Age of Information-Aware Scheduling for Timely and Scalable Internet of Things Applications

Lorenzo Corneo, Christian Rohner and Per Gunningberg
Department of Information Technology, Uppsala University, Sweden
Email: lorenzo.corneo@it.uu.se, christian.rohner@it.uu.se, per.gunningberg@it.uu.se

***Abstract*—We consider large scale Internet of Things applications requesting data from physical devices. We study the problem of timely dissemination of sensor data towards applications with freshness requirements by means of a cache. We aim to minimize direct access to the possibly battery powered physical devices, yet improving Age of Information as a data freshness metric.**

We propose an Age of Information-aware scheduling policy for the physical device to push sensor updates to the caches located in cloud data centers. Such policy groups application requests based on freshness thresholds, thereby reduces the number of requests and threshold misses, and accounts for delay variation. The policy is incrementally introduced as we study its behavior over ideal and more realistic communication links with delay variation. We numerically evaluate the proposed policy against a simple yet widely used periodic schedule. We show that our informed schedule outperforms the periodic schedule even under high delay variations.

I. Introduction

Internet of Things (IoT) enabled applications in the domains of health care, environmental and building monitoring must rely on timely and fresh information gathered from a wide plethora of sensing devices. Age of Information (AoI) [1], [2], the age of the data since it was generated, is a metric indicator of data freshness. AoI is formally defined as $\Delta(t) = t - U(t)$, where t is the current time and $U(t)$ the time stamp of the most recent information update.

Since its introduction in [2], the AoI concept has been studied with different approaches and in different contexts. AoI minimization problems have been broadly studied in the field of queuing theory in [2]–[8] and in the context of energy harvesting systems, where energy is intermittent due to uncertain environmental conditions [9]–[14]. In wireless communication links, AoI-aware scheduling policies have been proposed in [15]–[19]. Furthermore, AoI has been used for applications in the domain of cloud gaming [20] and the dissemination of content updates in mobile social networks [21].

Cloud computing [22] goes hand in hand with IoT. Cloud servers provide seemingly unlimited storage and computational power, is geographically distributed and accessible all around the globe. Cloud computing is particularly attractive for off-loading storage and computation from battery and battery-free operated sensing devices. Although there are many scalability advantages, there is a significant communication latency between the actual sensing event and the IoT applications in the cloud. To overcome this latency limitation, edge and fog computing [23] have been proposed to move cloud functionality closer to the physical devices, typically at the edge of the network [24]. Time and safety critical functionality can then be performed with lower latency.

Nonetheless, the intrinsic nature of IoT applications faces scalability challenges itself. In fact, the sensing device is a bottleneck as it is the main entity that generates information for the IoT applications. In addition, sensing devices may be energy constrained and should not have to reply to each and every application request. Cloud and edge infrastructures linking applications with the sensing devices are therefore exposed to a trade-off to either provide fresh information from the devices, by extensively accessing them, or provide applications with cached information that are aged.

In this paper, we present an AoI-aware scheduling policy aimed to solve the problem of 1-to-many information dissemination from sensors to several remote applications. In particular, we focus on reducing sensor energy consumption yet satisfying applications timing requirements in the presence of varying communication delays. We consider a system architecture composed by cloud servers, edge servers and physical systems of sensing devices. We use the storage and computational power provided by the cloud as a mean for content distribution of sensor updates towards several remote applications. Then, the edge aggregates applications subscriptions to sensor updates coming from geo-distributed cloud instances. As a result, the edge server generates "smart" schedules to be downloaded to the sensor devices for duty-cycling purposes. Our contributions for this work are:

- A novel AoI-aware scheduling policy for sensor networks which reduces energy consumption while still ensuring fresh enough information towards several remote applications.
- A method that allows remote applications to meet their freshness requirements despite the effect of delay variations, especially between edge server and cloud.

The remainder of this paper is organized as follows. Sec. II and III describe the system and performance metrics. Sec. IV and V incrementally introduce scheduling policies in ideal and delay constrained conditions, respectively. The performance of the AoI-aware schedule is evaluated and discussed in Sec. VI.

II. System

We consider the system illustrated in Fig. 1 that involves sensing devices, an edge server and one or more cloud data centers hosting different applications.

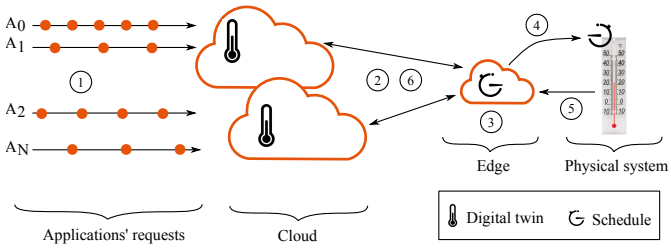


Fig. 1. System architecture.

① Generic **applications** with requirements on data freshness subscribe to the closest (in terms of latency) cloud instance. They specify the interest in particular sensors, an execution schedule and an upper-bound on freshness of the information. We consider applications that run periodically but can tolerate some variance in the scheduled execution times. Examples of such applications are monitoring and control systems.

② The **cloud** instances export sensor readings through an artifact commonly called "**digital twin**" sensor [25]. The digital twin allows the applications to access cached copies of sensor readings as if they were accessed directly at physical sensing devices, with some aging. The use of such artifact is motivated by latency. In fact, as the communication latency between the applications and the physical sensors may be significant, a *pull* update strategy may not be the best option for applications with strict timing requirements. As a result, we propose to update digital twins with a *push* policy from the physical sensing devices and edge device. Digital twins can be replicated at multiple cloud instances in order to minimize delay for data dissemination towards the applications. Furthermore, the subscriptions and the requirements on digital twin sensors are forwarded to an edge device on behalf of the applications.

③ The **edge** device is located close to the sensor network. As first responsibility, it receives from cloud instances all the subscriptions and requirements of the applications. Then, it aggregates all the requirements and combines them together creating a "smart" schedule. The aim of this schedule is to instruct the sensors on *when* to push updates toward the edge server. Such schedule is able to minimize the number of sensor update transmissions yet providing the required level of information freshness. Furthermore, it accounts for communication delay towards the cloud infrastructures. ④ After being computed, the schedule is pushed towards the physical sensor.

The **physical sensing devices** consist of a sensor network that provides sensor readings. The sensors in the network are duty-cycled in order to save energy and prolong battery lifetime. ⑤ The duty-cycling is dictated by the schedule received by the paired edge device. ⑥ Once the edge server receives the aforementioned sensor updates, it will push them towards the respective digital twins in the cloud. Without loss of generality, we consider sensor updates from one particular sensor to a possibly large number of applications.

III. Model

A. Notation and Parameters

Let $A = (a_1, a_2, ..., a_N)$ be the set of the N applications with timing requirements in the system. These applications are periodically executed and we indicate these periods with the set $T = (T_1, T_2, ..., T_N)$, where the subscript refers to applications in set A.

The applications in set A request sensors information at a cloud that provides digital twin services. The set of $M \in \mathbb{N}_+$ cloud instances is defined as $C = (c_1, c_1, ..., c_M)$. We indicate the latency (one way delay) between application a and sensing devices as $d_{a,s}$. To be noticed, every application specifies upper-bounds on AoI, τ, and may accept a delayed response by ϵ time units and wait for a fresher update. We make a simplifying assumption that all the applications have the same freshness threshold τ.

Let $x_a = \{t_0 + i \cdot T_a : i \in \mathbb{N}_+\}$ be the set of periodic execution times of application $a \in A$ with period $T_a \in T$ and start time t_0. As the application subscribes its interest in sensors information to a cloud $c \in C$, the latter knows the execution schedule of such application (x_a). Additionally, we define the set of applications subscribed to a particular cloud $c \in C$ as A_c, where $A_c \subseteq A$.

Finally, we define the execution schedule of all the applications subscribed at a cloud $c \in C$ as $X_c = \bigcup_{\forall a \in A_c} x_a$. To notice, all the elements in X_c are sorted such that $x_i < x_{i+1}, \forall x \in X_c \wedge i \in \mathbb{N}_+$. For simplicity of notation and readability, we assume that all the applications subscribe to the same cloud and we therefore avoid suffix c throughout the paper (i.e. $A_c \to A, X_c \to X$).

B. Evaluation Metrics

We take into account the following four evaluation metrics:

- The ratio between the number of transmitted sensor updates and the total number of application requests in the system, ρ, is defined as:

$$\rho = \frac{\text{\# sensor updates}}{\text{\# applications requests}} \tag{1}$$

- The average Age of Information of all the applications in the system, Δ_A, is formally defined as:

$$\Delta_A = \frac{1}{|X|} \sum_{i=0}^{|X|} \Delta(x_i) \tag{2}$$

  where x_i is the i:th scheduled application execution and $\Delta(\cdot)$ is the instantaneous age of information defined as $\Delta(t) = t - U(t)$, where t is the current time and $U(t)$ the update's generation time [2].
- The average response time of all the running applications, ReT, is defined as

$$\text{ReT} = \frac{1}{|X|} \sum_{i=0}^{|X|} \hat{x}_i - x_i \tag{3}$$

where $\hat{x_i}$ is the actual execution time of x_i and we also assume that $\hat{x_i} > x_i$. The main idea behind this metric is that the applications may have *delay budgets* and are able to tolerate longer time for the response to a request with the aim to obtain better AoI. The behavior of response time is further explained in the next section.

- The number of deadline misses is delivered by a threshold function that measures the number of stale updates that have been read by the applications. We formally define this metric as the cardinality of the elements older than τ, for all the applications requests:

$$\text{misses} = \frac{|\{x : x \in X \wedge \Delta(x) > \tau\}|}{|X|} \tag{4}$$

It is straightforward to extend the formula to a multi-threshold case by adding individual application identifier suffixes to thresholds (e.g., τ_a with $a \in A$). The resulting number of misses would be the summation of individual applications misses.

IV. Scheduling Policies

In this section we incrementally introduce scheduling policies to improve AoI. Schedules are created at the edge server after receiving timing requirements of the applications, forwarded by the cloud. Eventually, individual schedules are downloaded into sensing devices. To start with, we assume ideal communication links without propagation delay and packet loss. Therefore, the delay between applications and sensors $d_{a,s} = 0\,\text{ms}$ (including between edge and cloud) and, hence, AoI can reach $0\,\text{ms}$ as well.

A. Age of Information Optimal, π^*_Δ

π^*_Δ is a scheduling policy that delivers optimal AoI. It achieves this by matching every application execution in X with a sensor update. The Age of Information Optimal scheduling policy is formally defined as:

$$\pi^*_\Delta(X) = \bigcup_{x \in X} \{x - d_{a,s}\} \tag{5}$$

To put it in simple words, we can interpret equation 5 as a scheduling policy that instructs sensors to send one update for every application execution. The update should be sent in advance of a quantity equal to the delay between the sensing device and the application. Assuming a link with no delays, it is possible to skip the addend $d_{a,s}$. As a result, this will always deliver $\Delta_A = 0\,\text{ms}$. As a consequence, the ratio between applications requests and sensor reading transmissions is $\rho = 1$ and $\text{ReT} = 0\,\text{ms}$.

B. Periodic Updates, π_P

While the Optimal AoI policy requires a sensor update for each request, it is possible to schedule fewer updates if one update can satisfy multiple requests, given that they are all within the acceptable AoI. That is, we trade an increased average AoI against fewer updates.

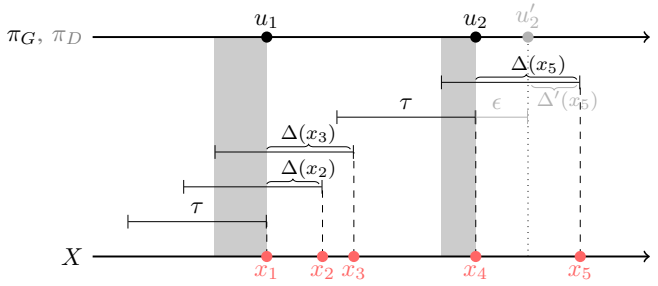


Fig. 2. Schedule π_G: Reducing sensor updates by exploiting intersections between τ segments. The scheduled sensor updates are appropriately scheduled at the end of the intersections (illustrated as gray areas). In light gray, an example of Schedule π_D for $x_{4,5}$ with a delayed update u'_2 to reduce the age of information for x_5.

Periodic delivery of sensor updates is one of the most common approaches in practice. We formally define a periodic scheduling policy as:

$$\pi_P(T_s) = \{t_0 + i \cdot T_s : i \in \mathbb{N}_+\} \tag{6}$$

where T_s is the updating period of sensor S and t_0 is the time of the first update.

What period should be chosen? The objective is to decrease ρ while keeping $\Delta(\cdot)$ bounded in the interval $[0, \tau]$. Hence, there is an upper-bound on the period when a request will provide the applications with stale data, i.e., older than τ.

The scheduling policies presented so far, namely, π^*_Δ and π_P, will be used as base lines for comparison between two other forthcoming policies that trade more effectively ρ against AoI.

C. Grouping Window, π_G

In the periodic policy, we adjust the update period to accommodate as many applications as possible for each update. In the Grouping Window policy, we will instead maximize the number of applications accessing the same update by breaking the periodicity between the updates.

How do we assign the times for the updates within a group of requests? Fig. 2 illustrates how they are grouped by using τ. We assume that all the applications in A share the same maximum acceptable τ.

In the figure, all the scheduled application executions in X are represented as (red) dots on the time line at the bottom. At the top time line, the corresponding sensor updates are marked (in black). Every execution can tolerate an AoI between 0 and τ. All the (red) dots within the shaded grey areas (grouping windows) can share the same sensor update. A schedule π_G is then composed by $\{u_1, u_2\}$ and so on. Intuitively, the algorithm finds a window that includeds as many requests as possible. These applications can be accommodated by a single update and the delivery of stale data is then avoided. In the figure, the updates are reduced from five to two. This algorithm is expected to do a better balancing between ρ and AoI than π_P.

Algorithm 1: Grouping Window with Delay Budget

Data: X, the schedule of applications executions
τ, applications timing requirement
ϵ, applications delay budget

1 $S \longleftarrow \emptyset$;
2 $i \longleftarrow 0$;
3 **while** $i < |X|$ **do**
4 $\quad j \longleftarrow 1$;
5 $\quad$ **while** $X_{i+j} \leq X_i + \tau \wedge i + j < |X|$ **do**
6 $\quad\quad j \longleftarrow j + 1$;
7 $\quad$ **if** $\epsilon > 0 \wedge \exists X_a \in [X_i, X_i + \epsilon]$ **then**
8 $\quad\quad S \longleftarrow S \cup \{X_{i+|X_a|}\}$;
9 $\quad$ **else**
10 $\quad\quad S \longleftarrow S \cup \{X_i\}$;
11 $\quad i \longleftarrow i + j$;
12 **return** S

D. Grouping Window with Delay Budget, π_D

In this scheduling policy we will extend the Grouping Window policy, π_G, by using the observation that many periodic application executions can tolerate a little delay, when getting the data from the digital twin. We denominate this policy *Grouping Window with Delay Budget*, π_D. The small delay acts as a delay budget for getting the data. The digital twin can deliberately postpone the response to the application up to the delay budget, we here call it ϵ. Why should we use such delay budget? One answer is that it allows a scheduler to detect that a digital twin can deliver a new update arriving within the ϵ interval. Thus, the application will experience better AoI than in the case of delivering the previous update. As a result, we can trade some longer response time, for each application and up to ϵ, against lower average AoI.

The operational difference between π_G and π_D is shown in Fig. 2. The application running at time x_4 can tolerate up to ϵ delay and we exploit this for delivering better AoI at time x_5. Algorithm 1 shows a pseudo code implementation of the Grouping Window with Delay Budget. At line 5-6, we define the grouping window including in it all the applications executions in the interval $[X_i; X_i + \tau]$. Then, at line 7, we check whether there are other applications scheduled starting from the first application up to ϵ. If that is the case, line 8, we delay the sensor update up to the latest execution in the aforementioned interval ($X_{i+|X_a|}$ in the code). If there are no additional application executions in $[X_i, X_i + \epsilon]$, the sensor update is scheduled at X_i. In the remainder of the algorithm, we create the schedule and we manipulate the indexes of set X so to process it from beginning to end.

E. Relation between schedules

The presented schedules are related to each others. π_G with $\tau = 0$ corresponds to π^*_Δ since no request can be grouped. Likewise, π_D with $\epsilon = 0$ corresponds to π_G and, for this reason, Algorithm 1 is valid also for π_G. For instance, π_G can be obtained by removing line 7-9 from Algorithm 1.

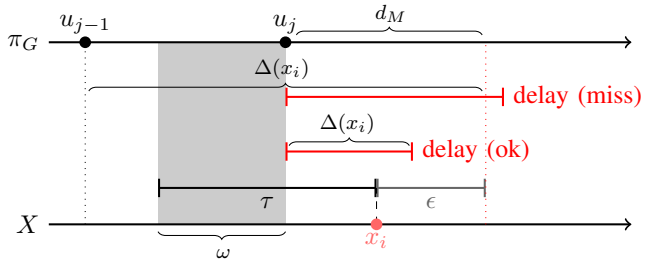


Fig. 3. The choice of the grouping window in π_G depends on the delay variation. We chose the grouping to minimize misses (i.e., $\Delta_A > \tau$), that is, the margins from maximum acceptable age of information plus delayed reply has to be larger than the grouping window plus a safety margin to account for the delay distribution: $\tau + \epsilon > \omega + d_M$.

V. Communication Delays

We assumed so far ideal communication channels without propagation delay and errors. In this section we discuss the impact of propagation delays on the performance of the schedules and introduce a delay-aware adaptation of the previously introduced policies. Delays can partly be compensated by adapting the schedule if their variation is small and can be tracked by an appropriate algorithm. In this paper, however, we focus on the worst case scenario when the delay variation is highly random. As a result, such variations cannot be tracked, yet we can compensate for the minimum delay.

We distinguish between two communication segments: between edge and cloud, and cloud and applications as they affect AoI in a different context. The delay between edge and cloud affects a set of grouped application requests. On the other hand, the delay between cloud and applications affects individual application requests. Studying these delays separately is motivated by the fact that we assume compensation for constant delay components; a dominating delay variation among cloud and applications can be considered a general case covering delay variation on both segments, and a dominating delay variation between edge and cloud includes the scenario where applications are hosted in the same cloud as the digital twin.

A. Digital Twin Strategies

The digital twins in the cloud are aware of the schedules produced by the edge and can measure delays in order to obtain delay averages and variance. On the basis of this knowledge, they can estimate when sensor updates are likely to come. In the best case, a twin could calculate a percentile, say 95:th, of the delay distribution and use that together with ϵ and τ to see if it is worth waiting for an update.

Given this knowledge, the twin can decide to either immediately deliver the last update to the application or decide to wait, if it is likely that a better update will come within the ϵ time. The choice depends on which of the previous and the

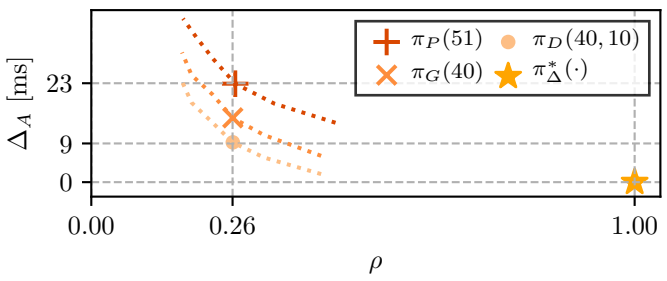


Fig. 4. Average age of information Δ_A against ratio samples/requests (assuming perfect links).

forthcoming updates that are expected will have the lowest AoI.

Furthermore, for unpredictable communication delays there is always a risk for stale data and out-of-order updates. We assume digital twins to be preemptive, meaning that outdated updates are automatically discarded.

For benchmarking our proposed policies we also constructed an "Oracle digital twin" who knows the exact communication delay for each individual update, now as well as in the future. This means that the Oracle can predict when all updates will arrive to the twin and always decide whether it is better to wait for an update or to deliver the previous one in order to minimize the AoI.

B. Delay between Edge and Cloud

When communication delays affect the network path between an edge and a cloud, all the applications subscribed to a particular digital twin will be out of synchronization with respect to the updates. For example, assuming a temporary delay at the path, the sensor updates will be received by all the applications later than expected and they may all experience stale information. Nonetheless, in case of frequent sensor updates, this misalignment may be insignificant since the application will get fresh enough data anyway in the near future. We propose a simple yet effective way to handle communication delays between edge and cloud in Sec. V-D.

C. Delay between Cloud and Applications

The delay between the twin in the cloud and the application will not be discussed further besides the observation that it only affects individual applications. The digital twin is aware of these delays (e.g., via round-trip time estimation using stamps). When the delays become significant the digital twin can notify this to the edge device that will then re-arrange the schedule. The method proposed in the next section can be applied in that case.

D. Miss Avoidance Grouping Window, π_ω

The Grouping Window policy suffers from communication induced delays in two ways. First, an update which is delayed for more than ϵ will lead to a deadline miss for that application. A deadline miss will also happens when an update arrives after the end of the grouping window. With this policy, even a small delay variations may significantly impact the number of misses. We now propose a method for mitigating this undesired sensitivity and suggest boundaries on the maximum number of deadline misses. The main idea is to reduce the length of the grouping window with a "safety margin" that accounts for the variations in the delays. Intuitively, the probability of misses will decrease with a smaller window.

Knowing the extent of delay variations and/or distributions would assist system's designers to target (and obtain) a limited amount of deadline misses while trading off with higher ρ. For example, assume that we aim at 5% misses over the total number of application requests. In order to achieve this, we must calculate the 95:th percentile of the delay distribution. The aforementioned safety time is then set at the 95:th percentile. For convenience in this paper we will call the safety margin d_M.

We now re-define the size of the grouping window with the safety margin d_M for varying delays as follow:

$$\omega = \tau + \epsilon - d_M \tag{7}$$

A graphical representation of the scheduling policies is shown in Fig. 3. It should be observed that the physical device should now send the update in advance of a quantity equal to $d_M - \epsilon$. Assuming that a schedule S' is the output of Algorithm 1 (with $\tau = \omega$ and $\epsilon = 0$[1]), a new schedule, S, with safety margin d_M can then be formalized in the following way:

$$S = \{s + \epsilon - d_M : s \in S'\} \tag{8}$$

Intuitively, while decreasing the number of misses, the new schedule will provide higher average AoI.

VI. Evaluation and Discussion

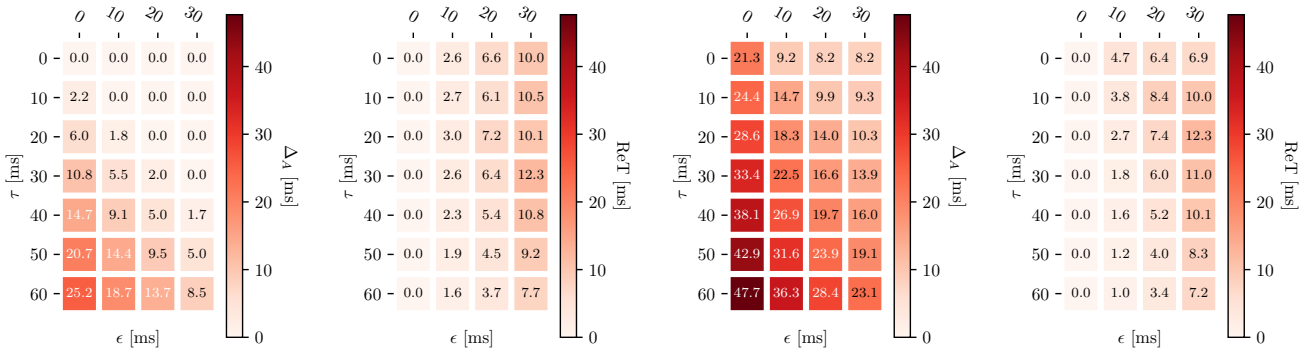
In this section we evaluate numerically the metrics introduced in Sec. III-B comparing scheduling policies presented in Sec. IV and V. For every simulation instance we provide 5 min of simulated time that provides enough statistical relevance as we operate at milliseconds granularity. If not explicitly mentioned, we assume N = 10 application with periodicity randomly selected in the interval $[100, 200)$ ms. When we account for delays, we assume samples from a Pareto distribution [26] with shape $a = 1.672$ and scale $m = 5$.

Whenever the periodic policy, π_P, is compared to one of our proposed schedule, we choose the updating period, T_s, to correspond to the mean inter-arrival time between the updates of such schedule. As a result, the periodic schedule and the proposed one will have approximately the same ρ.

A. Age of Information vs. Ratio Updates-Requests

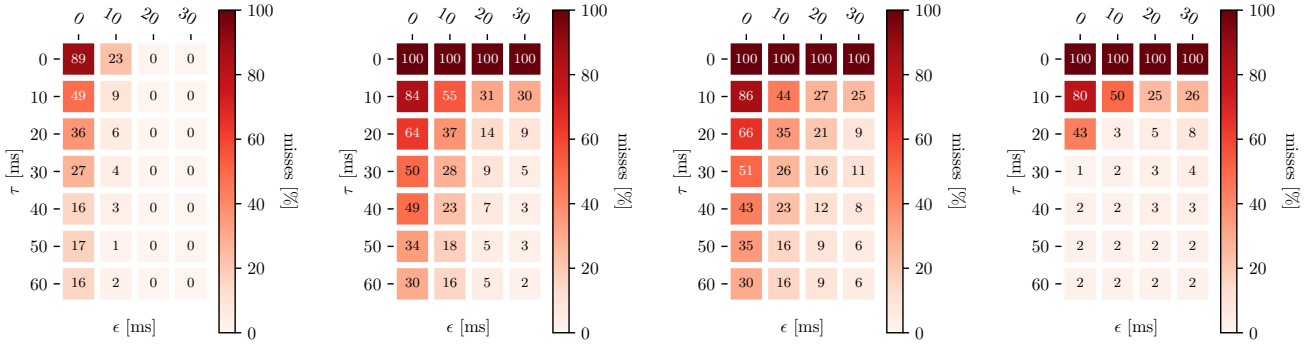
Figure 4 shows the hyper-plane average age of information vs. ratio updates-requests in the ideal communication channel case. We compare Age-Optimal π^*_Δ, Periodic π_P, Grouping Window π_G and Grouping Window with Delay Budget π_D

[1] ω as from Equation 7 with $\epsilon \geq 0$, while $\epsilon = 0$ only for pseudo-code in Algorithm 1, not for Equation 7 and 8.



(a) No delay: Age of Information (b) No delay: Response time (c) With delay: Age of Information (d) With delay: Response time

Fig. 5. System metrics evolution varying τ and ϵ on perfect links and links with random delay for the Grouping Window with Delay Budget schedule π_D. The data points ($\epsilon = 0$) correspond to π_G, the data point ($\tau = 0$, $\epsilon = 0$) to π^*_Δ.



(a) No delay: periodic π_P (b) With delay: periodic π_P (c) With delay: Grouping Window with Delay Budget π_D (d) With delay: Miss Avoidance Grouping Window π_ω

Fig. 6. Percentage of misses evolution varying τ and ϵ for different scheduling policies.

scheduling policies. The star on the bottom-right corner represents the Age-Optimal schedule which provides best possible AoI and maximum ρ, as this strategy schedules an update for every application request. The up most dotted line shows the evolution of the periodic schedule. We clearly see that the smaller the period, the lower the AoI at the cost of higher ρ. Roughly, $\Delta_A \approx T_s/2$. Both π_G and π_D deliver better AoI for the same updates/request ratio as they are designed to meet applications requirements. The applications will be scheduled in between a time window that, in the best case, provides AoI $0\,\text{ms}$ and, in the worst, $40\,\text{ms}$. Δ_A depends on the distribution and the number of applications in the window. In the illustrated example, $\pi_P(51)$, $\pi_G(40)$ and $\pi_D(40, 10)$ for $\rho = 0.26$, π_G and π_D improve Δ_A by 40% and 66% respectively. The reason for π_D to be better than π_G, in terms of AoI, is that the applications are willing to pay some response time ReT for fresher data.

B. Age of Information and Response Time

Figure 5 shows the evolution of the system metrics with several τ and ϵ configurations for the π_D schedule, both on a perfect link and a link affected by delays. Fig. 5a shows the average AoI, Δ_A. Reading the heat map from top to bottom, we evince that the bigger τ the higher Δ_A, intuitively because of the larger grouping window contributing to higher $\Delta(\cdot)$. For fixed τ values, the bigger ϵ the higher the response time ReT, see Fig. 5b. As a result, Δ_A decreases because the applications are willing to wait some time to get fresher information. To be noticed, ReT is significantly smaller than the tolerance ϵ. Reading the matrix from top to bottom, we notice an increment then decrement in ReT. In fact, when $\tau < \epsilon$, the applications are willing to wait time to fetch fresher data, that will come because of the smaller grouping window. As a result, the average response time increases. On the other hand, when $\tau > \epsilon$, updates come more seldom and the applications will not wait: this result in smaller average ReT.

Figure 5c and 5d present the same metrics on a link affected

by delays. The same behavior described for the ideal link applies in this case as well, with the difference that we must account for an additive factor coming from the delays. These contribute to increase Δ_A, especially for big τ and small ϵ. In fact, they affect particularly the applications that are scheduled between the beginning of the grouping window and the extent of the delay itself. To make things even worse, if these applications have no delay budget, they will not be able to compensate for delays, delivering poor AoI and higher chances of misses. The response time is not significantly affected by random delays as we assume an oracle digital twin, see Sec. V-A.

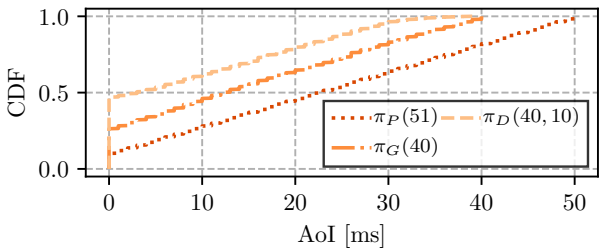
C. Misses

We now consider how π_P, π_D and π_ω are affected in terms of deadline misses. For this particular experiment, we aim at 5% misses and, because of the used Pareto distribution, we obtain $d_M = d_{95} = 30\,\text{ms}$. Fig. 6 provides an overview of the evolution of misses, while varying τ and ϵ for the aforementioned policies. As a remainder, a miss happens when the sensor update is consumed by the application when it is older than τ. We consider first π_P, whose period is calculated as the mean value of the inter-arrival time of the relative Grouping Window schedule used for comparison. To put it in other words, the two schedules will have the same ρ. Fig. 6a shows the performance of the periodic schedule on a perfect link. We clearly see that, without delay budget, the percentage of misses is rather high. On the other hand, if the applications have delay budget, the periodic schedule is able to compensate. When introducing random delays, see Fig. 6b, the periodic scheduling delivers many more misses than in the previous case. Again, if the applications do not account for some delay budget, the performance is very poor.

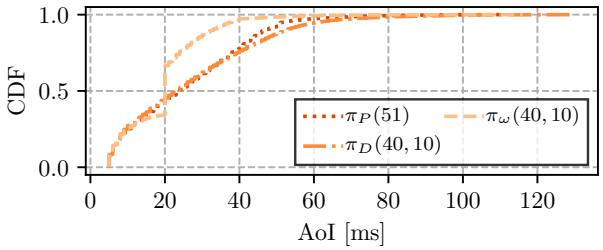
Fig. 6c shows how the Grouping Window with Delay Budget is affected by delays. The results are in line with the periodic schedule as, for some configuration of (τ, ϵ), it is slightly better and, for others, slightly worse. Nonetheless, both Fig. 6c and 6b exhibit degraded performance. For this reason, we introduced Miss Avoidance Grouping Window π_ω, and the obtained percentage of misses is shown in Fig. 6d. For $\tau \geq 20\,\text{ms}$[2] we notice a significant performance improvement both over π_P and π_D. Surprisingly, and contrarily to all the previous cases, some increasing ϵ deliver slightly higher number of misses. The explanation behind this is rooted in Equation 7. In fact, the bigger ϵ the bigger the grouping window, meaning more applications executions in it. When a random delay is greater than d_{95}, all the applications in the window will experience a deadline miss. That is, increasing ϵ in π_ω may lead to deliver higher number of misses, yet reducing ρ. Nonetheless, such increment of misses is negligible.

D. Age of Information Distribution

Fig. 7 presents the cumulative distribution function (CDF) of AoI on both an ideal channel and a channel affected by delays. The former case is depicted in Fig. 7a. We observe that all the schedules exhibit a linear distribution and the CDFs resemble parallel lines. The main difference between the schedules is the amount of zeros. In fact, zeros are distributed such that $\pi_G \approx 25\%$ and $\pi_D \approx 50\%$, while $\pi_P = 0\%$. The reason behind the higher number of zeros in π_G is that the schedule is optimized for delivering best AoI to the first application of every grouping window. On the other hand, π_D delivers even more zeros because it serves best AoI to *at least* one application in every grouping window. In fact, applications with delay budgets can be postponed such that more applications will read best AoI at the time of the new update. From the plot, we als notice an interesting property of AoI for periodic applications: π_P delivers AoI that is uniformly distributed in $[0, 51]$.



(a) Ideal channel



(b) Channel with random delays

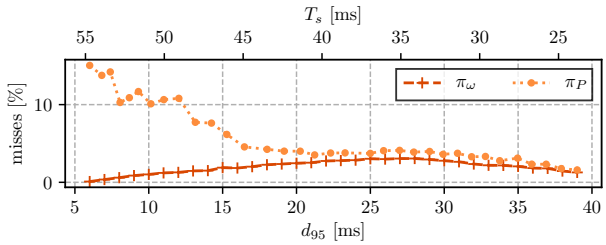
Fig. 7. CDF of average age of information for different scheduling policies.

Fig. 7b shows the CDF of AoI for π_P, π_D and π_ω on a channel affected by delays. π_D and its relative periodic, π_P, have a similar distributions as they approximately share the same ρ. The only difference between the two schedules is that π_P delivers updates regularly while π_D may deliver either a bit earlier or a bit later according to applications schedule. Furthermore, none of the policies implement a mechanism to mitigate delays. On the other hand, π_ω takes into account "safety margins" for avoiding to serve applications with stale data due to delays. As a result, we observe significant improvement on the X-axis between $20\,\text{ms}$ and $60\,\text{ms}$.

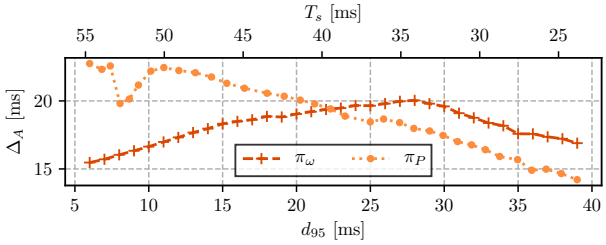
E. A broader picture

It is difficult to provide a complete overview of periodic and Grouping Window scheduling policies. In fact, there are many possible parameters combinations and the results are strongly dependent on the used delay distribution. In this section, we select one of these configurations ($\tau = 40\,\text{ms}, \epsilon = 10\,\text{ms}$) and we study it over a range of delay distributions. Our goal

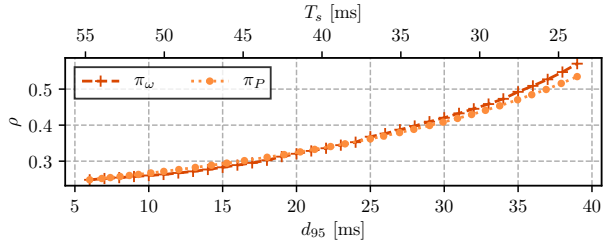
[2]For $\tau < 20$ the used delay distribution would make ω negative but, in that case, we enforce $\omega = 0$.



(a) Misses



(b) Age of Information



(c) Ratio samples-requests

Fig. 8. System metrics evolution for $\tau = 40$ms and $\epsilon = 10$ms on links with random delay. Small d_{95} corresponds to small delays and high d_{95} to long delays.

is to give insights on how to select a suitable scheduling policy on the basis of the delay distribution. To that end, we adapt the shape parameter α of the Pareto delay distribution to vary its 95:th percentile d_{95}. d_{95} is in linear relation to the grouping window ω of π_ω (c.f., equation 7), resulting in smaller grouping windows for higher delay variation.

The figures in Fig. 8 set the periodic and Grouping Window schedules in relation to each other in terms of misses, average system AoI, and overhead. The upper scale of the x-axis corresponds to the average sampling interval T_s resulting from π_ω, which is applied in π_P.

Fig. 8a displays the percentage of misses. Independently from the extent of the delay distribution, the periodic schedule is worse than our proposed schedule. In particular, when $T_s > 30\,\text{ms}$, the number of misses increases dramatically. Furthermore, π_ω is constantly below 3%, achieving the design goal of number of deadline misses mitigation.

Fig. 8b shows the average system AoI Δ_A for the aforementioned settings. We observe that, for the periodic schedule, the greater the period T_s the higher the AoI. On the other hand, the AoI for π_ω increases with the delay variation up to $d_{95} < 28\,\text{ms}$, and decreases after. In fact, the smaller the extent of delay variations the more we get closer to the perfect link scenario, where we already showed that the Grouping Window approach outperforms the periodic. We find particularly interesting that through this plot it is possible to see which schedule is best according to the delay distribution, e.g., when the two curves intersect at $d_{95} = 22\,\text{ms}$ and the AoI of the Grouping Window schedule exceeds the AoI of the periodic schedule.

Finally, in Fig. 8c we compare the ratio updates-requests. We here observe that both π_P and π_ω are similar. The gap between the curves is to be imputed to the approximation performed to make the two schedules comparable (π_ω mean inter-arrival time).

F. Discussion

We proposed π_G and π_D, two scheduling policies designed to deliver better Δ_A than the periodic schedule π_P. The proposed policies deliver better Δ_A when applied to an ideal communication channel without delays. Nonetheless, when introducing delays in the communication channel, the proposed schedules are not better than the periodic in terms of misses. For this reason, we introduced π_ω.

The Miss Avoidance Grouping Window, π_ω, is a scheduling policy designed to impose an upper bound on the number of misses induced by communication delays. We show that π_ω always delivers fewer misses than π_P. Nonetheless, when the delay variations are big, the periodic policy is able to deliver better Δ_A than π_ω. This is due to the fact that π_ω imposes a safety margin that results in sending updates unnecessarily early.

We now discuss some implementation challenges. First, the proposed scheduling policies are sensitive to the notion of time. To some extent, the safety margin introduced to handle random delays can also compensate for timing inaccuracies. Furthermore, as the number of applications grows, the schedule produced by the edge device may be very long and not suitable to be stored in resource constrained devices, e.g., sensors. In this case, the edge should fragment such schedule, push partial information to the sensor while keeping track of the progress and renew the schedule when needed, for every sensor. However, such strategy may be merged together with schedule updates sometimes required by new applications joining the system.

VII. Conclusion

We studied the problem of timely dissemination of sensor data to applications with freshness requirements by means of a digital twin. We minimize direct access to the possibly battery powered physical devices yet improving Age of Information as a data freshness metric. The proposed Age of Information-aware scheduling policies allow the physical devices to push sensor updates to the digital twin by grouping application requests based on freshness criterion, thereby reducing the number of sensor transmissions. Scheduling updates in a network with communication delays is non-trivial. In fact, random delays affect the Age of Information and are likely to result in deadline misses, if the requirements are too stringent. By estimating the random delay component, we introduce some

safety margin in the schedule to keep the misses below a target level. Delaying replies to application requests in the favor of awaiting fresh data has proven to be an efficient measure. We numerically evaluate the proposed policies against a simple yet widely used periodic scheduling and demonstrate that our schedule, even with high delay variation, outperforms the periodic schedule in terms of misses, at the cost of marginally higher Age of Information.

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers for their constructive comments. We are grateful to Bengt Ahlgren and Anders Lindgren from RISE for discussions. This work was supported by the Swedish Foundation for Strategic Research under the project "Future Factories in the Cloud (FiC)" with grant number GMT14-0032.

REFERENCES

[1] S. Kaul, M. Gruteser, V. Rai, and J. Kenney, "Minimizing age of information in vehicular networks," in *2011 8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, June 2011, pp. 350–358.

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

[3] K. Chen and L. Huang, "Age-of-information in the presence of error," in *2016 IEEE International Symposium on Information Theory (ISIT)*, July 2016, pp. 2579–2583.

[4] A. Kosta, N. Pappas, A. Ephremides, and V. Angelakis, "Age and value of information: Non-linear age case," in *2017 IEEE International Symposium on Information Theory (ISIT)*, June 2017, pp. 326–330.

[5] C. Kam, S. Kompella, G. D. Nguyen, and A. Ephremides, "Effect of message transmission path diversity on status age," *IEEE Transactions on Information Theory*, vol. 62, no. 3, pp. 1360–1374, March 2016.

[6] 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, April 2016.

[7] E. Najm and R. Nasser, "Age of information: The gamma awakening," in *2016 IEEE International Symposium on Information Theory (ISIT)*, July 2016, pp. 2574–2578.

[8] R. D. Yates and S. Kaul, "Real-time status updating: Multiple sources," in *2012 IEEE International Symposium on Information Theory Proceedings*, July 2012, pp. 2666–2670.

[9] R. D. Yates, "Lazy is timely: Status updates by an energy harvesting source," in *2015 IEEE International Symposium on Information Theory (ISIT)*, June 2015, pp. 3008–3012.

[10] B. T. Bacinoglu, E. T. Ceran, and E. Uysal-Biyikoglu, "Age of information under energy replenishment constraints," in *2015 Information Theory and Applications Workshop (ITA)*, Feb 2015, pp. 25–31.

[11] Y. Sun, E. Uysal-Biyikoglu, R. Yates, C. E. Koksal, and N. B. Shroff, "Update or wait: How to keep your data fresh," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, April 2016, pp. 1–9.

[12] T. Bacinoglu and E. Uysal-Biyikoglu, "Scheduling status updates to minimize age of information with an energy harvesting sensor," in *2012 IEEE International Symposium on Information Theory Proceedings (ISIT)*, 06 2017, pp. 1122–1126.

[13] X. Wu, J. Yang, and J. Wu, "Optimal status update for age of information minimization with an energy harvesting source," *IEEE Transactions on Green Communications and Networking*, vol. 2, no. 1, pp. 193–204, March 2018.

[14] S. Feng and J. Yang, "Optimal status updating for an energy harvesting sensor with a noisy channel," *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 348–353, 2018.

[15] B. Li, A. Eryilmaz, and R. Srikant, "On the universality of age-based scheduling in wireless networks," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, April 2015, pp. 1302–1310.

[16] I. Kadota, E. Uysal-Biyikoglu, R. Singh, and E. Modiano, "Minimizing the age of information in broadcast wireless networks," in *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Sept 2016, pp. 844–851.

[17] Q. He, D. Yuan, and A. Ephremides, "Optimizing freshness of information: On minimum age link scheduling in wireless systems," in *2016 14th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, May 2016, pp. 1–8.

[18] Y. P. Hsu, E. Modiano, and L. Duan, "Age of information: Design and analysis of optimal scheduling algorithms," in *2017 IEEE International Symposium on Information Theory (ISIT)*, June 2017, pp. 561–565.

[19] N. Lu, B. Ji, and B. Li, "Age-based scheduling: Improving data freshness for wireless real-time traffic," in *Proceedings of the Eighteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. Mobihoc '18. New York, NY, USA: ACM, 2018, pp. 191–200. [Online]. Available: http://doi.acm.org/10.1145/3209582.3209602

[20] R. D. Yates, M. Tavan, Y. Hu, and D. Raychaudhuri, "Timely cloud gaming," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, May 2017, pp. 1–9.

[21] S. Ioannidis, A. Chaintreau, and L. Massoulie, "Optimal and scalable distribution of content updates over a mobile social network," in *IEEE INFOCOM 2009*, April 2009, pp. 1422–1430.

[22] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb 2009. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

[23] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: ACM, 2012, pp. 13–16. [Online]. Available: http://doi.acm.org/10.1145/2342509.2342513

[24] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct 2009.

[25] K. M. Alam and A. E. Saddik, "C2ps: A digital twin architecture reference model for the cloud-based cyber-physical systems," *IEEE Access*, vol. 5, pp. 2050–2062, 2017.

[26] W. Zhang and J. He, "Modeling End-to-End Delay Using Pareto Distribution," in *International Conference on Internet Monitoring and Protection (ICIMP)*, 2007.