT

by Algorithm 5.4, and obtain an optimal solution $\boldsymbol{r}^*$. We have $\sum_{i=1}^{N} r_i^* = 0.976 < 1$, so we use Algorithm 5.1 to find a feasible scheduler π (with cycle length $c = 240$). Under π, we simulate $T = 10,000,000$ time slots, and calculate the violation rate ρ_i, as shown in Table 5.5. We can see that $\rho_i < \epsilon_i$ for all $i = 1, 2, \cdots, 100$. So π is indeed feasible for the given $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$.

5.6.2 Impact of Tolerance Rate

In this section we study the impact of tolerance rate ϵ for algorithms STS and UTS. We consider 50 sources, i.e., $N = 50$. For ease of presentation, we assume all sources have the same p_i and the same ϵ_i, i.e., $p_i = p$ and $\epsilon_i = \epsilon$ for all i's. We assume $p = 0.1$ and vary ϵ from 0.04 to 0.3, i.e., from a value smaller than p to a value greater than p. When $\epsilon < p$, we use UTS and when $\epsilon \geq p$, we use STS.

We assume $d_i \in \{10, 20, \cdots, 300\}$. For each ϵ, we randomly generate 100 different $\boldsymbol{d}$'s for each load interval $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) \in (0.3, 0.32], (0.32, 0.34], \cdots, (0.98, 1]$. For each load interval, we run the algorithm and calculate the percentage of $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$'s for which our algorithm (either STS or UTS, depending on ϵ) successfully finds a feasible scheduler. The results are shown in Fig. 5.2. As we can see, as ϵ increases, the corresponding curve shifts to the right, meaning that for the same success percentage in finding a feasible scheduler, the network can take a higher load when ϵ increases. Alternatively, for the same load, the success percentage in finding a feasible scheduler increases with ϵ. We also show the load guarantee for STS, $\ln 2$ in the figure. We can see in the stable tolerant case, when $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) \leq \ln 2$, the success percentage of STS is 100% (i.e., when ϵ is 0.1, 0.2, and 0.3), which confirms Lemma 5.4.

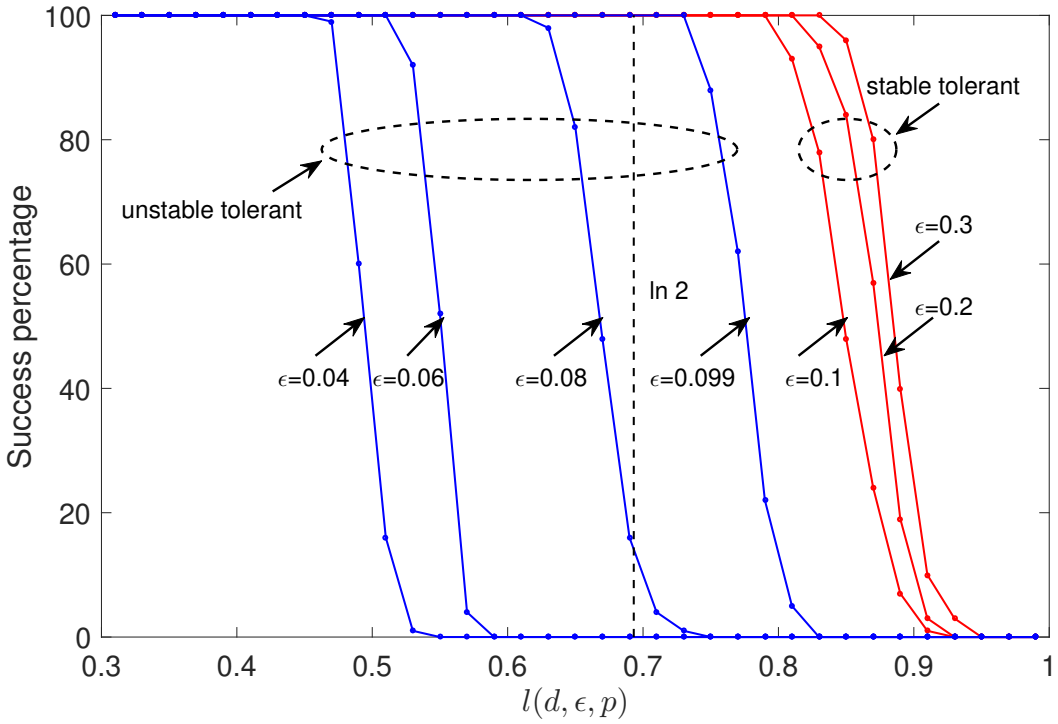


Figure 5.2: Success percentage for STS/UTS to find a feasible scheduler when $p = 0.1$.

5.6.3 Impact of Packet Loss Rate

In this section we study the impact of packet loss rate p for algorithms STS and UTS. We consider 50 sources, i.e., $N = 50$. Again, we assume all sources have the same p_i and the same ϵ_i, i.e., $p_i = p$ and $\epsilon_i = \epsilon$ for all i's. We assume $\epsilon = 0.1$ and vary p from 0.05 to 0.3, i.e., from a value smaller than ϵ to a value greater than ϵ. When $\epsilon < p$, we use UTS and when $\epsilon \geq p$, we use STS.

We assume $d_i \in \{10, 20, \cdots, 300\}$. For each ϵ, we randomly generate 100 different $\boldsymbol{d}$'s for each load interval $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) \in (0.3, 0.32], (0.32, 0.34], \cdots, (0.98, 1]$. For each load interval we run the algorithm and calculate the percentage of $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$'s for which our algorithm (either STS or UTS, depending on p) successfully finds a feasible scheduler. The results are shown in Fig. 5.3. As we can see, as p decreases, the corresponding curve shifts to the right, meaning

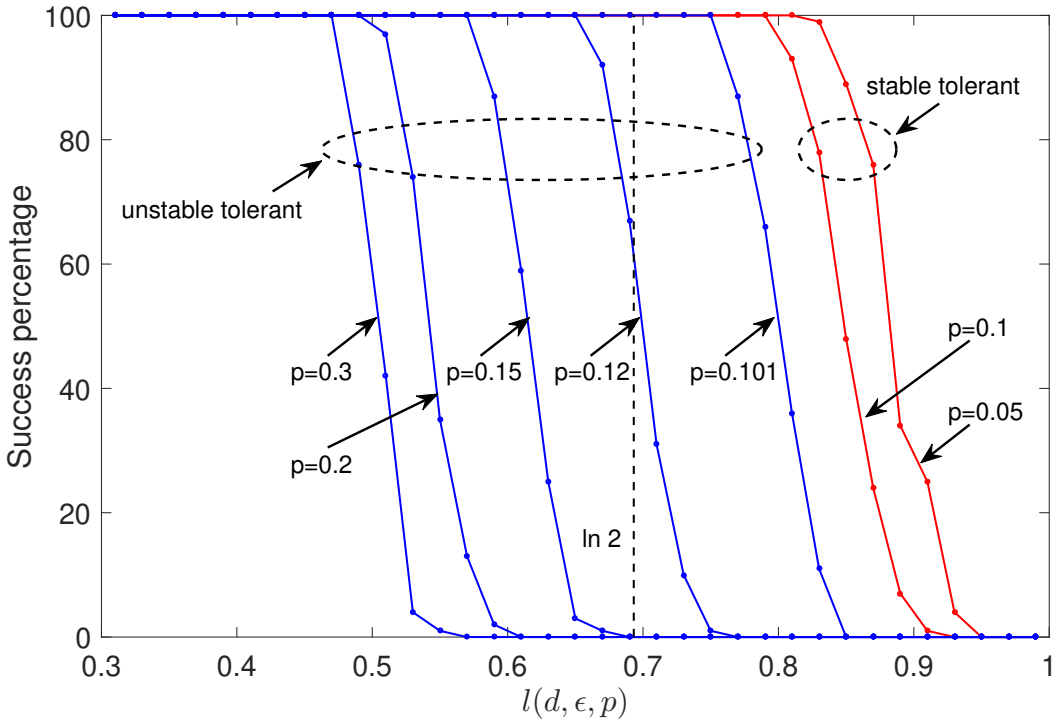


Figure 5.3: Success percentage for STS/UTS to find a feasible scheduler when $\epsilon = 0.1$.

that for the same success percentage in finding a feasible scheduler, the network can take a higher load when p decreases. Alternatively, for the same load, the success percentage in finding a feasible scheduler decreases with p. We also show the load guarantee for STS, $\ln 2$ in the figure. We can see in the stable tolerant case, when $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) \leq \ln 2$, the success percentage of STS is 100% (i.e., when p is 0.05 and 0.1), which confirms Lemma 5.4.

5.7 Chapter Summary

This chapter investigated an important AoI scheduling problem that considers soft AoI deadlines and tolerance of occasional deadline violations. We first presented system load $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ that seamlessly integrates AoI deadline requirement $\boldsymbol{d}$, channel condition $\boldsymbol{p}$ and violation tolerance $\boldsymbol{\epsilon}$ into one metric, and showed that $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ is schedulable only if $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) \leq 1$. Then

we addressed the scheduling problem by exploring the relationship between $\boldsymbol{\epsilon}$ and $\boldsymbol{p}$ and identified two cases: stable tolerant and unstable tolerant. For the stable tolerant case, we presented STS, which can find a feasible scheduler as long as $l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p}) \leq \ln 2$. For the unstable tolerant case, we presented UTS, and gave a sufficient condition for it to find a feasible scheduler. We used numerical results to validate our theoretical results.

Chapter 6

Scheduling with Soft AoI Deadlines in 5G Networks

6.1 Introduction

In Chapter 5, we studied the scheduling problem with a soft AoI deadline for each information source. In this chapter, we extend the problem in Chapter 5 to 5G network settings, i.e., we focus on designing 5G-compliant schedulers with soft AoI deadlines.

Again, we consider the settings in Fig. 1.1. At the BS, we assume that there is a soft AoI deadline for information (sample) from each source. Given the unknown nature of future channel conditions, it is unrealistic to demand a hard guarantee of AoI deadline for each source. Instead, it is more reasonable to minimize the proportion of time for each source when its information is outdated (i.e., AoI deadline is exceeded). In particular, it is plausible to ask for a scheduler that minimizes the maximum proportion of outdated information among all source nodes.

There are a number of challenges to design such a scheduler. First, an online scheduler is intrinsically difficult to design, due to the unknown knowledge of the future (e.g., channel conditions). Second, there are some unique considerations associated with 5G scheduling, such as frequency-time resource grids (RBs) and modulation and coding scheme (MCS) [31]. In each transmission time interval (TTI) the scheduler needs to allocate $\sim$100 RBs to $\sim$100 source nodes, as well as select one MCS (from 29 levels) for each source node, which

presents an extremely large search space. Finally, a 5G scheduler has a stringent real-time requirement: the scheduler must make scheduling decisions within one TTI, which is in sub-millisecond time scale. Designing a scheduler that meets all the above three requirements is not a trivial task, especially for the AoI optimization objective that we wish to achieve in this chapter. The main contributions of this chapter are the following:

- We investigate a 5G IoT network for data collection with the objective of finding an online scheduler to minimize the maximum proportion of outdated information among all source nodes. For performance benchmark, we develop an offline computational procedure to find a lower bound for the objective value. The lower bound is obtained via a series of relaxation and reformulation, following which an effective technique can be applied to solve the reformulated optimization problem.

- We develop an important and interesting property, called *uniform fairness*, that is associated with an offline optimal scheduler. It says that there exists an optimal offline scheduler to our objective, under which the proportion of outdated information by all source nodes should converge to the same value as time becomes large, regardless of the difference in AoI deadlines and sample sizes among the source nodes. This interesting property offers us an important guideline in our design of an online scheduler where we only have knowledge of the past and present, not the future.

- We present Aequitas—an online 5G scheduler to optimize our objective. At the heart of Aequitas is a novel priority metric that takes the AoIs, AoI deadlines, channel conditions and historical performance behavior into consideration. By computing this priority metric iteratively in each time slot (TTI), Aequitas performs RB allocation and MCS selection for the source nodes that are selected for transmission.

- Although the time complexity of Aequitas is polynomial, its running time is still too

long to meet the 5G timing requirement. We propose to exploit Aequitas' intrinsic property (amenable to parallel computation) and employ an off-the-shelf GPU platform in our implementation. Experimental results show that Aequitas can achieve excellent performance in terms of objective function (i.e., close to the lower bound) and its running time is under 1 ms.

6.2 System Model and Problem Statement

In this section we present the 5G data collection network model and state the min-max scheduling problem in this chapter. Table 6.1 lists the key notations in this chapter.

6.2.1 System Model

We consider a 5G-based IoT network where N source nodes collect information and forward it to a base station (BS) (see Fig. 1.1). We assume an on-demand sampling strategy, under which each source node collects a sample just before it is scheduled (notified) for transmission. Such an on-demand sampling strategy relies on a downlink control channel through which the BS sends its scheduling decision to the source nodes.

For source node i, denote L_i as the sample size (in unit of bits), which is the amount of information carried in a sample. We assume the sample size (L_i) only depends on its source and is invariant over time. The BS maintains the most recent (freshest) sample from each source. Once a new sample from a source node is received completely by the BS, the BS deletes the previous sample and replaces it with this new one.

The source nodes transmit information (collected samples) to the BS through a 5G uplink channel. In 5G, uplink transmission resource is organized into 2-dimensional grids of *resource blocks* (RBs) that span both time and frequency domains [31]. In the time domain, time

is equally slotted into *transmission time intervals* (TTIs), while in the frequency domain, bandwidth is equally slotted into a large number of sub-carriers, and 12 sub-carriers over a TTI is called an RB. RB is the smallest scheduling unit in 5G, and in each TTI there is a large number of RBs that can be allocated to the source nodes for uplink transmission. Due to channel fading (both time-selective and frequency-selective), the channel conditions on different RBs are different in general, even with respect to the same source node.

The BS employs a scheduler to allocate the uplink RBs to the source nodes based on the channel conditions in each TTI. Denote B as the total number of RBs that is available for uplink transmission in each TTI. We assume one RB can be allocated to at most one source node in each TTI. Denote $x_i^b(t)$ as a binary variable indicating whether RB $b \in \{1, 2, \cdots, B\}$ is allocated to source node i at TTI t. We have

$$x_i^b(t) = \begin{cases} 1 & \text{if RB } b \text{ is allocated to node } i \text{ at TTI } t, \\ 0 & \text{otherwise,} \end{cases}$$

and

$$\sum_{i \in \mathcal{N}} x_i^b(t) \leq 1, \ b \in \{1, 2, \cdots, B\}. \tag{6.1}$$

In each TTI, the scheduler also needs to choose a modulation and coding scheme (MCS) for each source node [31]. The MCS of each source node determines the modulation and coding rate—how much information (in unit of bits) is modulated and coded within each RB for this source node. The higher the MCS is, the higher the modulation and coding rate is. On the other hand, the maximum amount of information that can be successfully transmitted by an RB depends on the channel condition. If the channel condition for this RB is poor and the source node uses a high MCS, information carried in the RB will not be successfully received and decoded by the BS.

Under 5G, there are $M = 29$ different levels of MCSs [31]. We assume $m = 1$ is the lowest

Table 6.1: Notations

Symbol	Definition
$A_i(t)$	AoI for source node i at the BS at time t
B	Number of RBs
c^m	Modulation and coding rate under MCS m
d_i	AoI deadline for source node i
M	Number of MCSs
N	Number of source nodes
$q_i^b(t)$	Channel condition for source i and RB b at TTI t
$R_i(t)$	Data rate for source i at time t
v_i	Long-term average outdated proportion for source i
$v_{\max}$	Greatest one among all v_i's
$x_i^b(t)$	Variable for whether RB b is allocated to source i at t
$y_i^m(t)$	Variable for whether MCS m is selected for source i at t

MCS and $m = 29$ is the highest MCS. Denote $q_i^b(t)$ as the maximum MCS level that source node i's channel can support on RB b at TTI t. We have:

$$0 \leq q_i^b(t) \leq M.$$

In practice, $q_i^b(t)$ is determined by the channel quality indicator (CQI) report carried in the feedback from source node i at TTI $(t-1)$. Denote c^m as the modulation and coding rate under MCS level m, which can be found in Table 5.1.3.1-1 in [31]. Denote $r_i^{b,m}(t)$ as the achievable data rate by RB b w.r.t. source node i under MCS m. If $m \leq q_i^b(t)$, the transmission is successful and the achievable data rate is c^m. Otherwise, i.e., $m > q_i^b(t)$, the transmission is unsuccessful and the achievable data rate is 0. We have:

$$r_i^{b,m}(t) = \begin{cases} c^m & \text{if } m \leq q_i^b(t), \\ 0 & \text{otherwise.} \end{cases} \tag{6.2}$$

Note that although each RB can only be allocated to at most one source node in a TTI, a source node may be allocated with multiple RBs. For a source node allocated with multiple RBs, it must choose and use one MCS $m \in \{1, 2, \cdots, M\}$ for all its allocated RBs [31].

Denote $y_i^m(t)$ as a binary variable indicating whether MCS $m \in \{1, 2, \cdots, M\}$ is chosen by source node i at TTI t, i.e.,

$$y_i^m(t) = \begin{cases} 1 & \text{if MCS } m \text{ is chosen for source } \ i \text{ at TTI } t, \\ 0 & \text{otherwise.} \end{cases}$$

We have

$$\sum_{m=1}^{M} y_i^m(t) \leq 1, \ i \in \{1, 2, \cdots, N\}. \tag{6.3}$$

Denote $R_i(t)$ as the amount of information transmitted by source node i in TTI t across all RBs allocated to it. We have

$$R_i(t) = \sum_{b \in \mathcal{B}} \sum_{m \in \mathcal{M}} x_i^b(t) y_i^m(t) r_i^{b,m}(t), \ i \in \{1, \cdots, N\}. \tag{6.4}$$

Recall that the sample size for source i is L_i. In TTI t, if $R_i(t) \geq L_i$, then this sample can be transmitted completely within this TTI. Otherwise, part of the sample will be left to the following TTI(s) for transmission.

6.2.2 AoI Notation

AoI is location-dependent. AoI at the BS is defined as the elapsed time between the present and the time when the sample (at the BS) was generated at its source [5, 6]. For the most recent sample from source i that is currently maintained by the BS, denote $U_i(t)$ as its generation time (at its source node). Then the AoI for source node i at the BS, denoted as $A_i(t)$, is:

$$A_i(t) = t - U_i(t). \tag{6.5}$$

Under on-demand sampling strategy, each source node collects a sample just before it is scheduled for transmission. Recall the transmission of a sample under our 5G model may

span multiple TTIs. Denote $\tau_i(n)$ (in unit of TTIs) as the time duration from the beginning of the TTI when the transmission starts to the end of the TTI when the transmission ends for the n-th sample ($n = 1, 2, \cdots$) from source node i. Clearly, $\tau_i(n)$ is an integer with $\tau_i(n) \geq 1$.

- At the end of TTI t, if no sample from source node i arrives at the BS, then at the beginning of TTI $(t+1)$ its AoI at the BS will increase by one.

- On the other hand, if the n-th sample arrives at the end of TTI t (i.e., TTI t is the last time slot of $\tau_i(n)$), then the new sample's generation time is the beginning of TTI $t - \tau_i(n) + 1$.[1] Then at the beginning of TTI $(t+1)$, the BS has the complete new sample and $U_i(t+1) = t + 1 - \tau_i(n)$. We have:

$$A_i(t+1) = t + 1 - U_i(t+1) = \tau_i(n).$$

Combining the two cases, we have:

$$A_i(t+1) = \begin{cases} A_i(t) + 1, & \text{if no sample arrives in } t, \\ \tau_i(n), & \text{if } n\text{-th sample arrives in } t. \end{cases} \tag{6.6}$$

6.2.3 Problem Statement

In this chapter, we assume that each information source has an AoI deadline at the BS, denoted by d_i. For each source node i, when its AoI (of the stored sample) at the BS is no greater than d_i, we consider the information is *fresh*; when its AoI is greater than d_i, we consider the information as *outdated* (not fresh). Our goal is to minimize the proportion of time when the information at the BS is outdated across all source nodes.

[1] We assume the time to collect a sample at a source is instant and does not take any TTIs.

More formally, denote v_i as the proportion of TTIs when the information from source i is outdated at the BS. We have:

$$v_i = \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} [A_i(t) > d_i], \tag{6.7}$$

where "$[\cdot]$" is Iverson bracket, returning 1 if the inside statement is true and 0 otherwise. Denote $v_{\max}$ as the largest among all v_i's, i.e.,

$$v_{\max} = \max_{i\in\{1,2,\cdots,N\}} v_i. \tag{6.8}$$

So a plausible objective is to minimize $v_{\max}$. This is the objective of our optimization problem. Note that $v_{\max}$ is also a long-term metric, just as the v_i's.

Now we state the optimization problem studied in this chapter.

$$\begin{aligned}
\text{OPT:}\ \min\quad & v_{\max} \\
\text{s.t.}\quad & \text{RB allocation constraint (6.1)}, \\
& \text{MCS selection constraint (6.3)}, \\
& \text{Calculation of data rates (6.2), (6.4)}, \\
& \text{Calculation of AoI (6.6)}, \\
& \text{Calculation of outdated proportion (6.7), (6.8)}.
\end{aligned}$$

In problem OPT, the decision variables are $x_i^b(t)$'s and $y_i^m(t)$'s for each TTI t.

There are a number of technical challenges to the design of a scheduler. First and foremost, the scheduler will be an online algorithm, without any knowledge of future information (including channel conditions) So it is impossible to obtain a provably optimal scheduler. Second, the search space of OPT is very large. In each TTI there are B^N possibilities for $x_i^b(t)$'s and N^M possibilities for $y_i^m(t)$'s (e.g., when $B = N = 100$ and $M = 29$, the search space consists of 1.7×10^{346} possibilities). Finding a near-optimal solution from this search

space is challenging. Finally, a 5G scheduler has a stringent real-time requirement. The scheduler must make scheduling decisions within one TTI, which is in sub-millisecond time scale under 5G. It is very challenging to find a near-optimal scheduling solution in such a time scale.

6.3 Performance Bound

Since the scheduler that we will design addresses an online scheduling problem (with no knowledge of the future), it is impossible to find a provably optimal one. Nevertheless, we can derive a lower bound for the objective $v_{\max}$, which can serve as a benchmark for the performance of any proposed scheduler.

To find a lower bound for $v_{\max}$, we find that it is more convenient to use long term average data rate in the objective function. Therefore, in Section 6.3.1, we present a relaxation by replacing $v_{\max}$ with a function of data rates and form a new optimization problem for the lower bound. In Section 6.3.2, we present a series of reformulation and relaxation to make objective function more tractable. Finally, in Section 6.3.3, we exploit an effective technique in the literature to solve the reformulated optimization problem.

6.3.1 A New Objective Function for Lower Bound

For any scheduler π, denote $\overline{R}_i$ as the long-term average data rate for source node i, i.e.,

$$\overline{R}_i = \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} R_i(t). \tag{6.9}$$

Denote p_i as the fraction of TTIs when samples from source i arrive at the BS (counting only the last TTI that completes each sample's transmission). Recall the sample size for

source i is L_i. It is easy to see p_i is proportional to $\overline{R}_i$ with a factor L_i, i.e.,

$$\overline{R}_i = p_i \cdot L_i. \tag{6.10}$$

Note that a new sample from source i can ensure AoI at the BS for that source not to exceed its deadline limit for no more than d_i TTIs. So with p_i, the fraction of TTIs with AoI for source i being under its deadline limit is no greater than $p_i d_i$. Recall $(1-v_i)$ is the proportion of TTIs with source i's AoI being under d_i. Therefore, we have

$$1 - v_i \leq p_i d_i = \frac{\overline{R}_i d_i}{L_i}, \tag{6.11}$$

which is equivalent to

$$v_i \geq 1 - \frac{\overline{R}_i d_i}{L_i}. \tag{6.12}$$

For $v_{\max}$ in (6.8), we have

$$v_{\max} \geq \max_{i \in \{1,2,\cdots,N\}} \{1 - \frac{\overline{R}_i d_i}{L_i}\}. \tag{6.13}$$

To find a lower bound to OPT, we can relax its objective $v_{\max}$ to the RHS of (6.13). So we have the following new optimization problem for the lower bound (LB):

$$\text{OPT-LB: } \min \max_{i \in \{1,2,\cdots,N\}} \{1 - \frac{\overline{R}_i d_i}{L_i}\}$$
$$\text{s.t.} \quad \text{Constraints (6.1), (6.2), (6.3), (6.4).}$$

Since optimal objective value of OPT-LB is always no greater than the objective of OPT, it can can serve as a lower bound for $v_{\max}$. Note that constraints (6.6), (6.7) and (6.8) are no longer included in OPT-LB because they do not affect either the new objective function or other constraints in OPT-LB.

6.3.2 Reformulation and Relaxation

Instead of working with $\{1 - \frac{\overline{R}_i d_i}{L_i}\}$ in the min max function, we can rewrite the objective function in OPT-LB as max min function as follows:

$$\text{OPT-LB2: } \max \min_{i \in \{1,2,\cdots,N\}} \{\frac{\overline{R}_i d_i}{L_i}\}$$
$$\text{s.t. Constraints (6.1), (6.2), (6.3), (6.4).}$$

Clearly, OPT-LB and OPT-LB2 are equivalent.

OPT-LB2 is a scheduling problem to maximize a utility function of $\overline{R}_i$. In [61] the authors studied a similar problem and showed that a gradient scheduling algorithm can achieve the optimal objective value asymptotically (when the number of TTIs goes to infinity). However, the utility function in [61] is required to be a concave smooth function. But the utility function in OPT-LB2 ($\min_i\{\frac{\overline{R}_i d_i}{L_i}\}$) isn't smooth. To address this issue, we will perform a relaxation for the utility function as follows.

Define a *smooth min* function S_α with parameter $\alpha > 0$ for $x_1, x_2, \cdots, x_N > 0$ as:

$$S_\alpha(x_1, x_2, \cdots, x_N) = \frac{\sum_{i=1}^N x_i e^{-\alpha x_i}}{\sum_{i=1}^N e^{-\alpha x_i}}. \tag{6.14}$$

Since $S_\alpha(x_1, x_2, \cdots, x_N) > \min\{x_1, x_2, \cdots, x_N\}$ for any $x_1, x_2, \cdots, x_N > 0$, we can perform a relaxation to OPT-LB2 as following, which we denote as OPT-LB-α:

$$\text{OPT-LB-}\alpha\text{: } \max S_\alpha(\frac{\overline{R}_1 d_1}{L_1}, \frac{\overline{R}_2 d_2}{L_2}, \cdots, \frac{\overline{R}_N d_N}{L_i})$$
$$\text{s.t. Constraints (6.1), (6.2), (6.3), (6.4).}$$

Clearly, the optimal objective of OPT-LB-α, denoted by S^*_α, is always greater than the optimal objective of OPT-LB2. Then $(1 - S^*_\alpha)$ is always smaller than the optimal objective of OPT-LB, so it can serve as a lower bound for $v_{\max}$.

Note that when $\alpha \to \infty$, $S_\alpha(x_1, x_2, \cdots, x_N) = \min\{x_1, x_2, \cdots, x_N\}$. So we always choose

a large α to tighten the relaxation.

6.3.3 Solving OPT-LB-α

In OPT-LB-α, the objective function $S_\alpha(x_1, x_2, \cdots, x_N)$ is smooth and concave. It has been shown in [61] that the so-called *gradient scheduling algorithm* can be used to solve it.

Gradient Scheduling Algorithm In a gradient scheduling algorithm, an empirical data rate $R_i^e(t)$ is defined for each TTI t and updated as an exponentially smoothed average as follows:

$$R_i^e(t+1) = (1-\beta) \cdot R_i^e(t) + \beta \cdot R_i(t), \tag{6.15}$$

where β is a small positive constant (e.g., 0.01) and $R_i(t)$ is the instantaneous data rate at TTI t and is given in (6.4). It is easy to show that $R_i^e(t) \to \overline{R}_i$ as $t \to \infty$. In practice, t does not need to be very large for $R_i^e(t)$ to approach $\overline{R}_i$.

Denote $\boldsymbol{R}^e(t) = [R_1^e(t)\ R_2^e(t)\ \cdots\ R_N^e(t)]^T$ and $\boldsymbol{r} = [r_1\ r_2\ \cdots\ r_N]^T$. The gradient scheduling algorithm solves OPT-LB-α by maximizing the following gradient-based objective function:

$$\sum_{i=1}^{N} \frac{\partial S_\alpha(\frac{r_1 d_1}{L_1}, \frac{r_2 d_2}{L_2}, \cdots, \frac{r_N d_N}{L_N})}{\partial r_i}\Big|_{\boldsymbol{r}=\boldsymbol{R}^e(t)} R_i(t)$$

in each TTI t. Note that

$$\frac{\partial S_\alpha}{\partial r_i}\Big|_{\boldsymbol{r}=\boldsymbol{R}^e(t)} = \frac{d_i e^{-\frac{\alpha d_i R_i^e(t)}{L_i}} \left(1 + \alpha(S_\alpha - \frac{d_i R_i^e(t)}{L_i})\right)}{L_i \sum_{j=1}^{N} e^{-\frac{\alpha d_j R_j^e(t)}{L_j}}}. \tag{6.16}$$

Denote the RHS of (6.16) as $g_i(t)$. Then the gradient scheduling algorithm aims to maximize $\sum_{i=1}^{N} g_i(t) R_i(t)$ in every TTT t. With $R_i(t)$ given in (6.4), we have the following optimization

problem OPT-LB-t in every TTI t:

$$\text{OPT-LB-}t\text{: max} \quad \sum_{i\in\mathcal{N}}\sum_{b\in\mathcal{B}}\sum_{m\in\mathcal{M}} g_i(t)r_i^{b,m}(t)x_i^b(t)y_i^m(t)$$
$$\text{s.t.} \quad \text{Constraints (6.1), (6.2), (6.3),}$$

where the decision variables are $x_i^b(t)$'s and $y_i^m(t)$'s. Problem OPT-LB-t is an integer quadratic program (IQP), which can be solved by a commercial solver such as CPLEX [62]. Note that solving OPT-LB-t by CPLEX will cost 10s of seconds, which is far beyond the 5G timing requirement. Therefore, OPT-LB-t can be solved offline to get a performance bound of OPT, but its solution cannot be used as a real-world 5G online scheduler.

Note that we need to solve OPT-LB-t for a large number (say T) of TTIs. Then we let $\overline{R}_i = R_i^e(T)$ for each $i = 1, 2, \cdots, N$ and substitute $\overline{R}_i$'s into the objective of OPT-LB-α to get S_α^*—the optimal objective to OPT-LB-α. And we can use $(1 - S_\alpha^*)$ as a lower bound of $v_{\max}$.

Algorithm 6.1 presents a pseudocode for using the gradient scheduling algorithm to find a lower bound for the objective $v_{\max}$. Note that $v_{\max}$ cannot be less than 0. So if the algorithm gives a value less than 0, we can instead use 0 as the lower bound.

Example Consider a network with $N = 100$ and $B = 100$. For each source node, we assume a Rician fading channel with no time or frequency correlation. More details about the simulation settings can be found in Section 6.5. For $\alpha = 10$ and $\beta = 0.01$, we run Algorithm 6.1 for $T = 1,000$ TTIs and show the convergence behavior of $1 - S_\alpha^*$ (which we use as a lower bound for $v_{\max}$) in Fig. 6.1. We can see that setting $T = 1,000$ is adequate as the terminating time. At $T = 1,000$, the value of $(1 - S_\alpha^*)$ is 0.367, which is the lower bound for this example.

Algorithm 6.1 An Algorithm to Find a Lower Bound for $v_{\max}$.

Input: α, β, T.

Output: A lower bound for $v_{\max}$.

1: Set $R_i^e(0) = 0$ for each $i = 1, 2, \cdots, N$.

2: **for** $t = 1, 2, \cdots, T$ **do**

3: Solve OPT-LB-t and get the optimal solution $x_i^b(t)$'s and $y_i^m(t)$'s.

4: Use $x_i^b(t)$'s and $y_i^m(t)$'s to perform scheduling at TTI t and get the data rate $R_i(t)$ for each $i = 1, 2, \cdots, N$.

5: Use (6.15) to update $R_i^e(t)$ for $i = 1, 2, \cdots, N$.

6: **end for**

7: Let $\overline{R}_i = R_i^e(T)$ for each $i = 1, 2, \cdots, N$.

8: Substitute $\overline{R}_i$'s into the objective of OPT-LB-α to get S_α^*, and use $(1 - S_\alpha^*)$ as a lower bound for $v_{\max}$. If $S_\alpha^* > 1$, then use 0 as the lower bound.

6.4 Algorithm Design

In this section we present Aequitas[2]—a real-time scheduler to solve problem OPT. In Section 6.4.1, we derive an interesting property which we call *uniform fairness* for an ideal (offline) optimal scheduler. Although an ideal optimal scheduler cannot be obtained (as future channel information is unknown), we can use this uniform fairness property as a guideline in our design of an online scheduler. In Section 6.4.2, we outline the main ideas in our design of Aequitas. In Section 6.4.3, we present the details of Aequitas, along with a GPU-based implementation.

[2]Aequitas is the Latin concept of justice, equality, and fairness.

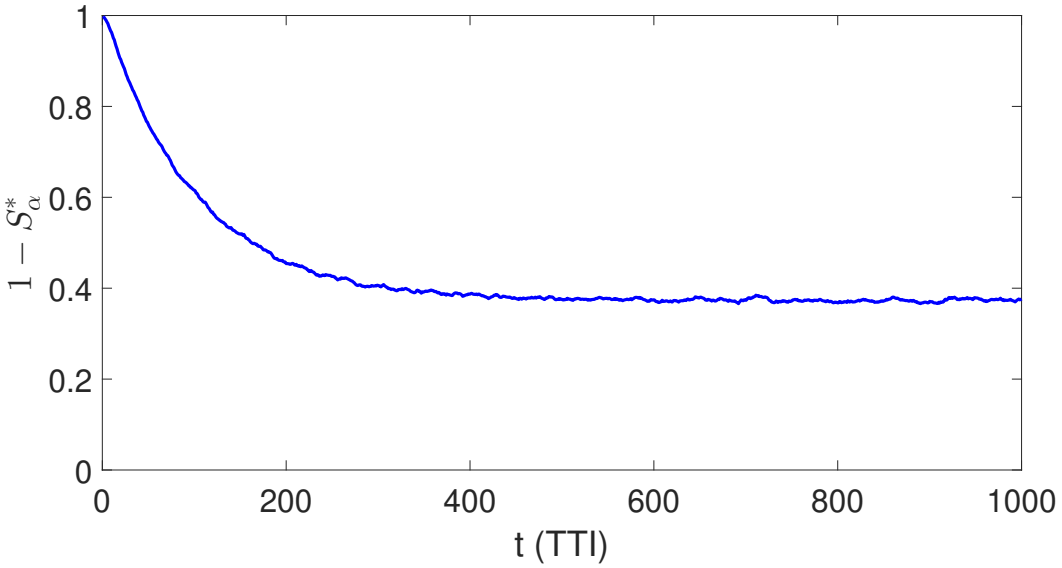


Figure 6.1: An illustration of convergence behavior of Algorithm 6.1 when $\alpha = 10$ and $\beta = 0.01$.

6.4.1 Uniform Fairness

Denote π^* as an ideal offline optimal scheduler to OPT. By "ideal", we assume π^* has all future channel information when performing scheduling. Denote $v_1^*, v_2^*, \cdots, v_N^*$ as the outdated proportions for sources 1 to N under π^*. Denote $v_{\max}^* = \max\{v_1^*, v_2^*, \cdots, v_N^*\}$, which is the optimal objective of OPT. Recall that each source node i has its own sample size L_i and the average channel condition varies for different sources. So we ask the following question: Will v_i^*'s be different for different source nodes? The answer to this question is given in the following lemma.

Lemma 6.1. *There always exists an optimal offline scheduler π^* such that $v_1^* = v_2^* = \cdots = v_N^*$.*

This result is both important and interesting. Before we offer a formal proof, one can give a simple (but intuitive) argument to show this is true based on contradiction. Suppose v_i^*'s

were not equal. Then one could always re-allocate the transmission resources (RBs) from the source(s) with lower v_i^*'s to those source(s) with higher v_i^*'s and the objective value $v_{\max}$ will decrease.

We now give a formal proof to Lemma 6.1.

Proof. Suppose π' is an offline optimal scheduler with outdated proportions $v_1', v_2', \cdots, v_N'$. We have two cases.

Case 1. If $v_1' = v_2' = \cdots = v_N'$, we just let $\pi^* = \pi'$ and we are done.

Case 2. Otherwise, suppose v_k' contains the largest value among $v_1', v_2', \cdots, v_N'$, i.e., $v_k' = \max\{v_1', v_2', \cdots, v_N'\}$. Then for any source i with $v_i' < v_k'$, we can remove a group of RBs that carry one sample from source i (these RBs may span multiple TTIs) without allocating these RBs to any other source nodes. After removing these RBs, the BS will receive one less sample from source i, and the outdated proportion for source i at the BS will increase. We continue to perform this RB removal on source i until its outdated proportion v_i' increases to v_k'. By doing this procedure for all source i's with $v_i' < v_k'$, we can get a new scheduler under which the outdated proportions for all sources are the same as v_k'. We call this newly constructed scheduler π^*. It is clear that π^* is also optimal (since v_k' remains the optimal objective value). This completes our proof. □

Now we give a definition of uniformly fair schedulers.

Definition 6.1. *A scheduler π is uniformly fair if $v_1 = v_2 = \cdots = v_N$.*

Lemma 6.1 says there exists an offline optimal scheduler π^* that is uniformly fair. Although an offline optimal scheduler cannot be designed without knowledge of future channel

information, we will exploit this uniformly fair property as a guideline when we design Aequitas.

6.4.2 Main Ideas

In this section we outline the main ideas of Aequitas. Aequitas is a priority-based scheduler. Within a TTI, for each iteration, Aequitas computes the priorities for all eligible source nodes and selects the source node with the highest priority for resource (RB) allocation.

The most important question to address is how to define and calculate priority in scheduling. Aequitas addresses this question with the following considerations:

- First, Aequitas aims to achieve uniform fairness, i.e., $v_1 = v_2 = \cdots = v_N$ as t increases. Therefore, the source nodes with higher outdated proportion in the past will have higher priorities.

- Recall that $A_i(t)$ will keep increasing until a sample is received in its entirety at the BS. So Aequitas will try to make this duration (i.e., $\tau_i(n)$) as small as possible. As a consequence, the source node with an unfinished sample carried from the previous TTI will have the highest priority.

- For a source i that already has $A_i(t) \geq d_i$ (i.e., either already outdated or about to be outdated), it should be assigned a higher priority.

- When transmitting a sample from a source, Aequitas tries to use a high MCS to obtain a high date rate per RB. Therefore, the source node that can use a higher MCS (based on its channel conditions) will have a higher priority.

6.4.3 Design Details

Within a TTI, for each iteration, Aequitas computes a priority metric, denoted by w_i, for each eligible source node i. Then it selects the source node with the largest w_i and then chooses an MCS and allocates RBs to it. After each iteration, the remaining available RBs will be fewer and Aequitas recomputes w_i's for all remaining eligible source nodes for the next iteration. To design w_i, we need to introduce some notations. When there is no ambiguity, we omit to include "t" in these notations in the rest of this section.

Notations Denote u_i as a binary indicator for whether or not a sample from source node $i = 1, 2, \cdots, N$ is being transmitted in the TTI t (1 for yes and 0 for no). We have

$$u_i = [\big(\sum_{b=1}^{B} x_i^b(t)\big) > 0]. \tag{6.17}$$

Recall that "$[\cdot]$" is Iverson bracket, returning 1 if the inside statement is true and 0 otherwise. Denote g^b ($b = 1, 2, \cdots B$) as a binary indicator for whether RB b is allocated to some source node (1 for yes, 0 for no). We have

$$g^b = [\big(\sum_{i=1}^{N} x_i^b(t)\big) > 0]. \tag{6.18}$$

Before the first iteration of Aequitas, we have $x_i^b(t) = 0$ for all i's and b's, so we have $u_i = 0$ and $g^b = 0$ for all i's and b's initially.

Denote s_i as a binary indicator (1 for yes, 0 for no) for whether source node i has a sample that started its transmission in previous TTI but still has an unfinished part to be transmitted in the current TTI t. Denote L^{R} as the number of bits in the remaining unfinished part for the source with $s_i = 1$. Since Aequitas aims to minimize transmission duration $\tau_i(n)$, this unfinished sample shall have the highest priority in the current TTI. Further, the number of unfinished samples at the end of each TTI is at most one, i.e.,

$\sum_{i=1}^{N} s_i \leq 1$ for every TTI.

Denote l_i^m as the minimum required number of RBs to transmit a sample from source node i under MCS m. We have:

$$l_i^m = \begin{cases} \lceil \frac{L_i}{c^m} \rceil & \text{if } \ s_i = 0, \\ \lceil \frac{L^{\mathrm{R}}}{c^m} \rceil & \text{if } \ s_i = 1. \end{cases} \tag{6.19}$$

where "$\lceil \cdot \rceil$" denotes the ceiling function. Therefore, if there are at least l_i^m un-allocated RBs with $q_i^b(t) \geq m$, then a sample from source i can be transmitted in its entirety under MCS m. Denote n_i^m as the number of un-allocated RBs with $q_i^b(t) \geq m$. Then

$$n_i^m = \sum_{b=1}^{B} \big((1 - g^b) \cdot [q_i^b(t) \geq m] \big). \tag{6.20}$$

Clearly, if $n_i^m \geq l_i^m$, then a sample from source i can be transmitted in full under MCS m.

Denote z_i as a binary indicator for whether a sample from source i can be transmitted under some MCSs in full in this TTI (1 for yes, 0 for no). Clearly, if there is at least one m making $n_i^m \geq l_i^m$, then $z_i = 1$. We have

$$z_i = \big[(\sum_{m=1}^{M} [n_i^m \geq l_i^m]) > 0 \big]. \tag{6.21}$$

In Aequitas, source nodes with $z_i = 1$ have a higher priority than those source nodes with $z_i = 0$.

When a source node is chosen for transmission, we need to select an MCS for it. Denote m_i^* as the optimal MCS level that source i should select.

1) If $z_i = 1$, i.e., the sample from source i can be transmitted to the BS in full under some MCSs, then Aequitas will selects an MCS as high as possible (as long as it can be transmitted

in full). That is,

$$m_i^* = \max_m \{m : n_i^m \geq l_i^m\}, \text{ if } z_i = 1. \tag{6.22}$$

2) If $z_i = 0$, we cannot finish transmitting the sample from source i under any MCS level. Per our design, Aequitas limits the number of unfinished samples in each TTI to at most one. Aequitas will select an MCS that can transmit the maximum amount of information (in bits) with the remaining RBs. Therefore, for source i with $z_i = 0$, m_i^* is given by

$$m_i^* = \arg\max_m \{c^m \cdot n_i^m\}, \text{ if } z_i = 0. \tag{6.23}$$

Denote λ_i as the amount of information that source i can transmit under MCS m_i^*, i.e.,

$$\lambda_i = c^{m_i^*} \cdot n_i^{m_i^*}. \tag{6.24}$$

Clearly, we have $\lambda_i < L_i$ if $z_i = 0$.

Priority Metric Now we are ready to present the priority metric w_i. Based on the design ideas discussed in the last section, w_i depends on s_i, $A_i(t)$, d_i, m_i^*, λ_i, and v_i. We propose the following construct for w_i:

$$w_i = f_1(s_i) \cdot f_2\big(A_i(t), d_i\big) \cdot f_3(z_i, m_i^*) \cdot e_i(t). \tag{6.25}$$

Some discussions are in order.

- $f_1(s_i)$: This component is about s_i, which indicates whether source node i has an unfinished sample from the previous TTI. In our design, the source node with $s_i = 1$ will have the highest priority (i.e., 1). So we define:

  $$f_1(s_i) = (1 - s_i) \cdot C_1 + 1\ , \tag{6.26}$$

  where C_1 is a constant and $C_1 \gg 1$.

- $f_2\big(A_i(t), d_i\big)$: This component is concerned with the current AoI $A_i(t)$ and the AoI deadline d_i. Those source nodes with $A_i(t) \geq d_i$ should have a greater $f_2\big(A_i(t), d_i\big)$ than those source nodes with $A_i(t) < d_i$. When $A_i(t) < d_i$, the closer the $A_i(t)$ is to its d_i, the higher the $f_2\big(A_i(t), d_i\big)$ should be. When $A_i(t) \geq d_i$, from the objective's perspective, it doesn't matter how much $A_i(t)$ is greater than d_i. So we define:

$$f_2\big(A_i(t), d_i\big) = \begin{cases} \frac{A_i(t)}{d_i} & \text{if } \ A_i(t) < d_i, \\ C_2 & \text{if } \ A_i(t) \geq d_i, \end{cases} \tag{6.27}$$

  where C_2 is a constant and $C_2 > 1$.

- $f_3(z_i, m_i^*)$: This component is about z_i, whether a sample from source i can be transmitted in full, and its MCS level m_i^*. Since we prefer those samples that can be transmitted in full, we will make those sources with $z_i = 1$ have a higher priority than those sources with $z_i = 0$. Among the source nodes with $z_i = 1$, we prioritize those with higher date rates per RB, i.e., $c^{m_i^*}$. Among the nodes with $z_i = 0$, we prioritize those with a larger proportion that can be transmitted in this TTI, i.e., λ_i / L_i. So we define:

$$f_3(z_i, m_i^*) = \begin{cases} c^{m_i^*} & \text{if } \ z_i = 1, \\ \frac{c^1}{C_3} \cdot \frac{\lambda_i}{L_i} & \text{if } \ z_i = 0, \end{cases} \tag{6.28}$$

  where C_3 is constant and $C_3 > 1$.

- $e_i(t)$: This component serves as an equalizer to ensure $v_1 = v_2 = \cdots = v_N$ as t increases. Denote $\bar{v}_i(t)$ as the outdated proportion for source i till TTI t, i.e.,

$$\bar{v}_i(t) = \frac{1}{t} \sum_{\tau=1}^{t} [A_i(\tau) > d_i]. \tag{6.29}$$

Clearly, $v_i = \lim_{t\to\infty} \bar{v}_i(t)$. We define $e_i(t)$ as:

$$e_i(t) = \left(1 - \gamma + \gamma \cdot \frac{N\bar{v}_i(t-1)}{\sum_{j=1}^{N} \bar{v}_j(t-1)}\right) \cdot e_i(t-1), \quad \text{for } t \geq 2, \tag{6.30}$$

and $e_i(1) = 1$. Here γ is a constant and $0 < \gamma \ll 1$. We can see when $\bar{v}_i(t-1)$ is higher than the average outdated proportion of other sources, it will have a high priority $e_i(t)$ in the next TTI, and vice versa.

For parameter settings, since we want to ensure the unfinished sample from the previous TTI will have the highest priority, the component $f_1(s_i)$ should dominate over other component when $s_i = 1$. To ensure this is the case, we should let $C_1 \gg C_2$ and $C_1 \gg C_3$. After these two conditions are satisfied, the settings of C_2 and C_3 are rather open and do not affect the performance of Aequitas very much (see numerical results in Section 6.5).

Complete Algorithm The complete Aequitas algorithm is summarized (in pseudocode) in Algorithm 6.2. In each TTI t, Aequitas iteratively computes w_i for all eligible i's (line 2). Note that since $C_1 \gg C_2$ and $C_1 \gg C_3$, the source node with $s_i = 1$ (with incomplete sample transmission from the last TTI) will always have the largest w_i. Then Aequitas selects a source node with the largest w_i (line 3), choose the optimal MCS for it (line 4), and allocate RBs to it for transmission (line 6). After each iteration (line 2–7), the remaining available RBs will be fewer. Finally, in the last iteration, the source node with the largest w_i cannot transmit its sample in its entirety (i.e., $z_k = 0$), and Aequitas will allocate all the remaining RBs to this source node (line 9).

Clearly, Algorithm 6.2 is a polynomial-time algorithm. For each iteration, the most time-consuming component is computation of w_i for all i's (line 2 in Algorithm 6.2), which requires computing NM different n_i^m (for all i's and all m's). The time complexity of computing one n_i^m (see (6.20)) is $O(B)$, so the time complexity of each iteration is $O(BNM)$. There are

Algorithm 6.2 Aequitas

Input: $A_i(t)$, d_i, $e_i(t)$ and s_i for all i's; $q_i^b(t)$ for all i's and $b's$. Parameters: C_1, C_2, C_3, and γ.

Output: $x_i^b(t)$'s and $y_i^m(t)$'s.

1: Set $x_i^b(t) = 0$, $y_i^m(t) = 0$, $u_i = 0$ and $g^b = 0$ for all i's, b's and m's.

2: For all i's with $u_i = 0$, compute z_i by (6.21), m_i^* by (6.22) or (6.23), and w_i by (6.25).

3: Set $k = \arg\max_i w_i$.

4: Set $u_k = 1$, $m = m_k^*$, and $y_k^m(t) = 1$.

5: **if** $z_k = 1$ **then**

6: Choose any n_k^m different b's with $g^b = 0$ and $q_k^b(t) \geq m$, and set $x_k^b(t) = 1$ and $g^b = 1$ for these b's.

7: **go to** line 2

8: **else**

9: Set $x_k^b(t) = 1$ for all b's with $g^b = 0$.

10: **end if**

11: **return** $x_i^b(t)$'s and $y_i^m(t)$'s.

at most B iterations in one TTI, so the time complexity of Algorithm 6.2 in each TTI is $O(B^2NM)$.

Real-Time Implementation Although Aequitas' time complexity is polynomial, as we will see in Section 6.5, its average running time cannot meet 5G's timing requirement. For example, when implemented on an Intel Xeon E5-2698 v4 CPU, the average running time of this algorithm is 23.48 ms when $N = 100$, $B = 100$ and $M = 29$. This is far beyond the real-time requirement in 5G (with 1 ms being the largest allowed time interval). Therefore, merely being a polynomial time algorithm is insufficient to meet 5G real-time requirement, which is measured by wall-clock time (rather than $O(\cdot)$ analysis).

We propose to exploit parallel computation to speed up Aequitas' execution time. Recall that the computation of NM different n_i^m's (when computing w_i's in line 2, Algorithm 6.2) is the most time-costing task in Aequitas. We observe that computations of these NM different n_i^m's are independent from each other. If we can compute them in parallel (e.g., using GPU), the total computation time can be reduced dramatically. In our implementation, we employ a commercial off-the-shelf (COTS) NVIDIA Tesla V100 GPU and CUDA programming platform to computer NM different n_i^m's in parallel. In Section 6.5, we will show that the average running time under the same settings is reduced to 0.31 ms, which can meet the scheduling time requirement in 5G.

6.5 Performance Evaluation

In this section we conduct experiments and evaluate the performance of our Aequitas implementation.

Table 6.2: AoI deadlines and sample sizes for different types of source nodes.

Type	d_i (TTIs)	L_i (bits)	Type	d_i (TTIs)	L_i (bits)
1	3	6400	6	5	9800
2	5	5400	7	9	4500
3	4	7800	8	4	3900
4	3	4200	9	6	6600
5	6	5600	10	3	5800

6.5.1 Experiment Setup

We implement Aequitas on an NVIDIA DGX Station with an Intel Xeon E5-2698 v4 CPU (2.20 GHz) and an NVIDIA Tesla V100 GPU (32 GB memory). We use Visual Studio 2019 and CUDA 10.2 for programming.

We assume there are 10 different types of source nodes, with AoI deadline d_i's and sample size L_i's given in Table 6.2. For each source node, we assume a Rician fading channel with factor $K = 1$ (i.e., the power of LOS equals to the power of NLOS) and randomly generate an average SNR between 10 and 20dB. We assume there is no correlation in time and frequency, i.e., for each TTI, we generate channel conditions independently and so as for each RB.

In Algorithm 6.2, we set $C_1 = 100,000$, $C_2 = 10$, $C_3 = 10$, and $\gamma = 0.001$. For each network setting, we run Aequitas over 100,000 TTIs. In Algorithm 6.1, we set $\alpha = 10$, $\beta = 0.01$, and $T = 1,000$.

6.5.2 A Case Study

We first consider a network with $N = 100$ source nodes (10 nodes for each type in Table 6.2) and $B = 100$ RBs. Define

$$\bar{v}_{\max}(t) = \max\{\bar{v}_1(t), \bar{v}_1(t), \cdots, \bar{v}_N(t)\}. \tag{6.31}$$

Table 6.3: Aequitas under different C_2 and C_3

(C_2, C_3)	$(10, 10)$	$(10, 100)$	$(100, 10)$	$(100, 100)$
$\bar{v}_{\max}(100,000)$	0.422	0.421	0.422	0.423
Average running time (ms)	0.310	0.311	0.316	0.317

Clearly, we have $\lim_{t\to\infty} \bar{v}_{\max}(t) = v_{\max}$.

Fig. 6.2(a) shows the evolution of $\bar{v}_i(t)$ for all 100 nodes under Aequitas over 100,000 TTIs. As we can see, after a warm-up period, all 100 $\bar{v}_i(t)$'s converge roughly to the same value. This shows that the uniform fairness property for an offline optimal scheduler is also achieved by Aequitas.

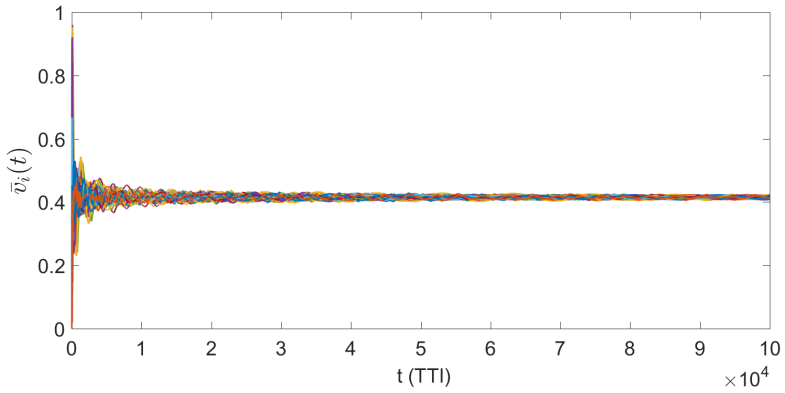
Fig. 6.2(b) shows the evolution of $\bar{v}_{\max}(t)$ under Aequitas over 100,000 TTIs. The lower bound found by Algorithm 6.1 is also shown in the figure. We can see that $\bar{v}_{\max}(t)$ by Aequitas is close to the lower bound. When $t = 100,000$, we have $\bar{v}_{\max}(100,000) = 0.422$ while the lower bound is 0.367.

Fig. 6.2(c) shows the running time of Aequitas with and without parallel implementation over 100,000 TTIs. Under parallel implementation, the average running time of Aequitas is 0.31 ms, which is under 1 ms (as required by 5G NR); while the average running time is 23.48 ms when parallel implementation is not used.

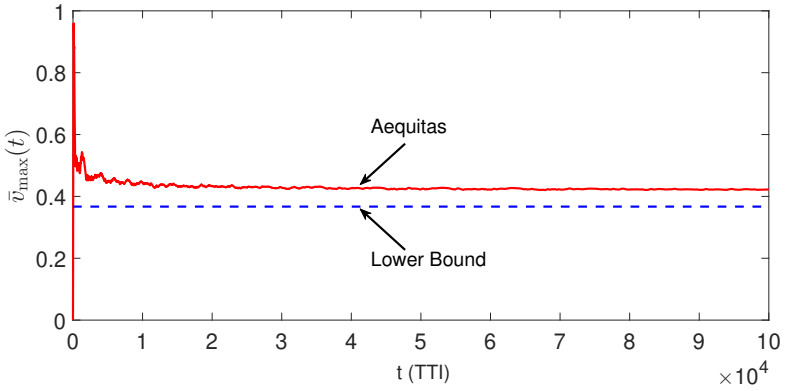
6.5.3 Varying Parameters

In this section we examine the behavior of Aequitas under varying system parameters.

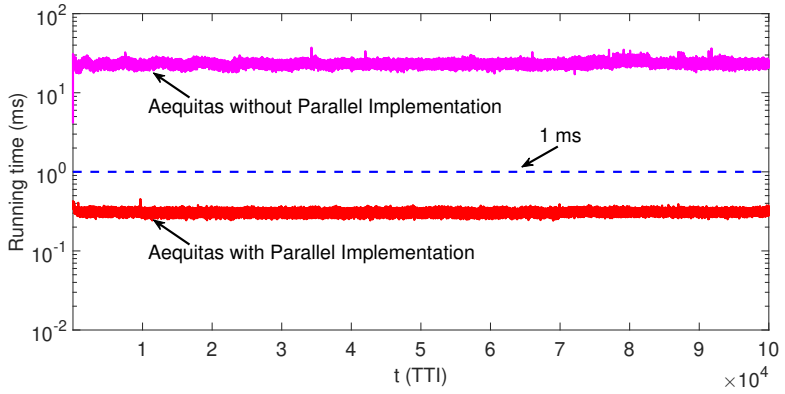
Varying Algorithm Parameters We first show the impact of algorithm parameter settings on Aequitas. For the network setting used in Section 6.5.2, we vary the values of C_2 and C_3 in Algorithm 6.2. Table 6.3 shows the performance of Aequitas under different values of C_2 and C_3. Recall that we set $C_1 = 100,000$ so we have $C_1 \gg C_2$ and $C_1 \gg C_3$. As we can see,



(a) Evolution of $\bar{v}_i(t)$ for all 100 nodes.

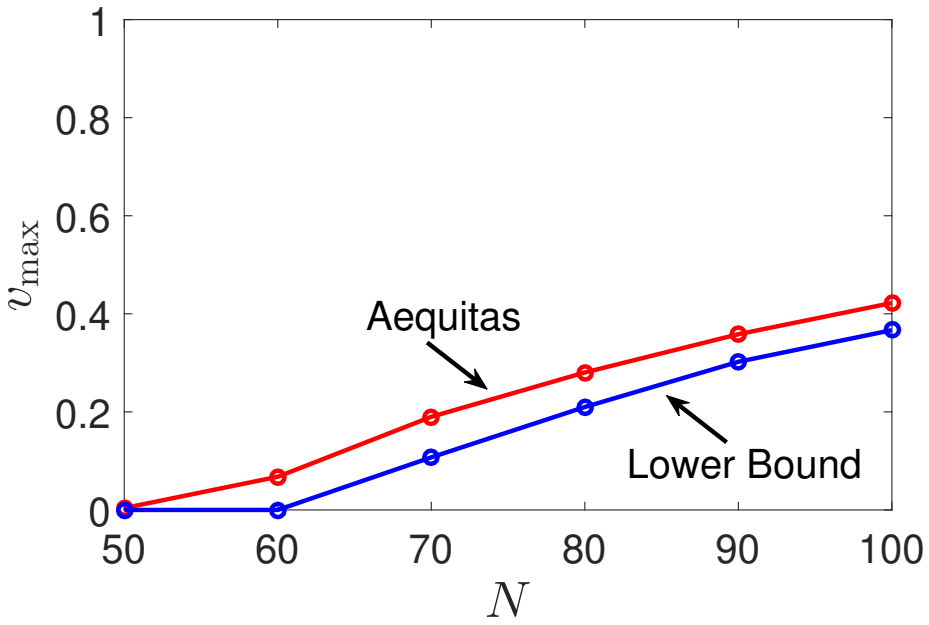


(b) [Evolution of $\bar{v}_{\max}(t)$.

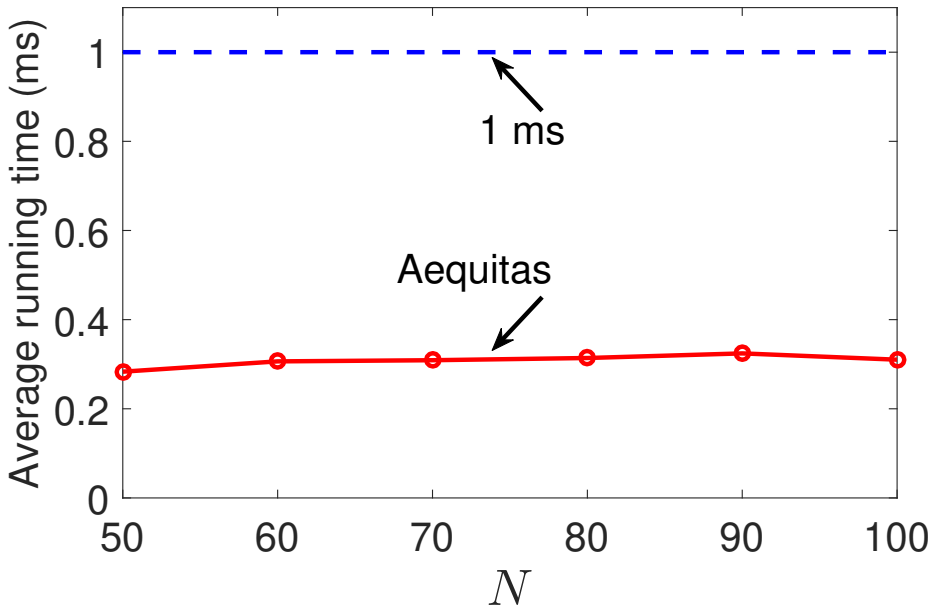


(c) Running time of Aequitas with and without parallel implementation

Figure 6.2: Results for a case study with $N = 100$, $B = 100$.



(a) $v_{\max}$



(b) Average running time

Figure 6.3: Aequitas under varying N when $B = 100$

the settings of C_2 and C_3 do not have much impact on Aequitas, which is what we would hope for.

Varying Number of Nodes We investigate the performance of Aequitas under varying N. We set $B = 100$ RBs. Other settings and parameters are described in Section 6.5.1.

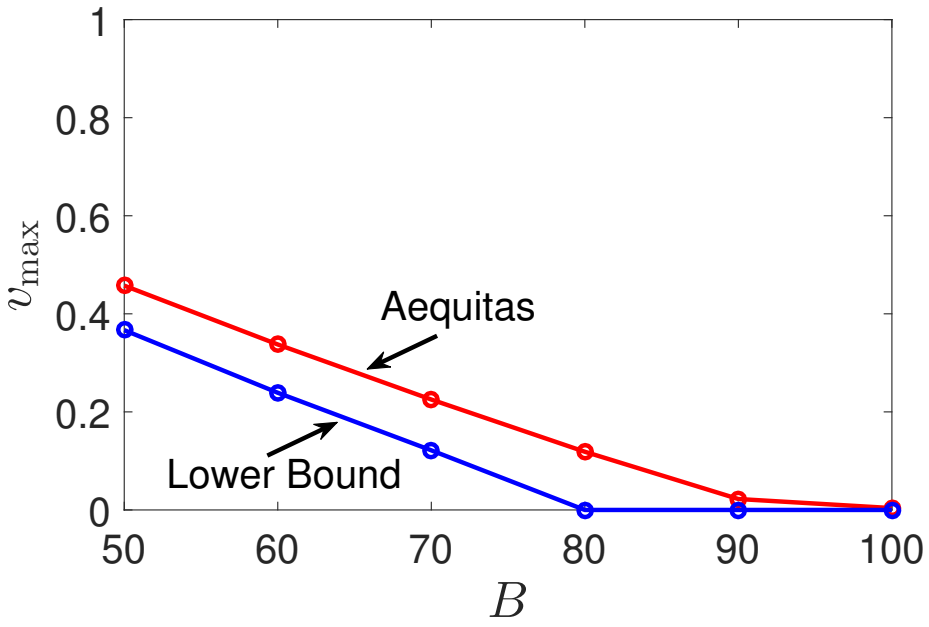
Fig. 6.3(a) shows the objective $v_{\max}$ under Aequitas for $N = 50, 60, 70, 80, 90$ and 100. Here $v_{\max}$ refers to $\bar{v}_{\max}(100,000)$ as we run $100,000$ time slots. The lower bound found by Algorithm 6.1 is also shown in the figure. As we can see, when N increases $v_{\max}$ under Aequitas also increases, which is intuitive. For all settings of N, $v_{\max}$ under Aequitas is close to the lower bound.

Fig. 6.3(b) shows the average running time of Aequitas (with parallel implementation) for $N = 50, 60, 70, 80, 90$ and 100. As we can see, the average running time is always under 1 ms and remains nearly flat when $N = 50$ to 100. Specifically, when $N = 50$ the average running time is 0.283 ms while when $N = 100$ it is 0.310 ms. This is because the number of source nodes will not affect running time much as long as we have enough GPU cores to accommodate them.

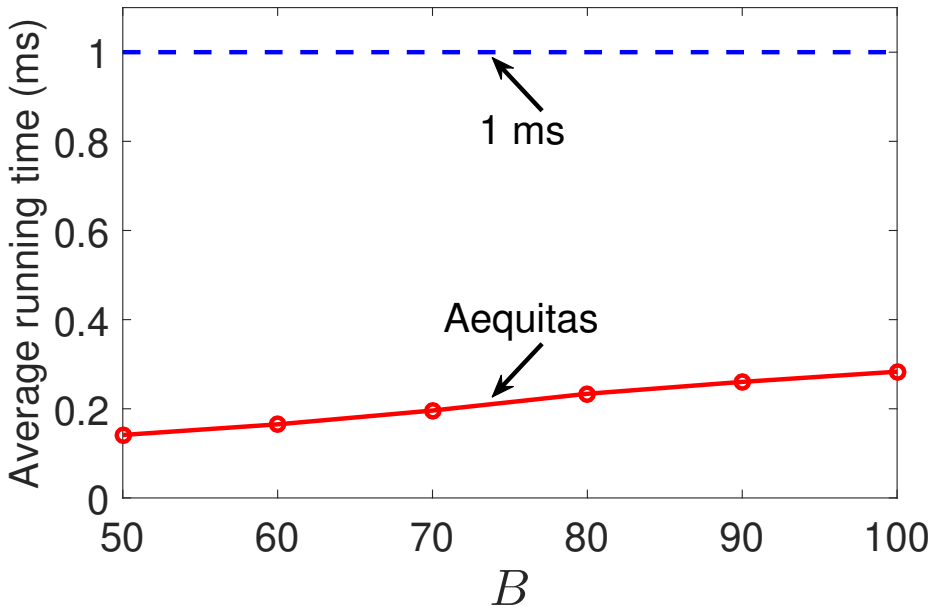
Varying Number of RBs We study the performance of Aequitas under varying B. We set $N = 50$ source nodes. Other settings and parameters are described in Section 6.5.1.

Fig. 6.4(a) shows the objective $v_{\max}$ under Aequitas for $B = 50, 60, 70, 80, 90$ and 100, as well as the lower bound found by Algorithm 6.1. As we can see, when B increases $v_{\max}$ under Aequitas decreases, which is intuitive. For all settings of B, $v_{\max}$ under Aequitas is close to the lower bound.

Fig. 6.4(b) shows the average running time of Aequitas (with parallel implementation) for $B = 50, 60, 70, 80, 90$ and 100. As we can see, the average running time is always under 1 ms and increases with B. Specifically, when $B = 50$ the average running time is 0.141 ms



(a) $v_{\max}$



(b) Average running time

Figure 6.4: Aequitas under varying B when $N = 50$

while when $B = 100$ it is 0.283 ms.

6.6 Chapter Summary

In this chapter, we investigated an online 5G scheduling problem with the goal of minimizing the violation rate of soft AoI deadlines among a group of source nodes. We first developed a computation procedure to find a lower bound for the objective, which can be used as a performance benchmark. Then we derived a uniform fairness property associated with an offline optimal scheduler, which indicates that each source may have the same proportion of outdated information when time becomes large. Using this property, we developed Aequitas—an online 5G scheduler that performs RB allocation and MCS selection in each TTI. By exploiting the intrinsic parallelism in Aequitas, we implemented it on a COTS GPU platform. Through extensive experimental study, we found that that can achieve excellent performance in finding objective value while meeting the 5G timing requirement.

Chapter 7

Eywa: A General Framework for Scheduler Design and Its Applications to A Family of AoI-Related Problems

7.1 Introduction

Since the AoI concept was introduced, there has been a growing body of research on designing scheduling algorithms to optimize AoI-related objectives (see, e.g., [9, 10, 14, 27, 32, 43, 44, 45, 46, 79, 81, 84, 88, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105]). The optimization objectives in these works include: minimize average/peak AoI (e.g., [10, 32, 43, 44, 45, 79, 88, 93, 94, 95, 97, 103, 104]), to minimize transmission bandwidth under AoI constraints (e.g., [102]), to determine the existence of feasible schedulers to satisfy AoI constraints (e.g, [46, 101]), and to optimize variants of the AoI metric (e.g., [14, 98, 99]), among others. Many different network settings (scenarios) have been considered, including IoT data collection (e.g., [9, 45, 46, 84, 88, 95, 99, 100, 101, 102, 103, 104]), queuing systems (e.g., [43, 44]), cache systems (e.g., [14, 92]), and so on.

For each of these AoI problems, typically a *custom-designed* scheduler was proposed. With so many custom-designed AoI schedulers around, it is important to explore the intrinsic connections among these problems and proposed solutions and extract some important properties for fundamental understanding. More important, it is desirable to develop a *general framework* that can be applied to design a wide range of schedulers.[1]

[1]The vision of this general framework for AoI research came from co-author Y.T. Hou's early work on

In this chapter, we make a first attempt to develop such as a general framework—code-named *Eywa*[2]. Eywa considers a data collection network setting (Fig. 1.1) and can be applied to construct high-performance schedulers for a family of AoI-related optimization and decision problems in this network setting. The problems that we have identified so far include: to minimize sum of AoIs, to minimize bandwidth under AoI constraints, and to determine the existence of feasible schedulers to satisfy AoI constraints. Other problems to which Eywa may be applied will be addressed in future work.

The design ideas of Eywas are inspired by the design ideas in Chapter 5: (i) *Almost Uniform Scheduler* (AUS)—a unique type of *cyclic* schedulers where the transmission pattern of each source node is at least *nearly uniform*, and (ii) *step-down* rate vectors, which can be used to construct AUS-based schedulers efficiently.

The general framework of Eywa consists of three steps. The first step is to transform the original (AoI-based) objective function and/or the constraints to a *rate-based* objective function and *rate-based* constraints. The second step is to find the optimal step-down rate vector by solving an optimization problem with the rate-based objective function and constraints. The third step is to construct an AUS-based scheduler using the optimal step-down rate vector.

We put the Eywa framework in action by solving a family of AoI-related problems in the literature: to minimize sum of AoIs [103, 104, 105], to minimize bandwidth under AoI constraints [102], and to determine the existence of feasible schedulers to satisfy AoI constraints (studied in Chapters 4 and 5). We show that for each problem, Eywa can either offer a stronger performance guarantee than state-of-the-art algorithms, or provide new/general results that are not available in the literature.

Internet QoS scheduling [106] and asymptotic capacity analysis in wireless networks [107, 108].

[2]Eywa is the guiding force of life and deity of Pandora and the Na'vi (from movie Avatar).

The remainder of this chapter is organized as follows. In Section 7.2, we present the general framework of Eywa and elaborate its three steps in details. In Sections 7.3 to 7.5, we apply the Eywa framework to solve several problems. Specifically, in Section 7.3, we apply Eywa to find a scheduler that minimizes the weighed sum of AoIs. In Section 7.4, we apply Eywa to find schedulers that minimize bandwidth requirement under peak and average AoI constraints, respectively. In Section 7.5, we apply Eywa to solve a decision problem, i.e., to determine the existence of feasible schedulers to satisfy hard/soft AoI deadlines. Section 7.6 summarizes this chapter.

7.2 Eywa: A General Approach

In this section, we present a general approach—code-named Eywa—to construct *high-performance* cyclic schedulers for many AoI optimization problems. By "high-performance" we mean some strong theoretical guarantees, either w.r.t the optimal objective value or some schedulability conditions. The types of AoI optimization problems that Eywa may be applied, include (but not necessarily limited to):

1. **Min-Sum:** Minimize (weighted) sum of AoI;
2. **Min-BW:** Minimize bandwidth requirement under AoI constraints;
3. **Decision Problems:** Determine the existence of feasible schedulers to satisfy AoI constraints.

A **side benefit** of Eywa is that its schedulers follow a "fixed" cyclic transmission pattern and do not require the BS to convey its scheduling decisions in every time slot. This will *lower communication overhead* in the control channel as well as *reduce delay* (the time to send the scheduling decision from the BS to each source).

Table 7.1: Notations

Symbol	Definition
	General Notations
$A_i(t)$	AoI for sample from source node i at the BS at time t
W	Transmission bandwidth
c	Cycle length for a cyclic scheduler
d_i	AoI deadline for source node i
N	Number of source nodes in the network
$\pi_i(t)$	Scheduling decision for source i at time t
p_i	Packet loss rate for source i
$p_{\max}$	The largest one among all p_i's
$q_i(t)$	Channel indicator for source i at time t
r_i	Average transmission rate w.r.t. source node i
$U_i(t)$	Generation time of the most recent sample at the BS from source node i at time t
	Notations for Min-Sum
$\bar{A}_i$	Long-term average AoI for source i
$\bar{A}$	Weighted sum of all $\bar{A}_i$'s
$\bar{A}_{\text{LB}}$	A lower bound for $\bar{A}$
$\bar{A}^*$	Minimum $\bar{A}$ for all schedulers
$\boldsymbol{r}^{\text{LB}}$	Optimal solution to OPT-LB
w_i	Weight for source i
	Notations for Min-BW
α_i	Threshold for average AoI for source i
$\boldsymbol{\alpha}$	A vector denoting $[\alpha_1, \alpha_2, \cdots, \alpha_N]$
W^*	Minimum W for all feasible schedulers
d_i	Threshold for peak AoI for source i
$\boldsymbol{d}$	A vector denoting $[d_1, d_2, \cdots, d_N]$
	Notations for Decision Problems
d_i	(Hard or soft) AoI deadline for source i
$\boldsymbol{d}$	A vector denoting $[d_1, d_2, \cdots, d_N]$
ϵ_i	Violation tolerance rate for source node i's AoI deadline
$\boldsymbol{\epsilon}$	A vector denoting $[\epsilon_1, \epsilon_2, \cdots, \epsilon_N]$
$l(\boldsymbol{d})$	System load for $\boldsymbol{d}$ under hard AoI deadlines
$l(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$	System load for $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ under soft AoI deadlines
p_i	Packet loss rate for source node i
$\boldsymbol{p}$	A vector denoting $[p_1, p_2, \cdots, p_N]$

7.2.1 System Model

Consider a data collection network where N source nodes collect information from the environment and send it to a base station (BS) through a shared wireless channel, as shown in Fig. 1.1. Tables 7.1 lists key notations used in this chapter.

We assume time is slotted, and each source node takes a sample at the beginning of each time slot. If scheduled for transmission, a source will send its freshest sample (collected at the beginning of the time slot) to the BS.

Denote W as the transmission bandwidth of the wireless network. The wireless channel allows at most W samples to be transmitted in each time slot, and the BS scheduler needs to decide which samples will be chosen for transmission. For scheduler π, we denote $\pi_i(t) \in \{0, 1\}$ as the scheduling decision at time t w.r.t. each source i ($i = 1, 2, \cdots, N$), i.e.,

$$\pi_i(t) = \begin{cases} 1, & \text{if source } i \text{ is scheduled to transmit at time } t, \\ 0, & \text{otherwise.} \end{cases} \tag{7.1}$$

For any scheduler π, at any $t = 0, 1, 2, \cdots$ we have

$$\sum_{i=1}^{N} \pi_i(t) \leq W. \tag{7.2}$$

The long-term average *transmission rate*, r_i for source i, can be defined as:

$$r_i = \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \pi_i(t). \tag{7.3}$$

Then we have

$$r_i \leq 1, \quad \text{for } i = 1, 2, \cdots, N, \tag{7.4a}$$

$$\sum_{i=1}^{N} r_i \leq W. \tag{7.4b}$$

We consider unreliable wireless channel in our model. We assume there is a (fixed) packet loss rate (due to transmission failure) for each source node i, which we denote as p_i. With the presence of packet loss rate, the transmission from a source may not always be successful. Denote $q_i(t)$ as a binary indicator for whether or not the transmission from source i at time t (if scheduled) is successful, i.e.,

$$q_i(t) = \begin{cases} 1, & \text{if transmission from } i \text{ is successful at } t \\ & \text{(should } i \text{ be scheduled for transmission),} \\ 0, & \text{otherwise.} \end{cases}$$

Then we have

$$\mathbb{P}\{q_i(t) = 0\} = p_i,$$
$$\mathbb{P}\{q_i(t) = 1\} = 1 - p_i.$$

The AoI of source i (at the BS) at time t, denoted by $A_i(t)$, is defined as the elapsed time between the current time t and $U_i(t)$— the sample's generation time at its source, i.e.,

$$A_i(t) = t - U_i(t). \tag{7.5}$$

If the BS does not receive a sample from source i in time t (i.e., $\pi_i(t) = 0$ or $q_i(t) = 0$), then by definition of AoI, at time $(t+1)$ we have $A_i(t+1) = A_i(t) + 1$. Otherwise, the AoI will drop to 1. We have,

$$A_i(t+1) = \begin{cases} 1, & \text{if } \pi_i(t) \cdot q_i(t) = 1, \\ A_i(t) + 1, & \text{otherwise.} \end{cases} \tag{7.6}$$

7.2.2 Main Idea: Almost Uniform Scheduler

Eywa focuses on a special class of cyclic schedulers,[3] which we call *Almost Uniform Schedulers (AUSs)*. The concept of AUS first appeared in Chapter 5 for the special case of unit channel bandwidth (i.e., $W = 1$). The AUS that we present here is general and can be applied to arbitrary channel bandwidth ($W \geq 1$).

Denote τ_i^k as the time slot when the k-th sample ($k = 1, 2, \cdots$) from source i is scheduled for transmission, i.e., $\pi_i(\tau_i^k) = 1$. Denote T_i^k as the time interval between the k-th and the $(k+1)$-th transmission for source i, i.e.,

$$T_i^k = \tau_i^{k+1} - \tau_i^k. \tag{7.7}$$

Then, AUS is defined as the following.

Definition 7.1. *A cyclic scheduler π is an AUS if for each source i there exists an integer b_i such that we have either $T_i^k = b_i$ or $T_i^k = b_i + 1$ for any $k \geq 1$.*

For example, consider four sources A, B, C, and D. Denote "$(\cdots)$" as scheduling decisions for one cycle. Then when $W = 1$, the scheduler

$$(\text{ABABCABDABC})$$

is an AUS with $b_A = 2$, $b_B = 2$, $b_C = 5$, and $b_D = 11$. Note that for source C, $b_C = 5$ because the interval length between the last transmission in this cycle and the first transmission in the next cycle is 5.

As the second example, when $W = 2$ [where for each time slot, 2 samples from 2 source

[3] A cyclic scheduler repeats the same scheduling decisions every c time slots (where c is the cycle length). More formally, a scheduler π is cyclic if there exists an integer c such that $\pi_i(t) = \pi_i(t + c)$ for all i's and $t \geq 0$. Cyclic schedulers do not require the BS to convey its scheduling decisions in every time slot, which will i) lower communication overhead in the control channel, and ii) reduce the delay for sending the scheduling decision to the sources.

nodes (a column in the scheduling bracket) are transmitted to the BS], the scheduler

$$\begin{pmatrix} \text{ACBACBBADBA} \\ \text{BADBAACBACB} \end{pmatrix}$$

is an AUS with $b_A = 1$, $b_B = 1$, $b_C = 2$, and $b_D = 5$.

We also identify a special case of AUS, which we call *Exact Uniform Scheduler* (EUS). We say an AUS is an EUS if for each source i, we have $T_i^k = b_i$ for all $k \geq 1$. That is, each source i is periodically scheduled for transmission with an *exact* period of b_i.

7.2.3 Eywa: Complete Procedure

Eywa is designed to solve a family of AoI optimization problems and decision problems, including Min-Sum, Min-BW, and the decision problems that we listed in the beginning of Section 7.2. Although these problems have different objectives and constraints, they all conform to the system model as described in Section 7.2.1. As such, we have discovered the following framework (Eywa) that can be used to custom design a high-performance AUS-based scheduler for each of these problems. Our proposed Eywa framework is given in Algorithm 7.1.

Eywa (Algorithm 7.1) consists of three steps. In the rest of this section, we elaborate the details in each step.

Step 1: Transform objective function and constriants The essence of Eywa is to design an AUS-based scheduler based on each source i's transmission rate r_i. We start with the objective function. Since the original objective function may not be rate-based (e.g., AoI-based), the first step of Eywa is to transform the original objective function to a *rate-based* objective function.

Algorithm 7.1 A General Framework of Eywa

1: **Transform objective function and constraints:** If the original objective function and/or the constraints are not rate-based (e.g., AoI-based), then transform them to a rate-based objective function and rate-based constraints.

2: **Find optimal transmission rates:** Find the optimal transmission rates by solving an optimization problem (OPT-SD) with the rate-based objective function.

3: **Construct an AUS-based scheduler:** Construct an AUS-based scheduler using the optimal transmission rates.

We denote $J_o(\cdot)$ as the original objective function that we want to minimize.[4] For example, $J_o(\cdot)$ could be a function of AoIs, i.e., $J_o(\cdot) = J_o\big(A_1(t),\, A_2(t), \cdots, A_N(t)\big) = \lim_{T\to\infty} \frac{1}{T} \cdot \sum_{t=1}^{T} \sum_{i=1}^{N} w_i A_i(t)$ as in the Min-Sum problem. We will transform $J_o(\cdot)$ to a rate-based objective function, which we denote as $J_R(r_1, r_2, \cdots, r_N)$. The goal of this transformation is to minimize the distance (gap) between the two functions such that

$$J_o(\cdot) \approx J_R(r_1, r_2, \cdots, r_N). \tag{7.8}$$

Similarly, if any of the original constraints are not rate-based, then transform them into rate-based constraints.

Step 2: Find optimal transmission rates The goal of Step 2 is to find the transmission rates r_i's that optimize $J_R(r_1, r_2, \cdots, r_N)$. In particular, we solve a special optimization problem (OPT-SD) that will ensure the optimal rates (r_i's) can be readily used for the design of AUS-based schedulers in Step 3.

[4]We focus on a minimization problem here to concretize our discussions. Note that a maximization problem can be easily reformulated as a minimization problem, i.e., minimize the negative of the objective function.

Denote the transmission rate vector $\boldsymbol{r} = [r_1\ r_2\ \cdots\ r_N]$. We define a sorted rate vector $\boldsymbol{\gamma} = [\gamma_1\ \gamma_2\ \cdots\ \gamma_N]$ as:

$$\boldsymbol{\gamma} = \text{sort}(\boldsymbol{r}), \tag{7.9}$$

where sort$(\cdot)$ is a function that sorts the elements of the input vector in a *non-increasing* order. We are interested in $\boldsymbol{\gamma}$'s with a special *"step-down"* property, which we define as follows:

Definition 7.2. *A sorted vector $\boldsymbol{\gamma}$ is step-down if $\gamma_i/\gamma_{i+1} \in \mathbb{N}^*$ for all $\gamma_i < 1$ and $i < N$.*

In a step-down vector, the leading elements in $\boldsymbol{\gamma}$ can be 1's, while the remaining elements (with $\gamma_i < 1$) must satisfy $\gamma_i/\gamma_{i+1} \in \mathbb{N}^*$. For example, $\boldsymbol{\gamma} = [1\ 1\ \frac{4}{5}\ \frac{4}{5}\ \frac{2}{5}\ \frac{1}{5}]$ is step-down.

We now solve the following optimization problem:

$$\begin{aligned}
\text{OPT-SD:}\ \min_{\boldsymbol{r}}\quad & J_R(r_1, r_2, \cdots, r_N) \\
\text{s.t.}\quad & \boldsymbol{\gamma} = \text{sort}(\boldsymbol{r}), \\
& \boldsymbol{\gamma} \text{ is step-down}, \\
& \sum_{i=1}^{N} \gamma_i = W, \\
& \text{Additional problem-specific rate-based} \\
& \text{constraints}, \\
& 0 < r_i \le 1, \quad i = 1, 2, \cdots, N.
\end{aligned}$$

Note that we purposely make $\sum_{i=1}^{N} \gamma_i = W$ (instead of $\le W$) so that the rates will fit perfectly into a discrete slot-based scheduler.

Denote the optimal solution to OPT-SD as $\boldsymbol{r}^* = [r_1^*\ r_2^*\ \cdots\ r_N^*]$. After solving OPT-SD and obtaining $\boldsymbol{r}^*$, we will use $\boldsymbol{r}^*$ to construct an AUS in Step 3.

Step 3: Construct an AUS-based Scheduler The goal of Step 3 is to construct an

AUS-based scheduler using the optimal rate vector $\boldsymbol{r}^*$ that we find in Step 2.

We first consider the special case where $W = 1$ and present an algorithm on how to construct an AUS-based scheduler with $\boldsymbol{r}^*$ (with $\sum_{i=1}^{N} r_i^* = 1$) and $\boldsymbol{\gamma}^* = \text{sort}(\boldsymbol{r}^*)$ is step-down. We use an example to show how our algorithm works. The complete pseudocode is given in Algorithm 7.2.

Example 7.1. Consider six sources A, B, C, D, E, F and $\boldsymbol{r}^* = [\frac{1}{10}\ \frac{3}{10}\ \frac{1}{10}\ \frac{3}{10}\ \frac{1}{10}\ \frac{1}{10}]$. We have $\boldsymbol{\gamma}^* = [\frac{3}{10}\ \frac{3}{10}\ \frac{1}{10}\ \frac{1}{10}\ \frac{1}{10}\ \frac{1}{10}]$, which is step-down, and we have $\sum_{i=1}^{N} r_i^* = 1$. We will show how to construct an AUS for this $\boldsymbol{r}^*$.

Denote c^{AUS} as the cycle length of the AUS we are going to construct, and N_i as the number of slots allocated to the i-th source (indexed w.r.t. $\boldsymbol{\gamma}^*$) in c^{AUS}. Clearly, we have $N_i = \gamma_i^* \cdot c^{\text{AUS}}$.

In this example, the sequence of the sources in $\boldsymbol{\gamma}^*$ is *B-D-A-C-E-F*. Since the smallest rate among the γ_i^*'s is $1/10$ (for source F), we set the cycle length of the AUS to $c^{\text{AUS}} = 10$. Then we have $N_B = 3$, $N_D = 3$, $N_A = 1$, $N_C = 1$, $N_E = 1$, and $N_F = 1$.

Let's start with the first source B. Within a cycle with $c^{\text{AUS}} = 10$ slots, there are $N_B = 3$ slots allocated to source B. Ideally, we want the 3 slots to be evenly spaced in 10 slots. But this is not possible under 10 slots. So we will *add* the minimum number of slots (which is 2) to the cycle to make this happen. This corresponds to adding $(\lceil \frac{c^{\text{AUS}}}{N_B} \rceil \cdot N_B - c^{\text{AUS}})$ extra slots. These extra slots will be removed in the end when we change the EUS scheduler to an AUS scheduler. With a cycle of 12 slots ($c = 12$), we can place source B evenly as follows:

π	(B □ □ □ B □ □ □ B □ □ □)
t	1 2 3 4 5 6 7 8 9 10 11 12

Now we we consider the second node—D. With $N_D = 3$, we can place D evenly among the empty slots left by B. Among the empty slots ($t = 2, 3, 4, 6, 7, 8, 10, 11, 12$), we select the time slots with the smallest number of elapsed time slots following its predecessor node B, i.e., $t = 2, 6, 10$. We allocate all of them ($t = 2, 6, 10$) to source D, and we have:

π	(B D □ □ B D □ □ B D □ □)
t	1 2 3 4 5 6 7 8 9 10 11 12

The next source node to consider is A. Since $N_A = 1$, we only need to allocate 1 empty slot to source A. Among all the remaining empty slots, we want to find one with the shortest distance (in number of time slots) to A's predecessor nodes, starting from the oldest one (node B). The time slots with the shortest distance to B are: $t = 3, 7, 11$. Among them ($t = 3, 7, 11$), we select the time slots with the shortest distance to D and we still have: $t = 3, 7, 11$. Since we only need one slot, we choose $t = 3$ to A.

π	(B D A □ B D □ □ B D □ □)
t	1 2 3 4 5 6 7 8 9 10 11 12

The next source node to consider is C. Since $N_C = 1$, we only need to allocate 1 empty slot to source C. Again, among all the remaining empty slots, our goal is to find one with the shortest distance to C's predecessor nodes (B, D, A), starting from the oldest one (node B). With respect to each B in the cycle, the time slots with the shortest distance (on the

right side) are: $t = 7, 11$. Between the two ($t = 7, 11$), we want to select the time slots with the shortest distance to a D and we still have: $t = 7, 11$. Now between the two ($t = 7, 11$), we want to select one time slot with the shortest distance to A and we only have one choice, which is $t = 7$. We allocate $t = 7$ to C. Note that the above process can be put into an iteratively procedure (as shown in Steps 4–10 in Algorithm 7.2).

π	(B	D	A	□	B	D	C	□	B	D	□	□)
t	1	2	3	4	5	6	7	8	9	10	11	12

Following the same token, we allocate $t = 11$ to source E and $t = 4$ to source F and we have:

π	(B	D	A	F	B	D	C	□	B	D	E	□)
t	1	2	3	4	5	6	7	8	9	10	11	12

Clearly, outcome of our recursion will ensure the scheduler is an EUS throughout the process. In the last step, we remove the unused slots in the EUS scheduler and obtain an AUS scheduler $\pi^{(1)}$. We have:

$\pi^{(1)}$	(B	D	A	F	B	D	C	B	D	E	)
t	1	2	3	4	5	6	7	8	9	10	

□

The ideas of Example 7.1 are stated as pseudocode in Algorithm 7.2. The time complexity of Algorithm 7.2 is $O(N^2/\gamma_N)$.

Now we extend Algorithm 7.2 to the general case when $W \geq 1$. The problem is to construct an AUS scheduler $\pi^{(M)}$ for a step-down vector $\boldsymbol{\gamma}$ with $\sum_{i=1}^{N} \gamma_i = W$.

Denote M as the number of elements in $\boldsymbol{\gamma}$ such that $\gamma_i = 1$. For each source node

Algorithm 7.2 Construct an AUS-based Scheduler with $W = 1$

Input: A step-down vector $\boldsymbol{\gamma}^{(1)}$ with $\sum_{i=1}^{N} \gamma_i^{(1)} = 1$.

Output: An AUS-based scheduler $\pi^{(1)}$.

1: Set $c^{\text{AUS}} = 1/\gamma_N^{(1)}$ and $N_i = \gamma_i^{(1)} \cdot c^{\text{AUS}}$ for each $1 \leq i \leq N$.

2: Initialize an EUS cycle with length $c = \lceil \frac{c^{\text{AUS}}}{N_1} \rceil \cdot N_1$ slots.

3: Set $b_1 = c/N_1$. Allocate time slots $t = 1, b_1 + 1, 2b_1 + 1, \cdots, c - b_1 + 1$ in the EUS cycle to the first source in $\boldsymbol{\gamma}$.

4: **for** source $i = 2, 3, \cdots, N$ in $\boldsymbol{\gamma}$ **do**

5: Let $\mathcal{S}_0$ be the set of empty time slots in the EUS cycle.

6: **for** $j = 1, 2, \cdots, i - 1$ **do**

7: Let $\mathcal{S}_j$ be the subset of $\mathcal{S}_{j-1}$ containing time slots with the shortest distance after a source j.

8: **end for**

9: Let τ be the first slot in $\mathcal{S}_{i-1}$. Set $b_i = c/N_i$. Allocate slots $t = \tau, b_i + \tau, 2b_i + \tau, \cdots, c - b_i + \tau$ to source i.

10: **end for**

11: Remove the $(c - c^{\text{AUS}})$ unassigned time slots in the EUS cycle and output the resulting scheduler $\pi^{(1)}$.

$1, 2, \cdots, M$, we allocate every slot to them and get an EUS $\pi^{(M)}$ with bandwidth M.

For other source nodes, denote $\boldsymbol{\gamma}' = [\gamma_{M+1}\ \gamma_{M+2}\ \cdots\ \gamma_N]$, and the problem is to construct an AUS $\pi^{(W-M)}$ for $\boldsymbol{\gamma}'$ with bandwidth $(W-M)$. To solve this problem, the idea is to first construct an AUS-based scheduler $\pi^{(1)}$ using Algorithm 7.2 for the special case $W=1$. Since Algorithm 7.2 requires $\sum_{i=1}^{N}\gamma_i^{(1)}=1$, we have to scale down $\boldsymbol{\gamma}$ by a factor $(W-M)$, i.e., $\boldsymbol{\gamma}^{(1)}=\boldsymbol{\gamma}'/(W-M)$. Clearly, we have $\sum_{i=1}^{N}\gamma_i^{(1)}=1$, and we can use Algorithm 7.2 to construct $\pi^{(1)}$.

With $\pi^{(1)}$ as the basic building block, we use a procedure called *Zigzag packing* to construct the final AUS scheduler $\pi^{(W-M)}$ by packing scheduler $\pi^{(1)}$ column-by-column into a frequency-time grid exactly W times. We use an example to show how it works.

Example 7.2. Consider six sources A, B, C, D, E, F, $W=3$, and $\boldsymbol{r}=[\frac{3}{10}\ \frac{9}{10}\ \frac{3}{10}\ \frac{9}{10}\ \frac{3}{10}\ \frac{3}{10}]$. We have $\boldsymbol{\gamma}=[\frac{9}{10}\ \frac{9}{10}\ \frac{3}{10}\ \frac{3}{10}\ \frac{3}{10}\ \frac{3}{10}]$, which is step-down, $\sum_{i=1}^{N}\gamma_i=3$, and $M=0$. We now show how to construct an AUS scheduler $\pi(3)$ for this $\boldsymbol{\gamma}$.

As the first step, we scale down $\boldsymbol{\gamma}$ by a factor 3, and get $\boldsymbol{\gamma}^{(1)}=[\frac{3}{10}\ \frac{3}{10}\ \frac{1}{10}\ \frac{1}{10}\ \frac{1}{10}\ \frac{1}{10}]$, with $\sum_{i=1}^{N}\gamma_i^{(1)}=1$. Then we use Algorithm 7.2 to find AUS $\pi^{(1)}$ with $\boldsymbol{\gamma}^{(1)}$ as input, which is identical to Example 7.1. To construct $\pi^{(3)}$, we roll $\pi^{(1)}$ into $\pi^{(3)}$'s frequency-time structure column-by-column 3 times, as shown in Fig. 7.1, and obtain the AUS scheduler $\pi^{(3)}$. □

After obtaining $\pi^{(M)}$ and $\pi^{(W-M)}$, we can combine them and get our target AUS $\pi^{(W)}$ with bandwidth W. A pseudocode on how to construct an AUS-based scheduler is given in Algorithm 7.3. This completes Step 3 of Eywa.

Now we have presented all details for the three steps of our proposed Eywa framework. In the rest of this chapter, we put this framework in action by solving a family of AoI

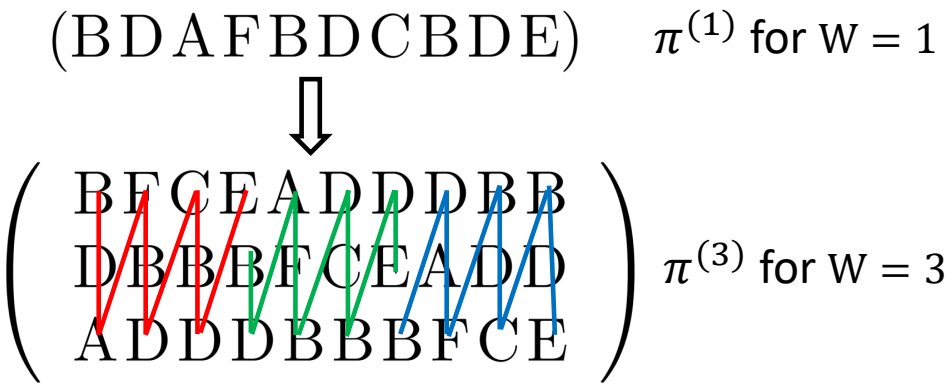


Figure 7.1: An example for Zigzag packing

optimization and decision problems.

7.3 Application to the Min-Sum Problem

In this section, we show how to use Eywa to solve Min-Sum problem that we mentioned in the beginning of Section 7.2. Then we will compare Eywa's scheduler with the state-of-the-art.

7.3.1 Problem Statement

The network setting for Min-Sum is the same as that in Section 7.2.1. We consider a general $W \geq 1$. The long-term average AoI for source node i is defined as:

$$\bar{A}_i = \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} A_i(t). \tag{7.10}$$

Denote w_i as the weight of source node i. The weighted sum of long-term average AoI over all source nodes, denoted as $\bar{A}$, is defined as:

$$\bar{A} = \sum_{i=1}^{N} w_i \bar{A}_i. \tag{7.11}$$

The objective for the Min-Sum problem is to design a scheduler π that can minimize $\bar{A}$.

Algorithm 7.3 Construct an AUS-based Scheduler with $W \geq 1$

Input: A step-down vector $\boldsymbol{\gamma}$ with $\sum_{i=1}^{N} \gamma_i = W$.

Output: An AUS-based scheduler $\pi^{(W)}$.

1: For each source node $1, 2, \cdots, M$, allocate every slot to them and get an EUS $\pi^{(M)}$ with bandwidth M.

2: For other source nodes, set $\boldsymbol{\gamma}' = [\gamma_{M+1}\ \gamma_{M+2}\ \cdots\ \gamma_N]$. Set $\boldsymbol{\gamma}^{(1)} = \boldsymbol{\gamma}'/(W-M)$. Construct an AUS scheduler $\pi^{(1)}$ by Algorithm 7.2 with $\boldsymbol{\gamma}^{(1)}$ as input.

3: If $W-M > 1$, use Zigzag packing to construct an AUS scheduler $\pi^{(W-M)}$ with bandwidth $(W-M)$.

4: Combine $\pi^{(M)}$ and $\pi^{(W-M)}$, and get AUS $\pi^{(W)}$.

7.3.2 A Lower Bound

To date, there is no known polynomial time optimal scheduler for the Min-Sum problem in the literature. This is partially due to p_i (packet loss probability) associated for each source, which make it hard to prove optimality. As such, a lower bound for the objective $\bar{A}$ is important because it can serve as a performance benchmark (in place of the optimal objective).

In [92], the authors showed that for each source i, we have

$$w_i \bar{A}_i \geq \frac{w_i}{2(1-p_i)r_i} + \frac{w_i}{2}. \tag{7.12}$$

Therefore, $\min \sum_{i=1}^{N} w_i \bar{A}_i \geq \min \sum_{i=1}^{N} (\frac{w_i}{2(1-p_i)r_i} + \frac{w_i}{2})$. So we solve the following optimiza-

tion problem:

$$\text{OPT-LB:} \quad \min_{\boldsymbol{r}} \quad \sum_{i=1}^{N} \frac{w_i}{2(1-p_i)r_i}$$
$$\text{s.t.} \quad \sum_{i=1}^{N} r_i \le W,$$
$$0 < r_i \le 1, \quad i = 1, 2, \cdots, N.$$

Denote $\boldsymbol{r}^{\text{LB}} = [r_1^{\text{LB}}, r_2^{\text{LB}}, \cdots, r_N^{\text{LB}}]$ as the optimal solution to OPT-LB. Since OPT-LB is a convex optimization problem, we can get $\boldsymbol{r}^{\text{LB}}$ easily. Then, by (7.12), a lower bound of $\bar{A}$ is given by

$$\bar{A}_{\text{LB}} = \sum_{i=1}^{N} \frac{w_i}{2(1-p_i)r_i^{\text{LB}}} + \sum_{i=1}^{N} \frac{w_i}{2}. \tag{7.13}$$

7.3.3 Eywa: Step 1—Transform Objective Function and Constraints

In this and the following two sections, we show how to apply Eywa framework to find a high-performance scheduling solution to the Min-Sum problem. As presented in Section 7.2.3, the first step of Eywa is to transform the AoI-based objective function $J_o\big(A_1(t), A_2(t), \cdots, A_N(t)\big) = \sum_{i=1}^{N} w_i \bar{A}_i$ to a rate-based objective function $J_R(r_1, r_2, \cdots, r_N)$. For the Min-Sum problem, we will approximate $\bar{A}_i$ as a rate-based function $q_i(r_i)$, i.e., $\bar{A}_i \approx q_i(r_i)$ for each i. Then we will have $J_R(r_1, r_2, \cdots, r_N) = \sum_{i=1}^{N} w_i \cdot q_i(r_i)$.

Consider an AUS π. To make a connection between $\bar{A}_i$ and $q_i(r_i)$, we propose to *decompose* the AUS scheduler into *two mini EUS* schedulers, for which we can calculate the AoI easily and express it as a rate-based function. We will use an example to show how this work.

Example 7.3. Consider an AUS π with $W = 1$:

$$\pi = (\text{BDAFBDCBDE}).$$

We use source B as an example. To find $\bar{A}_B$, we decompose π into two mini EUSs. We consider the simple case where packet loss rate $p_B = 0$ first.

Since we are only interested in calculating source B's average age, $\bar{A}_B$, we focus on the time slots that source B are scheduled for transmission, i.e.,

$$\pi^0 = (\text{B}\square\square\square\text{B}\square\square\text{B}\square\square).$$

In scheduler π^0, we notice that there are three elapsed intervals w.r.t. source B in the cycle: 4, 3, and 3. So we can decompose π^0 into two EUSs π^1 and π^2 as following:

$$\pi^1 = (\text{B}\square\square\square), \quad \pi^2 = (\text{B}\square\square\text{B}\square\square).$$

Denote $\bar{A}_B^1$ and $\bar{A}_B^2$ as the average AoIs under EUS π^1 and π^2, respectively. Clearly, we have $\bar{A}_B^1 = \frac{5}{2}$ (average of 1, 2, 3, and 4) and $\bar{A}_B^2 = 2$ (average of 1, 2, 3). Denote $q_B(r_B)$ as the weighted average (proportional to mini-cycle length) AoI of two mini EUS π^1 and π^2. Since there are 4 slots in π^1 and 6 slots in π_2, $q_B(r_B)$ is given by

$$q_B(r_B) = \frac{4 \cdot \bar{A}_B^1 + 6 \cdot \bar{A}_B^2}{4+6} = \frac{4 \cdot \frac{5}{2} + 6 \cdot 2}{10} = \frac{11}{5}. \quad \square$$

For a general π with rate r_i and $p_i = 0$ for each i, we now show how to find $q_i(r_i)$ based on the idea presented in Example 7.3. We first focus on the time slots that source i are scheduled for transmission and get π^0. Under π^0, the interval length for source i can only be

either $\lfloor \frac{1}{r_i} \rfloor$ or $\lfloor \frac{1}{r_i} \rfloor + 1$, where $\lfloor \cdot \rfloor$ is the floor function. Then we decompose π^0 into two mini EUSs π^1 and π^2. Denote $q_i(r_i)$ as the weighted average (proportional to mini-cycle length) AoI of π^1 and π^2. It is not hard to derive the following expression:

$$q_i(r_i) = \lfloor \frac{1}{r_i} \rfloor + 1 - \frac{r_i}{2}(\lfloor \frac{1}{r_i} \rfloor^2 + \lfloor \frac{1}{r_i} \rfloor), \quad (\text{for } p_i = 0). \tag{7.14}$$

For the general case where $p_i \geq 0$, it can be shown (by taking expectations) that

$$q_i(r_i) = \frac{1+p_i}{1-p_i} \cdot \big(\lfloor \frac{1}{r_i} \rfloor + \frac{1}{2} - \frac{r_i}{2}(\lfloor \frac{1}{r_i} \rfloor^2 + \lfloor \frac{1}{r_i} \rfloor)\big) + \frac{1}{2}. \tag{7.15}$$

Note that when packet loss rate $p_i = 0$, $q_i(r_i)$ is exactly equal to $\bar{A}_i$. But when $p_i > 0$, $q_i(r_i)$ will be slightly greater than $\bar{A}_i$. This is because there is a slight relaxation for AoI when we decompose AUS π^0 into two mini EUSs π^1 and π^2.

7.3.4 Eywa: Step 2—Find Optimal Transmission Rates

The second step of Eywa is to find the optimal rate vector $\boldsymbol{r}^*$. That is, we need to solve OPT-SD, with objective $J_R(r_1, r_2, \cdots, r_N) = \sum_{i=1}^{N} w_i \cdot q_i(r_i)$. Note that there are no additional constraints in OPT-SD for the Min-Sum problem.

Due to the complexity of the rate function $q_i(r_i)$ in (7.15), it is too difficult to solve OPT-SD directly. So we present an algorithm that can find a (provably) near-optimal solution to OPT-SD.

Our proposed near-optimal solution is based on $\boldsymbol{r}^{\text{LB}}$, the optimal solution to OPT-LB. Since $\boldsymbol{r}^{\text{LB}}$ may not be a feasible solution to OPT-SD, we propose to find a β $(0 < \beta \leq 1)$ and $\boldsymbol{r}$, such that $\boldsymbol{r} \geq \beta \cdot \boldsymbol{r}^{\text{LB}}$ (i.e., $r_i \geq \beta \cdot r_i^{\text{LB}}$ for each $i = 1, 2, \cdots, N$) and $\boldsymbol{r}$ is step-down and feasible to OPT-SD. Clearly, the greater the β is, the closer the $\boldsymbol{r}$ and $\boldsymbol{r}^{\text{LB}}$ will be. So we want to solve the following optimization problem:

Table 7.2: Comparison between Eywa and state-of-the-art for the Min-Sum problem.

Setting	Algorithm	Performance Guarantee	When $p_{\max} = 0.12$
$W = 1$	Eywa	$\bar{A} \leq (1 + p_{\max}) \log_2 e \cdot \bar{A}^*$	$\bar{A} \leq 1.62 \cdot \bar{A}^*$
	Stat. Rand. [103], [105]	$\bar{A} \leq 2 \cdot \bar{A}^*$	$\bar{A} \leq 2 \cdot \bar{A}^*$
	Max-Weight [103], [105]	$\bar{A} \leq 2 \cdot \bar{A}^*$	$\bar{A} \leq 2 \cdot \bar{A}^*$
	Whittle's Index [103], [105]	$\bar{A} \leq 4 \cdot (\frac{\sqrt{2}}{1-p_{\max}} + \frac{1}{\sqrt{2}})^2 \cdot \bar{A}^*$	$\bar{A} \leq 21.42 \cdot \bar{A}^*$
$W \geq 2$	Eywa	$\bar{A} \leq (1 + p_{\max}) \log_2 e \cdot \bar{A}^*$	$\bar{A} \leq 1.62 \cdot \bar{A}^*$
	Stat. Rand. [104], [105]	$\bar{A} \leq 2 \cdot \bar{A}^*$	$\bar{A} \leq 2 \cdot \bar{A}^*$

$$
\begin{aligned}
\text{OPT-}\beta\text{:} \quad \max_{\boldsymbol{r},\beta} \quad & \beta \\
\text{s.t.} \quad & \boldsymbol{\gamma} = \text{sort}(\boldsymbol{r}), \\
& \boldsymbol{\gamma} \text{ is step-down}, \\
& \beta \cdot r_i^{\text{LB}} \leq r_i \leq 1, \ i = 1, 2, \cdots, N, \\
& \sum_{i=1}^{N} \gamma_i = W.
\end{aligned}
$$

Denote $\boldsymbol{r}^*$ and β^* as the optimal solution to OPT-β. Clearly, $\boldsymbol{r}^*$ is feasible to OPT-SD, which we will use to construct an AUS-based scheduler in Step 3. An algorithm (based on bisection and dynamic programming) to solve OPT-β is given in Appendix D.

7.3.5 Eywa: Step 3—Construct AUS

The third step of Eywa is to construct AUS based on $\boldsymbol{r}^*$, which is exactly the same as what we have done in Section 7.2.3.

7.3.6 Performance and Comparison with State-of-the-Art

In this section, we show that Eywa offers a strong theoretical performance guarantee to the Min-Sum problem. Denote $p_{\max}$ as the maximum packet loss rate among all source nodes,

i.e.,

$$p_{\max} = \max_{i=1,2,\cdots,N} p_i.$$

Denote $\bar{A}^*$ as the optimal (unknown) objective value for the Min-Sum problem. We have the following theorem that guarantees the performance of $\bar{A}$—the objective value obtained by Eywa.

Theorem 7.1. *When $p_{\max} \leq 0.807$, the objective value obtained by Eywa satisfies:*

$$\bar{A} \leq (1 + p_{\max}) \log_2 e \cdot \bar{A}^*. \tag{7.16}$$

A proof of Theorem 7.1 is given in Appendix E.

In practice, 4G LTE and 5G NR (eMBB) use link adaption algorithms to make sure the BLER (packet loss rate) is about 10% for each user [109]. For $p_{\max} = 0.12$, we have $\bar{A} \leq 1.62 \cdot \bar{A}^*$ under Eywa.

Table 7.2 compares the performance guarantees offered by Eywa and the state-of-the-art algorithms for the Min-Sum problem. When $W = 1$, three solutions were proposed ([103], [105](Ch. 3)): the stationary randomized scheduler, the max-weight scheduler, and the Whittle's index scheduler. We can see that in practice (when $p_{\max} = 0.12$), Eywa offers a tighter bound than the other three schedulers. When $W \geq 2$, the max-weight scheduler and the Whittle's index scheduler are not available in the literature, while the stationary randomized scheduler was studied in [104] and [105] (Ch. 4). Again, we can see that in practice (when $p_{\max} = 0.12$), Eywa offers a tighter bound than the stationary randomized scheduler.

In addition to offering a stronger performance guarantee, Eywa also has a lower over-

head and latency than the state-of-the-art schedulers, as we discussed in the beginning of Section 7.2.

7.4 Application to the Min-BW Problem

In this section, we show how to apply Eywa to solve the Min-BW problem. Recall that the Min-BW problem (see Section 7.2) is to minimize the uplink bandwidth requirement under some AoI constraints. Specifically, we study the problems under two AoI constraints: i) Min-BW under *peak* AoI constraints, and ii) Min-BW under *average* AoI constraints. The first problem was studied in [102]. We show that Eywa can offer a stronger performance guarantee (a tighter bound) than that in [102]. The second problem has not yet been studied in the literature and thus our scheduler (by applying Eywa) is the first known solution to this problem.

7.4.1 Min-BW Under Peak AoI Constraints

Problem Statement

The network setting is the same as that in Section 7.2.1. For this problem, we assume there is a deadline (denoted as $d_i \in \mathbb{N}^*$) constraint for the peak AoI of each source $i = 1, 2, \cdots, N$. That is,

$$A_i(t) \leq d_i, \text{ for all } t \geq 0. \tag{7.17}$$

Since the above constraint is a deterministic one and applies to all $t \geq 0$ and all $i = 1, 2, \cdots, N$, it can hold only when $p_i = 0$ for $i = 1, 2, \cdots, N$. Therefore, we assume $p_i = 0$ for all i's in this problem. The objective is to find a scheduler π that minimizes bandwidth W, such that (7.17) is satisfied.

Eywa: Step 1—Transform Objective Function and Constraints

Since the original objective function is $J_o(\cdot) = W$, and that $W = \sum_{i=1}^{N} r_i$, we can just replace it by $J_R(r_1, r_2, \cdots, r_N) = \sum_{i=1}^{N} r_i$.

In addition, for constraint (7.17), which is AoI-based and specific for the Min-BW problem, we need to transform it to a rate-based constraint. For an AUS with rate r_i's, the transmission intervals T_i^k's for source i are upper bounded by $\lceil \frac{1}{r_i} \rceil$, where $\lceil \cdot \rceil$ is the ceiling function. Therefore, if $r_i \geq \frac{1}{d_i}$, we will have $T_i^k \leq d_i$ for all $k \geq 1$. That is, $r_i \geq \frac{1}{d_i}$ is a sufficient condition for constraint (7.17). As such, we can replace constraint (7.17) (AoI-based) by the following rate-based constraint:

$$r_i \geq \frac{1}{d_i}. \tag{7.18}$$

Eywa: Step 2—Find Optimal Transmission Rates

The second step of Eywa is to find the optimal rate vector $\boldsymbol{r}^*$, i.e., to solve the following optimization problem OPT-SD (Min-BW).

$$
\begin{aligned}
&\text{OPT-SD (Min-BW)} \\
&\min_{\boldsymbol{r},W} \quad \sum_{i=1}^{N} r_i \\
&\text{s.t.} \quad \boldsymbol{\gamma} = \text{sort}(\boldsymbol{r}), \\
&\qquad \boldsymbol{\gamma} \text{ is step-down}, \\
&\qquad \sum_{i=1}^{N} r_i = W, \\
&\qquad r_i \geq \frac{1}{d_i}, \quad i = 1, 2, \cdots, N, \\
&\qquad 0 < r_i \leq 1, \quad i = 1, 2, \cdots, N. \\
&\qquad W \in \mathbb{N}^*.
\end{aligned}
$$

Note that in OPT-SD (Min-BW), $r_i \geq \frac{1}{d_i}$ is the additional problem-specific rate-based constraint. Denote $\boldsymbol{r}^*$ as the optimal solution to OPT-SD (Min-BW), which we will use to construct an AUS-based scheduler in Step 3. The complete algorithm to solve OPT-SD (Min-BW) is given in Appendix F.

Eywa: Step 3—Construct an AUS-Based Scheduler

The third step of Eywa is to construct an AUS-based scheduler based on $\boldsymbol{r}^*$, which is exactly the same as what we have done in Section 7.2.3.

Performance and Comparison with State-of-the-Art

In this section, we show that Eywa offers a strong theoretical performance guarantee to the Min-BW problem under peak AoI constraints. Denote W^* as the optimal (unknown) objective value for the Min-BW problem under peak AoI constraints. We have the following theorem that guarantees the performance of W—-the objective value obtained by Eywa.

Theorem 7.2. *For the Min-BW problem under peak AoI constraints, the objective value obtained by Eywa satisfies:*

$$W \leq \lceil \log_2 e \cdot W^* \rceil. \tag{7.19}$$

A proof of Theorem 7.2 is given in Appendix G.

For the Min-BW problem under peak AoI constraints, a state-of-the-art solution (called Aion) was proposed in [102]. Aion can offer a performance guarantee of $W \leq 2 \cdot W^*$ (a proof is given in Appendix H). Since $\log_2 e \approx 1.44$, we can see that Eywa offers a tighter performance bound than Aion.

7.4.2 Min-BW under Average AoI Constraints

Problem Statement

For this problem, we assume there is an average AoI constraint (denoted as $\alpha_i \in \mathbb{R}^*$) for the average AoI $\bar{A}_i$. That is, for $i = 1, 2, \cdots, N$,

$$\bar{A}_i \leq \alpha_i. \tag{7.20}$$

For average AoI constraints, the packet loss rate p_i for each source i can be greater than 0, i.e., $p_i \geq 0$. Recall in our system model, we assume $r_i \leq 1$, i.e., there is *at most* one packet transmitted from source i in every time slot, regardless of how large W is. So to satisfy constraint (7.20), there must be a limit on the value p_i (channel condition). It can be shown that $\bar{A}_i \geq \frac{1}{1-p_i}$ when $r_i \leq 1$. So for constraint (7.20) to hold, we must have $\alpha_i \geq \frac{1}{1-p_i}$, i.e.,

$$p_i \leq 1 - \frac{1}{\alpha_i}, \tag{7.21}$$

for each $i = 1, 2, \cdots, N$.

The objective of the Min-BW problem under average AoI constraints is to find a scheduler π that minimizes bandwidth W such that constraint (7.20) is satisfied.

Eywa: Step 1—Transform Objective Function and Constraints

Again, in the first step of Eywa, we transform the original objective function $J_o(\cdot) = W$ to $J_R(r_1, r_2, \cdots, r_N) = \sum_{i=1}^{N} r_i$.

For the AoI-based constraint (7.20), we need to transform it to a rate-based constraint. Recall in Section 7.3.3 we have $\bar{A}_i \leq q_i(r_i)$. Then, to satisfy (7.20), its sufficient to have $q_i(r_i) \leq \alpha_i$ It can be shown that $q_i(r_i) \leq \alpha_i$ will hold if

$$r_i \geq \frac{2\lfloor x \rfloor + 1 - x}{\lfloor x \rfloor^2 + \lfloor x \rfloor}, \tag{7.22}$$

where $x = \frac{(1-p_i)(2\alpha_i - 1)}{1+p_i}$. So we can replace the original AoI-based constraint (7.20) by the new rate-based constraint (7.22).

Eywa: Step 2—Find Optimal Transmission Rates

The second step of Eywa is to find the optimal rate vector $\boldsymbol{r}^*$, i.e., to solve OPT-SD (Min-BW) by replacing problem-specific constraint "$r_i \geq \frac{1}{d_i}$" with constraint (7.22). The solution to this problem is identical to that shown in Appendix F (except the minor change in the RHS value of the problem-specific constraint).

Eywa: Step 3—Construct an AUS-based scheduler

The third step of Eywa is to construct an AUS-based scheduler based on $\boldsymbol{r}^*$, which is exactly the same as what we have done in Section 7.2.3.

Performance Guarantee

In this section, we show that Eywa offers a strong theoretical performance guarantee to the Min-BW problem under average AoI constraints. Denote W^* as the optimal (unknown) objective value for the Min-BW problem under average AoI constraints. We have the following theorem that guarantees the performance of W—-the objective value obtained by Eywa.

Theorem 7.3. *For the Min-BW problem under average AoI constraints, the objective value obtained by Eywa satisfies:*

$$W \leq \left\lceil \frac{9 \log_2 e \cdot (1 + p_{\max})}{8} \cdot W^* \right\rceil. \tag{7.23}$$

A proof of Theorem 7.3 is given in Appendix I.

To date, a solution to the Min-BW problem under average AoI constraints is not available in the literature. So the solution that we presented here is all new.

7.5 Application to Decision Problems

In the previous two sections, we applied Eywa to solve two optimization problems: Min-Sum and Min-BW. In addition to solving optimization problems, the Eywa framework can also be applied to solve decision problems related to AoI, which we will show in this section.

The decision problem we study is to determine the existence of feasible schedulers to satisfy AoI constraints. Specifically, we study the problems under two AoI constraints: i) *hard* AoI deadlines, and ii) *soft* AoI deadlines. The first problem was studied in Chapter 4, and the second problem was studied in Chapter 5, both for the special case of $W = 1$. We show that Eywa can be applied to solve both problems for the general case of $W \geq 1$.

7.5.1 A Decision Problem with Hard AoI Deadlines

Problem Statement

We consider the system model in Section 7.2.1 with fixed bandwidth $W \geq 1$. Denote $\boldsymbol{d} = [d_1, d_2, \cdots, d_N]$ as the deadline vector. Similar to Section 7.4.1, we assume a hard constraint (7.17) for each $i = 1, 2, \cdots, N$. Since constraint (7.17) is a deterministic one, it can hold only when $p_i = 0$ for $i = 1, 2, \cdots, N$.

Our goal is to solve the following decision problem: For a given $\boldsymbol{d}$, determine whether or not there exists a feasible scheduler that satisfies (hard) deadline constraint (7.17). If there exists a feasible scheduler, then we want to find one.

Eywa: Step 1—Transform Objective Function and Constraints

For the decision problem, there is no objective function and we don't need to make any transform.

For the constraints, just like in Section 7.4.1, we replace the AoI-based constraint (7.17) by the rate-based constraint $r_i \geq \frac{1}{d_i}$.

Eywa: Step 2—Find Optimal Transmission Rates

The second step of Eywa is to find the optimal rate vector $\boldsymbol{r}$. For a decision problem (no objective function), we need to solve the following problem (DEC-SD), along with the problem-specific constraint $r_i \geq \frac{1}{d_i}$.

$$
\begin{aligned}
\text{DEC-SD:} \quad & \text{Determine whether or not } \boldsymbol{r} \text{ exists.} \\
\text{s.t.} \quad & \boldsymbol{\gamma} = \text{sort}(\boldsymbol{r}), \\
& \boldsymbol{\gamma} \text{ is step-down}, \\
& \sum_{i=1}^{N} r_i = W, \\
& r_i \geq \frac{1}{d_i}, \quad i = 1, 2, \cdots, N, \\
& 0 < r_i \leq 1, \quad i = 1, 2, \cdots, N.
\end{aligned}
$$

The algorithm to solve DEC-SD is given in Appendix J. There are two possible outcomes. If we can find a feasible $\boldsymbol{r}$ to DEC-SD, then we will use it to construct an AUS-based scheduler in Step 3. Otherwise, we conclude we cannot find a feasible scheduler for $\boldsymbol{d}$.

Eywa: Step 3—Construct an AUS-based scheduler

The third step of Eywa is to construct an AUS-based scheduler based on $\boldsymbol{r}$, which is exactly the same as what we have done in Section 7.2.3.

Eywa: Performance and Comparison with State-of-the-Art

The following theorem gives a performance guarantee of Eywa:

Theorem 7.4. *For any $\boldsymbol{d}$ with $\sum_{i=1}^{N} \frac{1}{d_i} \leq \frac{W}{\log_2 e}$, Eywa can always find a feasible scheduler.*

Theorem 7.4 can be easily proved based on Lemma E.1 in Appendix E and the design of Algorithm J.1 in Appendix J.

For the decision problem with hard AoI deadlines, in Chapter 4 we proposed FPM. But FPM was only designed for the special case of $W = 1$ and it is not clear how to extend it to

the general case of $w \geq 1$. The main result of FPM is that for any $\boldsymbol{d}$ with $\sum_{i=1}^{N} \frac{1}{d_i} \leq \frac{1}{\log_2 e}$, FPM can always find a feasible scheduler, which is consistent to what Eywa can also deliver (when $W = 1$).

7.5.2 A Decision Problem with Soft AoI Deadlines

Problem Statement

Now we assume the AoI deadline d_i's are soft, i.e., occasional violations can be tolerated. Denote $\boldsymbol{\epsilon} = [\epsilon_1, \epsilon_2, \cdots, \epsilon_N]$ as the vector for violation rates, where $\epsilon_i \in [0, 1)$ for each $i = 1, 2, \cdots, N$. The soft AoI deadline constraint for each source node $i = 1, 2, \cdots, N$ is given by

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} [A_i(t) > d_i] \leq \epsilon_i, \tag{7.24}$$

where "$[\cdot]$" is Iverson bracket, returning 1 if the inside statement is true and 0 otherwise. Denote $\boldsymbol{p} = [p_1, p_2, \cdots, p_N]$ as the vector for packet loss rates. For soft AoI deadlines, p_i for each source i can be greater than 0, i.e., $p_i \geq 0$.

Similar to that in Section 7.5.1, our goal is to solve the following decision problem: For a given $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$, determine whether or not there exists a feasible scheduler that satisfies (soft) deadline constraint (7.24). If there exists a feasible scheduler, then we want to find one.

Eywa: Step 1—Transform Objective Function and Constraints

Again, for the decision problem, there is no objective function and we don't need to make any transform.

For the AoI-based constraint (7.24), we need to transform it to a rate-based constraint.

In Chapter 5, we showed that (7.24) is satisfied if

$$r_i \geq r_i^{\text{tar}}, \tag{7.25}$$

where r_i^{tar} is given by (5.29). So we can replace the original AoI-based constraint (7.24) by the new rate-based constraint (7.25).

Eywa: Step 2—Find Optimal Transmission Rates

The second step of Eywa is to find the optimal rate vector $\boldsymbol{r}$. Similar to that in Section 7.5.1, we need to solve DEC-SD by replacing problem-specific constraint "$r_i \geq \frac{1}{d_i}$" with constraint (7.25). The solution to this problem is identical to that shown in Appendix J. Again, if we can find a feasible $\boldsymbol{r}$ to DEC-SD, then we will use it to construct an AUS-based scheduler in Step 3. Otherwise, we conclude we cannot find a feasible scheduler for $\boldsymbol{d}$.

Eywa: Step 3—Construct an AUS-based scheduler

The third step of Eywa is to construct an AUS-based scheduler based on $\boldsymbol{r}$, which is exactly the same as what we have done in Section 7.2.3.

Eywa: Performance and Comparison with State-of-the-Art

The following theorem gives a performance guarantee of Eywa:

Theorem 7.5. *For any* $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ *with* $\sum_{i=1}^{N} r_i^{tar} \leq \frac{W}{\log_2 e}$, *Eywa can always find a feasible scheduler.*

Theorem 7.5 can be easily proved based on Lemma E.1 in Appendix E and the design of Algorithm J.1 in Appendix J.

Problems	Cases	State-of-the-art			Eywa	
		References	Algorithms	Performance Guarantee	Performance Guarantee	
Min-Sum	$W = 1$	[111], [113]	Stationary Randomized	$\bar{A} \leq 2 * A^*$	$\bar{A} \leq 1.62 * A^*$ ($p_{max} = 0.12$)	○
			Max-Weight	$\bar{A} \leq 2 * A^*$		
			Whittle's Index	$\bar{A} \leq 21.42 * A^*$ ($p_{max} = 0.12$)		
	$W > 1$	[112], [113]	Stationary Randomized	$\bar{A} \leq 2 * A^*$	$\bar{A} \leq 1.62 * A^*$ ($p_{max} = 0.12$)	○
Min-BW	Peak AoI: $A_i(t) \leq d_i$	[110]	Aion	$W \leq 2 * W^*$	$W \leq \lfloor 1.44 * W^* \rfloor$	○
	Average AoI: $\bar{A}_i \leq \alpha_i$	NA	NA	NA	$W \leq \lfloor 1.82 * W^* \rfloor$ ($p_{max} = 0.12$)	▲
Decision Problems	Hard AoI deadlines	Chapter 4	FPM	Schedulable when $\sum_{i=1}^{N} \frac{1}{d_i} \leq \ln 2$ (only for W = 1)	Schedulable when $\sum_{i=1}^{N} \frac{1}{d_i} \leq W \cdot \ln 2$	▲
	Soft AoI deadlines	Chapter 5	UTS	Schedulable when $\sum_{i=1}^{N} r_i^{tar} \leq \ln 2$ (only for $W = 1$)	Schedulable when $\sum_{i=1}^{N} r_i^{tar} \leq W \cdot \ln 2$	▲

○: Tighter performance bound than state-of-the-art ▲: New/general results that are not available in state-of-the-art

Figure 7.2: Applications of Eywa to a family of AoI problems and a comparison of performance guarantees. NA indicates results are not available in the literature.

For the decision problem with soft AoI deadlines, in Chapter 5 we proposed UTS. Again, UTS was only designed for the special case of $W = 1$. The main result of UTS is that for any $(\boldsymbol{d}, \boldsymbol{\epsilon}, \boldsymbol{p})$ with $\sum_{i=1}^{N} r_i^{\text{tar}} \leq \frac{1}{\log_2 e}$, UTS can always find a feasible scheduler, which is consistent to what Eywa can also deliver (when $W = 1$).

7.6 Chapter Summary

This chapter presented Eywa—a general design framework that can be applied to construct high-performance schedulers for AoI-related optimization and decision problems. The core design ideas of Eywa are the notions of AUS schedulers and step-down rate vectors, which are

inspried from the design ideas in Chapter 5. The framework of Eywa consists of three steps: (i) transform the (AoI-based) objective function and constraints to rate-based ones, (ii) find the optimal step-down rate vector by solving an optimization problem (OPT-SD), and (iii) construct an AUS-based scheduler using the optimal step-down rate vector. To validate the efficacy of the proposed Eywa framework, we applied it to solve a number of AoI optimization and decision problems, such as Min-Sum, Min-BW, and the decision problems with hard/soft AoI constraints. The results are summarized in Fig. 7.2. We found that for each problem, Eywa can either offer a stronger performance guarantee than the state-of-the-art algorithms, or provide new/general results that are not available in the literature.

Chapter 8

Summary and Future Work

8.1 Summary

In this dissertation, we focused on two key issues in AoI research: minimizing average AoI (Chapters 2 and 3) and guaranteeing AoI deadlines (Chapters 4, 5, and 6). In Chapter 7, a general approach that can address both types of AoI research problems was developed.

In Chapter 2, we studied an (average) AoI minimization problem with generalization of by three key elements in the system model: sampling behavior, sample size, and transmission capacity. Under these three generalizations, we developed two fundamental properties to reduce the search space and derived tight lower bounds for an optimal solution. Further, we developed a near-optimal low-complexity scheduling algorithm—Juventas. In Chapter 3, we extended the AoI minimization problem in Chapter 2 to 5G networks. We presented a near-optimal 5G-compliant scheduler—Kronos. To meet the stringent timing requirement in 5G, we implemented Kronos on COTS GPU by exploiting its massive parallel computing capability.

In Chapter 4, we studied AoI scheduling subject to hard AoI deadlines. We first proposed an error-free procedure brute-force solution CSD, which can be used to solve the problem when the network size is small. For a large network, we presented a procedure called FPM that can find a feasible scheduler for any deadline vector $\boldsymbol{d}$ when the system load is no greater than $l \ln 2$. In Chapter 5, we considered the scheduling problem with soft AoI deadlines and unreliable channel. We explored the relationship between the violation tolerance

rates and the packet loss rates, and identified two cases: stable tolerant and unstable tolerant. For the stable tolerant case, we presented STS that can find a feasible scheduler as long as the system load is no greater than $\ln 2$. For the unstable tolerant case, we presented UTS, and we gave a sufficient condition for it to find a feasible scheduler. In Chapter 6, we extended the scheduling problem in Chapter 5 to 5G networks, i.e., we studied a 5G scheduling problem problem to minimize the deadline violation rate. We derived a uniform fairness property associated with an offline optimal scheduler. Based on this property, we developed Aequitas—a near-optimal online 5G scheduler. By exploiting the intrinsic parallelism in Aequitas, we implemented it on a COTS GPU platform to meet the 5G timing requirement.

Inspired by the design ideas in Chapter 5, in Chapter 7 we presented Eywa—a general design framework that can be applied to construct high-performance schedulers for AoI-related optimization and decision problems. The core of Eywa hinges upon the notions of AUS schedulers and step-down rate vectors. The framework of Eywa consists of three steps: (i) transform the (AoI-based) objective function and constraints to rate-based ones, (ii) find the optimal step-down rate vector by solving an optimization problem (OPT-SD), and (iii) construct an AUS-based scheduler using the optimal step-down rate vector. To validate the efficacy of the proposed Eywa framework, we applied it to solve a number of problems, such as minimizing the sum of AoIs, minimizing the bandwidth requirement under AoI constraints, and determining the existence of feasible schedulers to satisfy AoI constraints. We found that for each problem, Eywa can either offer a stronger performance guarantee than the state-of-the-art algorithms, or provide new/general results that are not available in the literature.

8.2 Future Work

Although this dissertation has made a major advance in extending the state-of-the-art of AoI research, there are many open problems that remain to be explored. We discuss some of them as the followings:

- **Optimizing AoI Performance by Exploiting Advances at the Physical Layer** In this dissertation, we have studied AoI under more practical settings than the state-of-the-art. In Chapter 3 we minimized AoI under 5G, and in Chapter 6 we designed schedulers under 5G with AoI deadlines. However, more issues in practice deserve further investigation, especially new advances at the physical layer. For example, Chapters 3 and 6 did not consider MU-MIMO, which is a key technology in 5G and has been deployed in the field. However, considering MU-MIMO will bring a much larger problem space because the beamforming vector needs to be set up for each user, while the sub-millisecond real-time requirement still needs to be met. Under these advanced physical layer technologies, either minimizing average AoI or guaranteeing AoI deadlines is a very challenging problem.

- **New Age-Based Metrics beyond Existing AoI** Although AoI is a well-recognized metric to quantify information freshness, it has its limitations, especially for some specific applications. For example, AoI will increase with time if no new information samples arrive at the destination, regardless of whether the information content changes at the source. If the information content at the source doesn't change, then the destination should still have the freshest information, even though its AoI increases. In this case, the AoI metric does not exactly capture the freshness of information. There have been a few papers that proposed new metrics to improve the AoI definition, such as *Age of Synchronization* (AoS) [14], *Effective AoI* (EAoI) [70], and *Age of Incorrect*

Information (AoII) [98]. Each of them improves the AoI metric from a specific perspective. More research in this area is needed, especially on the design of new schedulers for each of these new metrics.

- **Machine Learning for Real-Time AoI Scheduling** Machine learning (ML) is a very powerful technique and it has been applied to solve problems in various research areas. In the area of wireless communication and networking, designing schedulers based on ML is an active area of research. The advantages of ML-based schedulers include: (i) they do not require a closed-form mathematical formulation, (ii) they can improve their performance adaptively by accumulating experience from the past, and (iii) they are suitable to solve problems with extremely large search space. So it would be important and interesting to explore ML-based scheduling algorithms to solve AoI-related problems, such as those the problems that we studied in Chapters 2 to 6.

Appendices

Appendix A

A Proof of Theorem 2.1

First we introduce an interesting property about $\Delta_i(t)$ for any scheduling algorithm (not specific to Juventas). Denote $y_i(t)$ as a binary indicator on whether or not source node i starts to transmit a sample at time slot t. Over an interval of T time slots, the sum of AoI decrease for source node i at the BS is $\sum_{t=1}^{T} y_i(t)\Delta_i(t)$. On the other hand, A_i^{B} increases by 1 at each time slot if it does not decrease (no new sample received in the time slot). When T becomes large, the sum of increase and decrease for A_i^{B} balance out. We have

$$\lim_{T\to\infty} \frac{\sum_{t=1}^{T} y_i(t)\Delta_i(t)}{\sum_{t=1}^{T} 1} = 1. \tag{A.1}$$

Equation (A.1) is equivalent to

$$\lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} y_i(t)\Delta_i(t) = 1. \tag{A.2}$$

We then consider Juventas. When $L_i \leq M$, under Juventas, each sample finishes its transmission within at most 2 time slots. we define $d_{il}^{\mathrm{c}}(t)$, $d_{il}^{\mathrm{p}}(t)$ and $d_{il}^{\mathrm{n}}(t)$ are the binary variables indicating whether the l-th unit from node i begins its transmission (i.e., the unit is at the first time slot of a sample's transmission), ends its previous transmission (i.e., the unit is at the second time slot of a sample's transmission), or isn't transmitted at time slot t. We have

$$d_{il}^{\mathrm{c}}(t) + d_{il}^{\mathrm{p}}(t) + d_{il}^{\mathrm{n}}(t) = 1, \quad \forall i, l, t. \tag{A.3}$$

From (2.17), at each time slot nodes $1, 2, \cdots, K$ can use at most $M - M_K$ transmission units,

so there are at least M_K transmission units for nodes $K+1, K+2, \cdots, N$ to use. Therefore we have

$$\sum_{i=K+1}^{N} \sum_{l=1}^{L_i} (d_{il}^{\mathrm{c}}(t) + d_{il}^{\mathrm{p}}(t)) \geq M_K, \quad \forall t. \tag{A.4}$$

Recall each sample from source node i consists of L_i units of data. Note that $y_i(t)$ is the binary indicator on whether or not source node i starts to transmit a sample at time slot t. Considering the definition of $d_{il}^{\mathrm{c}}(t)$ and $d_{il}^{\mathrm{p}}(t)$, we have

$$\lim_{T\to\infty} \frac{\sum_{t=1}^{T} L_i y_i(t) \Delta_i(t)}{\sum_{t=1}^{T} \sum_{l=1}^{L_i} (d_{il}^{\mathrm{c}}(t)\Delta_i(t) + d_{il}^{\mathrm{p}}(t)\Delta_i(t-1))} = 1. \tag{A.5}$$

For each node i, considering (A.2) and (A.5), we have

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \sum_{l=1}^{L_i} (d_{il}^{\mathrm{c}}(t)\Delta_i(t) + d_{il}^{\mathrm{p}}(t)\Delta_i(t-1)) = L_i. \tag{A.6}$$

For each node $1 \leq i \leq N$, we define

$$s_i(t) = A_i^{\mathrm{B}}(t+1) - A_i^{\mathrm{s}}(t-2) - 3.$$

Suppose $d_{jl}^{\mathrm{c}}(t) = 1$. For any source node i, at time slot t, there are 2 possibilities:

- The BS has completely received a sample from source node i at the end of t. In this case, we have $s_i(t) \leq 0$.
- The BS has partially received a sample at the end of t or doesn't receive a sample from source node i. In this case, from the principle of Juventas, we have $\sqrt{w_j/L_j}\Delta_j(t) \geq \sqrt{w_i/L_i}\Delta_i(t)$. Notice that $\Delta_i(t) \geq s_i(t)$. We have $\sqrt{w_j/L_j}\Delta_j(t) \geq \sqrt{w_i/L_i}s_i(t)$

So for any j, l, i, t, we have

$$d_{jl}^{\mathrm{c}}(t)\sqrt{\frac{w_j}{L_j}}\Delta_j(t) \geq d_{jl}^{\mathrm{c}}(t)\sqrt{\frac{w_i}{L_i}}s_i(t). \tag{A.7}$$

Suppose $d_{jl}^{\mathrm{p}}(t)=1$. For any source node i, at time slot $t-1$, there are 2 possibilities:

- The BS has completely received a sample from source node i at the end of $(t-1)$. In this case, we have $s_i(t)\leq 0$.
- The BS doesn't receive a sample from source node i at time slot $(t-1)$. In this case, from the principle of Juventas, we have $\sqrt{w_j/L_j}\Delta_j(t-1)\geq\sqrt{w_i/L_i}\Delta_i(t-1)$. Notice that $\Delta_i(t-1)\geq s_i(t)$. We have $\sqrt{w_j/L_j}\Delta_j(t-1)\geq\sqrt{w_i/L_i}s_i(t)$

So for any j,l,i,t, we have

$$d_{jl}^{\mathrm{p}}(t)\sqrt{\frac{w_j}{L_j}}\Delta_j(t-1)\geq d_{jl}^{\mathrm{p}}(t)\sqrt{\frac{w_i}{L_i}}s_i(t). \tag{A.8}$$

Combining (A.7) and (A.8) at each time slot t for each node i, we have

$$s_i(t)\leq\frac{\sum_{j=K+1}^{N}\sqrt{\frac{w_j}{L_j}}\sum_{l=1}^{L_j}(d_{jl}^{\mathrm{c}}(t)\Delta_j(t)+d_{jl}^{\mathrm{p}}(t)\Delta_j(t-1))}{\sqrt{\frac{w_i}{L_i}}\sum_{j=K+1}^{N}\sum_{l=1}^{L_j}(d_{jl}^{\mathrm{c}}(t)+d_{jl}^{\mathrm{p}}(t))}. \tag{A.9}$$

Combining (A.9) with (A.4) and (A.6), we have

$$\bar{A}_i^{\mathrm{B}}\leq\bar{A}_i^{\mathrm{s}}+3+\sqrt{\frac{L_i}{w_i}}\frac{\sum_{j=K+1}^{N}\sqrt{w_jL_j}}{M_K}. \tag{A.10}$$

For nodes $i>K$, adding all i's in (A.10) together, we have

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T}\sum_{i=K+1}^{N}w_iA_i^{\mathrm{B}}(t)\leq\frac{1}{M_K}(\sum_{i=K+1}^{N}\sqrt{w_iL_i})^2+\sum_{i=K+1}^{N}w_i(\bar{A}_i^{\mathrm{s}}+3). \tag{A.11}$$

For each node $i\leq K$, considering (A.10) and (2.18), we have

$$\bar{A}_i^{\mathrm{B}}\leq\bar{A}_i^{\mathrm{s}}+4\quad(1\leq i\leq K). \tag{A.12}$$

By combining (A.11) and (A.12), we can get the performance guarantee under Juventas

$$\begin{aligned}
\bar{A}^{\mathrm{B}} &\leq \frac{1}{M_K}\Big(\sum_{i=K+1}^{N}\sqrt{w_i L_i}\Big)^2 + \sum_{i=K+1}^{N} w_i(\bar{A}_i^{\mathrm{s}}+3) + \sum_{i=1}^{K} w_i(\bar{A}_i^{\mathrm{s}}+4) \\
&= 2\alpha_{(\mathrm{PTS},M)} + \alpha_{(\mathrm{ARB},\infty)} + \sum_{i=1}^{N} w_i \\
&\leq 3\alpha_{(\mathrm{ARB},M)} + \sum_{i=1}^{N} w_i \leq 3\bar{A}^{*} + \sum_{i=1}^{N} w_i.
\end{aligned}$$

This completes our proof of Theorem 2.1. □

Appendix B

A Proof of Theorem 5.1

The proof for the "only if" part follows directly from Lemma 5.1. So our proof focuses on the "if" part, i.e., an AUS π is feasible if $r_i \geq r_i^{\text{lb}}$ for each $i = 1, 2, \cdots, N$. In other words, we need to show that if $r_i \geq r_i^{\text{lb}}$, then (5.5) will be satisfied for source i.

By Definition 5.2, under AUS we have an integer b_i such that the interval length between any two consecutive transmissions of source i is either b_i or $(b_i + 1)$. We consider two cases.

- When $b_i < d_i$, we have $b_i + 1 \leq d_i$. Therefore, under AUS for any time slot t, within interval $[t-d_i, t-1]$ there will be at least one transmission for source i. Since the success probability for this transmission is $(1 - p_i)$, for any t we have $\mathbb{P}\{A_i(t) \leq d_i\} \geq 1 - p_i$. Therefore, we have
$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} [A_i(t) \leq d_i] \geq 1 - p_i,$$
which is equivalent to
$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} [A_i(t) > d_i] \leq p_i.$$
Since $\epsilon_i \geq p_i$ (in the stable tolerance case), we have
$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} [A_i(t) > d_i] \leq \epsilon_i.$$

- When $b_i \geq d_i$, under AUS for any time slot t, within interval $[t, t + d_i - 1]$ there must be at most one transmission for source i. Then for each t with $\pi_i(t) = 1$, we have $\pi_i(t+1) = \pi_i(t+2) = \cdots = \pi_i(t+d_i-1) = 0$, and $\mathbb{P}\{A_i(t+1) \leq d_i\} = \mathbb{P}\{A_i(t+2) \leq$

$d_i\} = \cdots = \mathbb{P}\{A_i(t+d_i) \leq d_i\} = (1-p_i)$. Therefore, each scheduled transmission for source i will benefit the following d_i time slots, i.e., make the probability that the AoI is no greater than the deadline $(1-p_i)$ on the following d_i time slots. Considering the scheduling rate r_i, for a large time window T, the number of time slots when the AoI is no greater than the deadline is $Tr_i(1-p_i)d_i$, i.e,

$$\lim_{T\to\infty} \sum_{t=1}^{T} [A_i(t) \leq d_i] = Tr_i(1-p_i)d_i.$$

Dividing both sides by T, we have

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} [A_i(t) \leq d_i] = r_i(1-p_i)d_i,$$

which is equivalent to

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} [A_i(t) > d_i] = 1 - r_i(1-p_i)d_i.$$

Since $r_i \geq r_i^{\text{lb}} = \frac{1-\epsilon_i}{(1-p_i)d_i}$, we have

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} [A_i(t) > d_i] \leq 1 - (1-p_i)d_i \cdot \frac{1-\epsilon_i}{(1-p_i)d_i} = \epsilon_i.$$

Combining both cases, we have completed the proof for the "if" part of the theorem. This completes our proof of Theorem 5.1. □

Appendix C

A Proof of Lemma 5.2

Denote π as the scheduler constructed by Algorithm 5.1. We will prove that under π, for each source i, after executing line 11, the transmission of source i is almost uniform (i.e., the interval lengths for source i differ by at most 1).

First, we will prove the following result: In Algorithm 5.1, after allocating the time slots to source i (i.e., after executing line 9 for source i), if we remove all the empty (unassigned) slots in the EUS cycle, the transmission of source i is exactly uniform (i.e., with exactly identical interval lengths).

The above result holds trivially when $i = 1$. When $i \geq 2$, consider any j with $1 \leq j < i$, we have $N_j/N_i \in \mathbb{N}^*$. Since the scheduler is exactly uniform (after executing line 9 for source i), there are $N_j/N_i \in \mathbb{N}^*$ time slots that have been assigned to source j between any two transmission of source i. Therefore, after executing line 9 for source i, if we remove all the empty (unassigned) slots in the EUS cycle, between any two transmission of source i there are $\sum_{j=1}^{i-1} N_j/N_i$ time slots that have been assigned to other sources. That is, the transmission of source i is exactly uniform. The above result has been proved.

Second, we will prove the following result: In Algorithm 5.1, for each source i, if we stop the algorithm at any j with $j > i$ (i.e., after executing line 9 for source j) and remove all the empty (unassigned) slots in the EUS cycle, the transmission of source i is almost uniform (i.e., the interval lengths for source i differ by at most 1).

The above result holds trivially when $i = N$. When $i < N$, we know that after allocating

time slots to source i (executing line 9 for source i, in the EUS, the numbers of empty slots between any two adjacent transmissions are identical. Before allocating time slots to source $(i+1)$, consider the set $\mathcal{S}_i$ in line 7. By design, we have $|\mathcal{S}_i| = N_i$. These N_i empty time slots are uniformly distributed in the N_i intervals (between transmissions of source i), i.e., one interval with one such empty time slot. By design, the time slot allocation for source $i+1, i+2, \cdots$ will first use these N_i slots in $\mathcal{S}_i$, before using any other empty slots. Therefore, if we stop at any j before all these N_i slot has been allocated, the numbers of empty slots between any two transmissions of i will differ by at most 1. This means the transmission of source i is almost uniform. If we don't stop before that, since $N_x/N_y \in \mathbb{N}^*$ for all $x < y$, we will just complete the allocation for one source node when all these N_i slot has been allocated, and now the transmission for source i is exactly uniform again. Then, for the new set $\mathcal{S}_i$ in line 7, we have $|\mathcal{S}_i| = N_i$ and these N_i empty slots are uniformly distributed in the N_i intervals. If we stop at any time instance before all these N_i slot has been allocated, the transmission for source i is almost uniform. For all further time instances we can do the same. Therefore, at any time instance, if we stop and remove all empty slots, the transmission for source i is almost uniform.

The above result holds when we stop at $j = N$. So after executing Algorithm 5.1, the transmission for any source i is almost uniform. This completes our proof of Lemma 5.2. □

Appendix D

An Algorithm to Solve OPT-β

We first transform the decision variable $\boldsymbol{r}$ to its sorted version γ, to make the problem easier. We define $\boldsymbol{\gamma}^{\mathrm{LB}} = \mathrm{sort}(\boldsymbol{r}^{\mathrm{LB}})$. It can be shown that in the optimal solution $\boldsymbol{r}^*$ to OPT-β, the order of elements should be the same as the order of elements. That is, for any $i \neq j$, if $r_i^{\mathrm{LB}} > r_j^{\mathrm{LB}}$, then we have $r_i^* > r_j^*$. Therefore, it's sufficient to solve the following OPT-β $(\boldsymbol{\gamma})$ with variable $\boldsymbol{\gamma}$ (rather than $\boldsymbol{r}$):

$$\begin{aligned} \text{OPT-}\beta\ (\boldsymbol{\gamma}):\ \max_{\boldsymbol{\gamma},\beta}\quad & \beta \\ \text{s.t.}\quad & \boldsymbol{\gamma} \text{ is step-down,} \\ & \beta \cdot \gamma_i^{\mathrm{LB}} \leq \gamma_i \leq 1,\ i = 1, 2, \cdots, N, \\ & \sum_{i=1}^{N} \gamma_i = W. \end{aligned}$$

After obtaining $\boldsymbol{\gamma}^*$ to OPT-β $(\boldsymbol{\gamma})$, we can re-sort it and obtain $\boldsymbol{r}^*$, the optimal solution to OPT-β.

We notice that there are two variables $\boldsymbol{\gamma}$ and β in OPT-β $(\boldsymbol{\gamma})$, which are hard for us to handle at the same time. So we propose to fix β first, and transform OPT-β $(\boldsymbol{\gamma})$ to a decision problem DEC-$\boldsymbol{\gamma}$, for whether or not there exists a feasible $\boldsymbol{\gamma}$ to OPT-β $(\boldsymbol{\gamma})$.

$$\text{DEC-}\boldsymbol{\gamma}\text{: Determine whether or not } \boldsymbol{\gamma} \text{ exists.}$$
$$\begin{aligned} \text{s.t.} \quad & \boldsymbol{\gamma} \text{ is step-down,} \\ & \beta \cdot \gamma_i^{\mathrm{LB}} \le \gamma_i \le 1, \ i = 1, 2, \cdots, N, \\ & \sum_{i=1}^{N} \gamma_i = W. \end{aligned}$$

If we have an algorithm to solve DEC-$\boldsymbol{\gamma}$ under fixed β, we could use bisection to find the optimal β^* to OPT-β ($\boldsymbol{\gamma}$).

To solve DEC-$\boldsymbol{\gamma}$, we further transform it to an easier optimization problem OPT-$\hat{\boldsymbol{\gamma}}$. we define $l_i = \beta \cdot \gamma_i^{\mathrm{LB}}$ for each $i = 1, 2, \cdots, N$. Clearly, we have $l_1 \ge l_2 \ge \cdots \ge l_N$. Then for fixed β, we consider the following optimization problem with objective $\hat{\boldsymbol{\gamma}}$.

$$\begin{aligned} \text{OPT-}\hat{\boldsymbol{\gamma}}\text{:} \quad \min_{\hat{\boldsymbol{\gamma}}} \quad & \sum_{i=1}^{N} \hat{\gamma}_i \\ \text{s.t.} \quad & \hat{\boldsymbol{\gamma}} \text{ is step-down,} \\ & l_i \le \hat{\gamma}_i \le 1, \ i = 1, 2, \cdots, N. \end{aligned}$$

Denote $\hat{\boldsymbol{\gamma}}^*$ as the optimal objective to OPT-$\hat{\boldsymbol{\gamma}}$. If $\sum_{i=1}^{N} \hat{\gamma}_i^* \le W$, then $\boldsymbol{\gamma} = \hat{\boldsymbol{\gamma}}^* \cdot W / (\sum_{i=1}^{N} \hat{\gamma}_i^*)$ is a feasible solution to DEC-$\boldsymbol{\gamma}$. On the other hand, if $\sum_{i=1}^{N} \hat{\gamma}_i^* \le W$, then there does not exist any feasible solution to DEC-$\boldsymbol{\gamma}$ for the fixed β.

OPT-$\hat{\boldsymbol{\gamma}}$ can be solved by a DP-based solution (see Algorithm 5.4 in Chapter 5). We only need the following minor change. In the step-down definition (Definition 7.2) in Chapter. 7, we allow the leading elements in γ to be 1, which is slightly different form the step-down definition in Chapter. 5. As a result, to solve OPT-$\hat{\boldsymbol{\gamma}}$, we need to set $\mathcal{R}_i = \{1\} \cup \{n\hat{r}_k \mid r_i^{\mathrm{lb}} \le n\hat{r}_k \le 1, n \in \mathbb{N}^*\}$ in Step 5 of Algorithm 5.4.

A pseudocode for a bisection-based solution to solve OPT-β is given in Algorithm D.1. Note that ϵ is an input parameter which stands for error tolerance of the objective β, with

Algorithm D.1 A Bisection-Based Solution to OPT-β

Input: $\boldsymbol{r}^{\mathrm{LB}}$, W, ϵ.

Output: β^* and $\boldsymbol{r}^*$.

1: Sort $\boldsymbol{r}^{\mathrm{LB}}$ and get $\boldsymbol{\gamma}^{\mathrm{LB}} = \mathrm{sort}(\boldsymbol{r}^{\mathrm{LB}})$. Set $\beta_{\mathrm{LB}} = 0$, $\beta_{\mathrm{UB}} = 1$.

2: **while** $\beta_{\mathrm{UB}} - \beta_{\mathrm{LB}} > \epsilon$ **do**

3: Set $\beta = (\beta_{\mathrm{UB}} + \beta_{\mathrm{LB}})/2$.

4: Set $l_i = \beta \cdot \gamma_i^{\mathrm{LB}}$ for each $i = 1, 2, \cdots, N$. Solve OPT-$\hat{\boldsymbol{\gamma}}$ (by Algorithm 5.4) and obtain an optimal solution $\hat{\boldsymbol{\gamma}}^*$.

5: **if** $\sum_{i=1}^{N} \hat{\gamma}_i^* \leq W$ **then**

6: Set $\boldsymbol{\gamma^*} = \hat{\boldsymbol{\gamma}}^* \cdot W/(\sum_{i=1}^{N} \hat{\gamma}_i^*)$. Set $\beta^* = \beta$. Set $\beta_{\mathrm{LB}} = \beta$.

7: **else**

8: Set $\beta_{\mathrm{UB}} = \beta$.

9: **end if**

10: **end while**

11: Re-sort $\boldsymbol{\gamma}^*$ and get $\boldsymbol{r}^*$.

$\epsilon << 1$.

Appendix E

A Proof of Theorem 7.1

Before proving Theorem 7.1, we first introduce the following lemma w.r.t. $\hat{\boldsymbol{\gamma}}^*$,the optimal objective to OPT-$\hat{\boldsymbol{\gamma}}$.

Lemma E.1. $\sum_{i=1}^{N} \hat{\gamma}_i^* \leq \log_2 e \cdot \sum_{i=1}^{N} l_i$.

Proof. We proof Lemma E.1 by contradiction. Suppose $\sum_{i=1}^{N} \hat{\gamma}_i^* > \log_2 e \cdot \sum_{i=1}^{N} l_i$. Then for any $\hat{\boldsymbol{\gamma}}$ that is feasible to OPT-$\hat{\boldsymbol{\gamma}}$, we have $\sum_{i=1}^{N} \hat{\gamma}_i > \log_2 e \cdot \sum_{i=1}^{N} l_i$.

For any $x > 0$, we define two vectors $\hat{\boldsymbol{\gamma}}^x = [\hat{\gamma}_1^x, \hat{\gamma}_2^x, \cdots, \hat{\gamma}_N^x]$ and $\boldsymbol{\gamma}^x = [\gamma_1^x, \gamma_2^x, \cdots, \gamma_N^x]$ as:

$$\hat{\gamma}_i^x = \min\{1, x \cdot 2^{\lceil \log_2(\frac{l_i}{x}) \rceil}\},\ i = 1, 2, \cdots, N. \tag{E.1}$$

$$\gamma_i^x = x \cdot 2^{\lceil \log_2(\frac{l_i}{x}) \rceil},\ i = 1, 2, \cdots, N. \tag{E.2}$$

It can be verified that $\hat{\boldsymbol{\gamma}}^x$ is feasible to OPT-$\hat{\boldsymbol{\gamma}}$. Therefore, for any $x > 0$ we have

$$\sum_{i=1}^{N} \hat{\gamma}_i^x > \log_2 e \cdot \sum_{i=1}^{N} l_i. \tag{E.3}$$

For each i, we have $\gamma_i^x \geq \hat{\gamma}_i^x$. So for any $x > 0$, we have

$$\sum_{i=1}^{N} \gamma_i^x > \log_2 e \cdot \sum_{i=1}^{N} l_i. \tag{E.4}$$

Define $g(x) = \frac{1}{x}$. We have

$$\int_{0.5}^{1} g(x)dx = \frac{1}{\log_2 e}. \tag{E.5}$$

Multiplying the two sides of (E.4) and (E.5) respectively, we have

$$\int_{0.5}^{1} g(x)dx \cdot \sum_{i=1}^{N} \gamma_i^x > \sum_{i=1}^{N} l_i. \tag{E.6}$$

Substituting $g(x) = \frac{1}{x}$ and (E.2) into (E.6), we have

$$\sum_{i=1}^{N} \int_{0.5}^{1} 2^{\lceil \log_2(\frac{l_i}{x}) \rceil} dx > \sum_{i=1}^{N} l_i. \tag{E.7}$$

It can be shown that the LHS of (E.7) equals to $\sum_{i=1}^{N} l_i$. So we have

$$\sum_{i=1}^{N} l_i > \sum_{i=1}^{N} l_i. \tag{E.8}$$

Clearly, (E.8) can not hold (i.e., a contradiction). This completes our proof of Lemma E.1. □

With Lemma E.1 in hands, we can further prove Lemma E.2 w.r.t. β^*, the optimal

solution to OPT-β. Note that Lemma E.2 holds when ϵ is sufficiently small (otherwise, Algorithm D.1 will terminate too early).

Lemma E.2. *As $\epsilon \to 0$, we can always have $\beta^* \geq \frac{1}{\log_2 e}$.*

Proof. We prove Lemma E.2 by contradiction. Suppose $\beta^* < \frac{1}{\log_2 e}$. Then for $\hat{\boldsymbol{\gamma}}^*$, the optimal objective to OPT-$\hat{\boldsymbol{\gamma}}$, we have

$$\sum_{i=1}^{N} \hat{\gamma}_i^* \geq W, \tag{E.9}$$

otherwise β^* is not the optimal objective to OPT-β. On the other hands, by Lemma E.1, we have

$$\sum_{i=1}^{N} \hat{\gamma}_i^* \leq \log_2 e \cdot \sum_{i=1}^{N} \beta^* \cdot \gamma_i^{\mathrm{LB}}. \tag{E.10}$$

Since $\boldsymbol{\gamma}$ is feasible to OPT-LB, we have $\sum_{i=1}^{N} \gamma_i^{\mathrm{LB}} \leq W$. Considering (E.10) and $\beta^* < \frac{1}{\log_2 e}$, we have

$$\sum_{i=1}^{N} \hat{\gamma}_i^* < W, \tag{E.11}$$

which contradicts to (E.9). This completes our proof of Lemma E.2. □

With Lemma E.2 in hands, now we can offer a proof of Theorem 1. Note that in Algorithm D.1, we must set ϵ small enough to make sure $\beta^* \geq \frac{1}{\log_2 e}$.

Proof. Considering $\bar{A}_{\mathrm{LB}} \leq \bar{A}^*$, to prove (7.16), it's sufficient to prove

$$\sum_{i=1}^{N} w_i \bar{A}_i \leq (1 + p_{\max}) \log_2 e \cdot \bar{A}_{\mathrm{LB}}. \tag{E.12}$$

Define $f(x) = 2\lfloor \frac{1}{x} \rfloor + 1 - x(\lfloor \frac{1}{x} \rfloor^2 + \lfloor \frac{1}{x} \rfloor)$. Then we have $q_i(r_i) = \frac{1+p_i}{1-p_i} f(r_i) + \frac{1}{2}$. Following the discussions in Section 7.3.3, it can be shown that $q_i(r_i) \geq \bar{A}_i$ for each i. Therefore, for the AUS with $\boldsymbol{r}^*$, to prove (E.12), it's sufficient to prove

$$\sum_{i=1}^{N} \frac{w_i(1+p_i)}{2(1-p_i)} f(r_i^*) + \sum_{i=1}^{N} \frac{w_i}{2} \leq (1+p_{\max}) \log_2 e \cdot \bar{A}_{\text{LB}}. \tag{E.13}$$

Considering (7.13), to prove (E.13), it's sufficient to prove

$$\frac{w_i(1+p_i)}{2(1-p_i)} f(r_i^*) + \frac{w_i}{2} \leq (1+p_{\max}) \log_2 e \cdot \left(\frac{w_i}{2(1-p_i) r_i^{\text{LB}}} + \frac{w_i}{2}\right)$$

for each $i = 1, 2, \cdots, N$. Since $p_{\max} \geq p_i$, it's sufficient to prove

$$\frac{1+p_i}{1-p_i} f(r_i^*) + 1 \leq (1+p_i) \log_2 e \cdot \left(\frac{1}{(1-p_i) r_i^{\text{LB}}} + 1\right).$$

By Lemma E.2, we have $r_i^* = \beta^* \cdot r_i^{\text{LB}} \geq \frac{1}{\log_2 e} \cdot r_i^{\text{LB}}$. So it's sufficient to prove

$$\frac{1+p_i}{1-p_i} f(r_i^*) + 1 \leq \frac{1+p_i}{(1-p_i) r_i^*} + (1+p_i) \log_2 e,$$

which is equivalent to

$$f(r_i^*) - \frac{1}{r_i^*} \leq (1-p_i) \log_2 e - \frac{1-p_i}{1+p_i}. \tag{E.14}$$

Recall $f(x) = 2\lfloor \frac{1}{x} \rfloor + 1 - x(\lfloor \frac{1}{x} \rfloor^2 + \lfloor \frac{1}{x} \rfloor)$. It can be proved that $f(x) - \frac{1}{x} \leq 3 - 2\sqrt{2}$ for all $0 < x < 1$ (the equality holds when $x = \frac{\sqrt{2}}{2}$). So the LHS of (E.14) is no greater than

$3-2\sqrt{2}=0.172$. On the other hand, the RHS of (E.14) is no less than $0.193*\log_2 e-\frac{0.193}{1.807}=0.172$ when $p_i \leq 0.807$. Therefore, (E.14) always holds. This completes our proof. □

Appendix F

An Algorithm to Solve OPT-SD (Min-BW)

Similar to what we we did in Appendix D, to solve OPT-SD (Min-BW), we first transform $\boldsymbol{r}$ to its sorted version $\boldsymbol{\gamma}$. Here we sort $\boldsymbol{d}$ in non-decreasing order. Then we solve problem OPT-$\hat{\boldsymbol{\gamma}}$ by letting $l_i = \frac{1}{d_i}$ for each $i = 1, 2, \cdots, N$ and let $W^* = \lceil \sum_{i=1}^{N} \hat{\gamma}_i^* \rceil$. A pseudocode for a solution to solve OPT-SD (Min-BW) is given in Algorithm F.1.

Algorithm F.1 A Solution to OPT-SD (Min-BW)

Input: $\boldsymbol{d}$.

Output: W and $\boldsymbol{r}^*$.

1: Sort $\boldsymbol{d}$ in non-decreasing order.

2: Set $l_i = \frac{1}{d_i}$ for each $i = 1, 2, \cdots, N$. Solve OPT-$\hat{\boldsymbol{\gamma}}$ (by Algorithm 5.4) and obtain an optimal solution $\hat{\boldsymbol{\gamma}}^*$.

3: Set $W = \lceil \sum_{i=1}^{N} \hat{\gamma}_i^* \rceil$. Set $\boldsymbol{\gamma^*} = \hat{\boldsymbol{\gamma}}^* \cdot W / (\sum_{i=1}^{N} \hat{\gamma}_i^*)$.

4: Re-sort $\boldsymbol{\gamma}^*$ and get $\boldsymbol{r}^*$.

Appendix G

A Proof of Theorem 7.2

First we develop a lower bound for the optimal objective W^*. For each source node i, to satisfy (7.17) for all t, there must be at least one transmission within any consecutive d_i time slots. Therefore,to satisfy (7.17), the transmission rate r_i must satisfy:

$$r_i \geq \frac{1}{d_i}. \tag{G.1}$$

Considering (G.1) and (7.4), we have

$$W^* \geq \sum_{i=1}^{N} \frac{1}{d_i}. \tag{G.2}$$

Then we derive an upper bound for the bandwidth W found by Eywa. By Lemma E.1, we have

$$\sum_{i=1}^{N} \hat{\gamma}_i^* \leq \log_2 e \cdot \sum_{i=1}^{N} \frac{1}{d_i}. \tag{G.3}$$

From Step 3 in Algorithm F.1, we have

$$W \leq \lceil \log_2 e \cdot \sum_{i=1}^{N} \frac{1}{d_i} \rceil. \tag{G.4}$$

Combining (G.2) and (G.4), we can complete the proof of Theorem 7.2. □

Appendix H

A Performance Guarantee for Aion in [102]

In [102, Theorem 2], Liu *et al.* proved that

$$\frac{W_{\text{Aion}}}{W^*} \leq \frac{\left\lceil \sum_{i=1}^{N} \frac{1}{l_i^*} \right\rceil}{\left\lceil \sum_{i=1}^{N} \frac{1}{d_i} \right\rceil}, \tag{H.1}$$

where W_{Aion} is the bandwidth achieved by their proposed algorithm Aion and the vector $\boldsymbol{l}^* = [l_1^* \; l_2^* \; \cdots \; l_N^*]$ is the optimal solution of the following problem:

$$\min_{l_1, l_2, \cdots, l_N \in \mathbb{R}} \quad \sum_{i=1}^{N} \frac{1}{l_i} \tag{H.2a}$$

$$\text{s.t.} \quad 1 \leq l_i \leq d_i, \text{ for } i \in \{1, 2, \cdots, N\}, \tag{H.2b}$$

$$\frac{l_i}{l_{i-1}} \in \mathbb{Z}^+, \text{ for } i \in \{2, 3, \cdots, N\}, \tag{H.2c}$$

$$l_i \in \mathbb{R}^+, \text{ for } i \in \{1, 2, \cdots, N\}. \tag{H.2d}$$

We note that it is easy to prove

$$\frac{W_{\text{Aion}}}{W^*} \leq \frac{\left\lceil \sum_{i=1}^{N} \frac{1}{l_i^*} \right\rceil}{\left\lceil \sum_{i=1}^{N} \frac{1}{d_i} \right\rceil} \leq 2 \tag{H.3}$$

as follows: Consider the vector $\boldsymbol{l}' = [l_1' \; l_2' \; \cdots \; l_N']$ where $l_i' = 2^{\lfloor \log_2 d_i \rfloor}$ for each $i = 1, 2, \cdots, N$.

Clearly, $\boldsymbol{l}'$ is a feasible solution of (H.2). As $\boldsymbol{l}^*$ is the optimal solution of (H.2), it holds that

$$\begin{aligned}\frac{\left\lceil \sum_{i=1}^{N} \frac{1}{l_i^*} \right\rceil}{\left\lceil \sum_{i=1}^{N} \frac{1}{d_i} \right\rceil} &\leq \frac{\left\lceil \sum_{i=1}^{N} \frac{1}{l_i'} \right\rceil}{\left\lceil \sum_{i=1}^{N} \frac{1}{d_i} \right\rceil} = \frac{\left\lceil \sum_{i=1}^{N} \frac{1}{2^{\lfloor \log_2 d_i \rfloor}} \right\rceil}{\left\lceil \sum_{i=1}^{N} \frac{1}{d_i} \right\rceil} \\ &\leq \frac{\left\lceil \sum_{i=1}^{N} \frac{1}{2^{(\log_2 d_i - 1)}} \right\rceil}{\left\lceil \sum_{i=1}^{N} \frac{1}{d_i} \right\rceil} = \frac{\left\lceil 2 \cdot \sum_{i=1}^{N} \frac{1}{d_i} \right\rceil}{\left\lceil \sum_{i=1}^{N} \frac{1}{d_i} \right\rceil} \leq 2,\end{aligned}$$

so we have $W_{\text{Aion}} \leq 2 \cdot W^*$. □

Appendix I

A Proof of Theorem 7.3

First we develop a lower bound for the optimal objective W^*. Recall that for any scheduler, we have (7.12). So to satisfy (7.20), we must have

$$r_i \geq \frac{1}{(1-p_i)(2\alpha_i - 1)},\ i = 1, 2, \cdots, N. \tag{I.1}$$

Considering (I.1) and (7.4), we have

$$W^* \geq \sum_{i=1}^{N} \frac{1}{(1-p_i)(2\alpha_i - 1)}. \tag{I.2}$$

Then we derive an upper bound for the bandwidth W found by Eywa. Define function $g(\cdot)$ as:

$$h(x) = \frac{2\lfloor x \rfloor + 1 - x}{\lfloor x \rfloor^2 + \lfloor x \rfloor}. \tag{I.3}$$

It can be shown that for all $x > 1$, we have

$$h(x) \leq \frac{9}{8x}, \tag{I.4}$$

where the equality holds when $x = \frac{3}{2}$. In OPT-$\hat{\boldsymbol{\gamma}}$, we have

$$\sum_{i=1}^{N} l_i = \sum_{i=1}^{N} h\big(\frac{(1-p_i)(2\alpha_i - 1)}{1+p_i}\big), \tag{I.5}$$

so we have

$$\sum_{i=1}^{N} l_i \leq \frac{9}{8} \cdot \sum_{i=1}^{N} \frac{(1-p_i)(2\alpha_i - 1)}{1+p_i}. \tag{I.6}$$

By Lemma E.1, we have

$$\sum_{i=1}^{N} \hat{\gamma}_i^* \leq \log_2 e \cdot \frac{9}{8} \cdot \sum_{i=1}^{N} \frac{(1-p_i)(2\alpha_i - 1)}{1+p_i}. \tag{I.7}$$

From Step 3 in Algorithm F.1, we have

$$W \leq \left\lceil \log_2 e \cdot \frac{9}{8} \cdot \sum_{i=1}^{N} \frac{(1-p_i)(2\alpha_i - 1)}{1+p_i} \right\rceil. \tag{I.8}$$

Combining (I.2) and (I.8), we can complete the proof of Theorem 7.3. □

Appendix J

An Algorithm to Solve DEC-SD

Similar to what we we did in Appendix D, to solve DEC-SD, we first transform $\boldsymbol{r}$ to its sorted version $\boldsymbol{\gamma}$. Here we sort $\boldsymbol{d}$ in non-decreasing order. Then we solve problem OPT-$\hat{\boldsymbol{\gamma}}$ by letting $l_i = \frac{1}{d_i}$ for each $i = 1, 2, \cdots, N$ and obtain an optimal solution $\hat{\boldsymbol{\gamma}}^*$. IF $\sum_{i=1}^{N} \hat{\gamma}_i^* \leq W$, then there exists a feasible $\boldsymbol{r}$ to DEC-SD. Otherwise, there does not. A pseudocode for a solution to solve DEC-SD is given in Algorithm J.1.

Algorithm J.1 A Solution to DEC-SD

Input: $\boldsymbol{d}$, W.

Output: Whether or not a feasible $\boldsymbol{r}$ exists. If yes, find it.

1: Sort $\boldsymbol{d}$ in non-decreasing order.

2: Set $l_i = \frac{1}{d_i}$ for each $i = 1, 2, \cdots, N$. Solve OPT-$\hat{\boldsymbol{\gamma}}$ (by Algorithm 5.4) and obtain an optimal solution $\hat{\boldsymbol{\gamma}}^*$.

3: **if** $\sum_{i=1}^{N} \hat{\gamma}_i^* \leq W$ **then**

4: Determine that there exist a feasible $\boldsymbol{r}$. Construct a step-down $\boldsymbol{\gamma}$ with $\sum_{i=1}^{N} \gamma_i = W$ by letting some leading elements in $\hat{\boldsymbol{\gamma}}^*$ be 1. Re-sort $\boldsymbol{\gamma}$ and get the feasible $\boldsymbol{r}$ to DEC-SD.

5: **else**

6: Determine that there does not exist a feasible $\boldsymbol{r}$.

7: **end if**

Bibliography

[1] L. Kong, M.K. Khan, F. Wu, G. Chen, and P. Zeng, "Millimeter-Wave Wireless Communications for IoT-Cloud Supported Autonomous Vehicles: Overview, Design, and Challenges," *IEEE Communications Magazine,* vol. 55, issue. 1, pp. 62–68, Jan. 2017.

[2] N.H. Motlagh, M. Bagga, T. Taleb, "UAV-based IoT Platform: A Crowd Surveillance Use Case," *IEEE Communications Magazine,* vol. 55, issue. 2, pp. 128–134, Feb. 2017.

[3] P. Schulz, M. Matthé, H. Klessig, M. Simsek, G. Fettweis, J. Ansari, S.A. Ashraf, B. Almeroth, J. Voigt, I. Riedel, A. Puschmann, A. Mitschele-Thiel, M. Müller, T. Elste, and M. Windisch, "Latency Critical IoT Applications in 5G: Perspective on the Design of Radio Interface and Network Architecture," *IEEE Communications Magazine,* vol. 55, issue 2, pp. 70–78, Feb. 2017.

[4] V.C. Gungor, D. Sahin, T. Kocak, S. Ergut, C. Buccella, C. Cecati, and G.P. Hancke, "A Survey on Smart Grid Potential Applications and Communication Requirements," *IEEE Transactions on Industrial Informatics,* vol. 9, issue 1, pp. 28–42, Feb. 2013.

[5] S. Kaul, M. Gruteser, V. Rai, and J. Kenney, "Minimizing Age of Information in Vehicular Networks," in *Proc. of IEEE SECON,* pp. 350–358, Salt Lake City, UT, USA, June 27–30, 2011.

[6] S. Kaul, R. Yates, and M. Gruteser, "Real-Time Status: How Often Should One Update?" in *Proc. of IEEE INFOCOM,* pp. 2731–2735, Orlando, FL, USA, Mar. 25–30, 2012.

[7] A. Kosta, N. Pappas, and V. Angelakis, "Age of Information: A New Concept, Metric,

and Tool," *Foundations and Trends in Networking,* vol. 12, issue 3, pp. 162–259, Nov. 2017, Now Publishers, Inc, ISBN-13: 978-1680833607.

[8] Y. Sun, "A Collection of Recent Papers on the Age of Information," available at http://www.auburn.edu/%7eyzs0078 [Online; accessed on 2021-12-2].

[9] Y. Hsu, E. Modiano, and L. Duan, "Age of Information: Design and Analysis of Optimal Scheduling Algorithms," in *Proc. of IEEE ISIT,* pp. 561–565, Archen, Germany, June 25–30, 2017.

[10] I. Kadota, E. Uysal-Biyikoglu, R. Singh, and E. Modiano, "Minimizing the Age of Information in Broadcast Wireless Networks," in *Proc. of Allerton Conference,* pp. 844–851, Monticello, IL, USA, Sept. 27–30, 2016.

[11] I. Kadota, A. Sinha, and E. Modiano, "Optimizing Age of Information in Wireless Networks with Throughput Constraints," in *Proc. of IEEE INFOCOM,* pp. 1844–1852, Honolulu, HI, USA, Apr. 16–18, 2018

[12] 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 Trans. on Information Theory,* vol. 63, issue 11, pp. 7492–7508, Nov. 2017.

[13] R.D. Yates, P. Ciblat, A. Yener, and M. Wigger, "Age-Optimal Constrained Cache Updating," in *Proc. of IEEE ISIT,* pp. 141–145, Archen, Germany, June 25–30, 2017.

[14] J. Zhong, R.D. Yates, and E. Soljanin, "Two Freshness Metrics for Local Cache Refresh," in *Proc. of IEEE ISIT,* pp. 1924–1928, Vail, CO, USA, June 17–22, 2018.

[15] M. Costa, M. Codreanu, and A. Ephremides, "Age of Information with Packet Management," in *Proc. of IEEE ISIT,* pp. 1583–1587, Honolulu, HI, USA, June 29–July 4, 2014.

[16] M. Costa, M. Codreanu, and A. Ephremides, "On the Age of Information in Status Update Systems with Packet Management," *IEEE Trans. on Information Theory,* vol. 62, issue 4, pp. 1897–1910, Apr. 2016.

[17] L. Huang and E. Modiano, "Optimizing Age-of-Information in a Multi-class Queueing System," in *Proc. of IEEE ISIT,* pp. 1681–1685, Hong Kong, China, June 14–19, 2015.

[18] Y. Inoue, H. Masuyama, T. Takine, and T. Tanaka, "The Stationary Distribution of the Age of Information in FCFS Single-Server Queues," in *Proc. of IEEE ISIT,* pp. 571–575, Archen, Germany, June 25–30, 2017.

[19] C. Kam, S. Kompella, G.D. Nguyen, and A. Ephremides, "Effect of Message Transmission Path Diversity on Status Age," *IEEE Trans. on Information Theory,* vol. 62, issue 3, pp. 1360–1374, Mar. 2016.

[20] A. Kosta, N. Pappas, A. Ephremides, and V. Angelakis, "Age and Value of Information: Non-Linear Age Case," in *Proc. of IEEE ISIT,* pp. 326–330, Archen, Germany, June 25–30, 2017.

[21] E. Najm and E. Telatar, "Status Updates in a Multi-Stream M/G/1/1 Preemptive Queue," *arXiv preprint,* arXiv:1801.04068, Jan. 2018.

[22] R.D. Yates and S. Kaul, "Real-Time Status Updating: Multiple Sources," in *Proc. of IEEE ISIT,* pp. 2666–2670, Cambridge, MA, USA, July 1–6, 2012.

[23] R.D. Yates, "Status Updates through Networks of Parallel Servers," in *Proc. of IEEE ISIT,* pp. 2281–2285, Vail, CO, USA, June 17–22, 2018

[24] C. Li, S. Li, and Y.T. Hou, "A General Model for Minimizing Age of Information at Network Edge," in *Proc. IEEE INFOCOM,* pp. 118–126, Paris, France, Apr. 29–May 2, 2019.

[25] Q. He, D. Yuan, and A. Ephremides, "Optimal Link Scheduling for Age Minimization in Wireless Systems," *IEEE Trans. on Information Theory,* vol. 64, issue 7, pp. 5381–5394, July, 2018.

[26] C. Joo and A. Eryilmaz, "Wireless Scheduling for Information Freshness and Synchrony: Drift-based Design and Heavy-Traffic Analysis," in *Proc. of WiOpt,* pp. 1–8, Paris, France, May 15–19, 2017.

[27] N. Lu, B. Ji, and B. Li, "Age-based Scheduling: Improving Data Freshness for Wireless Real-Time Traffic," in *Proc. of ACM MobiHoc,* pp. 191–200, Los Angeles, CA, USA, June 26–29, 2018.

[28] R. Talak, S. Karaman, and E. Modiano, "Optimizing Information Freshness in Wireless Networks under General Interference Constraints," in *Proc. of ACM MobiHoc,* pp. 61–70, Los Angeles, CA, USA, June 26–29, 2018.

[29] R. Talak, S. Karaman, and E. Modiano, "Distributed Scheduling Algorithms for Optimizing Information Freshness in Wireless Networks," in *Proc. of IEEE International Workshop on Signal Processing Advances in Wireless Communications,* pp. 1–5, Kalamata, Greece, June 25–28, 2018.

[30] R. Talak, S. Karaman, and E. Modiano, "Optimizing Age of Information in Wireless Networks with Perfect Channel State Information," in *Proc. of WiOpt,* pp. 1–8, Shanghai, China, May 7–11, 2018.

[31] 3GPP TS 38.214 version 15.0.0, "NR; NR and NG-RAN overall description," available at https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3216 [Online; accessed on 2022-12-2].

[32] A.M. Bedewy, Y. Sun, and N.B. Shroff, "Age-Optimal Information Updates in Multihop Networks," in *Proc. of IEEE ISIT,* pp. 576–580, Archen, Germany, June 25–30, 2017.

[33] R. Talak, S. Karaman, and E. Modiano, "Minimizing Age-of-Information in Multi-Hop Wireless Networks," in *Proc. of Allerton Conference,* pp. 486–493, Monticello, IL, USA, Oct. 3–6, 2017.

[34] 3GPP TR 21.101 version 7.0.0, "Physical Layer Aspects for Evolved UTRA," available at https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1247 [Online; accessed on 2022-12-2].

[35] 3GPP TS 38.300 version 15.0.0, "NR; NR and NG-RAN overall description," available at https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3191 [Online; accessed on 2022-12-2].

[36] S.K. Kaul and R.D. Yates, "Age of Information: Updates with Priority," in *Proc. of IEEE ISIT,* pp. 2644–2648, Vail, CO, USA, June 17–22, 2018.

[37] J. Zhong, R.D. Yates, and E. Soljanin, "Multicast with Prioritized Delivery: How Fresh is Your Data?" in *Proc. of IEEE International Workshop on Signal Processing Advances in Wireless Communication,* pp. 1–5, Kalamata, Greece, June 25–28, 2018.

[38] C. Kam, S. Kompella, and A. Ephremides, "Age of Information under Random Updates," in *Proc. of IEEE ISIT,* pp. 66–70, Istanbul, Turkey, July 7–12, 2013.

[39] C. Kam, S. Kompella, G.D. Nguyen, J.E. Wieselthier and A. Ephremides, "Controlling the Age of Information: Buffer Size, Deadline, and Packet Replacement," in *Proc. of IEEE MILCOM,* pp. 301–306, Baltimore, MD, USA, Nov. 1–3, 2016.

[40] S.K. Kaul and R.D. Yates, "Status Updates over Unreliable Multiaccess Channels," in *Proc. of IEEE ISIT,* pp. 331–335, Archen, Germany, June 25–30, 2017.

[41] S. Farazi, A.G. Klein, and D.R. Brown III, "Average Age of Information for Status Update Systems with an Energy Harvesting Server," in *Proc. of IEEE INFOCOM Workshops—Age of Information Workshop,* pp. 112–117, Honolulu, HI, USA, April 16, 2018.

[42] S. Farazi, A.G. Klein, and D.R. Brown III, "Age of Information in Energy Harvesting Status Update Systems: When to Preempt in Service?" in *Proc. of IEEE ISIT,* pp. 2436–2440, Vail, CO, USA, June 17–22, 2018.

[43] A.M. Bedewy, Y. Sun, and N.B. Shroff, "Optimizing Data Freshness, Throughput, and Delay in Multi-Server Information-Update Systems," in *Proc. of IEEE ISIT,* pp. 2569–2573, Barcelona, Spain, July 10–15, 2016.

[44] Y. Sun, E. Uysal-Biyikoglu, and S. Kompella, "Age-Optimal Updates of Multiple Information Flows," in *Proc. of IEEE INFOCOM Workshops—Age of Information Workshop,* pp. 136–141, Honolulu, HI, USA, April 15–19, 2018.

[45] C. Li, Y. Huang, Y. Chen, B. Jalaian, Y.T. Hou, and W. Lou, "Kronos: A 5G Scheduler for AoI Minimization under Dynamic Channel Conditions," in *Proc. IEEE ICDCS,* pp. 1466–1472, Dallas, TX, USA, July 7–9, 2019.

[46] C. Li, S. Li, Y. Chen, Y.T. Hou, and W. Lou, "AoI Scheduling with Maximum Thresholds," in *Proc. IEEE INFOCOM,* pp. 436–445, online conference, July 6–9, 2020.

[47] C. Kam, S. Kompella, G.D. Nguyen, J.E. Wieselthier, and A. Ephremides, "Modeling the Age of Information in Emulated Ad hoc Networks," in *Proc. IEEE MILCOM,* pp. 436–441, Baltimore, MD, USA, Oct. 23–25, 2017.

[48] X. Zheng, S. Zhou, Z. Jiang, and Z. Niu, "Closed-form Analysis of Nonlinear Age of Information in Status Updates with an Energy Harvesting Transmitter," *IEEE Trans. on Wireless Communications,* vol. 18, no. 8, pp. 4129–4142, 2019.

[49] V. Tripathi and E. Modiano, "A Whittle Index Approach to Minimizing Functions of Age of Information" in *Proc. IEEE Allerton,* pp. 1160–1167, Monticello, IL, USA, Sept. 24–27, 2019.

[50] B. Li, A. Eryilmaz, and R. Srikant, "On the Universality of Age-based Scheduling in Wireless Networks," in *Proc. IEEE INFOCOM,* pp. 1302–1310, Hong Kong, China, April 26–May 1, 2015.

[51] A. Maatouk, M. Assaad, and A. Ephremides, "On the Age of Information in a CSMA Environment," *IEEE/ACM Trans. on Networking,* vol. 28, no. 2, pp. 818–831, 2020.

[52] Y. Xiao and Y. Sun, "A Dynamic Jamming Game for Real-time Status Updates," in *Proc. IEEE INFOCOM Workshops—Age of Information Workshop,* Honolulu, HI, USA, April 15–19, 2018.

[53] G.D. Nguyen, S. Kompella, C. Kam, J. E. Wieselthier, and A. Ephremides, "Information Freshness over an Interference Channel: A Game Theoretic View," in *Proc. IEEE INFOCOM,* pp. 908–916, Honolulu, HI, USA, April 15–19, 2018.

[54] Z. Ning, P. Dong, X. Wang, X. Hu, L. Guo, B. Hu, Y. Guo, T. Qiu, and R. Kwok, "Mobile Edge Computing Enabled 5G Health Monitoring for Internet of Medical Things: A Decentralized Game Theoretic Approach," *IEEE J. Selected Areas in Commun.,* to appear, 2020.

[55] J. Zhong, R.D. Yates, and E. Soljanin, "Backlog-adaptive Compression: Age of Information," in *Proc. IEEE ISIT,* pp. 566–570, Aachen, Germany, June 25–30, 2017.

[56] R. Devassy, G. Durisi, G.C. Ferrante, O. Simeone, and E. Uysal- Biyikoglu, "Delay and peak-age violation probability in short-packet transmissions," in *Proc. IEEE ISIT,* pp. 2471–2475, Vail, Colorado, USA, June 17–22, 2018.

[57] P. Mayekar, P. Parag, and H. Tyagi, "Optimal Source Codes for Timely Updates," *IEEE Trans. on Information Theory,* vol. 66, no. 6, pp. 3714–3731, 2020.

[58] C. Kam, S. Kompella, and A. Ephremides, "Experimental Evaluation of the Age of Information via Emulation," in *Proc. IEEE MILCOM,* pp. 1070–1075, Tampa, FL, USA, Oct. 26–28, 2015,.

[59] V. Marbukh, "Towards Managing Age of Network State Information in Challenged Networks," in *Proc. IEEE INFOCOM Age of Information Workshop* Honolulu, HI, USA, April 15–19, 2018.

[60] R. Agrawal and V. Subramanian, "Optimality of Certain Channel Aware Scheduling Policies," in *Proc. Allerton,* pp. 1533–1542, Monticello, IL, USA, Oct. 2–4, 2002.

[61] A.L. Stolyar, "On the Asymptotic Optimality of the Gradient Scheduling Algorithm for Multiuser Throughput Allocation," *Operations Research,* vol. 53, issue 1, pp. 12–25, Feb. 2005.

[62] IBM ILOG CPLEX Optimizer, available at https://www.ibm.com/analytics/cplex-optimizer [Online; accessed on 2022-12-2].

[63] R.L. Garham, D.E. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science,* Chapter 2, *Addison-Wesley Professional,* 1989, ISBN: 978-0201558029.

[64] A.S. Tanenbaum and A.S. Woodhaul, *Operating Systems: Design and Implementation,* Chapter 4, *Englewood Cliffs: Prentice-Hall,* 1997, ISBN: 978-0136386773.

[65] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU programming,* Chapter 5, *Addison-Wesley Professional*, 2010, ISBN: 978-0131387683.

[66] Y. Kuo, "Minimum Age TDMA Scheduling," in *Proc. of IEEE INFOCOM,* pp. 2296–2304, Paris, France, Apr. 29–May 2, 2019.

[67] I. Kadota and E. Modiano, "Minimizing the Age of Information in Wireless Networks with Stochastic Arrivals," in *Proc. of ACM MobiHoc,* pp. 221–230, Catania, Italy, July 2–5, 2019

[68] A.M. Bedewy, Y. Sun, S. Kompella, and N.B. Shroff, "Age-optimal Sampling and Transmission Scheduling in Multi-Source Systems," in *Proc. of ACM MobiHoc,* pp. 121–130, Catania, Italy, July 2–5, 2019

[69] V. Tripathi, R. Talak, and E. Modiano, "Age Optimal Information Gathering and Dissemination on Graphs," in *Proc. of IEEE INFOCOM,* pp. 2422–2430, Paris, France, Apr. 29–May 2, 2019.

[70] B. Yin, S. Zhang, Y. Cheng, L.X. Cai, Z. Jiang, S. Zhou, and Z. Niu, "Only Those Requested Count: Proactive Scheduling Policies for Minimizing Effective Age-of-Information," in *Proc. of IEEE INFOCOM,* pp. 109–117, Paris, France, Apr. 29–May 2, 2019.

[71] Q. Liu, H. Zeng and M. Chen, "Minimizing Age-of-Information with Throughput Requirements in Multi-Path Network Communication," in *Proc. of ACM MobiHoc,* pp. 41–50, Catania, Italy, July 2–5, 2019.

[72] E. Altman, R. El-Azouzi, D.S. Menasche, and Y. Xu, "Forever Young: Aging Control

For Hybrid Networks," in *Proc. of ACM MobiHoc,* pp. 91–100, Catania, Italy, July 2–5, 2019.

[73] J. Liebeherr, D.E. Wrege, and D. Ferrari, "Exact Admission Control for Networks with a Bounded Delay Service," *IEEE/ACM Transactions on Networking,* vol. 4, issue 6, pp. 885–901, Dec. 1996.

[74] L. Georgiadis, R. Guérin, and A. Parekh, "Optimal Multiplexing on a Single Link: Delay and Buffer Requirements," *IEEE Transactions on Information Theory,* vol. 43, issue 5, pp. 1518–1535, Sep. 1997.

[75] M. Andrews, "Probabilistic End-to-End Delay Bounds for Earliest Deadline First Scheduling," in *Proc. IEEE INFOCOM,* vol. 2, pp. 603–612, Tel Aviv, Israel, Mar. 26–30, 2000.

[76] H. Hoang, M. Jonsson, U. Hagstrom, and A. Kallerdahl, "Switched Real-Time Ethernet with Earliest Deadline First Scheduling – Protocols and Traffic Handling," in *Proc. IEEE IPDPS,* Ft. Lauderdale, FL, USA, Apr. 15–19, 2001.

[77] A.B. Kahn, "Topological Sorting of Large Networks," *Communications of the ACM*, vol. 5, issue 11, pp. 558–561, Nov. 1962.

[78] R. Tarjan, "Depth-First Search and Linear Graph Algorithms," *SIAM Journal on Computing*, vol. 1, issue 2, pp. 146–160, June 1972.

[79] Z. Qian, F. Wu, J. Pan, K. Srinivasan, and N.B. Shroff, "Minimizing Age of Information in Multi-channel Time-sensitive Information Update Systems," in *Proc. IEEE INFOCOM,* pp. 446–455, online conference, July 6–9, 2020.

[80] J.P. Champati, R.R. Avula, T.J. Oechtering, and J. Gross, "On the Minimum Achiev-

able Age of Information for General Service-Time Distributions," in *Proc. IEEE INFOCOM,* pp. 456–465, online conference, July 6–9, 2020.

[81] J. Lou, X. Yuan, S. Kompellay, and N. Tzeng, "AoI and Throughput Tradeoffs in Routing-aware Multi-hop Wireless Networks," in *Proc. IEEE INFOCOM,* pp. 476–485, online conference, July 6–9, 2020.

[82] A.M. Bedewy, Y. Sun, R. Singh, and N.B. Shroff, "Optimizing Information Freshness Using Low-Power Status Updates via Sleep-Wake Scheduling," in *Proc. ACM MobiHoc,* pp. 51–60, online conference, Oct. 11–14, 2020.

[83] A. Maatouk, S. Kriouile, M. Assaad, and A. Ephremides, "Asymptotically Optimal Scheduling Policy for Minimizing the Age of Information," in *Proc. IEEE ISIT,* pp. 1747–1752, online conference, June 21–26, 2020.

[84] T. Park, W. Saad, and B. Zhou, "Centralized and Distributed Age of Information Minimization with Non-linear Aging Functions in the Internet of Things," *IEEE Internet of Things Journal,* to appear, DOI: 10.1109/JIOT.2020.3046448.

[85] B. Zhou and W. Saad, "Minimum Age of Information in the Internet of Things with Non-Uniform Status Packet Sizes," *IEEE Trans. on Wireless Communication,* vol. 19, issue. 3, pp. 1933–1947, March 2020.

[86] H. Ma, S. Zhou, X. Zhang, and L. Xiao, "Transmission Scheduling for Multi-loop Wireless Networked Control Based on LQ Cost Offset," in *Proc. IEEE INFOCOM Workshops—Age of Information Workshop,* online conference, July 6, 2020.

[87] Y. Wang and W. Chen, "Adaptive Power and Rate Control for Real-time Status Updating over Fading Channels," *IEEE Trans. on Wireless Communication,* to appear, DOI: 10.1109/TWC.2020.3047426.

[88] B. Sombabu and S. Moharir, "Age-of-Information Based Scheduling for Multi-Channel Systems," *IEEE Trans. on Wireless Communication,* vol. 19, issue. 7, pp. 4439–4448, July 2020.

[89] H.H. Yang, A. Arafa, T.Q.S. Quek, and V. Poor, "Optimizing Information Freshness in Wireless Networks: A Stochastic Geometry Approach," *IEEE Trans. on Mobile Computing,* to appear, DOI: 10.1109/TWC.2020.3047426.

[90] J. Sun, Z. Jiang, B. Krishnamachari, S. Zhou, and Z. Niu, "Closed-Form Whittle's Index-Enabled Random Access for Timely Status Update," *IEEE Trans. on Communications,* vol. 68, issue. 3, pp. 1538–1551, March 2020.

[91] H. Tang, J. Wang, L. Song, and J. Song, "Minimizing Age of Information With Power Constraints: Multi-User Opportunistic Scheduling in Multi-State Time-Varying Channels," *IEEE J. Selected Areas in Commun.,* vol. 38, issue. 5, pp. 854–868, May 2020.

[92] R.D. Yates, P. Ciblat, A. Yener, and M. Wigger, "Age-Optimal Constrained Cache Updating," in *Proc. IEEE ISIT,* pp. 141–145, Aachen, Germany, June 25–30, 2017.

[93] I. Kadota, A. Sinha, and E. Modiano, "Optimizing Age of Information in Wireless Networks with Throughput Constraints," in *Proc. of IEEE INFOCOM,* pp. 1844–1852, Honolulu, HI, USA, Apr. 16–18, 2018

[94] C. Li, S. Li, Y. Chen, Y.T. Hou, and W. Lou, "Minimizing Age of Information under General Models for IoT Data Collection," *IEEE Trans. on Network Science and Engineering*, vol. 7, no. 4, pp. 2256–2270, Oct. 2020.

[95] H. Tang, J. Wang, L. Song, and J. Song, "Minimizing Age of Information With Power Constraints: Multi-User Opportunistic Scheduling in Multi-State Time-Varying Chan-

nels," *IEEE Journal on Selected Areas in Communications,* vol. 38, no. 5, pp. 854-–868, May 2020.

[96] C. Xu, Q. Xu, J. Wang, K. Wu, K. Lu, and C. Qiao, "AoI-centric Task Scheduling for Autonomous Driving Systems," in *Proc. IEEE INFOCOM,* pp. 1019–1028, online conference, May 2–5, 2022.

[97] J. Pan, A.M. Bedewy, Y. Sun, and N.B. Shroff, "Minimizing Age of Information via Scheduling over Heterogeneous Channels," in *Proc. ACM MobiHoc,* pp. 111–120, Shanghai, China, July 26–-29, 2021.

[98] A. Maatouk, S. Kriouile, M. Assaad, and A. Ephremides, "The Age of Incorrect Information: A New Performance Metric for Status Updates," *IEEE/ACM Trans. on Networking,* vol. 28, no. 5, pp. 2215-–2228, Oct. 2020.

[99] Q. Liu, C. Li, Y.T. Hou, W. Lou, J.H. Reed, and S. Kompella, "Ao^{2}I: Minimizing Age of Outdated Information to Improve Freshness in Data Collection," in *Proc. IEEE INFOCOM,* pp. 1359–1368, online conference, May 2–5, 2022.

[100] D. Guo, K. Nakhleh, I. Hou, S. Kompella, and C. Kam, "A Theory of Second-Order Wireless Network Optimization and Its Application on AoI," in *Proc. IEEE INFOCOM,* pp. 999–1008, online conference, May 2–5, 2022.

[101] C. Li, Q. Liu, S. Li, Y. Chen, Y.T. Hou, and W. Lou, "On Scheduling with AoI Violation Tolerance," in *Proc. IEEE INFOCOM,* online conference, May 10–13, 2021.

[102] Q. Liu, C. Li, Y.T. Hou, W. Lou, and Sastry Kompella, "Aion: A Bandwidth Optimized Scheduler with AoI Guarantee," in *Proc. IEEE INFOCOM,* online conference, May 10–13, 2021.

[103] I. Kadota, A. Sinha, E. Uysal-Biyikoglu, R. Singh, and E. Modiano, "Scheduling Policies for Minimizing Age of Information in Broadcast Wireless Networks," *IEEE/ACM Trans. on Networking,* vol. 26, no. 6, pp. 2637-–26250, Dec. 2018.

[104] R. Talak, S. Karaman, E. Modiano, "Optimizing Information Freshness in Wireless Networks under General Interference Constraints," in *Proc. ACM MobiHoc,* pp. 61–70, Los Angeles, USA, June 25–-28, 2018.

[105] Y. Sun, I. Kadota, R. Talak, and E. Modiano, *Age of Information: A New Metric for Information Freshness, Synthesis Lectures on Communication Networks, Morgan & Claypool Publishers*, 2019, ISBN: 978-1681736808.

[106] Z. Zhang, Z. Duan, and Y.T. Hou, "Virtual Time Reference System: A Unifying Scheduling Framework for Scalable Support of Guaranteed Services," *IEEE Journal on Selected Areas in Communications,* vol. 18, no. 12, pp. 2684-–2695, Dec. 2000.

[107] C. Jiang, Y. Shi, Y.T. Hou, W. Lou, S. Kompella, and S.F. Midkiff, "Toward Simple Criteria to Establish Capacity Scaling Laws for Wireless Networks," in *Proc. IEEE INFOCOM,* pp. 774–782, Orlando, FL, USA, Mar. 25–30, 2012.

[108] C. Jiang, Y. Shi, Y.T. Hou, W. Lou, S. Kompella, and S.F. Midkiff, "A General Method to Determine Asymptotic Capacity Upper Bounds for Wireless Networks," *IEEE Trans. on Network Science and Engineering*, vol. 6, no. 1, pp. 2–15, Nov. 2017.

[109] A. Durán, M. Toril, F. Ruiz, and A. Mendo, "Self-Optimization Algorithm for Outer Loop Link Adaptation in LTE," *IEEE Communications Letters,* vol. 9, no. 11, pp. 2005–2008, Nov. 2015.