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. However, compared to the best delay-efficient policies (such as shortest remaining processing time (SRPT)), the AoI improvement is rather marginal in the settings with exogenous arrivals. Interestingly, we also prove sample-path equivalence between some size-based policies and AoI-based policies. This provides an intuitive explanation for why some size-based policies (such as SRPT) achieve a very good AoI performance.**

Index Terms: **Age of information, G/G/1 Queues, scheduling policies, update-size information.**

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 alerts, 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 [2]. 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., [2]).

Manuscript received April 11, 2020; revised December 21, 2020; approved for publication by Nele Noels, Division I Editor, February 4, 2021.

This work was supported in part by the NSF under Grants CCF-1657162, CNS-1651947, and CNS-1717108. A preliminary version of this work was presented at IEEE INFOCOM 2020 Age of Information Workshop [1].

Z. Liu and B. Ji are with the Department of Computer Science, Virginia Tech, Blacksburg, VA, email:{zhongdong, boji}@vt.edu.

L. Huang is with the College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou, China, email: lianghuang@zjut.edu.cn.

B. Li is with the Department of Electrical, Computer and Biomedical Engineering, University of Rhode Island, Kingston, Rhode Island, email: binli@uri.edu.

B. Ji is the corresponding author.

Digital Object Identifier: 10.23919/JCN.2021.000005

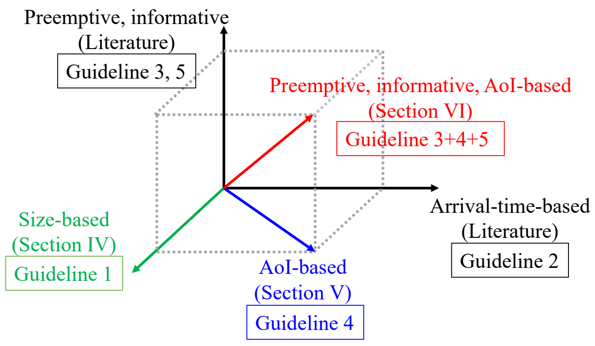


Fig. 1. Our position in the design space of AoI-efficient scheduling policies for a G/G/1 queue.

Although it is well-known that scheduling policies play an important role in reducing the delay in single-sever 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.

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 [3]. 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 [4]. 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

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

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

reduced AoI once delivered; see Section VI.A for a formal definition) yield a good AoI performance [5]–[7]. Intuitively, preemption prevents fresh updates from being blocked by a large and/or stale update in service; informative policies discard stale updates, which do not bring new information but may block fresh updates. To that end, we also consider AoI-based scheduling designs that both allow service preemption and prioritize informative updates.

In Fig. 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 1, 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 1.

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 1) for the design of AoI-efficient policies.

Second, leveraging both the update-size and arrival-time information, we introduce Guideline 4 and propose AoI-based scheduling policies (i.e., the blue arrow in Fig. 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. However, compared to the best delay-efficient policies (such as SRPT), the AoI improvement of the preemptive, informative, and AoI-based policies is rather marginal in the settings with exogenous arrivals. Moreover, when the AoI requirement is not stringent or the update-size information is not available, some simple delay-efficient policies (such as LCFS with preemption (LCFS_P)) are also good candidates for AoI-efficient policies.

The rest of this 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 [8], 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 [9], [10] 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 [11] 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 [12], [13] 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 [2], 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 [14], the AoI performance of the FCFS policy in the M/M/1/1 and M/M/1/2 queues is studied, where new arrivals are discarded if the buffer is full. In [15], the authors study the average AoI performance of a multi-source FCFS M/G/1 queue. They derive the exact expression and three approximations of the average AoI for a special case of an M/M/1 queue and a general case of an M/G/1 queue, respectively. The average AoI of the LCFS policy in the M/M/1 queue is also discussed in [14].

There has been some work that aims to reduce the AoI by making use of service preemption. In [16], the average AoI of LCFS in the M/M/1 queue with and without service preemption is analyzed. The work of [17] is quite similar to [16], but it considers the average AoI in the M/M/2 queue. In [18], 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 [5], 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 [6], [7] reveals that the AoI performance can be improved by prioritizing informative updates and discarding non-informative policies when making scheduling decisions. In [19], 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 [20], 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 [21], 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 AoI and the 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 ith 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 $U(t) \triangleq \max\left\{t_i : t_i' \leq t\right\}$ 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

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

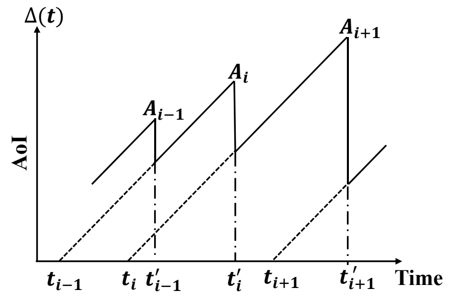


Fig. 2. An example of the AoI evolution under the FCFS policy.

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 [2]. We define the inter-arrival time between the ith update and $(i-1)$th update as $X_i \triangleq t_i - t_{i-1}$ and define the delay of the ith update as $T_i \triangleq t_i' - t_i$. Alternatively, the PAoI is also proposed as an information freshness metric [6], 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 ith 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 ith update and the previous update (i.e., X_i) and the delay of the ith update (i.e., T_i). Therefore, the PAoI of the ith update can also be expressed as $A_i = X_i + T_i$, and its expectation is $\mathbb{E}[A_i] = \mathbb{E}[X_i] + \mathbb{E}[T_i]$.

IV. SIZE-BASED POLICIES

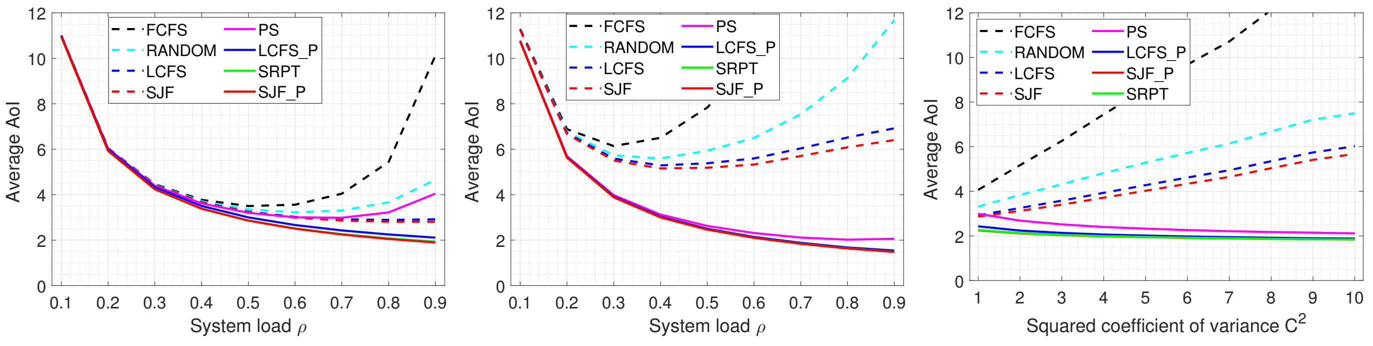
In this section, we investigate the AoI performance of several common scheduling policies, including size-based policies and non-size-based policies, via extensive simulations. Note that these common scheduling policies may serve the non-informative updates (which do not lead to a reduced AoI). This is because in some applications, such as news and social network, obsolete updates are still useful and need to be served [5]. In Section VI, we will discuss the case where obsolete updates are discarded.

Following [4], 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 [3]) 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.*

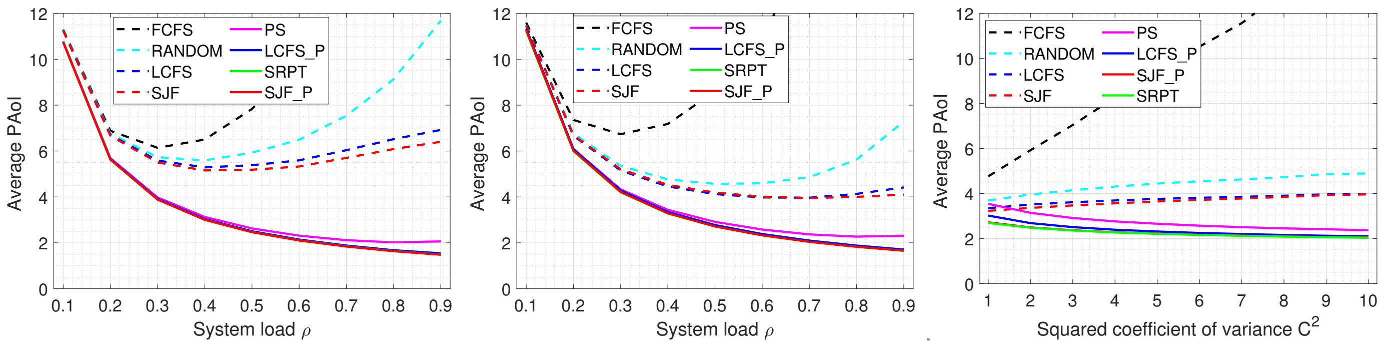
The first type consists of policies that are non-preemptive and blind to the update size:

- *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.



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

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

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

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

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

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

Previous work (see, e.g., [4, 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 [2]. 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) [2], [19]. Therefore, we attempt to investigate the AoI performance of size-based policies through extensive simulations.

In Figs. 3 and 4, we present the simulation results of the average AoI and PAoI performance under the scheduling policies we introduced above, respectively. There are three commonly used methods to conduct the simulation: Independent replications, batch means, and regeneration. Here, we use the independent replications for the following reasons: (i) The replication means are independent; (ii) it allows to start the individual replications in different initial states such that various different sample paths of the underlying stochastic process can be observed. Specifically, we conduct 50 simulation runs and take the average values. In each simulation run, we consider a total number of 10^5 updates to ensure that the steady state is reached. All the random numbers are generated using the default pseudorandom number generator (i.e., the Mersenne Twister) in the Python standard library. Here, we assume that a single source generates updates according to a Poisson process with rate λ, and the update size is independent and identically distributed (*i.i.d.*). In Fig. 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[1] 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

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

by the square of the mean [4]. 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 addition, the 95% confidence intervals of Figs. 3 and 4 are also provided in our online technical report [22], in which we observe that the margin of error is only a very small portion of the average (about 1%).

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

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

In Fig. 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, SJF_P, and SRPT, the average AoI decreases as the system load ρ increases. There are two reasons. First, when ρ increases, there will be more updates with small size arriving to the queue. Therefore, size-based policies 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 [5], [14], [23]. 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.*

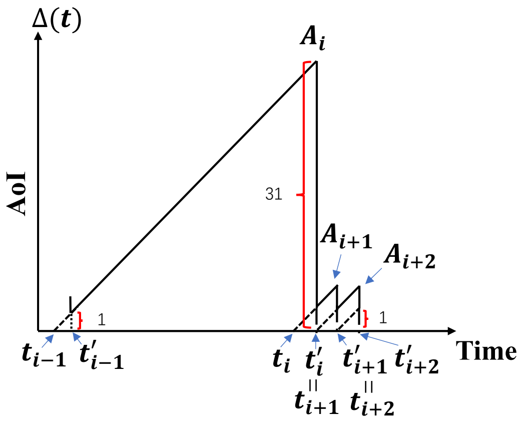


Fig. 5. An example of the AoI/PAoI evolution where the interarrival time has a large variability.

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.*

Note that above observations not only hold for the M/G/1 queue, but also can be made for the G/G/1 queue. More simulation results for the G/G/1 queue (i.e., Figs. 16–23) can be found in Appendix A and our technical report [22]. In addition, we make the following interesting observations regarding the average PAoI and AoI in a G/G/1 queue.

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

In Figs. 16(a) and 17(a), we can see that the average PAoI is much smaller than the average AoI for all the common scheduling policies we considered. This is due to the interarrival time has a large variability. We present an example in Fig. 5 to illustrate that this phenomenon comes from the large variability of the interarrival time. We consider three updates: The

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

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

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

In Fig. 4(c), we observe that while the average PAoI performance of FCFS is sensitive to the update-size variability, under several non-preemptive policies (such as RANDOM, LCFS, and SJF), the average PAoI performance is much less sensitive. An explanation for this observation is the following.

First, we explain why the average PAoI under FCFS is still sensitive to the update-size variability. Note that a key difference between FCFS and other non-preemptive policies is that under FCFS, every update leads to an AoI drop and thus corresponds to an AoI peak[2]. When a large update is in service, it will block all the following updates that are waiting in the queue, which results in a large delay for all such updates and thus a large PAoI corresponding to these updates. In contrast, under RANDOM, LCFS, and SJF, the impact of such a blocking issue is minimal for the updates that lead to an AoI drop.

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

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

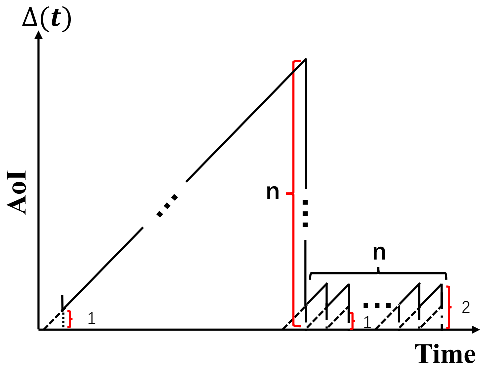


Fig. 6. An example of the AoI/PAoI evolution where the service time has a large variability.

To illustrate this fact, we provide an example in Fig. 6, where there is a large update of size $n-1$, immediately followed by n small updates of size 1. In this case, we can compute the average AoI as $\Delta = \left[1 \times (\frac{n^2}{2} - \frac{1^2}{2}) + n \times \left(\frac{2^2}{2} - \frac{1^2}{2}\right)\right] / ((n-1)+n) = \frac{n^2+3n-1}{4n-2} = O(n)$ and compute the average PAoI as $A = (n + 2 \times n)/(n+1) = 3n/(n+1) = O(3)$. This example shows that a larger update-size variability (i.e., a larger n in this example) results in a larger average AoI but only minimally affects the average PAoI. A similar explanation also applies to SJF and RANDOM.

V. AOI-BASED POLICIES

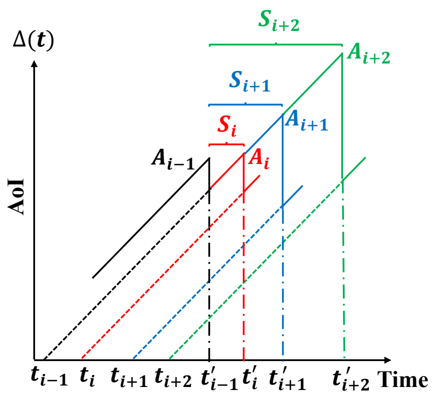
In Section IV, we have demonstrated that size-based policies achieve a better average AoI/PAoI performance than non-size-based policies. However, size-based policies do not utilize the arrival-time information, which also plays an important role in reducing the AoI. In this section, we propose three 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 drops as much as possible.

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

Although all of these AoI-based policies are quite intuitive, they behave very differently. In order to explain the differences of these AoI-based policies, we present an example in Fig. 7 to show how the AoI evolves under these policies. Suppose that when the $(i-1)$st update is being served, three new updates (i.e., the ith, $(i+1)$st, and $(i+2)$nd updates) arrive in sequence at times



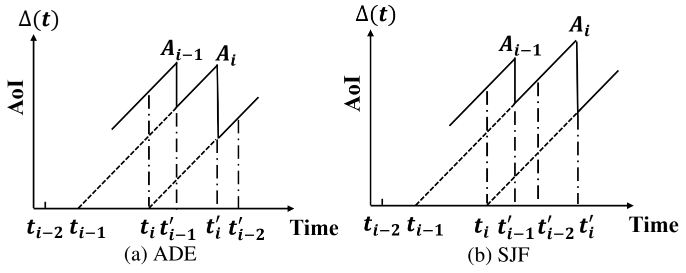
$\Delta(t)$
AoI
S_{i+2}
A_{i+2}
S_{i+1}
S_i
A_{i+1}
A_{i-1}
A_i
t_{i-1} t_i t_{i+1} t_{i+2} t'_{i-1} t'_i t'_{i+1} t'_{i+2} Time

Fig. 7. The AoI evolution under three AoI-based policies: ADE (red), ADS (blue), and ADM (green).

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 ith, $(i+1)$st, and $(i+2)$nd updates, respectively. This is because serving the ith update leads to the earliest AoI drop at time t'_i (following the red curve), serving the $(i+1)$st update leads to the AoI dropping to the smallest at time t'_{i+1} (following the blue curve), and serving the $(i+2)$nd update leads to the largest AoI drop at time t'_{i+2} (following the green curve). Clearly, ADE, ADS, and ADM aim to optimize AoI at a specific future time instant (i.e., the future delivery time of 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. 7). In addition, ADE and SJF may also look the same at first glance. Indeed, these two policies would make the same decision (i.e., choose the smallest update to serve) when the smallest update leads to an AoI drop. However, they make different decisions when the smallest update does not lead to an AoI drop. An example is provided in Fig. 8 to illustrate the key difference. In Fig. 8, after the $(i-1)$st update completes service at time t'_{i-1}, there are two updates waiting to be served: the $(n-2)$nd update and the ith update. Suppose that the update size and the arrival time of these two updates satisfy the following: $S_{i-2} < S_i$ and $t_{i-2} < t_{i-1} < t_i$. Clearly, ADE chooses to serve the ith update that leads to an earlier AoI drop (see Fig. 8(a)), while SJF chooses to serve the $(i-2)$nd update that has a smaller size (see Fig. 8(b)).

Next we conduct extensive simulations to investigate the AoI performance of these AoI-based policies. In Fig. 9, we present the simulation results of the average AoI performance of the AoI-based policies compared to a representative arrival-time-based policy (i.e., LCFS) and a representative size-based-policy (i.e., SJF). All the policies considered here are non-preemptive; the preemptive cases will be discussed in Section VI.

In Fig. 9(a), we observe that most AoI-based policies are slightly better than non-AoI-based policies, although their performances are very close. Among the AoI-based policies, ADE is the best, ADM is the worst, and ADS is in-between. This is not surprising that ADM is the worst: Although ADM has the largest AoI drop, this is at the cost that it may have to wait until the AoI 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. 9(b) and 9(c), similar observations can be made for update size following Weibull distributions.



$\Delta(t)$
AoI
A_{i-1} A_i
t_{i-2} t_{i-1} t_i t'_{i-1} t'_i t'_{i-2} Time
(a) ADE
$\Delta(t)$
AoI
A_{i-1}
A_i
t_{i-2} t_{i-1} t_i t'_{i-1} t'_{i-2} t'_i Time
(b) SJF

Fig. 8. An example of the AoI evolution under ADE and SJF.

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

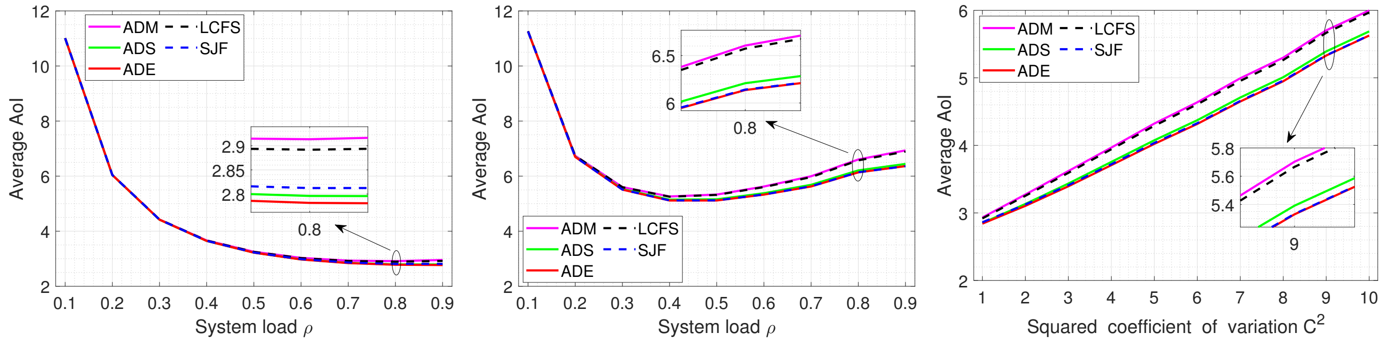
In Section IV, we have observed that preemptive policies have several advantages and perform better than non-preemptive policies. In this section, we first demonstrate that policies that prioritize informative updates (i.e., those that can lead to AoI drops once delivered) perform better than non-informative 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 [24]. *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 a G/M/1 queue, the informative version of LCFS is stochastically better than the original LCFS policy.

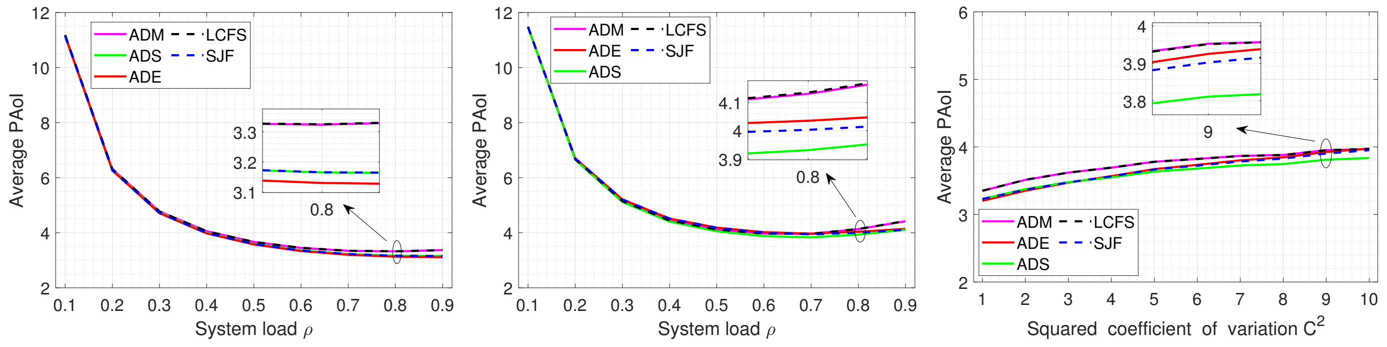
We use π_I to denote the informative version[3] of policy π. All the scheduling policies we consider have their informative versions. In some cases, the informative version is simply the same as the original policy (e.g., FCFS and LCFS_P).

[3]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.



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

Fig. 9. 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. 10. Comparisons of the average PAoI performance: AoI-based policies vs. non-AoI-based policies.

In Fig. 11, we show the simulation results of the average AoI performance of several informative policies compared to their non-informative counterparts. In order to evaluate the benefit of informative policies, we plot the informative AoI gain, which is the ratio of the difference between the average AoI of the non-informative version and the informative version to the average AoI of the non-informative version. Hence, a larger informative gain means a larger benefit of the informative version. One important observation from Fig. 11 is as follows.

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

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

Observation 8 leads to the following guideline:

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

Based on Observation 8, we conjecture that an informative policy is 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.

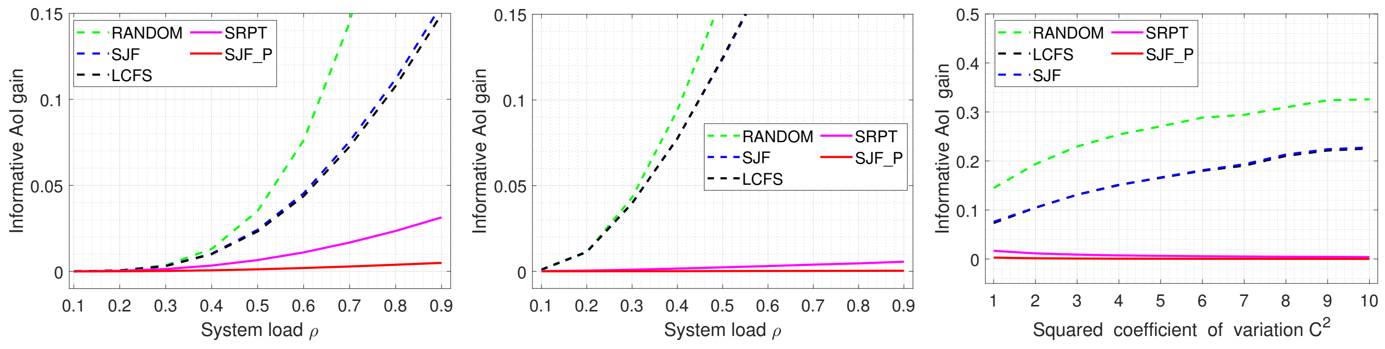
Definition 2. *Stochastic ordering of stochastic processes [25, Ch.6.B.7]: Let $\{X(t), t \in [0,\infty)\}$ and $\{Y(t), t \in [0,\infty)\}$ be two stochastic processes. Then, $\{X(t), t \in [0,\infty)\}$ is said to be stochastically less than $\{Y(t), t \in [0,\infty)\}$, denoted by $\{X(t), t \in [0,\infty)\} \leq_{\text{st}} \{Y(t), t \in [0,\infty)\}$, 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[4] $S^U \subseteq \mathbb{R}^n$:*

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

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

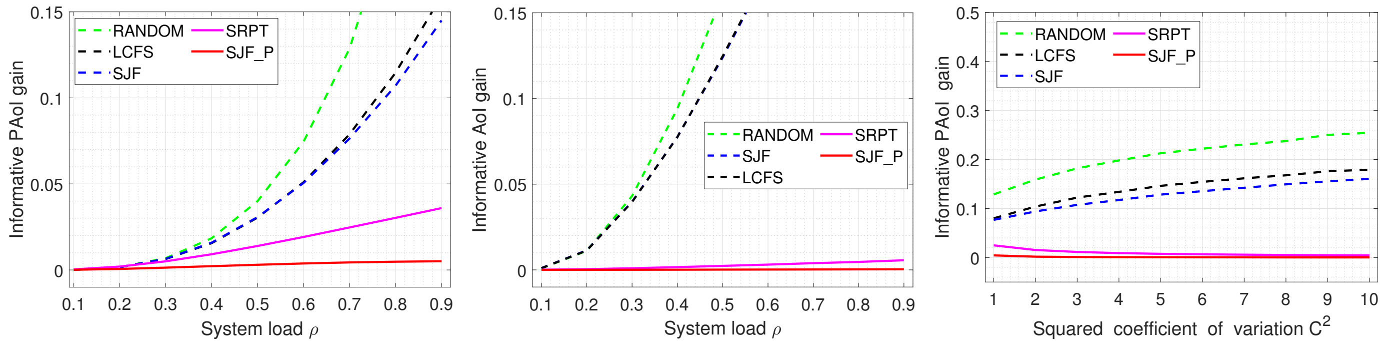
Roughly speaking, (2) implies that $\vec{X}$ is less likely than $\vec{Y}$ to take on large values, where "large" means any value in an upper set S^U. We also use $\Delta_\pi(t)$ to denote the AoI process under policy π. Furthermore, we define a set of parameters $\mathcal{I} = \{n, (t_i)_{i=1}^n\}$,

[4]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 $\vec{x} = (x_1, \cdots, x_n)$ and $\vec{y} = (y_1, \cdots, y_n)$ are two vectors in $\mathbb{R}^\kappa$ and $\vec{y} \geq \vec{x}$ if $y_i \geq x_i$ for all $i = 1, 2, \cdots, n$.



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

Fig. 11. Comparisons of the average AoI performance: Informative policies vs. non-informative policies.



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

Fig. 12. Comparisons of the average PAoI performance: Informative policies vs. non-informative policies.

where n is the number of updates and t_i is the generation time of update i. Having these definitions and notations, we are now ready to state Proposition 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.,*

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

Proof. Recall that we use t_i and t_i' to denote the arrival time and the delivery time of the ith update, respectively. In addition, we use s_i to denote the service start time of the ith update.

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

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

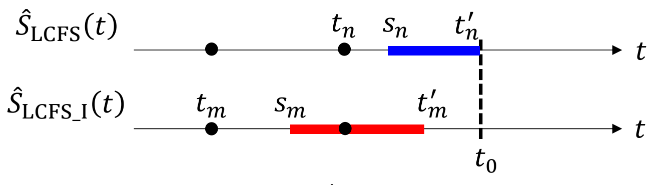
Next, we prove (4) by contradiction through a coupling argument. Suppose that stochastic processes $\hat{S}_{\text{LCFS\_I}}(t)$ and $\hat{S}_{\text{LCFS}}(t)$ have the same stochastic laws as $S_{\text{LCFS\_I}}(t)$ and $S_{\text{LCFS}}(t)$, respectively. We couple $\hat{S}_{\text{LCFS\_I}}(t)$ and $\hat{S}_{\text{LCFS}}(t)$ 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 [25], (4) holds if the following holds:

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

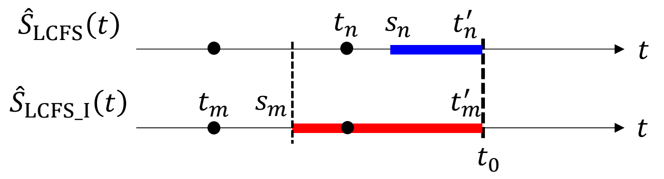
In the following, we want to show that $\hat{S}_{\text{LCFS\_I}}(t) \geq \hat{S}_{\text{LCFS}}(t)$ holds conditionally on an arbitrary sample path $\mathcal{I}$, which trivially implies (5). We prove it by contradiction. For the sake of contradiction, suppose that $\hat{S}_{\text{LCFS\_I}}(t) < \hat{S}_{\text{LCFS}}(t)$ does happen and that it happens for the first time at time t_0 (see Fig. 13 for illustration). Let m and n be the index of the served updates with the largest arrival time by t_0 in $\hat{S}_{\text{LCFS\_I}}(t)$ and $\hat{S}_{\text{LCFS}}(t)$, respectively. Then, we have $U_{\text{LCFS\_I}}(t_0) = t_m$ and $U_{\text{LCFS}}(t_0) = t_n$. Note that we also have $t_m < t_n$ due to $\hat{S}_{\text{LCFS\_I}}(t_0) < \hat{S}_{\text{LCFS}}(t_0)$ (i.e., $U_{\text{LCFS\_I}}(t_0) < U_{\text{LCFS}}(t_0)$). Since t_0 is the first time when $\hat{S}_{\text{LCFS\_I}}(t) < \hat{S}_{\text{LCFS}}(t)$ happens, a crucial observation is that t_0 must be immediately after an update is delivered in $\hat{S}_{\text{LCFS}}(t)$. Hence, we have $t_0 = (t_n')^+$, where $(t_n')^+$ denotes the time immediately after t_n'.

Due to the coupling between $\hat{S}_{\text{LCFS}}(t)$ and $\hat{S}_{\text{LCFS\_I}}(t)$, there are two cases in $\hat{S}_{\text{LCFS\_I}}(t)$: 1) The server is being idle at t_n'; 2) an update is delivered at t_n' too. We discuss these two cases separately and show that there is a contradiction in both cases.

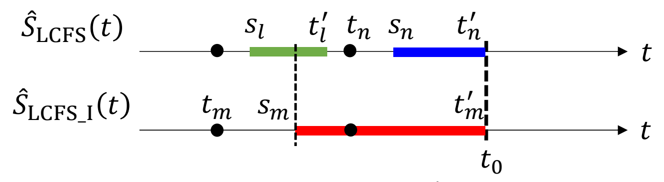
Case 1): The server in $\hat{S}_{\text{LCFS\_I}}(t)$ is being idle at t_n' (see Fig. 13(a)). Then, the most recently delivered update in $\hat{S}_{\text{LCFS\_I}}(t)$ (i.e., the mth update) must be delivered before t_n'. Hence, we have $t_m' < t_n'$ and that the server in $\hat{S}_{\text{LCFS\_I}}(t)$ stays in the idle state during $(t_m', t_n']$. Then, the server in $\hat{S}_{\text{LCFS\_I}}(t)$ could have started serving a newer update that arrives later than the



(a) Case 1): The server in $\hat{S}_{\text{LCFS\_I}}(t)$ is being idle at t'_n



(b) Case 2a): The mth update is delivered at t'_n in $\hat{S}_{\text{LCFS\_I}}(t)$, and the server in $\hat{S}_{\text{LCFS}}(t)$ is idle at time s_m



(c) Case 2b): The mth update is delivered at t'_n in $\hat{S}_{\text{LCFS\_I}}(t)$, and the server in $\hat{S}_{\text{LCFS}}(t)$ is busy at time s_m

Fig. 13. Part of sample path of $\hat{S}_{\text{LCFS\_I}}(t)$ and $\hat{S}_{\text{LCFS}}(t)$ in different cases.

mth update immediately after t'_m. (Such a newer update must exist as the nth update is a valid candidate due to $t_m < t_n$.) This results in a contradiction with the server being idle during $(t'_m, t'_n]$.

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

Case 2a): The server in $\hat{S}_{\text{LCFS}}(t)$ is idle at time s_m (see Fig. 13(b)). In this case, the mth update must have already been delivered by time s_m in $\hat{S}_{\text{LCFS}}(t)$. Otherwise, the server in $\hat{S}_{\text{LCFS}}(t)$ would have started serving the mth update (or a newer update) at or before s_m. This implies that $\hat{S}_{\text{LCFS\_I}}(t) < \hat{S}_{\text{LCFS}}(t)$ happens before s_m, which results in a contradiction with that t_0 is the first time at which $\hat{S}_{\text{LCFS\_I}}(t) < \hat{S}_{\text{LCFS}}(t)$ happens.

Case 2b): The server in $\hat{S}_{\text{LCFS}}(t)$ is busy at time s_m (see Fig. 13(c)). Assume that the lth update is being served at s_m in $\hat{S}_{\text{LCFS}}(t)$. In this case, the lth update must be delivered by time s_n in $\hat{S}_{\text{LCFS}}(t)$. This is because the nth update starts service at s_n in $\hat{S}_{\text{LCFS}}(t)$. Then, the mth update must also be delivered by time s_n in $\hat{S}_{\text{LCFS\_I}}(t)$, due to the coupling between $\hat{S}_{\text{LCFS}}(t)$ and $\hat{S}_{\text{LCFS\_I}}(t)$. This results in a contradiction that the mth update is delivered at t'_n.

Combining all the cases, we show that $\hat{S}_{\text{LCFS\_I}}(t) \geq \hat{S}_{\text{LCFS}}(t)$ holds conditionally on an arbitrary sample path $\mathcal{I}$. This trivially implies (5), which further implies (4) by Theorem 6.B.30 in [25]. This completes the proof. □

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.*

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

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

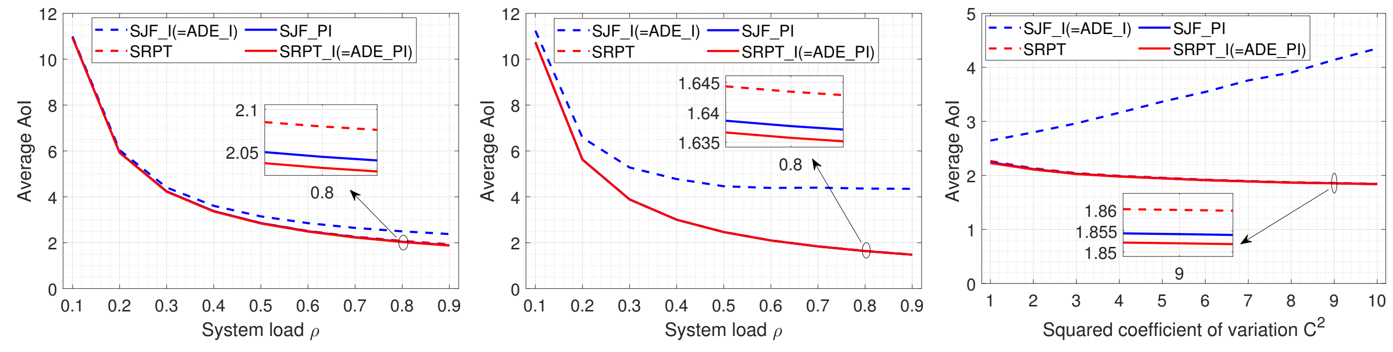
Claim: ADE_PI and SRPT_I always serve the same update at the same time, i.e., $(d_{\text{ADE\_PI}}(n), t_{\text{ADE\_PI}}(n)) = (d_{\text{SRPT\_I}}(n), t_{\text{SRPT\_I}}(n))$ for all n.

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

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

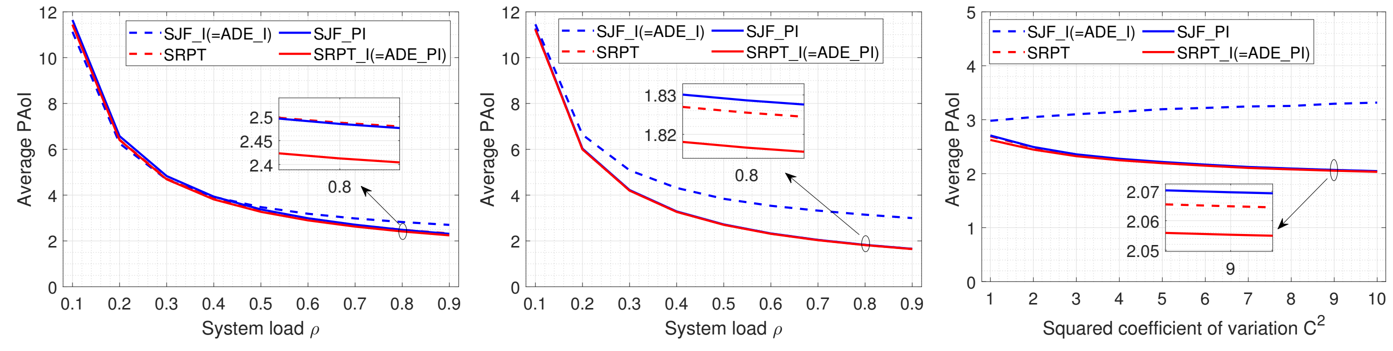
Case 1): The $(k+1)$st update preempts the kth update. During the service of the kth update, the $(k+1)$st update arrives. Under ADE_PI, in order to make AoI drop as early as possible, the server compares the remaining service time of the kth update with the original service time of the $(k+1)$st update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

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



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

Fig. 14. Comparisons of the average AoI performance: Preemptive, informative, AoI-based policies vs. others.



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

Fig. 15. Comparisons of the average PAoI performance: Preemptive, informative, AoI-based policies vs. others.

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

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

Proof. Similar to the proof of Proposition 2, we use strong induction to show that under the same sample path, ADE_I and SJF_I always choose the same update to serve at the same time. Here, we also only consider the informative updates.

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

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

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

Induction step: Suppose that for $n = k$ $(k \geq 1)$, we have $(d_{\text{ADE\_I}}(m), t_{\text{ADE\_I}}(m)) = (d_{\text{SJF\_I}}(m), t_{\text{SJF\_I}}(m))$ for the mth update for $1 \leq m \leq k$. We want to show that $(d_{\text{ADE\_I}}(n), t_{\text{ADE\_I}}(n)) = (d_{\text{SJF\_I}}(n), t_{\text{SJF\_I}}(n))$ still holds for $n = k+1$. Note that there are two cases for the $(k+1)$st update: 1) The $(k+1)$st update starts service from the idle state; 2) the $(k+1)$st update starts service immediately after the kth update is delivered. We discuss these two cases separately and show that $(d_{\text{ADE\_I}}(k+1), t_{\text{ADE\_I}}(k+1)) = (d_{\text{SJF\_I}}(k+1), t_{\text{SJF\_I}}(k+1))$ holds in both cases.

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

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

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

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

Our findings also raise several interesting questions that are worth investigating as future work. One important direction is to pursue more theoretical results beyond the simulation results we provided in this 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.

APPENDIX A
ADDITIONAL SIMULATION RESULTS FOR THE G/G/1 QUEUE

We present additional simulation results for the G/G/1 queue in Figs. 16–23. For all these simulations, we assume that the interarrival time follows a Weibull distribution with $C^2 = 10$. In subfigure (a), we assume that the update size follows an Exponential distribution with mean $1/\mu = 1$; in subfigures (b) and (c), we assume that the update size follows a Weibull distribution with mean $1/\mu = 1$. Note that in subfigures (a) and (b), we change the value of the system load ρ; in subfigure (c), we change the value of C^2 for the update size while fixing the system load at $\rho = 0.7$. Observations 1–8 can also be made for the setting of G/G/1 queue.

REFERENCES

[1] Z. Liu, L. Huang, B. Li, and B. Ji, "Anti-aging scheduling in single-server queues: A systematic and comparative study," in *Proc. INFOCOM WKSHPS*, 2020.

[2] S. Kaul, R. Yates, and M. Gruteser, "Real-time status: How often should one update?" in *Proc. IEEE INFOCOM*, 2012.

[3] 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, July 2017.

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

[5] A. M. Bedewy, Y. Sun, and N. B. Shroff, "Optimizing data freshness, throughput, and delay in multi-server information-update systems," in *Proc. IEEE ISIT*, 2016.

[6] M. Costa, M. Codreanu, and A. Ephremides, "Age of information with packet management," in *Proc. IEEE ISIT*, 2014.

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

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

[9] L. Schrage, "A proof of the optimality of the shortest remaining processing time discipline," *Operations Research*, vol. 16, no. 3, pp. 687–690, 1968.

[10] 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.

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

[12] A. Kosta, N. Pappas, and V. Angelakis, *Age of Information: A New Concept, Metric, and Tool*, 2017.

[13] Y. Sun, I. Kadota, R. Talak, and E. Modiano, *Age of Information: A New Metric for Information Freshness*, 2019.

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

[15] M. Moltafet, M. Leinonen, and M. Codreanu, "On the age of information in multi-source queueing models," *IEEE Trans. Commun.*, vol. 68, no. 8, pp. 5003–5017, May 2020.

[16] S. K. Kaul, R. D. Yates, and M. Gruteser, "Status updates through queues," in *Proc. CISS*, 2012.

[17] C. Kam, S. Kompella, and A. Ephremides, "Effect of message transmission diversity on status age," in *Proc. IEEE ISIT*, 2014, pp. 2411–2415.

[18] E. Najm and E. Telatar, "Status updates in a multi-stream m/g/1/1 preemptive queue," in *IEEE INFOCOM WKSHPS*, 2018.

[19] 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.

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

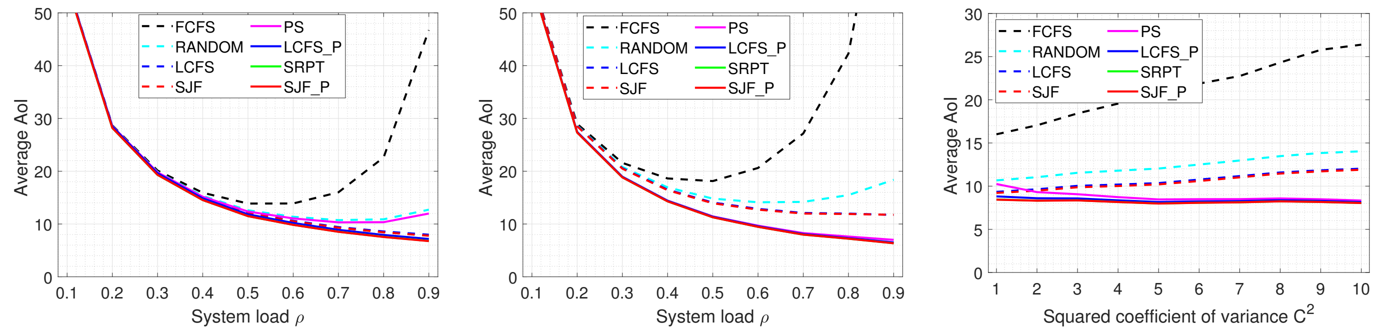
[21] R. Devassy, G. Durisi, G. C. Ferrante, O. Simeone, and E. Uysal-Biyikoglu, "Delay and peak-age violation probability in short-packet transmissions," in *Proc. IEEE ISIT*, 2018.

[22] Z. Liu, L. Huang, B. Li, and B. Ji, "Anti-aging scheduling in single-server queues: A systematic and comparative study," *arXiv e-prints*, p. arXiv:2003.04271, Oct. 2020.

[23] R. D. Yates and S. K. Kaul, "The age of information: Real-time status updating by multiple sources," *IEEE Trans. Inf. Theory*, vol. 65, no. 3, pp. 1807–1827, Mar. 2019.

[24] C. Kam, S. Kompella, and A. Ephremides, "Age of information under random updates," in *Proc. IEEE ISIT*, 2013.

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

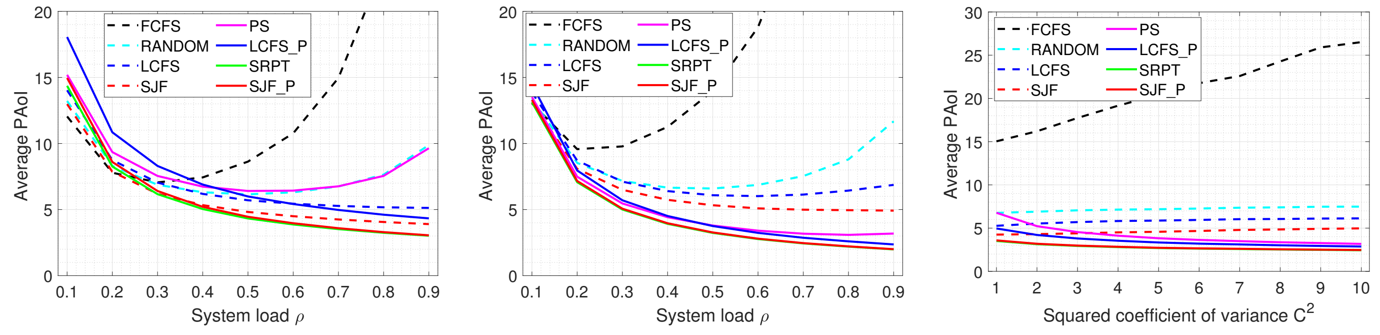


(a) Interarrival time: Weibull (C^2= 10); update size: Exponential (μ = 1)

(b) Interarrival time: Weibull (C^2= 10); update size: Weibull (μ = 1 and C^2= 10)

(c) Interarrival time: Weibull (C^2= 10); update size: Weibull (μ = 1 and ρ = 0.7)

Fig. 16. Comparisons of the average AoI performances of several common scheduling policies under different distributions.

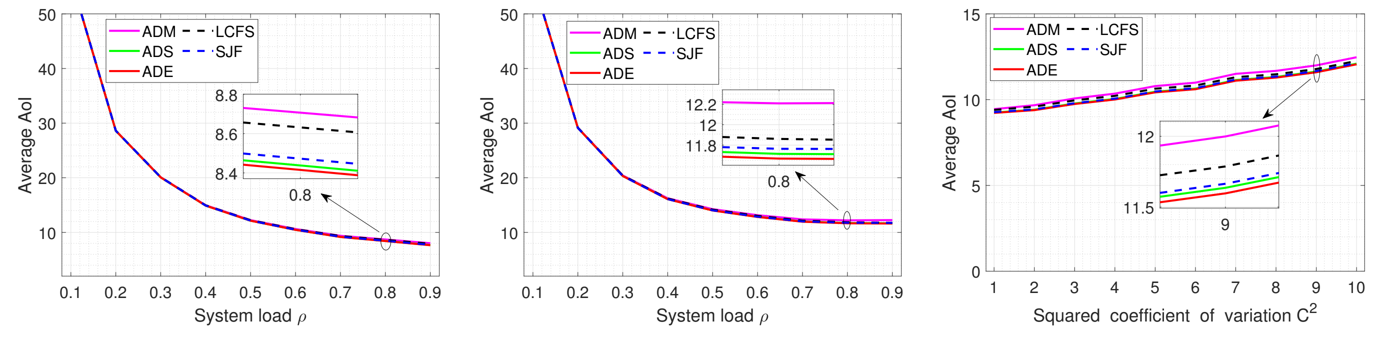


(a) Interarrival time: Weibull (C^2= 10); update size: Exponential (μ = 1)

(b) Interarrival time: Weibull (C^2= 10); update size: Weibull (μ = 1 and C^2= 10)

(c) Interarrival time: Weibull (C^2= 10); update size: Weibull (μ = 1 and ρ = 0.7)

Fig. 17. Comparisons of the average PAoI performances of several common scheduling policies under different distributions.

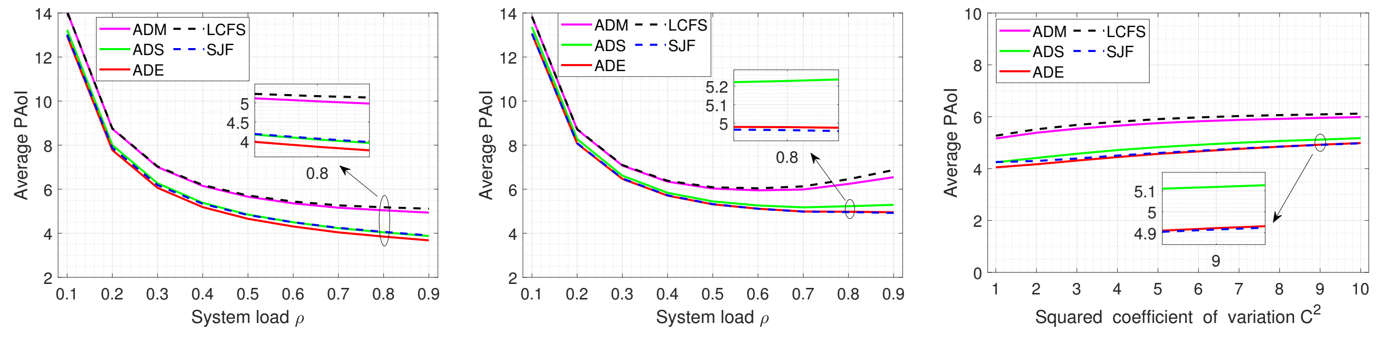


(a) Interarrival time: Weibull (C^2= 10); update size: Exponential (μ = 1)

(b) Interarrival time: Weibull (C^2= 10); update size: Weibull (μ = 1 and C^2= 10)

(c) Interarrival time: Weibull (C^2= 10); update size: Weibull (μ = 1 and ρ = 0.7)

Fig. 18. Comparisons of the average AoI performance under different distributions: AoI-based policies vs. non-AoI-based policies.

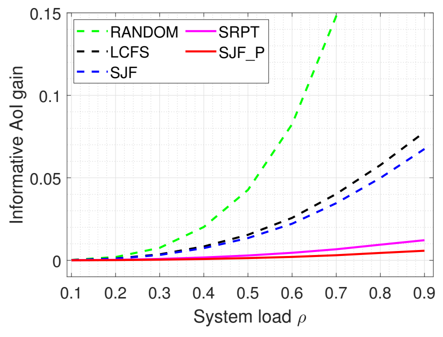


(a) Interarrival time: Weibull (C^2= 10); update size: Exponential (μ = 1)

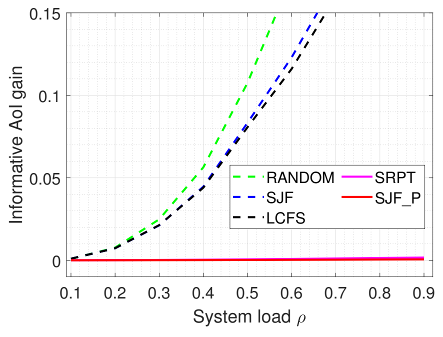
(b) Interarrival time: Weibull (C^2= 10); update size: Weibull (μ = 1 and C^2= 10)

(c) Interarrival time: Weibull (C^2= 10); update size: Weibull (μ = 1 and ρ = 0.7)

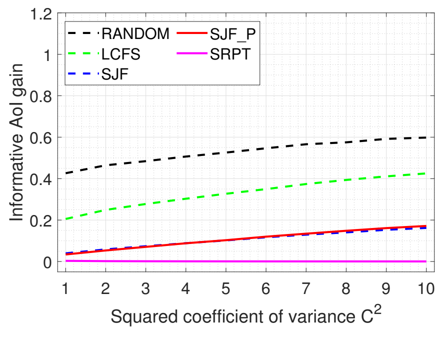
Fig. 19. Comparisons of the average PAoI performance under different distributions: AoI-based policies vs. non-AoI-based policies.



(a) Interarrival time: Weibull (C^2= 10); update size: Exponential (μ = 1)

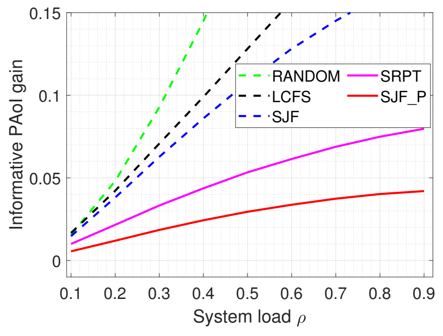


(b) Interarrival time: Weibull (C^2= 10); update size: Weibull (μ = 1 and C^2= 10)

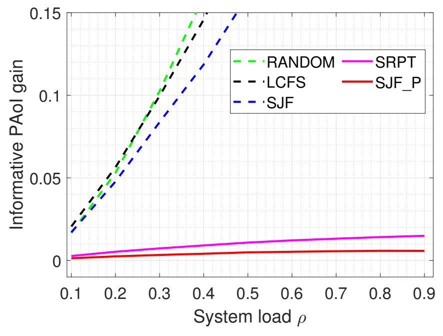


(c) Interarrival time: Weibull (C^2= 10); update size: Weibull (μ = 1 and ρ = 0.7)

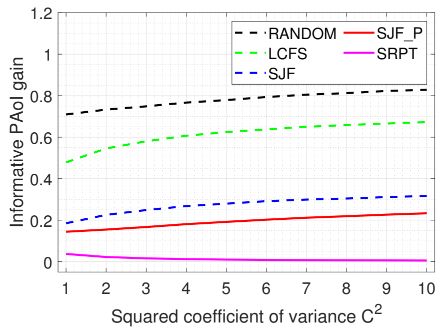
Fig. 20. Comparisons of the avg. AoI performance under different distributions: Informative policies vs. non-informative policies.



(a) Interarrival time: Weibull (C^2= 10); update size: Exponential (μ = 1)

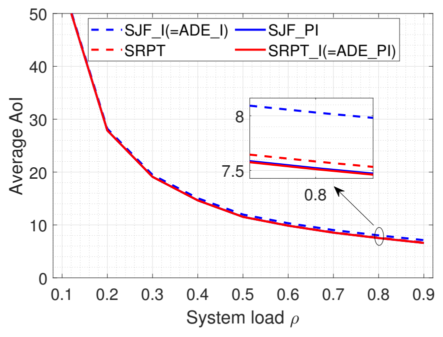


(b) Interarrival time: Weibull (C^2= 10); update size: Weibull (μ = 1 and C^2= 10)

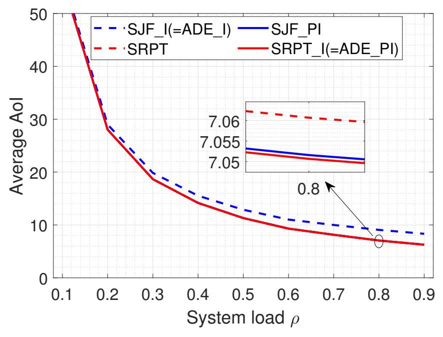


(c) Interarrival time: Weibull (C^2= 10); update size: Weibull (μ = 1 and ρ = 0.7)

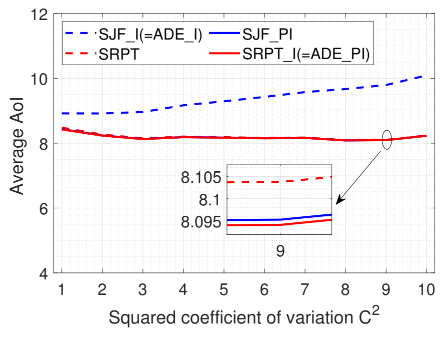
Fig. 21. Comparisons of the avg. PAoI performance under different distributions: Informative policies vs. non-informative policies.



(a) Interarrival time: Weibull (C^2= 10); update size: Exponential (μ = 1)

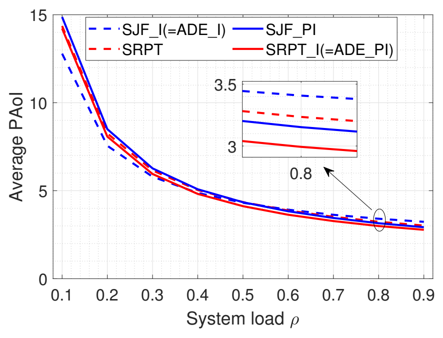


(b) Interarrival time: Weibull (C^2= 10); update size: Weibull (μ = 1 and C^2= 10)

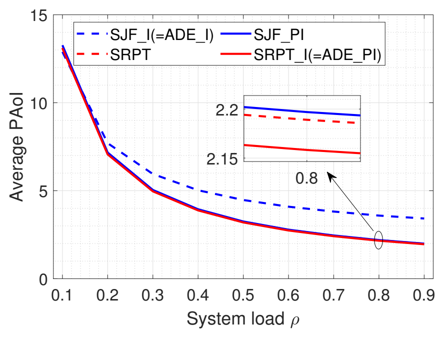


(c) Interarrival time: Weibull (C^2= 10); update size: Weibull (μ = 1 and ρ = 0.7)

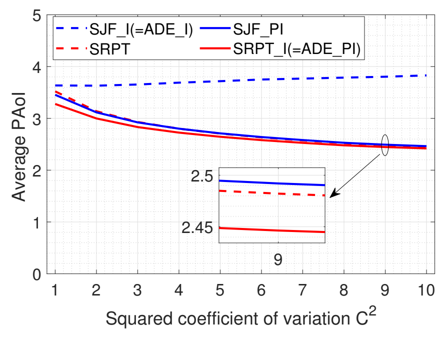
Fig. 22. Comparisons of the avg. AoI performance under different distributions: Preemptive, informative, AoI-based policies vs. others.



(a) Interarrival time: Weibull (C^2= 10); update size: Exponential (μ = 1)



(b) Interarrival time: Weibull (C^2= 10); update size: Weibull (μ = 1 and C^2= 10)

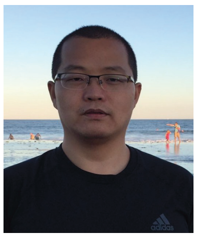


(c) Interarrival time: Weibull (C^2= 10); update size: Weibull (μ = 1 and ρ = 0.7)

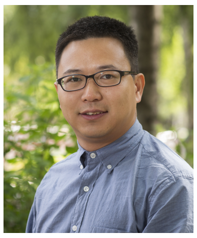
Fig. 23. Comparisons of the avg. PAoI performance under different distributions: Preemptive, informative, AoI-based policies vs. others.

Zhongdong Liu is a Ph.D. student in the Department of Computer Science at Virginia Tech. He received his B.S. degree in Mathematics and Applied Mathematics with honor from Northeast Forestry University in 2016. His research interests are in the modeling, analysis, control, and optimization of complex network systems.

Liang Huang received the BEng degree in communications engineering from Zhejiang University, Hangzhou, China, in 2009, and the Ph.D. degree in information engineering from The Chinese University of Hong Kong in 2013. He is currently an associate professor with the College of Computer Science and Engineering, Zhejiang University of Technology, China. His research interests include in the areas of queueing and scheduling in communication systems and networks.



Bin Li received his B.S. degree in Electronic and Information Engineering in 2005, M.S. degree in Communication and Information Engineering in 2008, both from Xiamen University, China, and Ph.D. degree in Electrical and Computer Engineering from The Ohio State University in 2014. Between 2014 and 2016, he worked as a Postdoctoral Researcher in the Coordinated Science Laboratory at the University of Illinois at Urbana-Champaign. He is currently an Assistant Professor in the Department of Electrical, Computer, and Biomedical Engineering at the University of Rhode Island. His research focuses on the intersection of networking, machine learning, and system developments, and their applications in networking for virtual/augmented reality, mobile edge computing, mobile crowd-learning, and Internet-of-things. He is a senior member of the IEEE and a member of the ACM. He received both the National Science Foundation (NSF) CAREER Award and Google Faculty Research Award in 2020, and ACM MobiHoc 2018 Best Poster Award.



Bo Ji received his B.E. and M.E. degrees in Information Science and Electronic Engineering from Zhejiang University, Hangzhou, China, in 2004 and 2006, respectively, and his Ph.D. degree in Electrical and Computer Engineering from The Ohio State University, Columbus, OH, USA, in 2012. He is an Associate Professor in the Department of Computer Science at Virginia Tech, Blacksburg, VA, USA. Prior to joining Virginia Tech, he was an Associate/Assistant Professor in the Department of Computer and Information Sciences at Temple University from July 2014 to July 2020. He was also a Senior Member of the Technical Staff with AT&T Labs, San Ramon, CA, from January 2013 to June 2014. His research interests are in the modeling, analysis, control, and optimization of computer and network systems, such as wired and wireless networks, large-scale IoT systems, high performance computing systems and data centers, and cyber-physical systems. He currently serves on the editorial boards of the IEEE/ACM Transactions on Networking, IEEE Transactions on Network Science and Engineering, IEEE Internet of Things Journal, and IEEE Open Journal of the Communications Society. He is a senior member of the IEEE and a member of the ACM. He is a National Science Foundation (NSF) CAREER awardee (2017) and an NSF CISE Research Initiation Initiative (CRII) awardee (2017). He is also a recipient of the IEEE INFOCOM 2019 Best Paper Award.