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

Zhongdong Liu, Liang Huang, Bin Li, and Bo Ji

***Abstract*—The Age-of-Information (AoI) is a new performance metric recently proposed for measuring the freshness of information in information-update systems. In this work, we conduct a systematic and comparative study to investigate the impact of scheduling policies on the AoI performance in single-server queues and provide useful guidelines for the design of AoI-efficient scheduling policies. Specifically, we first perform extensive simulations to demonstrate that the update-size information can be leveraged for achieving a substantially improved AoI compared to non-size-based (or arrival-time-based) policies. Then, by utilizing both the update-size and arrival-time information, we propose three AoI-based policies. Observing improved AoI performance of policies that allow service preemption and that prioritize informative updates, we further propose preemptive, informative, AoI-based scheduling policies. Our simulation results show that such policies empirically achieve the best AoI performance among all the considered policies. Interestingly, we also prove sample-path equivalence between some size-based policies and AoI-based policies. This provides an intuitive explanation for why some size-based policies, such as Shortest-Remaining-Processing-Time (SRPT), achieve a very good AoI performance.**

I. Introduction

Recently, the study of information freshness has received increasing attentions, especially for time-sensitive applications that require real-time information/status updates, such as road congestion information, stock quotes, and weather forecast. In order to measure the freshness of information, a new metric, called the *Age-of-Information (AoI)*, is proposed. The AoI is defined as the time elapsed since the generation of the freshest update among those that have been received by the destination [1]. Prior studies reveal that the AoI depends on both the inter-arrival time and the delay of the updates. Due to the dependency between the inter-arrival time and the delay, this new AoI metric exhibits very different characteristics than the traditional delay metric and is generally much harder to analyze (see, e.g., [1]).

Although it is well-known that scheduling policies play an important role in reducing the delay in single-server queues, it remains largely unknown how exactly scheduling policies impact the AoI performance. To that end, we aim to holistically study the impact of various aspects of scheduling policies on the AoI performance in single-server queues and provide useful guidelines for the design of scheduling policies that can achieve a small AoI.

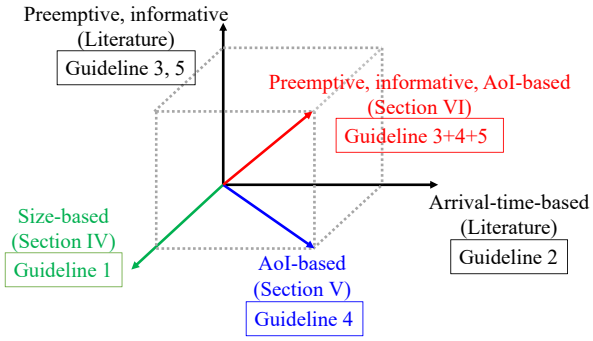


Fig. 1: Our position in the design space of AoI-efficient scheduling policies for a G/G/1 queue

While much research effort has already been exerted to the design and analysis of scheduling policies aiming to reduce the AoI, almost all of these policies are only based on the arrival time of updates, such as First-Come-First-Served (FCFS) and Last-Come-First-Served (LCFS), assuming that the update-size information is unavailable. Here, the size of an update is the amount of time required to serve the update if there were no other updates around. In some applications, such as smart grid and traffic monitoring, the update-size information can be obtained or fairly well estimated [2]. It has been shown that scheduling policies that leverage the size information can substantially reduce the delay, especially when the system load is high or when the size variability is large [3]. This motivates us to investigate the AoI performance of size-based policies in a G/G/1 queue. Note that the update-size information is "orthogonal" to the arrival-time information, both of which could significantly impact the AoI performance. Therefore, it is quite natural to further consider AoI-based policies that use both the update-size and arrival-time information of updates.

In addition, prior work has revealed that scheduling policies that allow service *preemption* and that prioritize *informative* updates (also called *effective* updates, which are those that lead to a reduced AoI once delivered; see Section VI-A for a formal definition) yield a good AoI performance [4]–[6]. Intuitively, preemption prevents fresh updates from being blocked by a large and/or stale update in service; informative policies discard stale updates, which do not bring new information but may block fresh updates. To that end, we also consider AoI-based scheduling designs that both allow service preemption

This work was supported in part by the NSF under Grants CCF-1657162, CNS-1651947, and CNS-1717108.

Zhongdong Liu (zhongdong.liu@temple.edu) and Bo Ji (boji@temple.edu) are with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA. Liang Huang (lianghuang@zjut.edu.cn) is with the College of Information Engineering, Zhejiang University of Technology, Hangzhou, China. Bin Li (binli@uri.edu) is with the Department of Electrical, Computer and Biomedical Engineering, University of Rhode Island, Kingston, Rhode Island.

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

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

and prioritize informative updates.

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

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

Second, leveraging both the update-size and arrival-time information, we introduce Guideline 4 and propose AoI-based scheduling policies (i.e., the blue arrow in Fig. 1). These AoI-based policies attempt to optimize the AoI at a specific future time instant from three different perspectives: the AoI-Drop-Earliest (ADE) policy, which makes the AoI drop the earliest; the AoI-Drop-to-Smallest (ADS) policy, which makes the AoI drop to the smallest; the AoI-Drop-Most (ADM) policy, which makes the AoI drop the most. The simulation results show that such AoI-based policies indeed have a good AoI performance.

Third, we observe that informative policies can significantly improve the AoI performance compared to their non-informative counterparts, which leads to Guideline 5. Integrating all the guidelines, we propose preemptive, informative, AoI-based policies (i.e., the red arrow in Fig. 1). The simulation results show that such policies empirically achieve the best AoI performance among all the considered policies.

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

To summarize, our study reveals that among various aspects of scheduling policies we investigated, prioritizing small updates, allowing service preemption, and prioritizing informative updates play the most important role in the design of AoI-efficient scheduling policies.

The rest of this paper is organized as follows. We first discuss related work in Section II. Then, we describe our system model in Section III. In Section IV, we evaluate the AoI performance of size-based scheduling policies. We further propose AoI-based scheduling policies in Section V. In addition, we evaluate the AoI performance of preemptive, informative, AoI-based policies in Section VI. Finally, we make concluding remarks in Section VII.

II. Related work

The traditional queueing literature on single-server queues is largely focused on the delay analysis. In [7], the authors prove that all non-preemptive scheduling policies that do not make use of job size information have the same distribution of the number of jobs in the system. The work of [8], [9] proves that for a work-conserving queue, the SRPT policy minimizes the number of jobs in the system at any point and is therefore delay-optimal. The work of [10] derives a formula of the average delay for several common scheduling polices (which will be discussed in Section IV).

On the other hand, although the AoI research is still in a nascent stage, it has already attracted a lot of interests (see [11], [12] for a survey). Here we only discuss the most relevant work, which is focused on the AoI-oriented queueing analysis. Much of existing work considers scheduling policies that are based on the arrival time (such as FCFS and LCFS). The AoI is introduced in [1], where the authors study the average AoI in the M/M/1, M/D/1, and D/M/1 queues under the FCFS policy. In [13], the AoI performance of the FCFS policy in the M/M/1/1 and M/M/1/2 queues is studied, where new arrivals are discarded if the buffer is full. The average AoI of the LCFS policy in the M/M/1 queue is also discussed in [13].

There has been some work that aims to reduce the AoI by making use of service preemption. In [14], the average AoI of LCFS in the M/M/1 queue with and without service preemption is analyzed. The work of [15] is quite similar to [14], but it considers the average AoI in the M/M/2 queue. In [16], the average AoI for the M/G/1/1 preemptive system with a multi-stream updates source is derived. The age-optimality of the preemptive LCFS (LCFS_P) policy is proved in [4], where the service times are exponentially distributed.

In addition to taking advantage of service preemption, some of the prior studies also consider the strategy of prioritizing informative updates for reducing the AoI. The work of [5], [6]

reveals that the AoI performance can be improved by prioritizing informative updates and discarding non-informative policies when making scheduling decisions. In [17], the authors consider a G/G/1 queue with informative updates and derive the stationary distribution of the AoI, which is in terms of the stationary distribution of the delay and the Peak AoI (PAoI). With the AoI distribution, one can analyze the mean or higher moments of the AoI in GI/GI/1, M/GI/1, and GI/M/1 queues under several scheduling policies (e.g., FCFS and LCFS).

Recent research effort has also been exerted to understanding the relation between the AoI and the delay. In [18], the authors analyze the tradeoff between the AoI and the delay in a single-server M/G/1 system under a specific scheduling policy without knowing the service time of each individual update. In [19], the violation probability of the delay and the PAoI is investigated under an additive white Gaussian noise (AWGN) channel, but the update size is assumed to be identical.

III. System Model

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

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

We use t_i and t_i' to denote the time at which the i-th update was generated at the source and the time at which it leaves the server, respectively. The AoI at time t is then defined as $\Delta(t) \triangleq t - U(t)$, where $U(t) \triangleq \max_i \{t_i : t_i' \leq t\}$ is the generation time of the freshest update among those that have been processed by the server. An example of the AoI evolution under the FCFS policy is shown in Fig. 2. Then, the average AoI can be defined as

$$\Delta = \lim_{t\to\infty} \frac{1}{t} \int_0^t \Delta(\tau) d\tau. \tag{1}$$

In general, the analysis of the average AoI is quite difficult since it is determined by two dependent quantities: the inter-arrival time and the delay of updates [1]. We define the inter-arrival time between the i-th update and $(i-1)$-th update as $X_i \triangleq t_i - t_{i-1}$ and define the delay of the i-th update as $T_i \triangleq t_i' - t_i$. Alternatively, the Peak AoI (PAoI) is also proposed as an information freshness metric [5], which is defined as the maximum value of the AoI before it drops due to a newly delivered fresh update. Let A_i be the i-th PAoI. From Fig. 2, we can see $A_i = t_i' - t_{i-1}$. This can be rewritten as the sum of the inter-arrival time between the i-th update and the previous update (i.e., X_i) and the delay of the i-th update (i.e., T_i). Therefore, the PAoI of the i-th update can also be expressed as $A_i = X_i + T_i$, and its expectation is $\mathbb{E}[A_i] = \mathbb{E}[X_i] + \mathbb{E}[T_i]$.

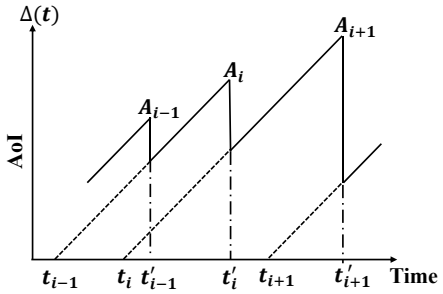


Fig. 2: An example of the AoI evolution under the FCFS policy

IV. Size-based policies

In this section, we investigate the AoI performance of several common scheduling policies, including size-based policies and non-size-based policies, via extensive simulations. Note that these common scheduling policies may serve the non-informative updates (which do not lead to a reduced AoI). This is because in some applications, such as news and social network, obsolete updates are still useful and need to be served [4]. In Section VI, we will discuss the case where obsolete updates are discarded.

Following [3], we first give the definitions of several common scheduling policies that can be divided into four types: depending on whether they are size-based or not, where the size-based policies use the update-size information (which is available in some applications, such as smart grid [2]) for making scheduling decisions; depending on whether they are preemptive or not. The definition of preemption is given below. In this paper, we do not consider the cost of preemption.

Definition 1. *A policy is preemptive if an update may be stopped partway through its execution and then restarted at a later time without losing intermediary work.*

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

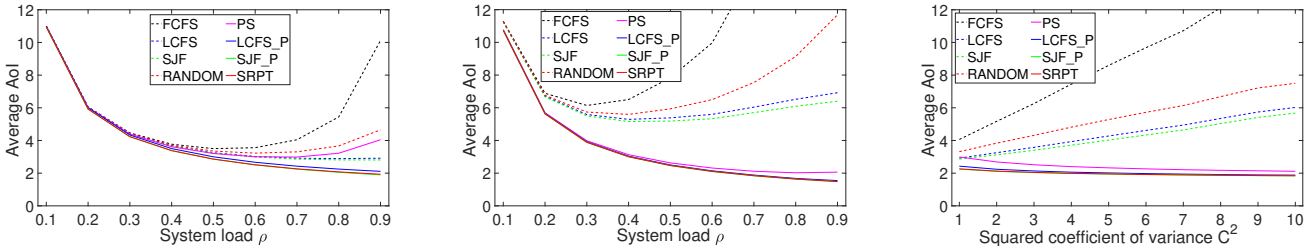
- *First-Come-First-Served (FCFS)*: When the server frees up, it chooses to serve the update that arrived first if any.
- *Last-Come-First-Served (LCFS)*: When the server frees up, it chooses to serve the update that arrived last if any.
- *Random-Order-Service (RANDOM)*: When the server frees up, it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

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

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

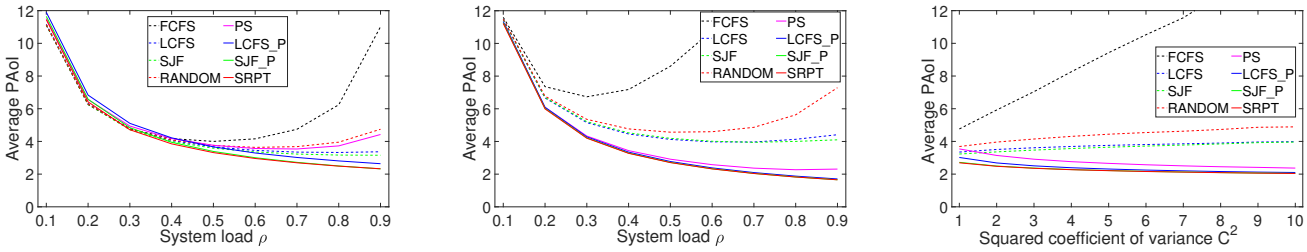
The third type consists of policies that are preemptive and blind to the update size:

- *Processor-Sharing (PS)*: All the updates in the system are served simultaneously and equally (i.e., each update receives an equal fraction of the available service capacity).
- *Preemptive Last-Come-First-Served (LCFS_P)*: This is the preemptive version of the LCFS policy. Specifically, a preemption happens when there is a new update.



(a) Exponential: $\mu = 1$ (b) Weibull: $\mu = 1$ and $C^2 = 10$ (c) Weibull: $\mu = 1$ and $\rho = 0.7$

Fig. 3: Comparisons of the average AoI performance under several common scheduling policies



(a) Exponential: $\mu = 1$ (b) Weibull: $\mu = 1$ and $C^2 = 10$ (c) Weibull: $\mu = 1$ and $\rho = 0.7$

Fig. 4: Comparisons of the average PAoI performance under several common scheduling policies

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

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

Previous work (see, e.g., [3, Section VII]) reveals that size-based policies can greatly improve the delay performance. Due to such results, we conjecture that size-based policies also achieve a better AoI performance given that the AoI is dominantly determined by the delay when the system load is high or when the size variability is large [1]. As we mentioned earlier, it is in general very difficult to obtain the exact expression of the average AoI except for some special cases (e.g., FCFS and LCFS) [1], [17]. Therefore, we attempt to investigate the AoI performance of size-based policies through extensive simulations.

In Fig. 3 and 4, we present the simulation results of the average AoI and PAoI performance under the scheduling policies we introduced above, respectively. Here we assume that a single source generates updates according to a Poisson process with rate λ, and the update size is independent and identically distributed (*i.i.d.*). In Fig. 3(a), we assume that the update size follows an exponential distribution with mean $1/\mu = 1$. In Figs. 3(b) and 3(c), we assume that the update size follows a Weibull distribution with mean $1/\mu = 1$. We define the squared coefficient of variation of the update size as $C^2 \triangleq \mathrm{Var}(S)/\mathbb{E}[S]^2$, i.e., the variance normalized by the square of the mean [3]. Hence, a larger C^2 means a larger variability. In Fig. 3(b), we fix $C^2 = 10$ and change the value of system load ρ, while in Fig. 3(c), we fix system load $\rho = 0.7$ and change the value of C^2. Note that throughout the paper, these simulation settings are used as default settings unless otherwise specified.

In the following, we will discuss key observations from the simulation results and propose useful guidelines for the design of AoI-efficient policies. Note that similar observations can also be made for the G/G/1 queue. An additional interesting observation is that the average PAoI could be much smaller than the average AoI when the interarrival time has a large variability. More detailed simulation results can be found in our technical report [20].

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

In Fig. 3, we can see that for the non-preemptive case, SJF has a better average AoI performance than FCFS, RANDOM, and LCFS in various settings. Similarly, for the preemptive case, SJF_P and SRPT have a better average AoI performance than PS and LCFS_P. Similar observations can be made for the average PAoI performance in Fig. 4.

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

In Figs. 3(a) and 3(b), we can see that under SJF_P and SRPT, the average AoI decreases as the system load ρ increases. There are two reasons. First, when ρ increases, there will be more updates with small size arriving to the queue when C^2 is fixed. Therefore, size-based policies that prioritize updates with small size lead to more frequent AoI drops.

Second, preemption operations prevent fresh updates from being blocked by a large or stale update in service. Similar observations can be made for the average PAoI performance in Figs. 4(a) and 4(b).

Observations 1 and 2 lead to the following guideline:

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

However, in certain application scenarios, the update-size information may not be available or is difficult to estimate. Hence, the scheduling decisions have to be made without the update-size information. In such scenarios, we make the following observations from Figs. 3 and 4.

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

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

Observations 3 and 4 have also been made in previous work [4], [13], [21]. It is quite intuitive that when the update-size information is unavailable, one should give a higher priority to more recent updates. This is because while all the updates have the same expected service time, the most recent update arrives the last and thus leads to the smallest AoI once delivered. Therefore, Observations 3 and 4 lead to the following guideline:

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

Note that Observations 2 and 4 also suggest that under preemptive policies, the average AoI/PAoI decreases as the system load ρ increases. This is because preemptions prevent fresh updates from being blocked by a large or stale update in service. In addition, we have also observed the following nice properties of preemptive policies.

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

In Figs. 3(a) and 3(b), we can see that preemptive policies (e.g., LCFS_P, SJF_P, and SRPT) generally have a better average AoI performance than non-preemptive ones (e.g., FCFS, RANDOM, LCFS, and SJF), especially when the system load is high. In Fig. 3(c), we can see that the advantage of preemptive policies becomes larger as the update-size variability (i.e., C^2) increases. Moreover, the AoI performance of preemptive policies is only very slightly impacted when the update-size variability changes, while that of non-preemptive policies varies significantly. Therefore, Observations 2, 4, and 5 lead to the following guideline:

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

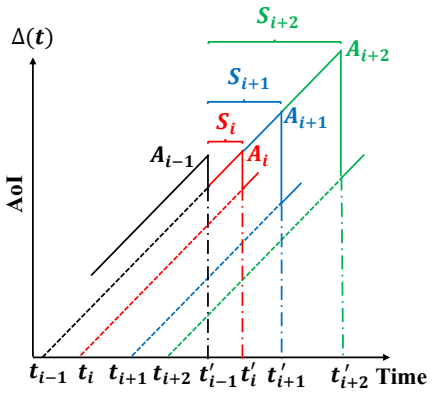


Fig. 5: The AoI evolution under three AoI-based policies: ADE (red), ADS (blue), and ADM (green)

V. AoI-based policies

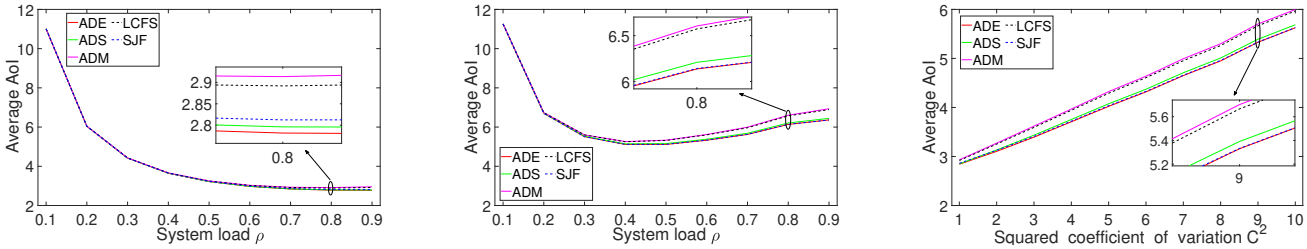
In Section IV, we have demonstrated that size-based policies achieve a better average AoI/PAoI performance than non-size-based policies. However, size-based policies do not utilize the arrival-time information, which also plays an important role in reducing the AoI. In this section, we propose three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

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

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

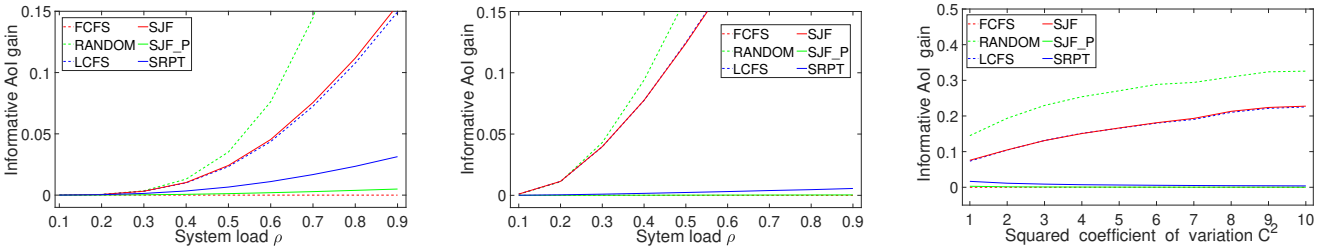
If all updates waiting in the queue are obsolete, then the above policies choose to serve an update with the smallest size.

Although all of these AoI-based policies are quite intuitive, they behave very differently. In order to explain the differences of these AoI-based policies, we present an example in Fig. 5 to show how the AoI evolves under these policies. Suppose that when the $(i-1)$-st update is being served, three new updates (i.e., the i-th, $(i+1)$-st, and $(i+2)$-nd updates) arrive in sequence at times t_i, t_{i+1}, and t_{i+2}, respectively. The sizes of these updates satisfy $S_i < S_{i+1} < S_{i+2}$. When the server frees up after it finishes serving the $(i-1)$-st update at time t'_{i-1}, ADE, ADS, and ADM choose to serve the i-th, $(i+1)$-st, and $(i+2)$-nd updates, respectively. This is because serving the i-th update leads to the earliest AoI drop at time t'_i (following the red curve), serving the $(i+1)$-st update leads to the AoI dropping to the smallest at time t'_{i+1} (following the blue curve), and serving the $(i+2)$-nd update leads to the largest



(a) Exponential: $\mu = 1$ (b) Weibull: $\mu = 1$ and $C^2 = 10$ (c) Weibull: $\mu = 1$ and $\rho = 0.7$

Fig. 6: Comparisons of the average AoI performance: AoI-based policies vs. non-AoI-based policies



(a) Exponential: $\mu = 1$ (b) Weibull: $\mu = 1$ and $C^2 = 10$ (c) Weibull: $\mu = 1$ and $\rho = 0.7$

Fig. 7: Comparisons of the average AoI performance: informative policies vs. non-informative policies

AoI drop at time t'_{i+2} (following the green curve). Clearly, ADE, ADS and ADM aim to optimize AoI at a specific future time instant (i.e., the future delivery time of chosen update) with different myopic goals. Note that at first glance, ADS and ADM may look the same. Indeed, they would be equivalent if the events of AoI drop have happened at the same time instant. However, these two policies are different as the time instants at which the AoI drops are not necessarily the same (e.g., t'_{i+1} vs. t'_{i+2} in Fig. 5).

Next we conduct simulations to investigate the AoI performance of these AoI-based policies. In this paper, we present the simulation results for the AoI performance only due to space limitation. The results for the PAoI performance can be found in our technical report [20]. In Fig. 6, we present the simulation results of the average AoI performance of the AoI-based policies compared to a representative arrival-time-based policy (i.e., LCFS) and a representative size-based-policy (i.e., SJF). All the policies considered here are non-preemptive; the preemptive cases will be discussed in Section VI.

In Fig. 6(a), we observe that most AoI-based policies are slightly better than non-AoI-based policies, although their performances are very close. Among the AoI-based policies, ADE is the best, ADM is the worst, and ADS is in-between. This is not surprising that ADM is the worst: although ADM has the largest AoI drop, this is at the cost that it may have to wait until the AoI become large first. ADE being the best suggests that giving a higher priority to small updates (so that the AoI drops as soon as possible) is a good strategy. In Figs. 6(b) and 6(c), similar observations can be made for update size following Weibull distributions.

The above observations lead to the following guideline:

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

VI. Preemptive, Informative, AoI-based Policies

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

A. Informative Policies

As far as the AoI is concerned, there are two types of updates: informative updates and non-informative updates [22]. *Informative updates lead to AoI drops once delivered while non-informative updates do not.* In some applications, such as autonomous vehicles and stock quotes, it is reasonable to discard non-informative updates (which do not help reduce the AoI but may block new updates). In this subsection, we introduce the "informative" versions of various policies, which prioritize informative updates and discards non-informative updates. Then, we use simulation results to demonstrate that informative policies generally have a better average AoI/PAoI performance than the original (non-informative) ones. Furthermore, we rigorously prove that in an G/M/1 queue, the informative version of LCFS is stochastically better than the original LCFS policy.

We use π_I to denote the informative version[1] of policy π. All the scheduling policies we consider have their informative

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

versions. In some cases, the informative version is simply the same as the original policy (e.g., FCFS and LCFS_P).

In Fig. 7, we show the simulation results of the average AoI performance of several informative policies compared to their non-informative counterparts. In order to evaluate the benefit of informative policies, we plot the informative AoI gain, which is the ratio of the difference between the average AoI of the non-informative version and the informative version to the average AoI of the non-informative version. Hence, a larger informative gain means a larger benefit of the informative version. One important observation from Fig. 7 is as follows.

Observation 6. *Informative policies achieve a better average AoI performance than their non-informative counterparts. The informative gain is larger for non-preemptive policies and increases as the system load increases.*

Intuitively, informative policies are expected to outperform their non-informative counterparts because serving non-informative updates cannot reduce the AoI but may block new updates. The simulation results verify this intuition as the informative AoI gain is always non-negative. Second, we can see that most non-preemptive policies (e.g., RANDOM, LCFS, and SJF) benefit more from prioritizing informative updates. Third, as the system load ρ increases, the informative AoI gain increases under most considered policies, especially those non-preemptive ones. This is because as the system load increases, the number of non-informative updates also increases, which has a larger negative impact on the AoI performance for non-preemptive, non-informative policies. Observation 6 leads to the following guideline:

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

Based on Observation 6, we conjecture that an informative policy is as least as good as its non-informative counterpart. As a preliminary result, we prove that this conjecture is indeed true for LCFS in a G/M/1 queue. In the following, we introduce the stochastic ordering notion, which will be used in the statement of Proposition 1.

Definition 2. *Stochastic Ordering of Stochastic Processes [23, Ch.6.B.7]: Let $\{X(t), t \in [0,\infty)\}$ and $\{Y(t), t \in [0,\infty)\}$ be two stochastic processes. Then, $\{X(t), t \in [0,\infty)\}$ is said to be stochastically less than $\{Y(t), t \in [0,\infty)\}$, denoted by $\{X(t), t \in [0,\infty)\} \leq_{\text{st}} \{Y(t), t \in [0,\infty)\}$, if, for all choices of integer n and $t_1 < t_2 < \cdots < t_n$ in $[0,\infty)$, the following holds for all upper sets[2] $S^U \subseteq \mathbb{R}^n$:*

$$\mathbb{P}(\vec{X} \in S^U) \leq \mathbb{P}(\vec{Y} \in S^U), \tag{2}$$

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

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

Roughly speaking, Eq. (2) implies that $\vec{X}$ is less likely than $\vec{Y}$ to take on large values, where "large" means any value in an upper set S^U. We also use $\Delta_\pi(t)$ to denote the AoI process under policy π. Furthermore, we let $\mathcal{I} = \{t_1, t_2 \ldots\}$ to denote a sample path specified by the arrival times of a sequence of updates. Having these definitions and notations, we are now ready to state Proposition 1.

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

$$[\{\Delta_{\text{LCFS\_I}}(t), t \in [0,\infty)\} | \mathcal{I}] \leq_{\text{st}} [\{\Delta_{\text{LCFS}}(t), t \in [0,\infty)\} | \mathcal{I}]. \tag{3}$$

We present a sketch of the proof in the following and provide the detailed proof in our online technical report [20].

Proof sketch. We define the system state at time t under policy π as $S_\pi(t) \triangleq U_\pi(t)$, where $U_\pi(t)$ is the largest arrival time of the updates that have been served under policy π by time t. Let $\{S_\pi(t), t \in [0,\infty)\}$ be the state process of policy π. By the definition of AoI, Eq. (3) holds if the following holds:

$$[\{S_{\text{LCFS\_I}}(t), t \in [0,\infty)\} | \mathcal{I}] \geq_{\text{st}} [\{S_{\text{LCFS}}(t), t \in [0,\infty)\} | \mathcal{I}]. \tag{4}$$

Next, we prove Eq. (4) by contradiction through a coupling argument. Suppose that stochastic processes $\hat{S}_{\text{LCFS\_I}}(t)$ and $\hat{S}_{\text{LCFS}}(t)$ have the same stochastic laws as $S_{\text{LCFS\_I}}(t)$ and $S_{\text{LCFS}}(t)$, respectively. We couple $\hat{S}_{\text{LCFS\_I}}(t)$ and $\hat{S}_{\text{LCFS}}(t)$ in the following manner: If an update i is delivered at t_i' in $\hat{S}_{\text{LCFS}}(t)$, then the update j being served at t_i' (if any) in $\hat{S}_{\text{LCFS\_I}}(t)$ is also delivered at the same time. This coupling is reasonable because: (i) the updates served in $\hat{S}_{\text{LCFS\_I}}(t)$ are not chosen based on update size; (ii) the service time of an update in both $\hat{S}_{\text{LCFS\_I}}(t)$ and $\hat{S}_{\text{LCFS}}(t)$ is exponentially distributed and has the memoryless property. By Theorem 6.B.30 in [23], Eq. (4) holds if the following holds:

$$\mathbb{P}(\hat{S}_{\text{LCFS\_I}}(t) \geq \hat{S}_{\text{LCFS}}(t), t \in [0,\infty) | \mathcal{I}) = 1, \tag{5}$$

which can be proven by contradiction. □

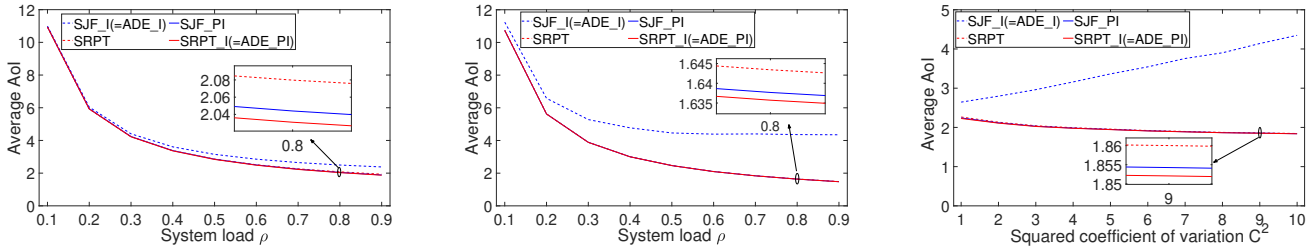
B. Preemptive, Informative, AoI-based Policies

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

We first consider preemptive, informative version of three AoI-based policies: ADE_PI, ADS_PI, and ADM_PI. Interestingly, we can show equivalence between ADE_PI and SRPT_I (i.e., the informative version of SRPT) and between ADE_I and SJF_I (i.e., the informative version of ADE and SJF, respectively) in the sample-path sense. These results are stated in Propositions 2 and 3.

Proposition 2. *ADE_PI and SRPT_I are equivalent in every sample path.*

Proposition 3. *ADE_I and SJF_I are equivalent in every sample path.*



(a) Exponential: $\mu = 1$ (b) Weibull: $\mu = 1$ and $C^2 = 10$ (c) Weibull: $\mu = 1$ and $\rho = 0.7$

Fig. 8: Comparisons of the average AoI performance: preemptive, informative, AoI-based policies vs. others

We prove Propositions 2 and 3 using the strong induction. The detailed proofs are provided in our online technical report [20]. Propositions 2 and 3 imply that although SRPT_I and SJF_I do not explicitly follow an AoI-based design, they are essentially AoI-based policies. This provides an intuitive explanation for why size-based policies, such as variants of SRPT and SJF, have a good empirical AoI performance.

In Fig. 8, we present the simulation results for the average AoI performance of the preemptive, informative, AoI-based policies (ADE_PI) compared to several other policies. We observe that in various settings we consider, ADE_PI achieves the best AoI performance.

VII. Conclusion

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

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

References

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

[2] S. Wu, X. Ren, S. Dey, and L. Shi, "Optimal scheduling of multiple sensors with packet length constraint," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 14 430–14 435, 2017.

[3] M. Harchol-Balter, *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[4] A. M. Bedewy, Y. Sun, and N. B. Shroff, "Optimizing data freshness, throughput, and delay in multi-server information-update systems," in *2016 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2016, pp. 2569–2573.

[5] M. Costa, M. Codreanu, and A. Ephremides, "Age of information with packet management," in *2014 IEEE ISIT*. IEEE, 2014, pp. 1583–1587.

[6] N. Pappas, J. Gunnarsson, L. Kratz, M. Kountouris, and V. Angelakis, "Age of information of multiple sources with queue management," in *2015 IEEE ICC*. IEEE, 2015, pp. 5935–5940.

[7] M. E. Crovella, R. Frangioso, and M. Harchol-Balter, "Connection scheduling in web servers," Boston University Computer Science Department, Tech. Rep., 1999.

[8] L. Schrage, "A Proof of the Optimality of the Shortest Remaining Processing Time Discipline," *Operations Research*, vol. 16, no. 3, pp. 687–690, June 1968.

[9] D. R. Smith, "A new proof of the optimality of the shortest remaining processing time discipline," *Operations Research*, vol. 26, no. 1, pp. 197–199, 1978.

[10] M. Harchol-Balter, "Queueing disciplines," *Wiley Encyclopedia of Operations Research and Management Science*, 2010.

[11] A. Kosta, N. Pappas, V. Angelakis *et al.*, "Age of information: A new concept, metric, and tool," *Foundations and Trends® in Networking*, vol. 12, no. 3, pp. 162–259, 2017.

[12] Y. Sun, I. Kadota, R. Talak, and E. Modiano, "Age of information: A new metric for information freshness," *Synthesis Lectures on Communication Networks*, vol. 12, no. 2, pp. 1–224, 2019.

[13] M. Costa, M. Codreanu, and A. Ephremides, "On the age of information in status update systems with packet management," *IEEE Transactions on Information Theory*, vol. 62, no. 4, pp. 1897–1910, 2016.

[14] S. K. Kaul, R. D. Yates, and M. Gruteser, "Status updates through queues," in *2012 46th Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2012, pp. 1–6.

[15] C. Kam, S. Kompella, and A. Ephremides, "Effect of message transmission diversity on status age," in *2014 IEEE ISIT*. IEEE, 2014.

[16] E. Najm and E. Telatar, "Status updates in a multi-stream m/g/1/1 preemptive queue," in *Ieee Infocom 2018-Ieee Conference On Computer Communications Workshops (Infocom Wkshps)*. IEEE, 2018.

[17] Y. Inoue, H. Masuyama, T. Takine, and T. Tanaka, "A general formula for the stationary distribution of the age of information and its application to single-server queues," *arXiv preprint arXiv:1804.06139*, 2018.

[18] R. Talak and E. Modiano, "Age-delay tradeoffs in single server systems," *arXiv preprint arXiv:1901.04167*, 2019.

[19] R. Devassy, G. Durisi, G. C. Ferrante, O. Simeone, and E. Uysal-Biyikoglu, "Delay and peak-age violation probability in short-packet transmissions," in *2018 IEEE ISIT*. IEEE, 2018, pp. 2471–2475.

[20] Z. Liu, L. Huang, B. Li, and B. Ji, "Anti-Aging Scheduling in Single-Server Queues: A Systematic and Comparative Study," *arXiv e-prints*, p. arXiv:2003.04271, Mar. 2020.

[21] R. D. Yates and S. K. Kaul, "The age of information: Real-time status updating by multiple sources," *IEEE Transactions on Information Theory*, vol. 65, no. 3, pp. 1807–1827, 2018.

[22] C. Kam, S. Kompella, and A. Ephremides, "Age of information under random updates," in *2013 IEEE ISIT*. IEEE, 2013, pp. 66–70.

[23] M. Shaked and J. G. Shanthikumar, *Stochastic orders*. Springer Science & Business Media, 2007.