AoI Scheduling with Maximum Thresholds

Chengzhang Li, Shaoran Li, Yongce Chen, Y. Thomas Hou, and Wenjing Lou
Virginia Tech, Blacksburg, VA, USA

***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 each source node has an AoI requirement (which we call Maximum AoI Threshold (MAT)). Specifically, we want to determine whether or not a vector of MATs 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 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.**

I. Introduction

Age of Information (AoI) is a new metric conceived by Kaul *et al.* [1], [2] to measure the *freshness* of information. It has since captured the attention of the research community and is now under intensive investigation (see a survey on AoI in [3] and an online bibliography in [4]). By definition [1], [2], AoI measures the elapsed time 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).

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 [5]–[28]. In [5]–[14], the authors considered a single-link network where multiple sources share a common link and only one source can use the link for transmission at one time instance. In [15]–[20], 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. In [21], [22], the authors considered a cellular-based network where the channel resource is organized as resource grids that can be allocated to the source nodes. In [23]–[25], the authors considered a multi-hop network.

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 the AoI from some input to meet certain freshness threshold. Although existing schedulers designed for AoI minimization has some remote relevance to AoI thresholds, 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 thresholds. Further, by scanning the literature, there is a serious lack of current 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. 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 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 [29]–[32]. In particular, *Earliest Deadline First* (EDF), the most well-known scheduler, was shown to be very efficient in the task scheduling problem [29], [30]. 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 follows:

- 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 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*

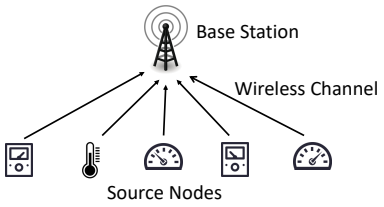

Figure 1: System model: N source nodes collect information and forward it to a BS.

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 a polynomial MAT 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 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 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.

To the best of our knowledge, this is the first paper that successfully addresses AoI scheduling problems subjected to maximum thresholds. As such, the theories and algorithms developed in this paper lay the groundwork for future studies on AoI scheduling problems subject to performance guarantees.

II. System Model

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 [8], [11], [12], [14], 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.

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.,

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

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 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.

III. Problem Statement

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 $i = 1, 2, \cdots, N$.[1] 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.

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

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. 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 [5]–[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 totally different problem, which we will address in this paper.

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), which will serve 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 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 1 *If an MAT vector $\boldsymbol{d}$ is schedulable, then there exists at least one cyclic scheduler that is feasible w.r.t. $\boldsymbol{d}$.*

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., $\boldsymbol{s}(t) = [A_1(t) \ A_2(t) \ \cdots \ A_N(t)]$. 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

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

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 MAT d_i will not be satisfied and thus infeasible). Denote $d_{\max}$ as the largest MAT 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.,

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

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

Proof of Lemma 1 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 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., 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" and "feasible cyclic scheduler" interchangeably in this paper.

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, which is a well-known problem with a known solution [33], [34].

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. $\boldsymbol{d}$ 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. $\boldsymbol{d}$, 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 [33] and Depth-First-Search (DFS) [34]. 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.[2] 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.

[2]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.

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.

Figure 2: CSD procedure

V. The Special Case of Polynomial MAT Vectors

Before we start out to design any scheduling algorithm,[3] 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 IV, 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 MAT vector, called *polynomial* MAT vector. We show that for this special MAT vector, we can design a procedure that meets all three objectives. 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

We first give a definition of polynomial MAT vector.

Definition 1 *An MAT 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 MAT vector with $b = 5$, $n_1 = n_2 = 0$, $n_3 = 1$, $n_4 = n_5 = 2$ and $n_6 = 3$. 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 [35]. 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

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

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

Eq. (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,

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

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 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

$$(ABCADEABCADF),$$

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**

Figure 3: A pseudocode for PSC

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 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. $\boldsymbol{d}$.*

A proof (based on contradiction) can be easily constructed. Due to space limitation, we omit it here. 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 $O(Nd_{\max})$.

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

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 meet all three objectives 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 MAT that is smaller than its requirement. If we can do this mapping and the 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 $1 \le i \le N$, 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 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 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 $l(\widetilde{\boldsymbol{d}}) \le 1$. 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.

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 $l(\widetilde{\boldsymbol{d}}) \le 1$, there exist a feasible scheduler w.r.t. $\boldsymbol{d}$.*

A proof of Theorem 1 is based on constructing one such feasible scheduler, which 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.

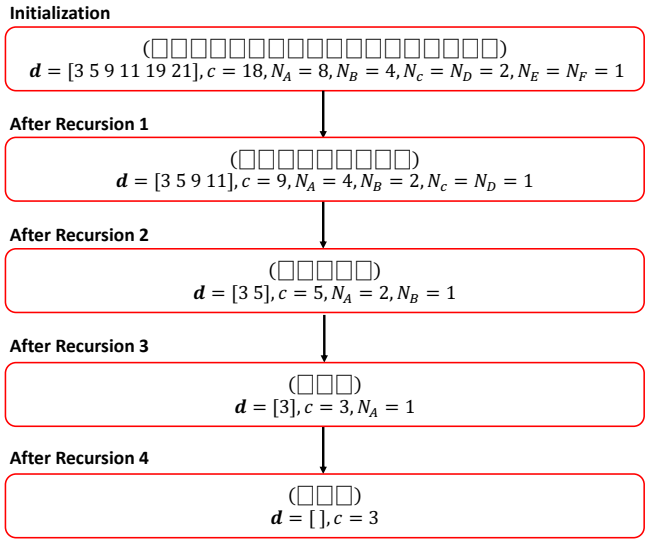

Figure 4: Recursion in the example.

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 $1 \le i \le N$.

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 $\sum_{i=1}^{N} r_i \le 1$ when $l(\widetilde{\boldsymbol{d}}) \le 1$. 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. 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, the proof of Theorem 1 can now be easily constructed (by following the above example and the pseudocode in Figure 5). We omit it here due to space limitation.

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.

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 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. ∎

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 $\boldsymbol{d}_2$.

Figure 5: A pseudocode for FSC.

Based on Lemma 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

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

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.

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 $d_{\max}$.*

FPM: For an MAT vector $\boldsymbol{d}$:
1: **for** $i = 1, 2, \cdots, N$ **do**
2: Compute $\widetilde{d}_j$ for each $1 \le j \le N$ by (9). Compute $l(\widetilde{\boldsymbol{d}}) = \sum_{j=1}^{N} 1/\widetilde{d}_j$.
3: **if** $l(\widetilde{\boldsymbol{d}}) \le 1$ **then**
4: Call FSC to construct a feasible scheduler w.r.t. $\boldsymbol{d}$ and **break**.
5: **end if**
6: **end for**

Figure 6: A Pseudocode of FPM.

Proof Suppose at iteration i, $\widetilde{d}_i = d_i$, $l(\widetilde{\boldsymbol{d}}) \le 1$ 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 $c = \widetilde{d}_{\max} \le d_{\max}$. ∎

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 $l(\widetilde{\boldsymbol{d}}) \le 1$, then FPM will call FSC (at most once). By Lemma 6, we have $c \le d_{\max}$ 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[4] 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 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 $l(\widetilde{\boldsymbol{d}}) \le 1$. However, since $d_i \ge \widetilde{d}_i$, we have $l(\boldsymbol{d}) \le l(\widetilde{\boldsymbol{d}})$. 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.

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 $l(\boldsymbol{d}) \le 0.5$, 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 *For any $\boldsymbol{d}$ with $l(\boldsymbol{d}) < \ln 2$, FPM can always construct a feasible scheduler w.r.t. $\boldsymbol{d}$.*

Proof Assume $\boldsymbol{d}$ is a vector that FPM cannot find a feasible scheduler for. To prove Proposition 1, it's sufficient to prove $l(\boldsymbol{d}) > \ln 2$.

Since FPM cannot find a $\widetilde{\boldsymbol{d}}$ with $l(\widetilde{\boldsymbol{d}}) \le 1$ that $\boldsymbol{d}$ can be mapped to, we have

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

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

Based on (10), by using the same method in the proof of Lemma 5, we can prove that for any $x > 0$,

$$\sum_{j=1}^{N} \frac{1}{x \cdot 2^{\lfloor \log_2(\frac{d_j}{x}) \rfloor}} > 1. \quad (11)$$

Denote $t = 1/x$. We have for any $t > 0$,

$$\sum_{j=1}^{N} t \cdot 2^{-\lfloor \log_2(d_j t) \rfloor} > 1. \quad (12)$$

Denote $g(t) = \frac{1}{\ln 2 \cdot t}$. We have $\int_{0.5}^{1} g(t)dt = 1$. Denote

$$u_j = \frac{1}{d_j} \cdot 2^{\lfloor \log_2(d_j) \rfloor}. \quad (13)$$

We have $u_j \in (0.5, 1]$ for each j.

Considering (12), we have

$$\begin{aligned}
&\int_{0.5}^{1} g(t) \sum_{j=1}^{N} t \cdot 2^{-\lfloor \log_2(d_j t) \rfloor} dt > 1 \\
\Longrightarrow\ &\frac{1}{\ln 2} \sum_{j=1}^{N} \int_{0.5}^{1} 2^{-\lfloor \log_2(d_j t) \rfloor} dt > 1 \\
\Longrightarrow\ &\frac{1}{\ln 2} \sum_{j=1}^{N} \Big(\int_{0.5}^{u_j} 2^{-\lfloor \log_2(d_j) \rfloor + 1} dt \\
&\quad + \int_{u_j}^{1} 2^{-\lfloor \log_2(d_j) \rfloor} dt \Big) > 1 \\
\Longrightarrow\ &\frac{1}{\ln 2} \sum_{j=1}^{N} u_j \cdot 2^{-\lfloor \log_2(d_j) \rfloor} > 1 \\
\Longrightarrow\ &\frac{1}{\ln 2} \sum_{j=1}^{N} \frac{1}{d_j} > 1 \\
\Longrightarrow\ &l(\boldsymbol{d}) > \ln 2.
\end{aligned} \quad (14)$$

This completes the proof. ∎

Proposition 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 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 $l(\widetilde{\boldsymbol{d}}) \le 1$, 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 guarantee that FPM can offer.

VIII. Numerical Results

In this section, we show some numerical results for FPM and compare it to CSD (in Section IV) and EDF. Here the EDF scheduler is given as

$$\pi^{\text{EDF}}(t) = \arg\min_i \big(d_i - A_i(t)\big), \quad (15)$$

which selects the source with the earliest deadline for transmission at each t. We organize our numerical results separately for small N and large N. Under a small N, CSD can be used

Table I: Performance of EDF, FPM and CSD on different $\boldsymbol{d}$'s.

$\boldsymbol{d}$	$l(\boldsymbol{d})$	FPM	CSD	EDF
[3 12 13 13]	0.571	✓$c=8$	✓$c=117$	✗
[5 8 10 12 13]	0.585	✓$c=8$	✓$c=429$	✓$c=131$
[3 7 8]	0.601	✓$c=8$	✓$c=48$	✓$c=6$
[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	✗	✓$c=24$	✗

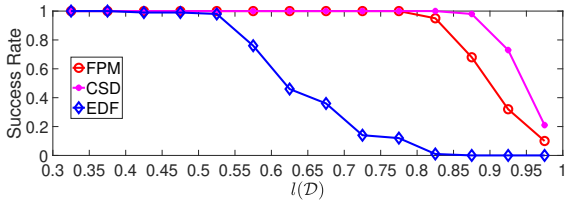

Figure 7: Success rates for FPM, CSD, and EDF under different $l(\boldsymbol{d})$ when $N=5$.

to find a feasible solution; while under a large N, CSD is not useful due to its exponential time complexity.

Small N. We consider small networks where we are able to execute CSD even with exponential time complexity. In Table I, we present the 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 $l(\boldsymbol{d})<\ln 2$, 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.

What are missing in Table I are the feasible schedulers that are found by FPM. To conserve space, we present the schedulers for two entries of $\boldsymbol{d}$'s in the table as follows: (i) For $\boldsymbol{d}=[3\ 5\ 7\ 10\ 12]$, the scheduler found by FPM is $(ABACDABACE)$; (ii) For $\boldsymbol{d}=[3\ 6\ 6\ 7\ 13\ 14]$, the scheduler found by FPM is $(ABCADEABCADF)$. The readers can easily verify their feasibility.

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 $l(\boldsymbol{d})\in(0.3,0.35],(0.35,0.4],\cdots,(0.95,1]$. For each $\boldsymbol{d}$, we run FPM, CSD and EDF and calculate the rate (percentage) of success for finding a feasible scheduler. In Fig. 7, we see that the performance of FPM is close to CSD (the optimal algorithm). In particular, when $l(\boldsymbol{d})<\ln 2$, the success rate of FPM is 100%, which confirms the result in Proposition 1. Again, EDF has the worst performance.

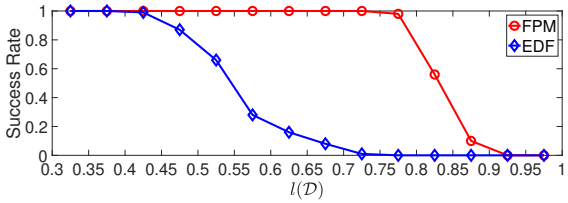

Figure 8: Success rate for FPM and EDF under different $l(\boldsymbol{d})$ when $N=100$.

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 $N=100$. We assume $d_i\in\{10,20,30,\cdots,800\}$ for each source node $i=1,2,\cdots,100$, and we randomly generate 100 different $\boldsymbol{d}$'s for each load interval $l(\boldsymbol{d})\in(0.3,0.35],(0.35,0.4],\cdots,(0.95,1]$. For each $\boldsymbol{d}$, we run FPM and EDF[5] 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 $l(\boldsymbol{d})<\ln 2$, the success rate of FPM is 100%, which confirms the result in Proposition 1.

IX. Conclusions

This paper studied AoI scheduling subject to 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 $l(\boldsymbol{d})\le 1$. 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 $l(\widetilde{\boldsymbol{d}})\le 1$. 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.

Acknowledgment

This research was supported in part by ONR MURI grant N00014-19-1-2621.

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

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] Y. Sun, "A Collection of Recent Papers on the Age of Information," available at http://www.auburn.edu/\%7eyzs0078

[5] Y. Kuo, "Minimum Age TDMA Scheduling," in *Proc. of IEEE INFOCOM,* pp. 2296–2304, Paris, France, Apr. 29 – May 2, 2019.

[6] 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

[7] 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

[8] 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

[9] 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.

[10] 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.

[11] 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.

[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] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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. of IEEE ICDCS,* pp. 1466–1472, Dallas, TX, USA, July 7–9, 2019.

[22] C. Li, S. Li, and Y.T. Hou, "A General Model for Minimizing Age of Information at Network Edge," in *Proc. of IEEE INFOCOM,* pp. 118–126, Paris, France, Apr. 29 – May 2, 2019.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] A.B. Kahn, "Topological Sorting of Large Networks," *Communications of the ACM*, vol. 5, issue 11, pp. 558–561, Nov. 1962.

[34] R. Tarjan, "Depth-First Search and Linear Graph Algorithms," *SIAM Journal on Computing*, vol. 1, issue 2, pp. 146–160, 1972.

[35] R.L. Garham, D.E. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, Chapter 2, Addison-Wesley, 1989.