Scheduling With Age of Information Guarantee

Chengzhang Li, *Student Member, IEEE*, Qingyu Liu, *Member, IEEE*, Shaoran Li, *Student Member, IEEE*, Yongce Chen, *Member, IEEE*, Y. Thomas Hou, *Fellow, IEEE*, Wenjing Lou, *Fellow, IEEE*, and Sastry Kompella, *Senior Member, IEEE*

***Abstract*— Age of Information (AoI) is an application layer performance metric that quantifies the freshness of information. This paper investigates scheduling problems at network edge when there is an AoI requirement for each source node, which we call Maximum AoI Threshold (MAT). Specifically, we want to determine whether or not a vector of MATs corresponding to 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 MAT vector when the load is under $\ln 2$. We use extensive numerical results to validate our theoretical results and show that the performance of FPM is significantly better than a state-of-the-art scheduling algorithm.**

***Index Terms*— Age of information (AoI), scheduling, deadline.**

I. Introduction

AGE of Information (AoI) is a new metric used to measure the *freshness* of information [2], [3]. It has since captured the attention of the research community and is now an area of active research (see a survey in [4] and an online bibliography in [5]). By definition in [2], [3], 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 that are used by the networking community as the latter only considers the transit time for a packet through a network component or the network. In other words, delay or latency typically refers to the time required to move the information from one point to another in the network. In contrast, AoI includes delay or latency as its components and advances with time if there are no new updates. From layered perspective, AoI is an application layer metric while delay or latency is at a lower layer (e.g., transport or link layer).

There has been active research on designing scheduling algorithms to minimize AoI (see, e.g., [5]). Although a clever design of a scheduler to minimize AoI is important, it does not address some important application areas where there is a hard performance requirement on AoI metric. Imagine some AoI-critical applications such as autonomous vehicles and unmanned aerial vehicles, where the applications require certain freshness guarantee from some sources. Although existing schedulers designed for AoI minimization has some relevance to AoI thresholds, they are fundamentally different problems. Simply put, existing schedulers designed for AoI minimization cannot offer any guarantee on AoI thresholds. A close scanning of the literature [5] shows that there is a serious lack of research in this area.

In this paper, we address this issue by studying AoI scheduling under a *Maximum AoI Threshold* (MAT) for each source node.[1] Specifically, this paper addresses the following problems:

(i) *For a vector of MAT 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 a simultaneous investigation of both is needed. Further, they 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 [42]–[45]. In particular, *Earliest Deadline First* (EDF), the most well-known scheduler, was shown to be very efficient in the task scheduling problem [42], [43]. But we shall see that it performs poorly for our AoI problem in this paper, due to the fundamental difference between the definitions of AoI and delay.

We summarize the main contributions of this paper as following:

- First, we prove that if there exists a feasible scheduler w.r.t. an MAT vector $\boldsymbol{d}$, then there must exist at least one feasible *cyclic* scheduler. This result allows us to

Manuscript received March 11, 2021; revised November 4, 2021 and January 19, 2022; accepted February 28, 2022; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Moharir. This work was supported in part by ONR MURI under Grant N00014-19-1-2621, in part by the Virginia Commonwealth Cyber Initiative (CCI), and in part by the Virginia Tech Institute for Critical Technology and Applied Science (ICTAS). An abridged version of this paper appeared in Proc. IEEE INFOCOM, online conference, July 6–9, 2020 [1] [DOI: 10.1109/INFOCOM41043.2020.9155514]. *(Corresponding author: Y. Thomas Hou.)*

Chengzhang Li, Qingyu Liu, Shaoran Li, Y. Thomas Hou, and Wenjing Lou are with the Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061 USA (e-mail: thou@vt.edu).

Yongce Chen is with NVIDIA Corporation, Santa Clara, CA 95051 USA.

Sastry Kompella is with the U.S. Naval Research Laboratory, Washington, DC 20375 USA.

Digital Object Identifier 10.1109/TNET.2022.3156866

[1] We use MAT to make a distinction from traditional notion of deadline for transport or link layer.

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 "MAT 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 MAT vector, called *polynomial* MAT vector, and present a low-complexity procedure called *Polynomial Scheduler Construction* (PSC) that can find a feasible scheduler for any polynomial MAT vector $\boldsymbol{d}$ with $l(\boldsymbol{d}) \leq 1$.
- For a general (non-polynomial) MAT vector $\boldsymbol{d}$, we propose to map it to another polynomial MAT vector with load no greater than 1 and subsequently construct a feasible scheduler for $\boldsymbol{d}$ based on this mapping. To do this, we generalize the definition of polynomial MAT 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 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 able to find such a mapping if it exists. We prove that FPM is able to find a feasible scheduler for any $\boldsymbol{d}$ with $l(\boldsymbol{d}) \leq \ln 2$ ($\approx 69.3\%$). We also show that the cycle length of a feasible scheduler found by FPM is no greater than the largest element in MAT vector $\boldsymbol{d}$.

The rest of the paper is organized as follows. In Section II, we review related work on AoI scheduling. In Section III, we describe the data collection network model in this paper and state the scheduling problem that we investigate in this paper. In Section IV, we present an exhaustive procedure that can determine the schedulability of any MAT vector. Due to its high complexity, it is only useful when the number of nodes is small. In Section V, we present a procedure that can find a feasible scheduler for a special group of MAT vectors when its load is no greater than 1. In Section VI, we generalize the procedure in Section V and present a new procedure for general MAT vectors. In Section VII we prove that the procedure in Section VI can find a feasible scheduler for any MAT vector when its load is no greater than $\ln 2$. In Section VIII, we perform extensive simulations to validate the theoretical results and examine the behavior of our proposed solution procedures. Section IX concludes this paper.

II. Related Work

There has been a flourish of research on AoI in recent years [5]. A main line of research is to design schedulers that minimize AoI (e.g., [6]–[28]). Another line of research focuses on modeling, analysis, and optimization of AoI (e.g., [29]–[33]). There are some other branches on AoI research, such as game theory for AoI (e.g., [34]–[36]), channel coding for AoI (e.g. [37]–[39]), and AoI applications (e.g. [40], [41]), to name a few.

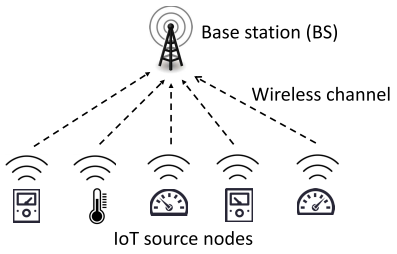


Fig. 1. System model: N source nodes collect information and forward it to a BS.

Since this paper is on scheduler design, we will limit our literature review in this area. In [6]–[9], the authors designed schedulers to minimize AoI under the same transmission model in Fig. 1 where information sources share a common channel. Specifically, in [6], Hsu *et al.* assumed a Bernoulli packet arrival model for the sources. In [7], Kadota *et al.* considered an unreliable channel where there is a fixed packet loss probability for each transmission. In [8], Zhong *et al.* studied a new metric called age of synchronization (AoS) along with AoI. In [9] Jhunjhunwala and Moharir studied how to minimize a long-term cost function of AoI.

Extensions beyond the model in [6]–[9] have also been explored in recent works. For example, in [10]–[14], the authors extended the assumption that only one source can transmit at one time instance to a multi-channel model where multiple sources can send packets to the BS. In [15], the authors extended the time-slotted network model to a more general case where the transmission from each source can begin at any time instance, and carrier sensing is utilized to reduce collisions. In [16] Zhou and Saad generalized the sample sizes, i.e., considered varying sample sizes among the source nodes. In [17]–[20], the authors considered OFDM channels where the channel resource is organized as a 2-dimensional resource gird. In [21]–[24], the authors considered a multi-link ad hoc network where multiple information sources send information updates to multiple destinations. In [25], Yang *et al.* studied many device-to-device communication pairs interfering with each other depending on their geography locations on a 2-dimensional plane. In [26]–[28], the author considered a multi-hop network environment where the sources update information to their destinations through relay nodes.

Although these efforts have made important contributions to the design of schedulers to minimize AoI, there is a lack of research on scheduler design under maximum AoI thresholds, which motivates us to investigate in this paper.

III. System Model and Problem Statement

We consider a network (see Fig. 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 [6]–[8], we assume the transmission of a sample takes one time slot. Therefore, at most one sample from a source node can be transmitted in a time slot.

Depending on the objective, a scheduler is needed to decide which sample will be chosen for transmission in each time slot. Denote $\pi(t) \in \{0, 1, 2, \cdots, N\}$ as the scheduling decision for time slot t ($t = 0, 1, 2, \cdots$). When $\pi(t) = 0$, it denotes that none of the source nodes is chosen for transmission at t; when $\pi(t) = i$ and $1 \leq i \leq N$, it denotes that the scheduler chooses source node i for transmission at 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 written as:

$$A_i(t) = t - U_i(t). \tag{1}$$

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:

$$A_i(t+1) = \begin{cases} 1, & \text{if } \pi(t) = i, \\ A_i(t) + 1, & \text{otherwise.} \end{cases} \tag{2}$$

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 source 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.

Tables I and II list key notation and acronyms in this paper. 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 paper, we assume there is a Maximum AoI threshold (MAT), 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 $i = 1, 2, \cdots, N$.[2] 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 IV-A.

Denote $\boldsymbol{d} = [d_1 \ d_2 \ \cdots \ d_N]$ as the vector of MATs for all source nodes. We say $\boldsymbol{d}$ is *schedulable* if there exists at least one feasible scheduler π. We want to study the following two problems in this paper: (i) For a given $\boldsymbol{d}$, is it schedulable? (ii) If it is schedulable, then find a feasible scheduler for it.

The above problem is very different from the existing research on minimizing AoI [6]–[8], [10]–[28]. 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 threshold 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 very different problem.

TABLE I
NOTATION

Symbol	Definition
$A_i(t)$	AoI for source node i at the BS at time t
c	Cycle length for a cyclic scheduler
$\boldsymbol{d}$	An MAT vector
d_i	MAT for source node i
$d_{\max}$	The element with the largest value in $\boldsymbol{d}$
$l(\boldsymbol{d})$	Load corresponding to $\boldsymbol{d}$
N	Number of source nodes in the network
$\pi(t)$	Scheduling decision at time t
r_i	Long term average data rate for source node i
$\mathcal{S}$	State space for feasible scheduler
$\boldsymbol{s}(t)$	State vector (consisting of $A_i(t)$ for all i's) at time t
$U_i(t)$	Generation time of the most recent sample from source node i at the BS

TABLE II
ACRONYMS

Acronym	Full Name
CSD	Cyclic Scheduler Detection
EDF	Earliest Deadline First
FCM	Fictitious Common Multiple
FPM	Fictitious Polynomial Mapping
FSC	Fictitious Scheduler Construction
LCM	Least Common Multiple
MAT	Maximum AoI Threshold
PSC	Polynomial Scheduler Construction
STG	State Transition Graph

[2]In this paper we assume a hard threshold that must not be violated. The case where the threshold is soft, e.g., allowing a percentage of threshold violation, will be explored in our future work.

IV. 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).[3] This limitation serves as the motivation of our work in Sections V and VI.

A. 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 $t \geq 0$ with some constant c. 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 1: If an MAT vector $\mathbf{d}$ is schedulable, then there exists at least one cyclic scheduler that is feasible w.r.t. $\mathbf{d}$.

To prove Lemma 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., $\boldsymbol{s}(t) = [A_1(t)\ A_2(t)\ \cdots\ A_N(t)]$. Clearly, 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 we have $\boldsymbol{s}(t_1+1) = \boldsymbol{s}(t_2+1)$.

We now consider possible state space under a feasible scheduler. After some 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

$$\mathcal{S} = \{\boldsymbol{s}(t) : |\ 1 \leq A_i(t) \leq d_i,\ i = 1, 2, \cdots, N\}. \tag{3}$$

Clearly, there is a total of $d_1 \cdot d_2 \cdots d_N$ unique states in $\mathcal{S}$. It's easy to see that under a feasible cyclic scheduler, the evolution of state also exhibits a cyclic behavior, with a cycle length equal to the length of the scheduling cycle, i.e.,

$$\boldsymbol{s}(t) = \boldsymbol{s}(t+c), \quad \text{for } t > d_{\max}. \tag{4}$$

Based on the above analysis, we are now ready to prove Lemma 1.

Proof Lemma 1: Our proof is based on the construction of a feasible cyclic scheduler. Since $\boldsymbol{d}$ is schedulable, there exists a feasible scheduler $\pi(t)$ with a warm-up time t_0. Since there are 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 the decisions for one cycle, and repeat it to construct a feasible cyclic scheduler. ∎

[3]Incidentally, when N is small, the MAT scheduling problem in this section can also be solved by the procedure proposed by Jhunjhunwala and Moharir in [9].

By Lemma 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 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), 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., there exists a t_o such that for any $t > t_0$, $A_i(t) \leq d_i$). This completes the proof. ∎

Based on Lemmas 1 and 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" as an abbreviation of "feasible cyclic scheduler" when there is no confusion.

B. 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 [46], [47].

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 3: The existence of a feasible cyclic scheduler w.r.t. $\mathbf{d}$ is equivalent to the existence of a cycle in STG.

The proof of Lemma 3 follows directly from the above discussion and is thus omitted here.

The following corollary follows from Lemma 3, which shows us a way to construct a feasible cyclic scheduler (if $\boldsymbol{d}$ is schedulable).

Corollary 3.1: A cycle in STG constitutes a feasible cyclic scheduler w.r.t. $\mathbf{d}$, with each edge in the cycle corresponding to the scheduling decision for that state.

Now we outline the CSD procedure in Fig. 2. Based on Lemma 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 [46] and Depth-First-Search (DFS) [47].

CSD: For an MAT 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.

Fig. 2. CSD procedure.

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.[4] Therefore, the time complexity of $O(N d_1 d_2 \cdots d_N)$ is no less than $O(N \cdot 2^N)$, which is exponential w.r.t. N. For space complexity, since there are $O(N d_1 d_2 \cdots d_N)$ edges in STG, the space complexity of CSD is also $O(N d_1 d_2 \cdots d_N)$. Therefore, although CSD can determine $\boldsymbol{d}$'s schedulability with absolute certainty, its exponential complexity poses a serious problem when N is large.

V. Special Case: Polynomial MAT Vectors

In this section, we consider a special MAT vector, called *polynomial* MAT vector. In Section VI, we will use this procedure as a basis to design a procedure for the general (non-polynomial) MAT vectors.

A. Polynomial MAT Vectors and MAT Load

Definition 1: An MAT vector $\mathbf{d}$ 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 MAT vector with $b = 5$, $n_1 = n_2 = 0$, $n_3 = 1$, $n_4 = n_5 = 2$ and $n_6 = 3$. On the other hand, $\boldsymbol{d} = \{2, 4, 6, 8\}$ is not polynomial since we cannot find a b and n_i such that $d_i = b \cdot 2^{n_i}$.

In Section V-B, we will design a procedure that can find a feasible scheduler for a polynomial MAT vector $\boldsymbol{d}$ under a very general condition. To do this, we need a definition to tie traffic load and MAT. First, we define the long-term average data rate for source node i under scheduler π as

$$r_i = \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} [\pi(t) = i], \tag{5}$$

where "$[\cdot]$" is Iverson bracket, returning 1 if the statement within is true and 0 otherwise [48]. 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

$$\sum_{i=1}^{N} r_i \leq 1. \tag{6}$$

[4]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 is not possible.

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

$$r_i \geq \frac{1}{d_i}, \quad i = 1, 2, \cdots, N. \tag{7}$$

Intuitively, $1/d_i$ represents the minimum guaranteed rate that a feasible scheduler should provision to source node i. We define *MAT load* for an MAT vector $\boldsymbol{d}$ as

$$l(\boldsymbol{d}) = \sum_{i=1}^{N} \frac{1}{d_i}, \tag{8}$$

which represents the sum of minimum guaranteed rate that a feasible scheduler should provide to all source nodes. Clearly, by (6), any $\boldsymbol{d}$ with $l(\boldsymbol{d}) > 1$ is unschedulable, which is quite intuitive.

Naturally, we would like to use the MAT 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.

B. Scheduling for Polynomial $\boldsymbol{d}$

A Motivating Example: Consider six source nodes A, B, C, D, E, F and a polynomial MAT vector $\boldsymbol{d} = [3\ 6\ 6\ 6\ 12\ 12]$ corresponding to these six sources. It can be easily verified (based on Definition 1) 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 for that 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 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 solution for $\boldsymbol{d}$. ∎

PSC: For a polynomial MAT 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**

Fig. 3. A pseudocode for PSC.

Based on the key ideas in the above motivating example, we outline a scheduling algorithm for polynomial MAT vectors, which we call *Polynomial Scheduler Construction* (PSC). Fig. 3 shows the pseudocode of PSC.

For PSC, we have the following lemma.

Lemma 4: For any polynomial MAT 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 MAT set $\boldsymbol{d}$ 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 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 is that for a polynomial MAT vector $\boldsymbol{d}$, PSC is guaranteed to find a feasible scheduler for MAT 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 $O(Nd_{\max})$.

VI. Scheduling for General MAT 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 always find a feasible scheduler when $l(\boldsymbol{d}) < \ln 2$ ($\approx 69.3\%$) regardless of whether $\boldsymbol{d}$ is polynomial or not.

A. Basic Idea

The basic idea is to "map" a general (non-polynomial) MAT vector $\boldsymbol{d}$ that is under consideration to a polynomial MAT vector by "tightening" one or more elements in $\boldsymbol{d}$. In other words, we can always offer a source with a new MAT that is smaller than its requirement. If we can do this mapping and the new reference polynomial MAT vector has a load no greater than 1, then we can apply Lemma 4 and use PSC to find a feasible scheduler.

Example: For a general MAT vector $\boldsymbol{d} = [3\ 6\ 7\ 8\ 12\ 13]$, we can map (tighten) it to the polynomial MAT vector $\boldsymbol{d}_1 = [3\ 6\ 6\ 6\ 12\ 12]$. Since the polynomial MAT 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 MAT vector $\boldsymbol{d}$ to some polynomial MAT 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 MAT vector $\boldsymbol{d} = [3\ 5\ 5\ 5]$. For this $\boldsymbol{d}$, the smallest MAT (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 MATs will be tightened to 4 (based on Definition 1), and thus $\boldsymbol{d}$ is mapped to the polynomial MAT 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). If it is mapped to 3 (unchanged), then the other MATs will be tightened to 3, and thus $\boldsymbol{d}$ is mapped to the polynomial MAT 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 MAT vector with a load no greater than 1.

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

Definition 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 2 and Definition 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 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 1, the smallest MAT in $\boldsymbol{d}$ can only be mapped to 2 or 3, and both mapping will have an MAT load greater than 1, which is not helpful. However, under fictitious polynomial vector definition, we will have more options to map the smallest MAT (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 MAT 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 VI-B we will present a low-complexity procedure to find a feasible scheduler w.r.t. any MAT 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.

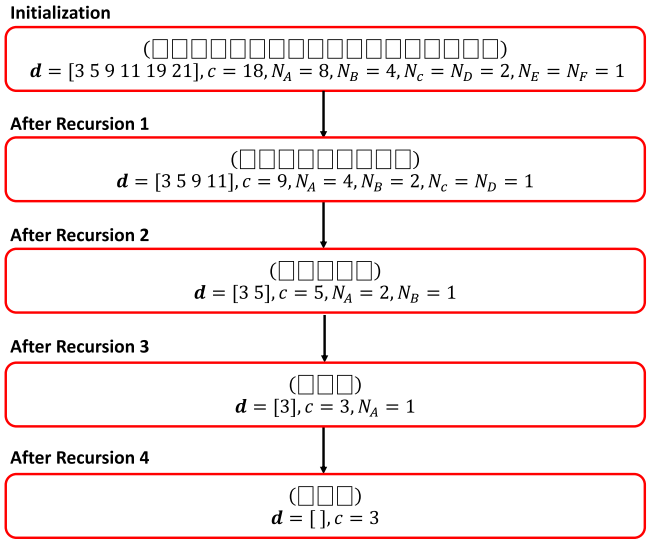


Fig. 4. Recursion in the example.

B. Scheduling for Fictitious Polynomial $\widetilde{\boldsymbol{d}}$

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

Theorem 1: For any MAT 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 1 is based on constructing one such feasible scheduler, which should also be of low-complexity. 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 VI-C). Now, we focus on showing how to construct a feasible scheduler w.r.t. $\boldsymbol{d}$ based on $\widetilde{\boldsymbol{d}}$.

In Section V-B, 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 V-B, 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 Fig. 4. 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 Fig. 5.

With FSC in hand, we now prove Theorem 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 Theorem 1: 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 $\dim(\boldsymbol{d}_1) < \dim(\boldsymbol{d})$.

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 $\boldsymbol{d}_2$.

Fig. 5. A pseudocode for FSC.

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. ∎

C. Mapping $\boldsymbol{d}$ to $\widetilde{\boldsymbol{d}}$

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.

FPM: For an MAT vector $\boldsymbol{d}$:

1: **for** $i = 1, 2, \cdots, N$ **do**
2: Compute $\widetilde{d}_j$ for each $1 \leq j \leq N$ by (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**

Fig. 6. A Pseudocode of FPM.

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 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 constructing $\widetilde{\boldsymbol{d}}_0 = [\widetilde{d}_{01}\ \widetilde{d}_{02}\ \cdots\ \widetilde{d}_{0N}]$ that 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 $\widetilde{\boldsymbol{d}}_0 = \min_j\{d_j/\widetilde{d}_j\} \cdot \widetilde{\boldsymbol{d}}$. Clearly, $\widetilde{\boldsymbol{d}}_0$ is a fictitious polynomial vector to which $\boldsymbol{d}$ can be mapped, $l(\widetilde{\boldsymbol{d}}_0) \leq l(\widetilde{\boldsymbol{d}}) \leq 1$, and for $i = \arg\min_j\{d_j/\widetilde{d}_j\}$ we have $d_i = \widetilde{d}_{0i}$. This completes the proof. ∎

Based on Lemma 5, to map $\boldsymbol{d}$ into $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \leq 1$, it is 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

$$\widetilde{d}_j = d_i \cdot 2^{\lfloor \log_2(\frac{d_j}{d_i}) \rfloor}, \quad \text{for each } j \in \{1, 2, \cdots, N\}. \tag{9}$$

Eq. (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 Fig. 6. Note that when the deadlines are identical (i.e., $d_1 = d_2 = \cdots = d_N$) and $l(\boldsymbol{d}) \leq 1$, FPM will always find a round-robin feasible scheduler.

The following corollary directly follows from Lemma 5.

Corollary 5.1: If $\boldsymbol{d}$ can be mapped to any $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \leq 1$, FPM can always find a feasible scheduler w.r.t. $\boldsymbol{d}$.

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

Proof: Suppose in 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 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[5] 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 $O(N^2) + O(d_{\max} \cdot \log d_{\max})$.

VII. MAT Load vs. Schedulability

In the last section, by introducing the notion of mapping from physical $\boldsymbol{d}$ to fictitious $\widetilde{\boldsymbol{d}}$, we showed that FPM 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 FPM can find a feasible scheduler when $l(\widetilde{\boldsymbol{d}}) \leq 1$, it is still not clear how large $l(\boldsymbol{d})$ can be for FPM to find a feasible scheduler. A natural question becomes: What is the maximum load $l(\boldsymbol{d})$ while FPM is guaranteed to find a feasible scheduler? We answer this question in this section.

By the mapping in (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 1. The following proposition shows that we can in fact do better than this.

Proposition 1: If $l(\boldsymbol{d}) \leq \ln 2$, then FPM can 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

$$l(\widetilde{\boldsymbol{d}}) = \sum_{j=1}^{N} \frac{1}{d_i \cdot 2^{\lfloor \log_2(\frac{d_j}{d_i}) \rfloor}} > 1, \quad \text{for } i = 1, 2, \cdots, N. \tag{10}$$

Define the following function $f(x)$:

$$f(x) = \sum_{j=1}^{N} \frac{1}{x \cdot 2^{\lfloor \log_2(\frac{d_j}{x}) \rfloor}} \quad \text{for } x \in \mathbb{R}_{>0}. \tag{11}$$

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 (10) and (11), we have $f(d_i) > 1$ for $i = 1, 2, \cdots, N$. So we have

$$f(x) > 1 \quad \text{for } x \in \mathbb{R}_{>0}. \tag{12}$$

Define $y = 1/x$. Replacing x with $1/y$ in (12), along with (11), we have

$$\sum_{j=1}^{N} y \cdot 2^{-\lfloor \log_2(d_j y) \rfloor} > 1, \quad \text{for } y \in \mathbb{R}_{>0}. \tag{13}$$

[5]We assume $N \leq d_{\max}$. Since for any $\boldsymbol{d}$ with $N > d_{\max}$, $\boldsymbol{d}$ is clearly unschedulable (with $l(\boldsymbol{d}) > 1$).

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

$$\int_{0.5}^{1} g(y) dy = 1 \tag{14}$$

Define u_j as

$$u_j = \frac{1}{d_j} \cdot 2^{\lfloor \log_2(d_j) \rfloor}, \quad j = 1, 2, \cdots, N. \tag{15}$$

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

Multiplying the two sides of (13) and (14) respectively, we have

$$\int_{0.5}^{1} g(y) \cdot \sum_{j=1}^{N} y \cdot 2^{-\lfloor \log_2(d_j y) \rfloor} dy > 1. \tag{16}$$

Substituting $g(y) = \frac{1}{y \cdot \ln 2}$ into (16), we have

$$\frac{1}{\ln 2} \sum_{j=1}^{N} \int_{0.5}^{1} 2^{-\lfloor \log_2(d_j y) \rfloor} dy > 1. \tag{17}$$

Breaking the integral in (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,

$$\frac{1}{\ln 2} \sum_{j=1}^{N} u_j \cdot 2^{-\lfloor \log_2(d_j) \rfloor} > 1. \tag{18}$$

Substituting (15) into (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 1 tells us $\ln 2$ is a valid guarantee for FPM. We will then show it is also the largest guarantee that FPM can offer. Consider a special group of MAT 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 $l(\widetilde{\boldsymbol{d}}) \leq 1$. Thus FPM cannot find a feasible scheduler for $\boldsymbol{d}^n$. Considering $\lim_{n \to \infty} l(\boldsymbol{d}^n) = \ln 2$, we conclude that $\ln 2$ is indeed the largest guarantee that FPM can offer.

VIII. 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

$$\pi^{\text{EDF}}(t) = \arg\min_i \big(d_i - A_i(t) \big), \tag{19}$$

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

TABLE III
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 IV
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

A. 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.

1) Small N: First we consider $N = 5$ source nodes, with $\boldsymbol{d} = [3\ 5\ 7\ 10\ 12]$. The MAT 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:

$$\pi_1 = (ABABACEABDAACBACBAEDABCAAC\ BADEABCAABACDABEAACBAADABCA\ EABACD).$$

For our algorithm FPM, for each $i = 1, 2, \cdots, N$ we compute $\widetilde{\boldsymbol{d}}$ by (9) and try to find one $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \le 1$. 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 III. We can see that $A_i(t) \le d_i$ 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).

2) Large N: We now consider $N = 100$ source nodes that are organized in 10 groups, each each group having the same MAT, as shown in Table IV. 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 (9) and try to find one $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \le 1$. We find that when $i = 1$, $l(\widetilde{\boldsymbol{d}}) = 1$. We use FSC to construct a feasible 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 IV. We can see that $A_i(t) \le d_i$ for all $i = 1, 2, \cdots, 100$. So our scheduler constructed by FPM is indeed feasible.

TABLE V
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 VI
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	($ABCADEABCADF$)

B. Compare to EDF: Small N

In this section we compere our schedulers to EDF when N is small. In Table V, 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 $l(\boldsymbol{d}) \le \ln 2$, FPM can find a feasible scheduler, which confirms the result in Proposition 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 6.

For those $\boldsymbol{d}$'s in Table V for which FPM can find feasible schedulers, we present the feasible schedulers in Table VI. 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. 7, 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

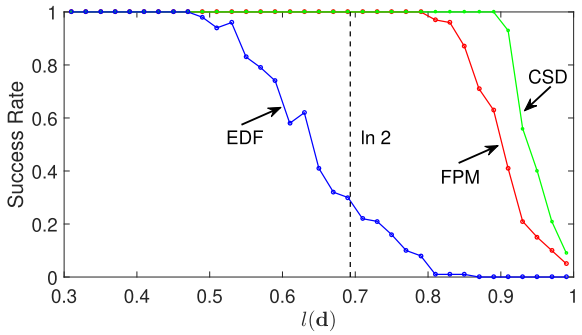


Fig. 7. Success rates for FPM, CSD, and EDF under different $l(\boldsymbol{d})$ when $N = 5$.

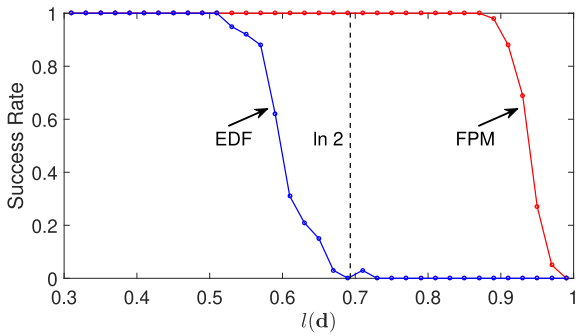


Fig. 8. Success rate for FPM and EDF under different $l(\boldsymbol{d})$ when $N = 20$.

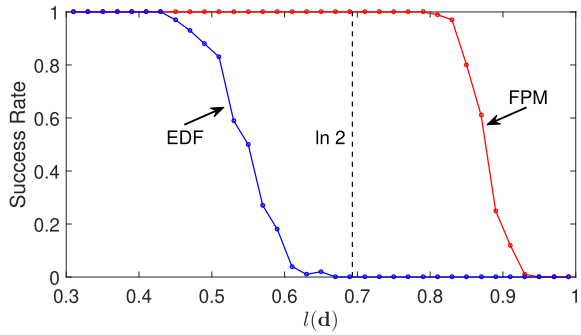


Fig. 9. Success rate for FPM and EDF under different $l(\boldsymbol{d})$ when $N = 50$.

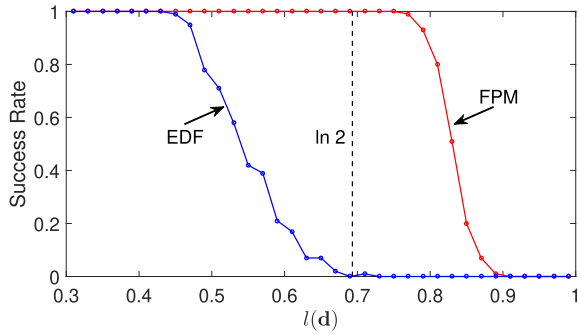


Fig. 10. Success rate for FPM and EDF under different $l(\boldsymbol{d})$ when $N = 100$.

success for finding a feasible scheduler. In Fig. 7, we see that the performance of FPM is close to CSD (the optimal algorithm). In particular, when $l(\boldsymbol{d}) \leq \ln 2$, the success rate of FPM is 100%, which confirms the result in Proposition 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 MAT guarantee problem in this paper, 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 MAT guarantee problem and our simulation results show that.

C. 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. 8 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[6] and calculate the rate (percentage) of success for finding a feasible scheduler. In Fig. 8, 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 1.

Fig. 9 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. 10 shows the performance of FPM and EDF 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%.

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

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

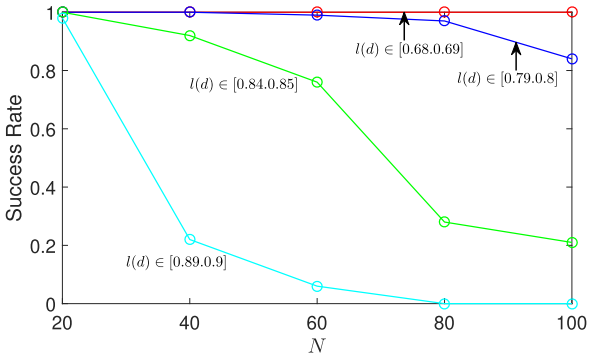


Fig. 11. Success rate for FPM under different N.

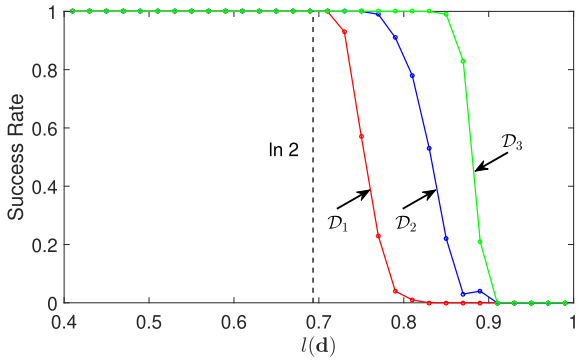


Fig. 12. Success rate for FPM under different $\boldsymbol{d}$ distribution.

The results are shown in Fig. 11. For load interval $[0.68, 0.69]$ the success rates are indeed 1 for all N. This affirms that Proposition 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 MATs of all source nodes.

E. FPM Under Different $\boldsymbol{d}$ 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 in Fig. 12.

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 MATs, the worse the FPM's performance.

IX. Conclusion

We studied scheduling subject to AoI performance guarantee. Specifically, we investigated the following two intertwined problems for AoI scheduling at network edge: (i) For a given MAT 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, 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 MAT 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 $l(\boldsymbol{d}) \leq \ln 2$, 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.

References

[1] C. Li, S. Li, Y. Chen, Y. Thomas Hou, and W. Lou, "AoI scheduling with maximum thresholds," in *Proc. IEEE INFOCOM*, Jul. 2020, pp. 436–445.

[2] S. Kaul, M. Gruteser, V. Rai, and J. Kenney, "Minimizing age of information in vehicular networks," in *Proc. IEEE SECON*, Salt Lake City, UT, USA, Jun. 2011, pp. 350–358.

[3] S. Kaul, R. Yates, and M. Gruteser, "Real-time status: How often should one update?" in *Proc. IEEE INFOCOM*, Orlando, FL, USA, Mar. 2012, pp. 2731–2735.

[4] A. Kosta, N. Pappas, and V. Angelakis, "Age of information: A new concept, metric, and tool," *Found. Trends Netw.*, vol. 12, no. 3, pp. 162–259, 2017.

[5] Y. Sun. *A Collection of Recent Papers on the Age of Information*. Accessed: Mar. 14, 2022. [Online]. Available: http://www.auburn.edu/%7eyzs0078

[6] Y.-P. Hsu, E. Modiano, and L. Duan, "Age of information: Design and analysis of optimal scheduling algorithms," in *Proc. IEEE ISIT*, Archen, Germany, Jun. 2017, pp. 561–565.

[7] I. Kadota, E. Uysal-Biyikoglu, R. Singh, and E. Modiano, "Minimizing the age of information in broadcast wireless networks," in *Proc. 54th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Monticello, IL, USA, Sep. 2016, pp. 844–851.

[8] J. Zhong, R. D. Yates, and E. Soljanin, "Two freshness metrics for local cache refresh," in *Proc. IEEE ISIT*, Vail, CO, USA, Jun. 2018, pp. 1924–1928.

[9] P. R. Jhunjhunwala and S. Moharir, "Age-of-information aware scheduling," in *Proc. Int. Conf. Signal Process. Commun. (SPCOM)*, Bangalore, India, Jul. 2018, pp. 222–226.

[10] A. M. Bedewy, Y. Sun, and N. B. Shroff, "Optimizing data freshness, throughput, and delay in multi-server information-update systems," in *Proc. IEEE ISIT*, Barcelona, Spain, Jul. 2016, pp. 2569–2573.

[11] Y. Sun, E. Uysal-Biyikoglu, and S. Kompella, "Age-optimal updates of multiple information flows," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Honolulu, HI, USA, Apr. 2018, pp. 136–141.

[12] A. Maatouk, S. Kriouile, M. Assaad, and A. Ephremides, "Asymptotically optimal scheduling policy for minimizing the age of information," in *Proc. IEEE ISIT*, Jun. 2020, pp. 1747–1752.

[13] B. Sombabu and S. Moharir, "Age-of-information based scheduling for multi-channel systems," *IEEE Trans. Wireless Commun.*, vol. 19, no. 7, pp. 4439–4448, Jul. 2020.

[14] 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. Sel. Areas Commun.*, vol. 38, no. 5, pp. 854–868, Mar. 2020.

[15] 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*, Oct. 2020, pp. 51–60.

[16] B. Zhou and W. Saad, "Minimum age of information in the Internet of Things with non-uniform status packet sizes," *IEEE Trans. Wireless Commun.*, vol. 19, no. 3, pp. 1933–1947, Mar. 2020.

[17] C. Li, S. Li, and Y. T. Hou, "A general model for minimizing age of information at network edge," in *Proc. IEEE INFOCOM*, Paris, France, Apr. 2019, pp. 118–126.

[18] 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*, Dallas, TX, USA, Jul. 2019, pp. 1466–1472.

[19] 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*, Jul. 2020, pp. 446–455.

[20] T. Park, W. Saad, and B. Zhou, "Centralized and distributed age of information minimization with nonlinear aging functions in the Internet of Things," *IEEE Internet Things J.*, vol. 8, no. 10, pp. 8437–8455, May 2021.

[21] Q. He, D. Yuan, and A. Ephremides, "Optimal link scheduling for age minimization in wireless systems," *IEEE Trans. Inf. Theory*, vol. 64, no. 7, pp. 5381–5394, Jul. 2018.

[22] C. Joo and A. Eryilmaz, "Wireless scheduling for information freshness and synchrony: Drift-based design and heavy-traffic analysis," in *Proc. 15th Int. Symp. Modeling Optim. Mobile, Ad Hoc, Wireless Netw. (WiOpt)*, Paris, France, May 2017, pp. 1–8.

[23] N. Lu, B. Ji, and B. Li, "Age-based scheduling: Improving data freshness for wireless real-time traffic," in *Proc. 18th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, Los Angeles, CA, USA, Jun. 2018, pp. 191–200.

[24] R. Talak, S. Karaman, and E. Modiano, "Optimizing information freshness in wireless networks under general interference constraints," in *Proc. ACM MobiHoc*, Los Angeles, CA, USA, Jun. 2018, pp. 61–70.

[25] H. H. Yang, A. Arafa, T. Q. S. Quek, and V. Poor, "Optimizing information freshness in wireless networks: A stochastic geometry approach," *IEEE Trans. Mobile Comput.*, vol. 20, no. 6, pp. 2269–2280, Jun. 2020.

[26] A. M. Bedewy, Y. Sun, and N. B. Shroff, "Age-optimal information updates in multihop networks," in *Proc. IEEE ISIT*, Archen, Germany, Jun. 2017, pp. 576–580.

[27] R. Talak, S. Karaman, and E. Modiano, "Minimizing age-of-information in multi-hop wireless networks," in *Proc. 55th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Monticello, IL, USA, Oct. 2017, pp. 486–493.

[28] J. Lou, X. Yuan, S. Kompella, and N.-F. Tzeng, "AoI and throughput tradeoffs in routing-aware multi-hop wireless networks," in *Proc. IEEE INFOCOM*, Jul. 2020, pp. 476–485.

[29] 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*, Baltimore, MD, USA, Oct. 2017, pp. 436–441.

[30] X. Zheng, S. Zhou, Z. Jiang, and Z. Niu, "Closed-form analysis of non-linear age of information in status updates with an energy harvesting transmitter," *IEEE Trans. Wireless Commun.*, vol. 18, no. 8, pp. 4129–4142, Aug. 2019.

[31] V. Tripathi and E. Modiano, "A whittle index approach to minimizing functions of age of information," in *Proc. 57th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Monticello, IL, USA, Sep. 2019, pp. 1160–1167.

[32] B. Li, A. Eryilmaz, and R. Srikant, "On the universality of age-based scheduling in wireless networks," in *Proc. IEEE INFOCOM*, Hong Kong, Apr. 2015, pp. 1302–1310.

[33] A. Maatouk, M. Assaad, and A. Ephremides, "On the age of information in a CSMA environment," *IEEE/ACM Trans. Netw.*, vol. 28, no. 2, pp. 818–831, Apr. 2020.

[34] Y. Xiao and Y. Sun, "A dynamic jamming game for real-time status updates," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Honolulu, HI, USA, Apr. 2018, pp. 354–360.

[35] 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*, Honolulu, HI, USA, Apr. 2018, pp. 908–916.

[36] Z. Ning *et al.*, "Mobile edge computing enabled 5G health monitoring for Internet of Medical Things: A decentralized game theoretic approach," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 12, pp. 463–478, Feb. 2021.

[37] J. Zhong, R. D. Yates, and E. Soljanin, "Backlog-adaptive compression: Age of information," in *Proc. IEEE ISIT*, Aachen, Germany, Jun. 2017, pp. 566–570.

[38] 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*, Vail, CO, USA, Jun. 2018, pp. 2471–2475.

[39] P. Mayekar, P. Parag, and H. Tyagi, "Optimal source codes for timely updates," *IEEE Trans. Inf. Theory*, vol. 66, no. 6, pp. 3714–3731, Jun. 2020.

[40] C. Kam, S. Kompella, and A. Ephremides, "Experimental evaluation of the age of information via emulation," in *Proc. IEEE MILCOM*, Tampa, FL, USA, Oct. 2015, pp. 1070–1075.

[41] V. Marbukh, "Towards managing age of network state information in challenged networks," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Honolulu, HI, USA, Apr. 2018, pp. 1–2.

[42] J. Liebeherr, D. E. Wrege, and D. Ferrari, "Exact admission control for networks with a bounded delay service," *IEEE/ACM Trans. Netw.*, vol. 4, no. 6, pp. 885–901, Dec. 1996.

[43] L. Georgiadis, R. Guerin, and A. Parekh, "Optimal multiplexing on a single link: Delay and buffer requirements," *IEEE Trans. Inf. Theory*, vol. 43, no. 5, pp. 1518–1535, Sep. 1997.

[44] M. Andrews, "Probabilistic end-to-end delay bounds for earliest deadline first scheduling," in *Proc. IEEE INFOCOM*, vol. 2, Tel Aviv, Israel, Mar. 2000, pp. 603–612.

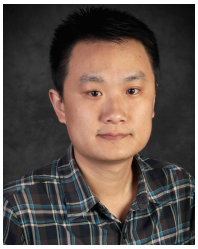
[45] H. Hoang, M. Jonsson, U. Hagstrom, and A. Kallerdahl, "Switched real-time Ethernet with earliest deadline first scheduling protocols and traffic handling," in *Proc. 16th Int. Parallel Distrib. Process. Symp.*, Ft. Lauderdale, FL, USA, 2002, pp. 1–7.

[46] A. B. Kahn, "Topological sorting of large networks," *Commun. ACM*, vol. 5, no. 11, pp. 558–561, Nov. 1962.

[47] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM J. Comput.*, vol. 1, no. 2, pp. 146–160, Jul. 1972.

[48] R. L. Garham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, Reading, MA, USA: Addison-Wesley, 1989, ch. 2.

Chengzhang Li (Student Member, IEEE) received the B.S. degree in electronics engineering from Tsinghua University, Beijing, China, in 2017, and the M.S. degree in computer engineering from Virginia Tech, Blacksburg, VA, USA, in 2020, where he is currently pursuing the Ph.D. degree with the Bradley Department of Electrical and Computer Engineering. His current research interests are modeling, analysis, and algorithm design for wireless networks, with a focus on age of information (AoI), 5G, and ultra-low latency research.



Qingyu Liu (Member, IEEE) received the B.S. degree in computer science from Qingdao University, Qingdao, China, in 2011, the M.S. degree in computer science from Tsinghua University, Beijing, China, in 2014, and the Ph.D. degree in computer engineering from Virginia Tech, Blacksburg, VA, USA, in 2019. In 2019, he joined the Bradley Department of Electrical and Computer Engineering, Virginia Tech, where he is currently a Research Assistant Professor. His research interests include wireless and mobile networks, 5G/next-G, CPS/IoT, intelligent transportation, and optimization/algorithm design in networked systems.

Shaoran Li (Student Member, IEEE) received the B.S. degree from Southeast University, Nanjing, China, in 2014, and the M.S. degree from the Beijing University of Posts and Telecommunications (BUPT), China, in 2017. He is currently pursuing the Ph.D. degree with Virginia Tech, Blacksburg, VA, USA. His research interests include algorithm design and real-time implementation for wireless networks. In 2019, his paper won the Fred W. Ellersick MILCOM Award for the best paper in the unclassified technical program.

Yongce Chen (Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from the Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2013 and 2016, respectively, and the Ph.D. degree in electrical engineering from Virginia Tech, Blacksburg, VA, USA, in 2021. During his Ph.D. study at Virginia Tech, he was awarded a VT Wireless Fellowship in 2016 and a Pratt Fellowship in 2021. He is currently a Senior System Software Engineer at NVIDIA Corporation, Santa Clara, CA, USA. His current research interests include optimization, MIMO techniques, and real-time implementation for wireless networks. He received a Best Paper Award at IEEE INFOCOM 2021.

Y. Thomas Hou (Fellow, IEEE) received the Ph.D. degree from the NYU Tandon School of Engineering (formerly Polytechnic University) in 1998. From 1997 to 2002, he was a member of Research Staff at Fujitsu Laboratories of America, Sunnyvale, CA, USA. He is currently Bradley Distinguished Professor of electrical and computer engineering at Virginia Tech, Blacksburg, VA, USA, which he joined in 2002. He has over 300 papers published in IEEE/ACM journals and conferences. His papers were recognized by nine best paper awards from IEEE and ACM. He holds six U.S. patents. He has authored/coauthored two graduate textbooks, including *Applied Optimization Methods for Wireless Networks* (Cambridge University Press, 2014) and *Cognitive Radio Communications and Networks: Principles and Practice* (Academic Press/Elsevier, 2009). His current research focuses on developing innovative solutions to complex science and engineering problems arising from wireless and mobile networks. He is also interested in wireless security. He was named an IEEE Fellow for contributions to modeling and optimization of wireless networks. He was a member of the IEEE Communications Society Board of Governors and served as the Steering Committee Chair for IEEE INFOCOM Conference. He was/is on the editorial boards of a number of ACM and IEEE TRANSACTIONS and journals. He was a Distinguished Lecturer of the IEEE Communications Society.

Wenjing Lou (Fellow, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of Florida. She is currently the W. C. English Endowed Professor of computer science at Virginia Tech, Arlington, VA, USA. Her research interests cover many topics in the cybersecurity field, with current research interests focusing on wireless networks, privacy protection in machine learning systems, and security and privacy problems in the Internet of Things (IoT) systems. She is a Highly Cited Researcher by the Web of Science Group. She is a Steering Committee Member of IEEE INFOCOM and IEEE TRANSACTIONS ON MOBILE COMPUTING. She served as the Program Director for the U.S. National Science Foundation (NSF) from 2014 to 2017. She received the Virginia Tech Alumni Award for Research Excellence in 2018, which is the highest university level faculty research award. She received the INFOCOM Test-of-Time Paper Award in 2020. She was a TPC Chair for IEEE INFOCOM 2019 and ACM WiSec 2020. She was the Steering Committee Chair for IEEE CNS Conference from 2013 to 2020.

Sastry Kompella (Senior Member, IEEE) received the Ph.D. degree in electrical and computer engineering from Virginia Polytechnic Institute and State University, Blacksburg, VA, USA, in 2006. He is currently the Section Head of the Wireless Network Research Section under the Information Technology Division, U.S. Naval Research Laboratory, Washington, DC, USA. His research interests include various aspects of wireless networks, including cognitive radio, dynamic spectrum access, and age of information.