Kronos: A 5G Scheduler for AoI Minimization under Dynamic Channel Conditions

Chengzhang Li†, Yan Huang†, Yongce Chen†, Brian Jalaian‡, Y. Thomas Hou†, and Wenjing Lou†
† Virginia Polytechnic Institute and State University, Blacksburg, VA
‡ U.S. Army Research Laboratory, Adelphi, MD

***Abstract*—Age of information (AoI) is a powerful new metric to quantify the freshness of information and has gained increasing popularity in IoT applications. Existing models on AoI remain primitive and do not consider state-of-the-art transmission technologies such as 5G. They also fail to consider the impact of dynamic channel conditions. In this paper, we present Kronos, a 5G-compliant AoI scheduling algorithm that can cope with highly dynamic channel conditions. Kronos is capable of performing RB allocation and selecting MCS for each source node based on channel conditions, with the objective of minimizing long-term AoI. To meet the stringent real-time requirement for 5G, we propose a GPU-based implementation of Kronos on low-cost off-the-shelf GPUs. Through simulations and experiments, we show that Kronos can find near-optimal AoI scheduling solutions in sub-millisecond time scale. To the best of our knowledge, this is the first 5G-compliant real-time AoI scheduler that can cope with dynamic channel conditions.**

I. Introduction

With the proliferation of IoT and its deployment for massive information gathering and sharing through edge/cloud computing, users are no longer satisfied with merely obtaining the information they desire, but rather, how fresh the information is when it is consumed. To address this trend, the concept of "Age of Information" (AoI) was conceived in [1], [2] and has since gained its acceptance in the research community. AoI is defined as the elapsed time for a sample (stored at a particular location, e.g., edge or cloud) between current time (now) and the time when the sample was first generated (collected) at its source. AoI measures the *freshness* of the sample from the time it was initially generated, which is of greater interest from a consumer's perspective than merely delay (or latency) of the sample to transit through the network. In this sense, the new AoI concept represents a new performance metric that has the potential to transform traditional throughput/delay-based networking research.

There has been active research on designing scheduling algorithms to minimize AoI [3]. However, existing research on AoI has been largely limited to information-theoretic exploration. Most notably, few existing efforts have considered the capability of state-of-the-art transmission technologies such as cellular (e.g., 4G LTE [4] or 5G NR [5]) or Wi-Fi (e.g., [6]) in AoI modeling and analysis. A bulk of existing research has been concerning extremely simple toy models (see, e.g., [7]–[9]) which are hardly applicable to real-world IoT systems. Further, there is hardly much research on AoI scheduling that addresses the impact of time-varying channel conditions. In [10], [11], the authors considered time-varying channels that only employed extremely unrealistic models (e.g., binary channel under which one can either transmit a sample or nothing). In [12], Lu *et al.* assumed channel coherence time could last an entire frame (consisting of a large number of time slots). But in reality, channel condition can change rapidly (e.g., for each TTI in 5G) and hardly holds constant over an entire frame.

In this paper, we focus on the design of a 5G-compliant AoI scheduler and address the impact of time-varying channel conditions. Our BS at edge IoT network is designed to conform to the state-of-the-art transmission technology in 5G cellular standard [5], which is what major carriers (e.g., AT&T [13], Verizon [14]) are supporting. Further, our scheduler is designed to cope with highly dynamic channel conditions (e.g., time-selective fading and frequency-selective fading), which is a major challenge in real-world environment. Finally, we will ensure that our AoI scheduler strictly meet the stringent timing requirement (i.e., sub-millisecond running time for computing scheduling solution) as specified in 5G standard.

There are a number of technical challenges in this research. First, as we shall see in Section III, the AoI scheduling problem in our model entails the allocation of resource blocks (RBs) and the selection of modulation and coding scheme (MCS) for each source node in each TTI based on channel conditions. This presents a much larger search space for an optimal solution than any of those problems considered to date in the AoI literature. Second, the stringent timing requirement for real world 5G systems (i.e., sub-millisecond time scale) sets a hard performance measure against any new design of an AoI scheduler. As we shall see, it is extremely challenging to find a near-optimal solution for a problem of such size and complexity in such a time scale.

The main contributions of this paper are the following:

- This paper studies AoI with consideration of varying channel conditions under 5G-based IoT network. Specifically, we model uplink transmission resource as grids of RBs that span both time and frequency domains with different channel conditions. The scheduling problem under our model entails RB allocation to each source node and selection of MCS by each source node based on channel condition on each RB, with the goal of minimizing long-term average AoI.
- Since channel conditions for the future is unknown, we pursue the design of an online scheduling algorithm. For performance benchmark, it is necessary to develop a lower bound for the objective function. We propose a novel computational procedure to find an asymptotic lower bound for the objective. Specifically, we first relax the original AoI minimization problem to a data rate mini-

mization problem. Then we employ a gradient scheduling algorithm to find an asymptotic lower bound for this problem. The gradient scheduling minimizes an empirical data rate for each TTI, which can be formulated as an integer quadratic programming problem and be solved by the CPLEX solver.

- For our AoI scheduling problem, we present Kronos, an online algorithm that conforms to 5G transmission standard and can cope with varying channel conditions. The essence of Kronos is to iteratively select a source node for RB allocation until all RBs in a TTI are allocated. We propose a novel metric that takes into consideration of AoI outage and channel conditions for the source node. By using this metric, we can identify the next source node for RB allocation and determine its MCS.
- To ensure Kronos can meet the stringent timing requirement in 5G, we propose to employ commercial off-the-shelf GPUs for implementation. This approach allows us to take advantage of the massive number of GPU processing cores to compute and compare the scheduling metric for all possible combinations of source nodes and MCSs. For proof-of-concept, we successfully implement Kronos on an Nvidia Quadro P6000 GPU using the CUDA programming model. Through extensive performance evaluation under various channel fading models, we find that Kronos can achieve near-optimal performance (when compared with our lower bound) in sub-millisecond time scale, thus meeting 5G timing requirement.

II. A 5G-BASED IOT ARCHITECTURE

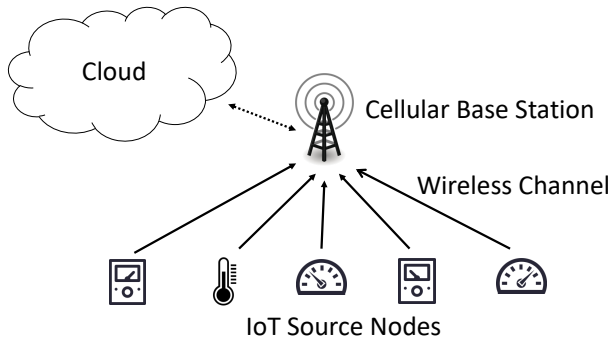


Figure 1: System Model: N nodes collect information and update it to a BS.

Consider a 5G cellular-based IoT network where a set $\mathcal{N}$ of source nodes collect information and forward it to a base station (BS) (see Fig. 1). At each source node, it produces samples of information periodically (through measurement, sensing, or information capture). Denote T_i as the sampling period (in unit of time slots) for source node i. Due to the heterogeneity of IoT applications, the sampling periods are generally different among different source nodes. For source node i, denote L_i as the sample size (in bits), which is the amount of information carried in the sample. Again, due to heterogeneity of IoT devices, sample sizes are generally different among different source nodes.

Once a sample is produced at a source node i, it is stored in a local memory. To ensure the cellular BS can received the latest sample, a source node always selects the freshest sample (i.e., the most recently generated sample) for transmission. Once a sample has started its transmission to the BS, it must be transmitted in its entirety, regardless how many time slots it may take. That is, any newly generated sample afterward cannot preempt an ongoing transmission of an older sample.

Since the uplink transmission from IoT source nodes to the BS follow 5G cellular technology [5], transmission resource is organized as grids of resource blocks (RBs) that span both time and frequency domains. In the time domain, time is equally slotted into transmission time intervals (TTIs), while in the frequency domain, bandwidth is equally slotted into a large number of tiny slices, and each tiny slice over a TTI is called an RB. That is, for each TTI, there is a large number of RBs that can be allocated to the IoT source nodes for uplink transmission.

Due to varying channel conditions in time (across different TTIs, i.e., time-selective fading) and frequency (across different RBs, i.e., frequency-selective fading), channel feedback from each source node is necessary for optimal scheduling of transmission resources. Based on such feedback, scheduling of RBs among the source nodes can be performed for each TTI. In addition to channel variation over time and frequency, new samples from different sources may be produced within each TTI. So it is utmost necessary to perform scheduling for each TTI, the smallest time resolution for 5G transmission.

Since the number of RBs within each TTI is limited, not every sample from each source node will be transmitted to the BS. Recall that to minimize AoI at the BS, the BS always selects the freshest sample at a source for the next transmission. As a result, only a fraction of samples generated at each source node will be transmitted while the rest will be eventually discarded at the source nodes.

At the BS, the collected information can be either processed and stored locally (edge computing) and/or be forwarded to a cloud, where the information can be further processed and accessed broadly by users at any location. Since many time-sensitive IoT applications need to access the latest sampled information from each source, it is desirable to maintain the freshest sample (from each source) at the edge BS. So it is necessary to design a specialized scheduler to minimize AoI for the maintained samples at the BS. Clearly, solving a complex scheduling problem such as AoI minimization within each TTI in real time is not a trivial problem. This is the focus of this paper.

III. MODELING AND PROBLEM STATEMENT

A. AoI Notation

Recall that at each source node i, it produces a sample for every T_i time slots (TTIs). Denote $U_i^s(t)$ as the generation time of the most recently generated sample at source node i.

Clearly, $U_i^{\mathrm{s}}(t) \leq t$. Then the AoI at source node i at time t, denoted as $A_i^{\mathrm{s}}(t)$, is defined as:

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

Since sampling at source node i has a period T_i, function $A_i^{\mathrm{s}}(t)$ exhibits a zigzag shape with slope 1 and period T_i.

AoI is location dependent. The sample maintained at the BS may be older than the freshest sample stored at the source node. Denote $U_i^{\mathrm{B}}(t)$ is the generation ("birth") time of the most recently received sample from source node i at the BS. Then the AoI at the BS for source i at time slot t, denoted as $A_i^{\mathrm{B}}(t)$, is defined as

$$A_i^{\mathrm{B}}(t) = t - U_i^{\mathrm{B}}(t). \tag{2}$$

To analyze $A_i^{\mathrm{B}}(t)$, it is necessary to make a connection between $A_i^{\mathrm{B}}(t)$ (AoI at edge BS) and $A_i^{\mathrm{s}}(t)$ (AoI at a source node). From source node i, for the k-th transmitted sample, denote its beginning transmission TTI as $b_i(k)$ and ending transmission TTI as $e_i(k)$. By the definition of $U_i^{\mathrm{s}}(t)$, the generation time of this k-th sample is $U_i^{\mathrm{s}}\big(b_i(k)\big)$. After the last unit of data of this sample is completely sent to the BS at TTI $e_i(k)$, at the next TTI $(e_i(k)+1)$, $U_i^{\mathrm{B}}(t)$ is updated and we have $U_i^{\mathrm{B}}(e_i(k)+1) = U_i^{\mathrm{s}}\big(b_i(k)\big)$. By the definitions of $A_i^{\mathrm{B}}(t)$ and $A_i^{\mathrm{s}}(t)$, it can be shown that AoI evolution at the BS follows the following expression:

$$A_i^{\mathrm{B}}(t+1) = \begin{cases} A_i^{\mathrm{s}}\big(b_i(k)\big) + e_i(k) - b_i(k) + 1, & \text{if } t = e_i(k), \\ A_i^{\mathrm{B}}(t) + 1, & \text{otherwise.} \end{cases} \tag{3}$$

The long term average of $A_i^{\mathrm{B}}(t)$ for source node i at the BS is defined as:

$$\bar{A}_i^{\mathrm{B}} = \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} A_i^{\mathrm{B}}(t). \tag{4}$$

Denote w_i as the weight of source node i. Then the weighted sum of long term average of $A_i^{\mathrm{B}}(t)$ over all source node $i \in \mathcal{N}$, denoted as $\bar{A}^{\mathrm{B}}$, is:

$$\bar{A}^{\mathrm{B}} = \sum_{i\in\mathcal{N}} w_i \bar{A}_i^{\mathrm{B}}. \tag{5}$$

In this paper, we want to minimize $\bar{A}^{\mathrm{B}}$.

B. Uplink Transmission

As shown in Fig. 1, the set $\mathcal{N}$ of IoT source nodes (users) share an uplink channel to transmit to the BS. Denote $\mathcal{B}$ as the set of RBs in one TTI for uplink transmission. The scheduler at the BS must allocate this set $\mathcal{B}$ of RBs to a subset of source nodes within each TTI to minimize $\bar{A}^{\mathrm{B}}$.

Denote $x_i^b(t)$ as a binary variable indicating whether RB $b \in \mathcal{B}$ is allocated to source node i at TTI t, i.e.,

$$x_i^b(t) = \begin{cases} 1 & \text{if RB } b \text{ is allocated to node } i \text{ at TTI } t, \\ 0 & \text{otherwise.} \end{cases}$$

Since each RB can only be allocated to at most one source node [5], we have:

$$\sum_{i\in\mathcal{N}} x_i^b(t) \leq 1 \quad (b \in \mathcal{B}). \tag{6}$$

Besides RB allocation, for each TTI, the scheduler also needs to choose a modulation and coding scheme (MCS) for each source node [5]. The MCS of each source node directly determines the modulation and coding rate – how much information (in unit of bits) is modulated and coded in each RB for this source node. The higher the MCS is, the higher the modulation and coding rate is. On the other hand, the maximum amount of information can be transmitted on one RB also depends on the channel condition. If the channel condition for this RB is poor and the source uses a high MCS, information carried in the RB will not be successfully received and decoded by the BS. Therefore, the achievable data rate by an RB $b \in \mathcal{B}$ depends on both the MCS selected by the scheduler as well as the channel condition for this RB.

Based on [5], there are 29 levels of MCSs for transmission. Denote $\mathcal{M}$ as the set of these available MCSs (i.e., $\mathcal{M} = \{1, 2, ...29\}$), where we assume $m = 1$ corresponds to the lowest MCS and $m = 29$ corresponds to the highest MCS. Denote $q_i^b(t)$ as the maximum MCS that can be used for RB $b \in \mathcal{B}$ with respect to source node i so that information carried in RB can be successfully received by the BS. We have:

$$1 \leq q_i^b(t) \leq |\mathcal{M}|.$$

In practice, $q_i^b(t)$ is determined by the channel quality indicator (CQI) report carried in the feedback from source node i at TTI $(t-1)$. Denote c^m as the modulation and coding rate for an RB under MCS m and $r_i^{b,m}(t)$ as the achievable data rate by RB b w.r.t. source node i under MCS m. If $m \leq q_i^b(t)$, the transmission is successful and the achievable data rate is c^m. Otherwise, i.e., $m > q_i^b(t)$, the transmission is unsuccessful the achievable data rate is 0. We have:

$$r_i^{b,m}(t) = \begin{cases} c^m & \text{if } m \leq q_i^b(t), \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

Note that although each RB can only be allocated to at most one source node within a TTI, a source node may be allocated with multiple RBs. For a source node allocated with multiple RBs, it must choose and use one MCS $m \in \mathcal{M}$ for all its RBs [5]. Denote $y_i^m(t)$ as a binary variable indicating whether MCS $m \in \mathcal{M}$ is chosen to source node i at TTI t, i.e.,

$$y_i^m(t) = \begin{cases} 1 & \text{if MCS } m \text{ is chosen for source } \ i \text{ at TTI } t, \\ 0 & \text{otherwise,} \end{cases}$$

and

$$\sum_{m\in\mathcal{M}} y_i^m(t) \leq 1 \quad (i \in \mathcal{N}). \tag{8}$$

Denote $R_i(t)$ as the amount of information transmitted by source node i at TTI t across all RBs allocated to it. We have

$$R_i(t) = \sum_{b\in\mathcal{B}} \sum_{m\in\mathcal{M}} x_i^b(t) y_i^m(t) r_i^{b,m}(t) \quad (i \in \mathcal{N}). \tag{9}$$

Based on (7) and (8), there is a clear trade-off between the choice of m and number of RBs allocated to source node i that can contribute to $R_i(t)$, due to the differences in channel

conditions on each RB allocated to the same source node. That is, the higher the MCS m is chosen, the fewer number of RBs can help contribute to $R_i(t)$. In Fig. 2, we use an example to show the trade-off. For a source node, suppose there are 4 RBs, and the channel conditions on the 4 RBs are respectively 2, 4, 3 and 4. If we choose MCS 4 for transmission, RB 2 and 4 can contribute to $R_i(t)$ and the total data rate is $2 \times 4 = 8$. If we choose MCS 3 for transmission, RB 2, 3 and 4 can contribute to $R_i(t)$ and the total data rate is $3 \times 3 = 9$. Here when we want to maximize the data rate for the source node, we should choose MCS 3. From the example we can see that judicious choice of MCS is necessary to balance the achievable bit rates from each RB and the number of RBs that can actually contribute to achievable bit rates.

Figure 2: An example for MCS choosing

C. Problem Statement

In this paper, we want to design a scheduling algorithm to minimize the long term average AoI at the BS, i.e., $\bar{A}^{\mathrm{B}}$. The scheduling algorithm needs to allocate $|\mathcal{B}|$ RBs to $|\mathcal{N}|$ source nodes, and choose the MCS for each source node in each TTI. That is, to determine the decision variables $x_i^b(t)$ and $y_i^m(t)$ for each TTI so that $\bar{A}^{\mathrm{B}}$ is minimized.

There are a number of challenges associated with this problem. First, the search space of the scheduling problem is enormous. Within each TTI, the BS needs to allocate $|\mathcal{B}|$ RBs (e.g., 100) among $|\mathcal{N}|$ source nodes (e.g., 100), and assign each source node an optimal MCS (among 29 possible levels). The solution space consists of $|\mathcal{N}|^{|\mathcal{B}|} \cdot |\mathcal{M}|^{|\mathcal{N}|}$ possibilities. None of the existing AoI research (see [3]) has studied problems of such size and complexity. Second, this is an online algorithm. A scheduler can only makes a scheduling decision for the next TTI and does not have any knowledge of channel conditions for future TTIs. Since we are minimizing a long term average AoI, it is not possible for a schedule to make an optimal decision without knowledge of the future. Therefore, we can only design a near-optimal scheduler at best. Finally, the timing requirement of our scheduling solution is critical. Our scheduler must make its scheduling decision in each TTI, which is typically in sub-millisecond time scale under 5G [5]. It is extremely challenging to find a near-optimal solution for problem of such size and complexity in such a time scale. To date, none of existing AoI research has considered such timing requirement in the design of a scheduling solution (see the webpage [3]).

IV. Performance Bound

Given that it is impossible to find an optimal online scheduling algorithm, it is therefore important to develop a lower bound for the objective $\bar{A}^{\mathrm{B}}$. This lower bound (if tight) can be used as a benchmark to measure the performance of a scheduling algorithm that we will design later in Section V.

In this section, we develop a novel computational procedure that can be used to find a tight lower bound for $\bar{A}^{\mathrm{B}}$. Denote $\overline{R}_i$ as the long term average data rate for source node $i \in \mathcal{N}$, i.e.,

$$\overline{R}_i = \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} R_i(t). \tag{10}$$

In Section IV-A, we develop a lower bound for $\bar{A}^{\mathrm{B}}$ under all possible scheduling algorithms that offer the same $\overline{R}_i$ (for all $i \in \mathcal{N}$), where we assume $\overline{R}_i$'s are given *a priori*. In Section IV-B, we remove this assumption (i.e., knowledge of $\overline{R}_i$'s) by offering a computational procedure and find a lower bound for $\bar{A}^{\mathrm{B}}$.

A. A Lower Bound Assuming Explicit Knowledge of $\overline{R}_i$

When $\overline{R}_i$'s are given, there may exist multiple different scheduling algorithms. We consider the following question: Under the given $\overline{R}_i$'s, how do we find a scheduling algorithm with the minimum $\bar{A}^{\mathrm{B}}$?

Note that in (5), $\bar{A}^{\mathrm{B}}$ is a weighted sum of all $\bar{A}_i^{\mathrm{B}}$'s. A lower bound of $\bar{A}^{\mathrm{B}}$ under the given $\overline{R}_i$'s can be found by the following relaxation. If we can find a lower bound for each $\bar{A}_i^{\mathrm{B}}$ (for $i \in \mathcal{N}$) under the given $\overline{R}_i$ independent of the other $\bar{A}_j^{\mathrm{B}}$'s or $\overline{R}_j$'s (for $j \neq i$), then we can use the weight of each $\bar{A}_i^{\mathrm{B}}$ to sum them up and this is clearly a lower bound of $\bar{A}^{\mathrm{B}}$.

We now show how to find a lower bound for $\bar{A}_i^{\mathrm{B}}$ under a given $\overline{R}_i$ independent of other $\overline{R}_j$ (for $j \neq i$). For ease of exposition, denote p_i as the fraction (in percentage) of successfully transmitted samples over all generated samples in the long term. Clearly, $p_i \leq 1$. With p_i, we can rewrite $\overline{R}_i$ as:

$$\overline{R}_i = \frac{p_i L_i}{T_i}. \tag{11}$$

Note that $\overline{R}_i$ is proportional to p_i via a constant factor. Therefore, minimizing $\bar{A}_i^{\mathrm{B}}$ under a given $\overline{R}_i$ is equivalent to minimizing $\bar{A}_i^{\mathrm{B}}$ under a given p_i.

Since we want to find a lower bound for $\bar{A}_i^{\mathrm{B}}$ under a given p_i, let us consider the following fictitious scenario. Instead of updating $A_i^{\mathrm{B}}(t)$ at the end of TTI $e_i(k)$, let's make a fictitious update at the end of TTI $b_i(k)$. Clearly, $\bar{A}_i^{\mathrm{B}}$ at the BS under such a fictitious updating mechanism is a smaller than that when the update is made at the end of TTI $e_i(k)$ (since the update is performed earlier than it should be). Therefore, we will use $\bar{A}_i^{\mathrm{B}}$ obtained under such a fictitious update mechanism as a lower bound.

Ideally, to minimize this new lower bound of $\bar{A}_i^{\mathrm{B}}$ under a given p_i, we would like to have each of those samples that are eventually transmitted be transmitted *immediately* after they

are generated. Clearly, such a hypothesized (ideal) scheduler would offer a new lower bound for $\bar{A}_i^{\mathrm{B}}$ under a given p_i.

Denote $T_i^U(k)$ as the k-th *update interval* between the k-th and $(k+1)$-th samples that are transmitted by source node i to the BS, i.e.,

$$T_i^U(k) = b_i(k+1) - b_i(k). \tag{12}$$

Then, under a hypothesized scheduler, $T_i^U(k)$ is an integral number of the sampling period T_i. Clearly, such a hypothesized scheduler is not unique and many (each with different behavior of $T_i^U(k)$'s) may offer the same p_i. Among this group of hypothesized schedulers, we want to identify a scheduler that minimizes $\bar{A}_i^{\mathrm{B}}$. The following lemma identifies such a scheduler.

Lemma 1 *Within the class of hypothesized schedulers that provide a lower bound for $\bar{A}_i^{B}$ for a given p_i, define h_i as:*

$$h_i = \lfloor \frac{1}{p_i} \rfloor, \tag{13}$$

where $\lfloor \cdot \rfloor$ is the floor function. Denote AUS as the almost-uniform scheduler that performs updates with:

$$T_i^U(k) = \begin{cases} h_i T_i \text{ with a percentage of } (h_i + 1 - \frac{1}{p_i}), \\ (h_i+1)T_i \text{ with a percentage of } (\frac{1}{p_i} - h_i). \end{cases}$$

Then AUS minimizes $\bar{A}_i^{B}$ within the given class of hypothesized schedulers and

$$\bar{A}_i^{B} = \frac{T_i}{2} f(p_i) + \frac{1}{2}, \tag{14}$$

where

$$f(p_i) = 2\lfloor \frac{1}{p_i} \rfloor + 1 - (\lfloor \frac{1}{p_i} \rfloor^2 + \lfloor \frac{1}{p_i} \rfloor) \cdot p_i. \tag{15}$$

This lemma says the hypothesized scheduler that employs almost uniform (or exactly uniform in the case when $1/p_i$ is an integer) update interval minimizes $\bar{A}_i^{\mathrm{B}}$. This result is very intuitive. It can be shown (as in the proof sketch below) that for any other scheduler with a larger variance in $T_i^U(k)$, we can always find a scheduler with a smaller variance in $T_i^U(k)$ that reduces $\bar{A}_i^{\mathrm{B}}$.

A Sketch of Proof For any scheduler, if there are two update intervals with length $n_1 T_i$ and $n_2 T_i$ such that $n_1 \geq n_2 + 2$, then we can construct a new hypothesized scheduler by changing the intervals to $(n_1-1)T_i$ and $(n_2+1)T_i$, and its $\bar{A}_i^{\mathrm{B}}$ is smaller than the original one. By repeatedly doing so we can construct a new hypothesized scheduler, a.k.a. AUS, where the length difference between any two update intervals isn't greater than T_i. In other words, there exists an integer h_i such that for all k, $T_i^U(k)$ is either $h_i T_i$ or $(h_i+1)T_i$.

Considering the fact that the sample rate is p_i, we have $h_i = \lfloor \frac{1}{p_i} \rfloor$. Denote a as the percentage of those intervals with length nT_i, then $(1-a)$ is the percentage of those intervals with length $(n+1)T_i$. During a long time interval $T \to \infty$, the sample rate is p_i, and the occurrence rates (the number of occurrences over the number of TTIs in long term) of those two kinds of interval are respectively $\frac{p_i a}{T_i}$ and $\frac{p_i(1-a)}{T_i}$. We have

$$\lim_{T\to\infty} T\frac{p_i a}{T_i} h_i T_i + T\frac{p_i(1-a)}{T_i}(h_i+1)T_i = T.$$

That means

$$h_i a + (h_i+1)(1-a) = \frac{1}{p_i}. \tag{16}$$

Then we have

$$a = h_i + 1 - \frac{1}{p_i}. \tag{17}$$

Then we can calculate $\bar{A}_i^{\mathrm{B}}$ under AUS:

$$\begin{aligned} \bar{A}_i^{\mathrm{B}} &= \frac{a h_i T_i (1 + h_i T_i) + (1-a)(h_i+1)T_i(1 + (h_i+1)T_i)}{2 \cdot (a h_i T_i + (1-a)(h_i+1)T_i)} \\ &= \frac{T_i}{2}(2h_i + 1 - (h_i^2 + h_i)p_i) + \frac{1}{2} \\ &= \frac{T_i}{2} f(p_i) + \frac{1}{2}. \end{aligned}$$

∎

Combining (5), (11) and (14), we have the following lower bound for $\bar{A}^{\mathrm{B}}$ as a functions of $\overline{R}_i$:

$$\bar{A}^{\mathrm{B}} \geq \sum_{i\in\mathcal{N}} w_i \cdot (\frac{T_i}{2} f(\frac{\overline{R}_i T_i}{L_i}) + \frac{1}{2}). \tag{18}$$

In the next subsection, we will remove the assumption of prior knowledge of $\overline{R}_i$.

B. Finding A Lower Bound of $\bar{A}^{B}$

Based on (18), a lower bound of $\bar{A}^{\mathrm{B}}$ can be found by minimizing the RHS of (18), i.e.,

$$\begin{aligned} \min_{x_i^b(t), y_i^m(t)} \quad & \sum_{i\in\mathcal{N}} w_i (\frac{T_i}{2} f(\frac{\overline{R}_i T_i}{L_i}) + \frac{1}{2}) \\ \text{s.t.} \quad & \text{Constraints (6), (7), (8), (9), (10).} \end{aligned} \tag{19}$$

For ease of exposition, we define

$$J_i(\overline{R}_i) = w_i (\frac{T_i}{2} f(\frac{\overline{R}_i T_i}{L_i}) + \frac{1}{2}). \tag{20}$$

Then (19) becomes:

$$\begin{aligned} \min_{x_i^b(t), y_i^m(t)} \quad & \sum_{i\in\mathcal{N}} J_i(\overline{R}_i) \\ \text{s.t.} \quad & \text{Constraints (6), (7), (8), (9), (10).} \end{aligned} \tag{21}$$

We will design an optimal scheduling algorithm to problem (21). Then we can substitute the optimal result for $\overline{R}_i$ into (19) and obtain the lower bound.

Problem (21) is a scheduling problem to minimize a function of $\overline{R}_i$. Similar problems have been studied in the information theory community (see, e.g., [15], [16]), where it has been shown that a gradient scheduling algorithm can achieve the same optimal objective value asymptotically (when the number of TTIs goes to infinity). Specifically, in a gradient scheduling algorithm, we define an empirical data rate $R_i^e(t)$ for each TTI t and it is updated as a moving average as follows:

$$R_i^e(t+1) = (1-\beta)R_i^e(t) + \beta R_i(t), \tag{22}$$

where β is a small positive constant (e.g., 0.01) and $R_i(t)$ is the instant data rate at TTI t. It can be easily shown that under the moving average updating algorithm in (22), when $\beta \to 0$,

$$\lim_{t\to\infty} R_i^e(t) = \overline{R}_i \tag{23}$$

That is, $R_i^e(t)$ asymptotically approaches $\overline{R}_i$ when $t \to \infty$. In practice, t does not need to be very large to achieve this approximation. In our simulation results, we find that $t = 500$ is sufficient to achieve a good approximation.

Based on (23), (21) becomes

$$\begin{aligned} \min_{x_i^b(t), y_i^m(t)} \quad & \lim_{t\to\infty} \sum_{i\in\mathcal{N}} J_i\big(R_i^e(t)\big) \\ \text{s.t.} \quad & \text{Constraints (6), (7), (8), (9), (22).} \end{aligned} \tag{24}$$

The idea of the gradient scheduling algorithm is to minimize $\sum_{i\in\mathcal{N}} J_i\big(R_i^e(t+1)\big)$ at every t. It can be shown that by performing such minimization for every TTI, $\lim_{t\to\infty} \sum_{i\in\mathcal{N}} J_i\big(R_i^e(t)\big)$ is also minimized when $\beta \to 0$ [15], [16].

We now show how to minimize $\sum_{i\in\mathcal{N}} J_i\big(R_i^e(t+1)\big)$ at TTI t. When $\beta \to 0$, considering (22), we have

$$\begin{aligned} &\sum_{i\in\mathcal{N}} J_i\big(R_i^e(t+1)\big) = \sum_{i\in\mathcal{N}} J_i\big((1-\beta)R_i^e(t) + \beta R_i(t)\big) \\ &= \sum_{i\in\mathcal{N}} J_i\Big(R_i^e(t) + \beta\big(R_i(t) - R_i^e(t)\big)\Big) \\ &= \sum_{i\in\mathcal{N}} J_i\big(R_i^e(t)\big) + \sum_{i\in\mathcal{N}} \frac{dJ_i(R)}{dR}\Big|_{R=R_i^e(t)} \beta\big(R_i(t) - R_i^e(t)\big), \end{aligned} \tag{25}$$

where the last equality follows from the definition of derivative for $J_i(\cdot)$. Since $R_i^e(t)$ can be computed at TTI t, $J(R_i^e(t))$ and $\frac{dJ_i(R)}{dR}\Big|_{R=R_i^e(t)} R_i^e(t)$ can also be computed. Therefore, to minimize $\sum_{i\in\mathcal{N}} J_i(R_i^e(t+1))$ at TTI t, we only need to solve the following problem.

$$\begin{aligned} \min_{x_i^b(t), y_i^m(t)} \quad & \sum_{i\in\mathcal{N}} \frac{dJ_i(R)}{dR}\Big|_{R=R_i^e(t)} R_i(t) \\ \text{s.t.} \quad & \text{Constraints (6), (7), (8), (9), (27),} \end{aligned} \tag{26}$$

where the derivative of $J_i(\cdot)$ is computed as

$$\begin{aligned} \frac{dJ_i(R)}{dR}\Big|_{R=R_i^e(t)} &= \frac{w_i T_i}{2} \frac{df(\frac{RT_i}{L_i})}{dR}\Big|_{R=R_i^e(t)} \\ &= \frac{w_i T_i^2}{2L_i}\big(\lfloor \frac{L_i}{R_i^e(t)T_i} \rfloor^2 + \lfloor \frac{L_i}{R_i^e(t)T_i} \rfloor\big). \end{aligned} \tag{27}$$

To ensure $f(\cdot)$ is continuously differentiable at every point, we need to define how to perform derivative at certain points. Recall $f(\cdot)$ is a piecewise linear function. When $\frac{L_i}{R_i^e(t)T_i}$ is exactly an integer, the function $f(\frac{RT_i}{L_i})$ is continuous but not differentiable at $R = R_i^e(t)$ (i.e., the left derivative doesn't equal to the right derivative). For these points, we use the left derivative as the derivative at $R = R_i^e(t)$, as shown in the RHS of (27).

Since each term in the RHS of (27) is either a constant or a known value at TTI t, let's denote the RHS of (27) as $W_i(t)$. Using (9), the minimization problem (26) can be written as

$$\begin{aligned} \min_{x_i^b(t), y_i^m(t)} \quad & \sum_{i\in\mathcal{N}} \sum_{b\in\mathcal{B}} \sum_{m\in\mathcal{M}} W_i(t) r_i^{b,m}(t) x_i^b(t) y_i^m(t) \\ \text{s.t.} \quad & \text{Constraints (6), (7), (8).} \end{aligned} \tag{28}$$

The optimization problem (28) is an integer quadratic programming (IQP) problem, which can be solved by commercial optimizers such as the IBM CPLEX [17]. Solving the problem in CPLEX is time-costing and will solely be used for offline benchmark purpose.

Recall that after solving the optimization problem (28) for sufficient number of TTIs (e.g., 500 TTIs in our simulations), we let $\overline{R}_i = R_i^e(t)$ and substitute $\overline{R}_i$ into (19) to get the lower bound.

V. Kronos: Design and Implementation of a Real-Time Scheduler

In this section, we develop a scheduling algorithm, code named Kronos[1], to achieve near-optimal performance for the AoI scheduling problem in real time.

A. Basic Idea

The design of Kronos is based on the following key ideas.

1) For the objective function in (5), it is obvious that we need to minimize $\bar{A}_i^{\mathrm{B}}$ from each source node $i \in \mathcal{N}$. For A_i^{B}, its value at the BS is not reduced until a new sample is received by the BS in its entirety. That is, a partially transmitted sample will not reduce (update) A_i^{B} at the BS. Based on this observation, we should minimize the number of partially (incomplete) transmission of samples at the end of each TTI. As an extreme, we can limit the number of samples that are partially (incompletely) transmitted at the end of a TTI to no more than one. This can be done by devoting all the remaining RBs to one sample, rather than spreading out to multiple samples.
2) Following the last idea, at the beginning of a new (the next) TTI, we can inherit at most one partially (incomplete) transmission of a sample from the previous TTI. Recall that we cannot preempt a sample once it starts transmission, even if there is a newly generated sample from the same source node. Further, for our IoT applications, a sample size is relatively small. So the remaining portion of the partially transmitted sample is not large (in most cases) and it makes sense to complete its transmission before starting to transmit any other samples.
3) After we complete transmission of the remaining (incomplete) sample (carried from the last TTI), we need to decide which sample to transmit next in the current TTI. To do this, we need a metric to compare among the samples from different source nodes and decide which sub-set of samples that we will allocate the remaining

[1]Kronos is the god of time in Greek mythology.

RBs. Clearly, this metric should consist of the weight and the "outage" (difference between AoI at the BS and the source, i.e., $A_i^{\mathrm{B}}(t) - A_i^{\mathrm{s}}(t)$) for each source node $i \in \mathcal{N}$. In our previous work [18], in the absence of considering channel conditions, we use the metric $w_i\Delta_i^2(t)$ for scheduling, where $\Delta_i(t)$ is defined as

$$\Delta_i(t) = A_i^{\mathrm{B}}(t) - A_i^{\mathrm{s}}(t). \tag{29}$$

It was shown in [18] that a scheduler based on this metric can offer near-optimal performance (under simplified channel conditions). Therefore, it would be wise to have Kronos to inherit this basic trait before we add additional features to cope with dynamic channel conditions.

4) To incorporate channel conditions into the scheduling decision metric, we must consider the impact of MCS setting on RBs. As shown in the example in Fig. 2, the higher the MCS m is chosen, the fewer number of RBs (with a higher rate) can be used for transmission. Intuitively, we prefer to use as few RBs as possible to transmit a sample. Therefore, the scheduling metric should also include the number of RBs required to transmit a sample, i.e., the more RBs required, the lower the priority (or smaller the metric) associated with a source node. We will elaborate the details of how to incorporating channel conditions into the scheduling metric in the next section.
5) With the scheduling metric (see next section), we can compare samples and perform scheduling, i.e. RB allocation. Clearly, RB allocation is an iterative process, where in each iteration, we will consider how to allocate a subset of RBs among the *remaining* unallocated RBs to a sample in the *remaining* unscheduled samples. Eventually (after a number of iterations), all RBs are allocated and the algorithm terminates.

B. Algorithm Details: Incorporating Channel Condition in Scheduling Metric

We devote this section to the discussion of how channel conditions are incorporated into the scheduling metric, which is the heart of our design.

Recall that the choice of MCS value m at a source node will set the corresponding coding rate c^m, which will in turn determine two parameters:

- the set of RBs in the remaining un-allocated RBs that can contribute at this bit rate c^m. We denote the number in this set as $n_i^m(t)$.
- the number of RBs that is needed to transmit a sample for source node i, which we denote as s_i^m.

That is,

$$n_i^m(t) = \sum_{\text{un-allocated } b} [q_i^b(t) \geq m], \tag{30}$$

where $q_i^b(t)$ (see Section III) is the maximum MCS that can be used for RB b and source node i for transmission (which is determined by the channel condition on RB b), and "$[\cdot]$" is the notation for Iverson bracket, returning 1 if the inside statement is true and 0 otherwise [19]. And we have

$$s_i^m = \lceil \frac{L_i}{c^m} \rceil, \tag{31}$$

where "$\lceil \cdot \rceil$" is the ceiling function.

Clearly, the scheduling metric for a sample is dependent on m and is a function of $n_i^m(t)$, and s_i^m, in addition to $w_i\Delta^2(t)$ (as discussed in the last section). As a start, denote $V_i^m(t)$ as the scheduling metric under MCS m with the following general form:

$$V_i^m(t) = g(w_i\Delta_i^2(t), n_i^m(t), s_i^m), \tag{32}$$

where "g" is a function of $w_i\Delta^2(t)$, $n_i^m(t)$ and s_i^m.

For each sample from source node $i \in \mathcal{N}$, we have the pair $(n_i^m(t), s_i^m)$ for each $m \in \mathcal{M}$. If $n_i^m(t) \geq s_i^m$, it means that this sample can possibly be transmitted in its entirety in this TTI. Otherwise (i.e., $n_i^m(t) < s_i^m$), this sample can only be partially transmitted even if we allocate all the remaining RBs to it. Now we have a dilemma: shall we transmit a partial sample (while holding back one or more other samples that can otherwise be transmitted in their entirety) or shall we transmit one or more complete samples first?

Since our goal is to minimize (5), based on the the *shortest-job-first* principle in queuing theory [20], we should first schedule one or more samples that can be fully transmitted. Therefore, we purposely design the function $g(w_i\Delta_i^2(t), n_i^m(t), s_i^m) > 0$ when $n_i^m(t) \geq s_i^m$ and $g(w_i\Delta_i^2(t), n_i^m(t), s_i^m) < 0$ when $n_i^m(t) < s_i^m$. Under such definition, the priority for a sample that can be fully transmitted within this TTI is always higher than that for a sample that can only be transmitted partially. After RBs have been allocated to those samples that can be fully transmitted, we move on to consider how to allocate the remaining RBs to those samples that cannot be fully transmitted (i.e., samples with $V_i^m(t) < 0$). Recall that in each TTI we only schedule at most one partially transmitted sample. So when $V_i^m(t) < 0$ for all remaining source nodes i and MCS m, we will choose one with the largest value of $V_i^m(t) < 0$ for transmission.

Based on the above discussion, we now show how to design function $g(w_i\Delta_i^2(t), n_i^m(t), s_i^m)$ as follows.

- When $n_i^m(t) \geq s_i^m$, sample i can be fully transmitted with un-allocated RBs under MCS m in this TTI. In this case, the fewer RBs required for transmission (i.e., s_i^m), the higher the priority it should have. Therefore, we define function g as

$$g(w_i\Delta_i^2(t), n_i^m(t), s_i^m) = w_i\Delta_i^2(t) \cdot \frac{1}{s_i^m}. \tag{33}$$

- When $n_i^m(t) < s_i^m$, sample i cannot be fully transmitted under MCS m. In this case, the greater the fraction of the sample that can be transmitted (i.e., $n_i^m(t)/s_i^m$), the higher the priority it should have. Based on this idea, the function g should be proportional to the term $n_i^m(t)/s_i^m$. On the other hand, as discussed earlier, $g(w_i\Delta_i^2(t), n_i^m(t), s_i^m)$ should be negative when

$n_i^m(t) < s_i^m$. To ensure this is the case, we can add a negative offset constant and define function g as

$$g(w_i\Delta_i^2(t), n_i^m(t), s_i^m) = w_i\Delta_i^2(t) \cdot \frac{n_i^m(t)}{s_i^m} - C, \quad (34)$$

where C is a large (offset) constant that can ensure $g(n_i^m(t), s_i^m) < 0$ for all i and m when $n_i^m(t) < s_i^m$. For example, we can set $C = \sum_{i\in\mathcal{N}} w_i\Delta_i^2(t)$ or $C = \max_{i\in\mathcal{N}} w_i\Delta_i^2(t)$.

Combining (32), (33) and (34), the scheduling metric $V_i^m(t)$ is given as

$$V_i^m(t) = \begin{cases} \frac{w_i\Delta_i^2(t)}{s_i^m}, & \text{if } n_i^m(t) \geq s_i^m, \\ \frac{w_i\Delta_i^2(t)n_i^m(t)}{s_i^m} - C, & \text{otherwise.} \end{cases} \quad (35)$$

In summary, within each TTI, Kronos first allocates RBs to complete transmission of the incomplete sample from the last TTI. Then Kronos selects samples (one at a time) iteratively for RB allocation. In each iteration, Kronos finds $n_i^m(t)$ by (30) and computes $V_i^m(t)$ for all $i \in \mathcal{N}$ and $m \in \mathcal{M}$ by (35). Then Kronos chooses i^* and m^* with the largest $V_{i^*}^{m^*}(t)$. If $V_{i^*}^{m^*}(t) > 0$, Kronos allocated RBs for the entire sample transmission with MCS m^* and then moves on to the next iteration; if $V_{i^*}^{m^*}(t) < 0$, Kronos allocated all remaining RBs for the sample with MCS m^* and terminates afterwards.

We now discuss the complexity of Kronos. To allocate RBs to complete transmission of the incomplete sample from the last TTI, the time complexity is $O(|\mathcal{B}||\mathcal{M}|)$. After that, if Kronos is implemented sequentially, then in each iteration, Kronos needs to compute $|\mathcal{N}||\mathcal{M}|$ different $V_i^m(t)$'s, which has a time complexity $O(|\mathcal{N}||\mathcal{M}||\mathcal{B}|)$. After that, Kronos selects i^* and m^* with the largest $V_{i^*}^{m^*}(t)$, which has a time complexity $O(|\mathcal{N}||\mathcal{M}|)$. Then Kronos allocates RBs to the selected source node i^*, which has a time complexity $O(|\mathcal{B}|)$. Therefore, the time complexity for each iteration is $O(|\mathcal{N}||\mathcal{M}||\mathcal{B}|) + O(|\mathcal{N}||\mathcal{M}|) + O(|\mathcal{B}|) = O(|\mathcal{N}||\mathcal{M}||\mathcal{B}|)$. Since there are at most $|\mathcal{N}|$ iterations in each TTI, the time complexity for scheduling new samples is $O(|\mathcal{N}|^2|\mathcal{M}||\mathcal{B}|)$. Thus, the total time complexity in each TTI is $O(|\mathcal{B}||\mathcal{M}|) + O(|\mathcal{N}|^2|\mathcal{M}||\mathcal{B}|) = O(|\mathcal{N}|^2|\mathcal{M}||\mathcal{B}|)$.

In one of our simulations in Section VI, we find that when $|\mathcal{N}| = 100$, $|\mathcal{B}| = 100$ and $|\mathcal{M}| = 29$, the average running time for sequential Kronos is about $\sim$10 ms, which can't meet the 5G timing requirement (sub-millisecond time scale). In the next section, we will incorporate parallel computation into Kronos implementation to speed up its timing performance.

C. Algorithm Speedup: A GPU-based Implementation

We observe that in each iteration of Kronos, the computation of $V_i^m(t)$'s for each i and m is independent from each other. This hints that we could compute them in parallel rather than in sequence. A low-cost, off-the-shelf solution to compute $V_i^m(t)$'s in parallel is to employ GPU. Today's commercial GPUs typically consist of a large number (1000s) of processing cores and are highly optimized for massive parallel computation. However, unlike a CPU core, each GPU core processor has very limited computational capability and is designed to handle very simple computations (and thus has low cost). To best utilize a GPU's capability, it is utmost important to ensure that each sub-problem handled by a GPU core processing is of extremely low complexity and requires very few iterations to find a solution. To calculate $V_i^m(t)$'s for all i's and m's in an iteration, we can decompose this problem into $|\mathcal{N}||\mathcal{M}|$ independent sub-problems, each of which is to calculate $V_i^m(t)$ under a specific value of i and m. Recall that computational complexity of this sub-problem is $O(|\mathcal{B}|)$, which can be done quickly by GPU cores.

In our implementation, we employ an off-the-shelf Nvidia Quadro P6000 GPU and the CUDA programming platform. This GPU consists of 30 streaming multi-processors (SMs), with each SM consisting of 128 small processing cores (CUDA cores). These cores are capable of performing concurrent computation tasks involving arithmetic and logic operations. Under CUDA, the sub-tasks to compute $V_i^m(t)$'s are handled by a grid of thread blocks, each with a certain number of threads. We limit each SM to handle at most one thread block to avoid sequential execution of thread blocks on the same SM. Specifically, with $|\mathcal{M}| = 29$ (under 5G standard [5]), we use 29 SMs and assign each SM to a specific value m, each with $|\mathcal{N}|$ sub-tasks. Then within each SM, $|\mathcal{N}|$ sub-tasks are being solved in parallel threads. Since we are not able to synchronize different thread blocks among the SMs within the GPU under CUDA, we rely on the CPU to perform synchronization after all $V_i^m(t)$'s have been computed.

After computing $V_i^m(t)$'s, we need to choose i^* and m^* corresponding to the largest $V_{i^*}^{m^*}(t)$. We can use parallel reduction [21] to reduce the complexity. For example, when $|\mathcal{N}||\mathcal{M}|$ is a power of 2, we can construct an elimination tournament, where only a half of $V_i^m(t)$'s survive after each round. After $\log_2(|\mathcal{N}||\mathcal{M}|)$ rounds, the champion (with the largest $V_i^m(t)$ among all $|\mathcal{N}||\mathcal{M}|$ $V_i^m(t)$'s) will be found. This parallel reduction procedure significantly reduces the time complexity (compared with sequentially search) from $|\mathcal{N}||\mathcal{M}|$ to $\log_2(|\mathcal{N}||\mathcal{M}|)$. When $|\mathcal{N}||\mathcal{M}|$ is not a power of 2, we can add fictitious elements in the beginning to increase the number of elements to a power of 2 and then use parallel reduction to find the champion.

With GPU implementation, in each iteration, the computation complexity of $V_i^m(t)$'s is $O(|\mathcal{B}|)$. The complexity of choosing the largest $V_i^m(t)$ (with parallel reduction) is $O\big(\log(|\mathcal{N}||\mathcal{M}|)\big)$. The computation complexity of RB allocating remains $O(|\mathcal{B}|)$. Therefore, the time complexity of each iteration is $O(|\mathcal{B}|) + O\big(\log(|\mathcal{N}||\mathcal{M}|)\big)$. Recall there are at most $|\mathcal{N}|$ iterations and the complexity of completing the unfinished sample from the last TTI is $O(|\mathcal{B}||\mathcal{M}|)$. Then the total computation complexity in each TTI is $O(|\mathcal{B}||\mathcal{N}|) + O\big(\log(|\mathcal{N}||\mathcal{M}|)\big)\times|\mathcal{N}|+O(|\mathcal{B}||\mathcal{M}|) = O\big(|\mathcal{B}|\cdot(|\mathcal{N}|+|\mathcal{M}|)\big)+ O\big(|\mathcal{N}|\cdot(\log|\mathcal{N}|+\log|\mathcal{M}|)\big)$. In one of our simulations in Section VI, we find that when $|\mathcal{N}| = 100$, $|\mathcal{B}| = 100$ and $|\mathcal{M}| = 29$, the average running time for Kronos implemented with GPU is about $\sim$0.4 ms, which decreases one order of magnitude compared with implementation without GPU.

Table I: Simulation Parameters

Type	w_i	L_i (bits)	T_i (TTIs)	Expected power (MCS)
1	8	5400	2	26
2	2	7200	5	28
3	10	6800	3	24
4	6	6200	6	23
5	5	7600	1	20
6	2	8200	11	22
7	9	6000	4	25
8	1	7100	5	18
9	4	9600	6	24
10	3	8400	3	21

VI. Performance Evaluation

The objective of this section is twofold. First, we will evaluate Kronos in terms of its ability to achieve our objective function. The primary benchmark for this purpose is the lower bound that we developed in Section IV. Second, we will examine the timing performance of Kronos and see if it can meet the real time requirement under 5G.

A. Network Setting

We assume there are 10 different types of IoT source nodes. The weight, sample size, and sampling period for each type of nodes are given in Table I. We consider 100 source nodes (10 from each type), i.e., $|\mathcal{N}| = 100$. For ease of presentation, we normalize the weight of each source node w.r.t. $\sum_{i\in\mathcal{N}} w_i$. We assume the uplink transmission consists of 100 RBs, i.e., $|\mathcal{B}| = 100$.

Although any setting of channel condition for each source node can be used, for ease of reproducibility, we pre-assign an expected channel condition (in terms of the corresponding MCS of the expected power) for each type of source nodes, as shown in Table I. Note that the pre-assigned channel condition is statistical and in each TTI the channel is randomly generated (i.e., time-varying channel).

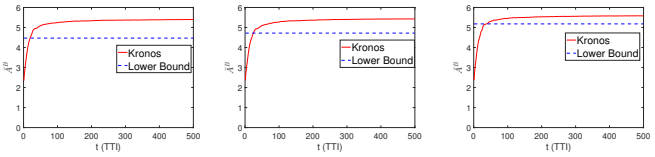
For each MCS m, we get the corresponding modulation and coding rate c^m from [5] (Table 5.1.3.1-1).

In each network setting, we run simulations for Kronos over 500 TTIs and then calculate the average AoI $\bar{A}^{\mathrm{B}}$. For initialization, $A_i^{\mathrm{s}}(0)$ for each i is set to a random number. For comparison, we simulate Kronos algorithm with and without GPU implementation. Our GPU implementation is done on a Dell Precision Tower 7910 with an Intel Xeon E5-2687W v4 CPU (3.0 GHz) and an Nvidia Quadro P6000 GPU. All the programmings are done in Microsoft Visual Studio 2017. We use CUDA 10.0 to program Kronos in our GPU.

B. Results

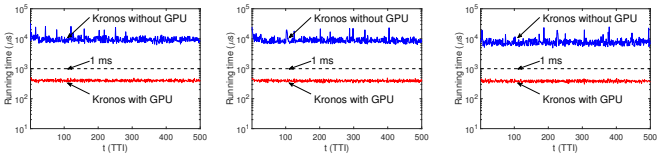
Varying Channel Propagation. We first evaluate Kronos under channels with different LOS signal strength. We assume Rician fading channel with no frequency nor time correlation.

Fig. 3 shows the evolution of $\bar{A}^{\mathrm{B}}$ under Kronos across 500 TTIs for different Rician factor K. In terms of objective value, there is no difference in Kronos implementation with and without GPU. Also shown in each sub-figure is the lower bound by the gradient scheduling algorithm (with $\beta = 0.01$). Clearly, we see Kronos can achieve near-optimal performance. In particular, when Rician factor $K = 0$ (i.e., Rayleigh fading),



(a) $K = 0$ (b) $K = 2$ (c) $K = 10$

Figure 3: $\bar{A}^{\mathrm{B}}$ under different Rician factors.



(a) $K = 0$ (b) $K = 2$ (c) $K = 10$

Figure 4: Running time under different Rician factors.

2 and 10, the objectives of Kronos are 20.8%, 15.0% and 7.7% within their respective lower bounds.

Fig. 4 shows the running time for Kronos in each TTI with and without GPU implementation. As a benchmark, we also show the 5G timing requirement for numerology 0 (1 ms) in each sub-figure. We can see under all K, the running time of Kronos falls below 1 ms when it is implemented with GPU. When it is implemented only with CPU, it is about $\sim$10 ms. In particular, when $K = 0$, 2 and 10, the average running times of Kronos (with GPU implementation) are 402 μs, 396 μs and 384 μs respectively.

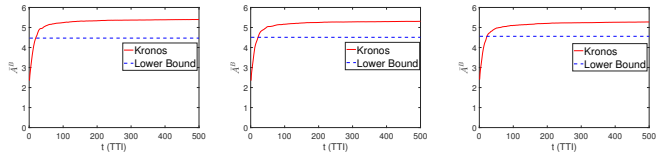
Varying Frequency Correlation. We now evaluate Kronos under channels with different frequency correlation. We assume Rayleigh fading channels with no time correlation, and the coherence bandwidth is B_c, i.e., the channel conditions on adjacent B_c RBs are identical for each source node.

Fig. 5 shows evolution of $\bar{A}^{\mathrm{B}}$ under Kronos across 500 TTIs for different coherence bandwidth B_c. Also shown in each sub-figure is the lower bound by the gradient scheduling algorithm. Clearly, we see Kronos can achieve near-optimal performance. In particular, when $B_c = 1$ (i.e, no frequency correlation), 4 and 10, the objectives of Kronos are respectively 20.8%, 17.7% and 15.8% within their respective lower bounds.

Fig. 6 shows the running time for Kronos in each TTI with and without GPU implementation. We can see under all B_c, the running time of Kronos falls below 1 ms when it is implemented with GPU. When it is implemented only with CPU, it is about $\sim$10 ms. In particular, when $B_c = 1$, 4 and 10, the average running times of Kronos (with GPU implementation) are respectively 402 μs, 392 μs and 377 μs.

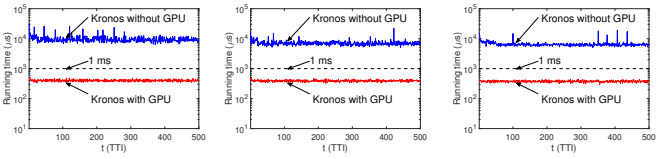
Varying Time Correlation. Finally, we evaluate Kronos under channels with different time correlation. We assume Rayleigh fading channels with no frequency correlation, and the coherence bandwidth is T_c, i.e., the channel conditions on adjacent T_c TTIs are identical for each source node.

Fig. 7 shows evolution of $\bar{A}^{\mathrm{B}}$ under Kronos across 500 TTIs for different coherence bandwidth T_c. Also shown in each sub-figure is the lower bound by the gradient scheduling algorithm. Clearly, we see Kronos can achieve near-optimal performance. In particular, when $T_c = 1$ (i.e, no time correlation), 2 and 5,



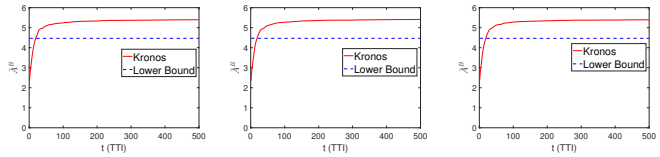
(a) $B_c = 1$ (b) $B_c = 4$ (c) $B_c = 10$

Figure 5: $\bar{A}^{\mathrm{B}}$ under different coherence bandwidth.



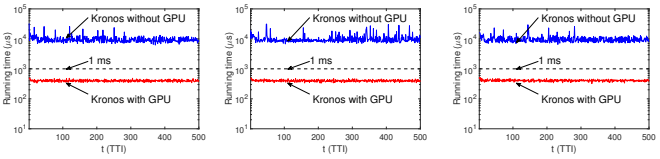
(a) $B_c = 1$ (b) $B_c = 4$ (c) $B_c = 10$

Figure 6: Running time under different coherence bandwidth.



(a) $T_c = 1$ (b) $T_c = 2$ (c) $T_c = 5$

Figure 7: $\bar{A}^{\mathrm{B}}$ under different coherence time.



(a) $T_c = 1$ (b) $T_c = 2$ (c) $T_c = 5$

Figure 8: Running time under different coherence time.

the objectives of Kronos are respectively 20.8%, 20.6% and 21.0% within their respective lower bounds.

Fig. 8 shows the running time for Kronos in each TTI with and without GPU implementation. We can see under all T_c, the running time of Kronos falls below 1 ms when it is implemented with GPU. When it is implemented only with CPU, it is about $\sim$10 ms. In particular, when $T_c = 1$, 2 and 5, the average running times of Kronos (with GPU implementation) are respectively 402 μs, 402 μs and 408 μs.

We also tested the performance of Kronos under many other different settings, including varying $|\mathcal{N}|$, $|\mathcal{B}|$, L_i's and T_i's. Under all settings we tested, Kronos can achieve near optimal AoI performance, and the running time (with GPU implementation) is below 1 ms. Due to paper length limitation we don't show the results.

VII. Conclusions

This paper presents the first successful design of a 5G-compliant real-time AoI scheduler, Kronos, which can cope with dynamic channel conditions. Kronos is capable of performing RB allocation and MCS selection for each source node based on channel conditions within each TTI. To cope with the enormous search space for optimal solution and the unknown nature of channel conditions, we developed a novel computation procedure to find an asymptotic lower bound for the objective as a performance benchmark. We further developed a novel metric that can be used by Kronos to select the next source node to allocate RBs and determine MCS in each iteration. To meet the stringent timing requirement in 5G, we proposed to implement Kronos on a low cost GPU platform by exploiting its massive parallel computing capability. Through extensive simulations and experiments, we show that Kronos can find a near-optimal AoI scheduling solution in sub-millisecond time scale.

Acknowledgments

This research was supported in part by US Army Research Laboratory under Cooperative Agreement No. W911NF1820293. The work of W. Lou and Y.T. Hou was also supported in part by NSF under Grant CNS-1800650.

References

[1] S. Kaul, M. Gruteser, V. Rai, and J. Kenney, "Minimizing Age of Information in Vehicular Networks," in *Proc. IEEE SECON,* 2011.

[2] S. Kaul, R. Yates, and M. Gruteser, "Real-Time Status: How Often Should One Update?" in *Proc. IEEE INFOCOM,* 2012.

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

[4] 3GPP TR 21.101 version 7.0.0, "Physical Layer Aspects for Evolved UTRA."

[5] 3GPP TS 38.214 version 15.0.0, "NR; Physical layer procedures for data."

[6] IEEE P802.11ac. Specification framework for TGac. IEEE 802.11-09/0992r21.

[7] R.D. Yates, P. Ciblat, A. Yener, and M. Wigger, "Age-Optimal Constrained Cache Updating," in *Proc. IEEE ISIT,* 2017.

[8] C. Joo and A. Eryilmaz, "Wireless Scheduling for Information Freshness and Synchrony: Drift-based Design and Heavy-Traffic Analysis," in *Proc. WiOpt,* 2017.

[9] J. Zhong, R.D. Yates, and E. Soljanin, "Two Freshness Metrics for Local Cache Refresh," in *Proc. IEEE ISIT,* 2018.

[10] I. Kadota, A. Sinha, and E. Modiano, "Optimizing Age of Information in Wireless Networks with Throughput Constraints," in *Proc. IEEE INFOCOM,* 2018

[11] R. Talak, S. Karaman, and E. Modiano, "Optimizing Age of Information in Wireless Networks with Perfect Channel State Information," in *Proc. WiOpt,*, 2018.

[12] N. Lu, B. Ji, and B. Li, "Age-based Scheduling: Improving Data Freshness for Wireless Real-Time Traffic," in *Proc. ACM MobiHoc,* 2018.

[13] AT&T Labs & AT&T Foundry, "AT&T Edge Cloud (AEC) - White Paper," available at https://about.att.com/content/dam/innovationdocs/Edge_Compute_White_Paper\%20FINAL2.pdf

[14] Verizon, "The Internet of Things Will Thrive on 5G Technology," available at https://www.verizon.com/about/our-company/5g/internet-things-will-thrive-5g-technology

[15] R. Agrawal and V. Subramanian, "Optimality of Certain Channel Aware Scheduling Policies," in *Proc. Allerton,* 2002.

[16] A.L. Stolyar, "On the Asymptotic Optimality of the Gradient Scheduling Algorithm for Multiuser Throughput Allocation," *Operations Research,* 2005.

[17] IBM ILOG CPLEX Optimizer, available at https://www.ibm.com/analytics/cplex-optimizer

[18] C. Li, S. Li, Y.T. Hou, "A General Model for Minimizing Age of Information at Network Edge," in *Proc. IEEE INFOCOM,* 2019.

[19] R.L. Garham, D.E. Knuth, and O. Patashnik, "Concrete Mathematics," *Addison-Wesley,* 1989.

[20] A.S. Tanenbaum and A.S. Woodhaul, *Operating systems: design and implementation,* Vol. 2. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[21] J. Sanders and E. Kandrot, "CUDA by Example: An Introduction to General-Purpose GPU Programming," *Addison-Wesley Professional*, 2010.