Timely Processing Of Updates From Multiple Sources

Vishakha Ramani, Ivan Seskar, Roy D. Yates
WINLAB, Rutgers University
Email: {vishakha, seskar, ryates}@winlab.rutgers.edu

***Abstract*—We consider a system where the updates from independent sources are disseminated via a publish-subscribe mechanism. The sources are the publishers and a decision process (DP), acting as a subscriber, derives decision updates from the source data. We derive the stationary expected age of information (AoI) of decision updates delivered to a monitor. We show that a lazy computation policy in which the DP may sit idle before computing its next decision update can reduce the average AoI at the monitor even though the DP exerts no control over the generation of source updates. This AoI reduction is shown to occur because lazy computation can offset the negative effect of high variance in the computation time.**

I. Introduction

A dense metropolis is a complex traffic environment, and autonomous cars with a plethora of attached sensors still have limited situational awareness. Hence, holistic situational awareness of a cloud-connected vehicle is facilitated by

1) timely collection of sensory inputs from different sources (e.g., other vehicles, pedestrians, and smart city infrastructure sensors) , and
2) employing this sensor data to provide timely feedback or decision updates to participating mobile clients.

For example, cameras at a smart-city intersection [1] can capture video or images of the intersection and send this data to a processing system at the edge. The system can then analyze the data to identify and inform potential public safety risks such as vehicles running red lights or pedestrians crossing the street outside of designated areas.

Although, it is possible that the system delivers source updates to interested clients using traditional synchronous request/reply communication paradigm, the rigid structure of such interaction renders the system inefficient for real-time decisions [2]. Publish-Subscribe (Pub-Sub) systems are an alternative communication paradigm that enables efficient and scalable communication among various components. The delivery of information from publishers to subscribers is decoupled, meaning that they need not be aware of each other's existence. For example, cameras can publish video frames to the system while a processing system subscribes to this data and uses it to make real-time decisions about intersection safety, such as adjusting the timing of traffic lights or alerting drivers to pedestrians in the area. A middleware (also known as a broker) acts as an intermediary to manage this distribution of information from publishers to subscribers.

The presence of different types of subscribers with varying time scales of operation in an edge computing system allows for flexibility and adaptability to different requirements and use cases. For instance, a real-time subscriber could be an analytics module that processes sensor data in real-time to detect anomalies or trigger immediate actions. On the other hand, a batch processing subscriber could be a processing module that handles a batch of sensor data collected over a certain time window to generate periodic reports or perform long-term trend analysis. Consequently, such heterogeneity requires decoupling subscribers from publishers.

In order to facilitate the required information dissemination, the data is pushed to the middleware from publisher. However, two complementary communication modes between middleware and subscriber exist: push and pull modes. Under the push mode, the subscriber passively receives information from the pub-sub middleware. In a pull-based approach, subscribers request messages from the middleware when they are ready to receive them, rather than waiting for the middleware to push messages to them.

There are pros and cons to both approaches. However, the pull mode is better suited for handling a diverse range of subscribers, without requiring a broker to determine the data transfer rate for each of them. Subscribers have more control over the rate at which they consume messages, making it easier to manage their individual needs [3].

Along with this, a real-time database system, acting as a middleware system, can support information dissemination between publishers and subscribers. With these design choices in mind, we present an analytical framework based on the publish-subscribe interaction paradigm, that is to be used for disseminating source updates to interested clients that process these source updates and arrive at decisions in a timely way.

Considered Problem: The freshness of status information received by subscriber plays an important role in decision making. In this work, we focus on a fundamental problem: What is the average Age of Information (AoI) [4] of decision updates that are computed from time-varying set of sensor data published in the database[1]? We model a class of systems (see Fig. 1) in which two independent sources submit time-stamped updates to a writer that is responsible for publishing the source measurements as updates in the memory. A decision process (DP), as a subscriber, reads the pair of source 1 and source

[1]Going forward, we refer to the database as *shared memory*, or simply as *memory*.

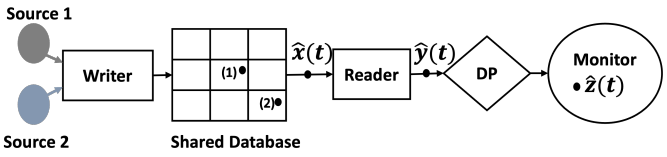


Fig. 1. A writer updates shared database with information fetched from two external sources. A decision process (DP) requests a reader process to read the pair of source updates from the memory. Monitors that track the age of source 1 and 2 updates in the memory are denoted $\bullet(1)$ and $\bullet(2)$ respectively; $\bullet(\hat{x}(t))$ tracks the age of max-age process in the memory, $\bullet(\hat{y}(t))$ tracks the age of sampled max-age process, and $\bullet(\hat{z}(t))$ tracks the age of computed decision updates at the external monitor.

2 updates from memory and derives a computational result, a *decision update*, from this pair that is delivered to a monitor.

Related Work: The issue of update timeliness using Age of Information (AoI) metric has been extensively studied; see the surveys [5], [6] and references therein. A majority of studies focus on analyzing the average AoI of different sources in single-server multi-source queueing models under different service policies [7]–[9].

More recently, there have been efforts to employ age optimization in diverse applications such as edge cloud processing for low-latency edge-assisted applications [10], [11], and timely mobile routing [12], [13]. Authors in [10] discuss a greedy traffic scheduling policy that selects the next processing request (job) that offers maximum age penalty reduction. [11] provides an analytical framework for the problem of optimizing frame rate and lag synchronization of server and player in a real-time cloud-assisted gaming application. Authors in [12], [13] study the effects of concurrency constructs on timely updating of shared data structures used in network software and how this, in turn, affects timely routing of information updates.

In previous works, the system models assumed a tight-coupling between the source and monitor. The primary goal was to maintain the status information of the physical process at the destination nodes. For instance, the just-in-time update policy [4] generated a fresh update instantaneously by the source, which started serving as soon as the current update in service was delivered to the destination node. Many works [14]–[17] have extended upon this idea and aimed to find optimal policies for determining sampling times and updating processes to minimize the Age of Information (AoI) at the destination node.

In publish-subscribe systems with decoupled producers and consumers, where producers use independent status updating policies, and consumers compute on this status information, a key open problem is to study optimal policies for minimizing the Age of Information (AoI) of computed updates. This work aims to study how the decoupling of publishers and subscribers affect the timeliness of updates computed from base set of sensor (source) updates.

II. System Overview

In this work, we employ a model that captures the asynchronous operation of the writer and reader of updates in the shared memory Pub-Sub system. There are three aspects to the system depicted in Fig. 1: 1) writing the time-varying data received from two sources into the memory, 2) the arbitration between reader and writer to access memory, 3) reading the source data from memory and generating a decision update. We now give a brief overview of the writing, reading and decision computation processes.

A. Writing source updates to the memory

We assume each source $i \in \{1, 2\}$ independently submits updates as a rate λ_i Poisson process to the network and that these updates arrive fresh at the writer, i.e. with age 0. The write operations to memory have independent exponential (μ) service times. We model the writer as a buffer-less service facility with blocking discipline. Under this model, a source update arriving at the writer will be served only if the writer is idle; otherwise, the update is discarded.

Remark 1: In the present study, we investigate a computational regime characterized by relatively longer decision update times compared to the write times of any update in the memory. Our focus is not on regimes where writing to the memory is the overloaded process. Instead, we are primarily interested in examining the delays associated with computational processing. Whether we adopt a buffer-less or a queuing model, the impact of queuing at the writer is expected to be minimal.

B. Read-Copy-Update

We assume that arbitration between the reader and writer for the shared memory is facilitated by Read-Copy-Update (RCU) mechanism. RCU is a lock-less concurrency construct that allows concurrent forward progress for both reader and writer. [18]. RCU can be broadly described in two steps [19]: 1) To publish a newer version of a data item, the writer creates a copy of the RCU protected data item, modifies this copy with the newer version of this data item, and atomically replaces the old reference with a reference to this newer version. This publishing process runs concurrently with ongoing read processes that continue to read the old copy/version using the old reference. However, new read requests read the most recent version. 2) Since some readers in progress hold reference to "stale" data, the system defers memory reclamation of old data until after each reader in progress has finished executing its read-side critical section.

Remark 2: RCU read operations can be performed concurrently without any locks, allowing for high concurrency and low contention. This can be particularly useful in systems with multiple subscribers and few publishers. In combination, RCU and Pub-Sub can enable efficient and scalable communication where multiple components need to access and update shared data in a concurrent and asynchronous manner.

C. Computing decision updates

We view the decision process (DP) reader as one of many subscribers to the updates in the memory system. The DP reader becomes aware of fresher updates in the memory only when it chooses to query the memory for a fresh sample of the source update pair. We assume a reader can fetch the updates of both source 1 and source 2 from memory in negligible time[2]. With this assumption, the DP reader is an observer that is *sampling* the pair of source updates from the memory as a point process. Based on this sample, the decision process derives a decision update which is sent to the monitor, as shown in Fig. 1. The reader process fetches the next sample of update pair from the memory only after the computation in progress is completed.

When the DP reader's inter-sample times form a renewal process, this is an example of the model of renewal process sampling of updates introduced in [20]. In this model, the DP reader generates an age process $\hat{y}(t)$ at the input to the DP that is a sampled version of the max-age process $\hat{x}(t)$ in the memory. Specifically, in the absence of a read, $\hat{y}(t)$ continues to grow at unit rate. However, if the DP reader makes a read at time τ, then $\hat{y}(t)$ is reset to $\hat{y}(\tau) = \hat{x}(\tau)$. This update pair is then processed by the DP for a time T so that at time $\tau + T$ a decision update with age $\hat{y}(\tau) + T$ is delivered to the monitor. The age at the monitor, $\hat{z}(t)$, is then reduced to $\hat{z}(\tau+T) = \hat{x}(\tau)+T$. At this time, the DP reader may choose to fetch a new sample pair from the memory, or it may choose to wait for a time W before fetching the next sample pair. When the DP reader employs non-zero waiting times, we say the DP is using a *lazy sampling* policy [16]. Fig. 2 illustrates the evolution of age processes $\hat{x}(t), \hat{y}(t)$, and $\hat{z}(t)$.

D. Paper Overview and Contributions

We divide our AoI analysis into two stages: 1) We analyze the average age of updates in shared memory. 2) We analyze the additional delay induced by the decision process computations. First, section III presents a stochastic hybrid system (SHS) evaluation of the update age processes in the memory. For the system with sources $i = 1, 2$, we derive the stationary expected ages $\mathrm{E}[x_i(t)]$ as well as the expected age of the *max-age* process $\hat{x}(t) = \max(x_1(t), x_2(t))$, .

In section IV, stage two of our analysis, we evaluate the age $\hat{z}(t)$ of the decision update process at the monitor. The decision process is said to be *sampling* the source updates from the memory as it holds a sample of updates that were written to the memory. Even though the sampling and computation of the DP makes no attempt to use the age of its sampled updates to optimize its operation, we show that a lazy sampling policy will be able to reduce $\hat{z}(t)$. Here we will see that analysis of $\hat{z}(t)$ is separable from the prior SHS analysis of the max-age process $\hat{x}(t)$ in the shared memory. In particular, the AoI reduction afforded by lazy sampling can be applied to any stationary update age process that is sampled by the DP.

[2]This assumption is consistent with RCU reads being lightweight and fast, so that the heavier load is indeed induced by actual decision computation. Further, our model assumes that the DP reader fetches updates from the memory at some finite average rate such that the combined read request process of all subscribers does not overload the shared memory system.

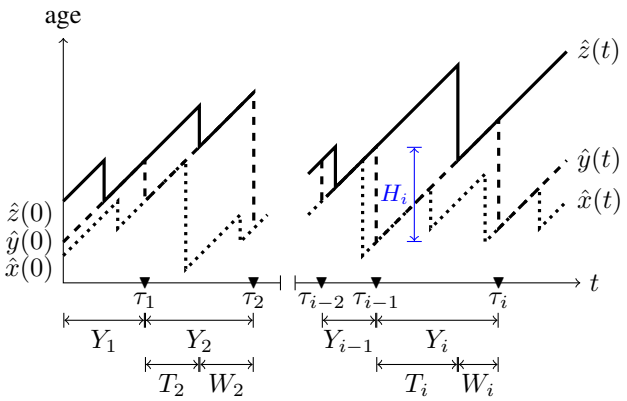


Fig. 2. Example AoI evolution of the max-age process $\hat{x}(t)$ at the memory, the sampled max-age process $\hat{y}(t)$ with *lazy sampling* at the input to the DP, and the age process $\hat{z}(t)$ at the monitor. The DP reader samples updates from the memory at times $\tau_1, \tau_2, \ldots$, marked by $\blacktriangledown$. Y_i is the sampling period for sample i, T_i is the computation time for decision update based on sample $i-1$, and W_i is the waiting time to get the i^{th} sample.

III. Age of source updates in the memory

Let $U_{i,1}, U_{i,2} \ldots$ be the sequence of source i update publication times. At any time t, $N_i(t)$ source i updates have been published in the memory, and the most recent update is published at time $U_{i,N_i(t)}$. It follows that the source i update process has age $x_i(t) = t - U_{i,N_i(t)}$ in the memory. Under this model, the update age $x_i(t)$ is reset to the write time $W \sim \exp(\mu)$ when it is published at time $U_{i,N_i(t)}$. When the writer writes a fresh source i update at time t', the max-age process $\hat{x}(t)$ is reset to $\hat{x}(t') = x_j(t')$, with $j \neq i$. In the following, we use a Stochastic Hybrid System (SHS) to capture the evolution of update age processes in the memory.

A. SHS Overview

To evaluate AoI of source updates, we use a Stochastic Hybrid Systems (SHS) [21] approach, a technique introduced for AoI evaluation in [22] and since employed in AoI evaluation of a variety of status updating systems [20], [23]–[29]. A stochastic hybrid system has a state-space with two components – a discrete component $q(t) \in \mathcal{Q} = \{0, 2, \ldots, M\}$ that is a continuous-time finite-state Markov Chain and a continuous component $\mathbf{x}(t) = [x_0(t), \ldots, x_n(t)] \in \mathbb{R}^{n+1}$. In AoI analyses using SHS, each $x_j(t) \in \mathbf{x}(t)$ describes an age process of interest. Each transition $l \in \mathcal{L}$ is a directed edge (q_l, q_l') with a transition rate $\lambda^{(l)}$ in the Markov chain. The age process vector evolves at a unit rate in each discrete state $q \in \mathcal{Q}$, i.e., $\frac{d\mathbf{x}}{dt} = \dot{\mathbf{x}}(t) = \mathbf{1}_n$. A transition l causes a system to jump from discrete state q_l to q_l' and resets the continuous state from $\mathbf{x}$ to $\mathbf{x}'$ using a linear transition reset map $\mathbf{A}_l \in \{0, 1\}^{(n\times n)}$ such that $\mathbf{x}' = \mathbf{x}\mathbf{A}_l$. For simple queues, examples of transition reset mappings $\{\mathbf{A}_l\}$ can be found in [22].

l	$q_l \to q_l'$	$\lambda^{(l)}$	$\mathbf{x}\mathbf{A}_l$
1	$0_1 \to 1$	λ_1	$[0, x_1, x_2, \hat{x}]$
2	$1 \to 0_2$	μ	$[x_0, x_0, x_2, x_2]$
3	$0_2 \to 1$	λ_1	$[0, x_1, x_2, \hat{x}]$
4	$0_2 \to 2$	λ_2	$[0, x_1, x_2, \hat{x}]$
5	$2 \to 0_1$	μ	$[x_0, x_1, x_0, x_1]$
6	$0_1 \to 2$	λ_2	$[0, x_1, x_2, \hat{x}]$

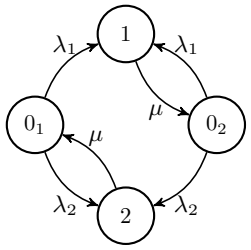


Fig. 3. The SHS transition/reset maps and Markov chain for the update age in the shared memory.

For a discrete state $\bar{q} \in \mathcal{Q}$, let

$$\mathcal{L}_{\bar{q}} = \{l \in \mathcal{L} : q_l' = \bar{q}\}, \qquad \mathcal{L}'_{\bar{q}} = \{l \in \mathcal{L} : q_l = \bar{q}\}. \tag{1}$$

denote the respective sets of incoming and outgoing transitions. Age analysis using SHS is based on the expected value processes $\{\mathbf{v}_q(t) \colon q \in \mathcal{Q}\}$ such that $\mathbf{v}_q(t) = \mathrm{E}[\mathbf{x}(t)\delta_{q,q(t)}]$, with $\delta_{i,j}$ denoting the Kronecker delta function. For the SHS models of age processes considered here, each $\mathbf{v}_q(t)$ will converge to a fixed point $\bar{\mathbf{v}}_q$. The fixed points $\{\bar{\mathbf{v}}_q \colon q \in \mathcal{Q}\}$ are the solution to a set of age balance equations. The following theorem provides a simple way to calculate the age-balance fixed point and then the average age.

Theorem 1. *[22, Theorem 4] If the discrete-state Markov chain $q(t) \in \mathcal{Q} = \{0, \ldots, M\}$ is ergodic with stationary distribution $\bar{\boldsymbol{\pi}} = [\bar{\pi}_0 \cdots \bar{\pi}_M] > 0$ and there exists a non-negative vector $\bar{\mathbf{v}} = [\bar{\mathbf{v}}_0 \cdots \bar{\mathbf{v}}_M]$ such that*

$$\bar{\mathbf{v}}_{\bar{q}} \sum_{l \in \mathcal{L}_{\bar{q}}} \lambda^{(l)} = \mathbf{1}\bar{\pi}_{\bar{q}} + \sum_{l \in \mathcal{L}'_{\bar{q}}} \lambda^{(l)} \bar{\mathbf{v}}_{q_l} \mathbf{A}_l, \quad \bar{q} \in \mathcal{Q}, \tag{2}$$

then the average age vector is $\mathrm{E}[\mathbf{x}] = \lim_{t\to\infty} \mathrm{E}[\mathbf{x}(t)] = \sum_{\bar{q}\in\mathcal{Q}} \bar{\mathbf{v}}_{\bar{q}}$.

B. SHS Analysis of Age in Shared Memory

The age of updates in a shared memory system with bufferless service at the writer can be described by the SHS Markov chain and table of state transitions shown in Fig. 3. The continuous age state vector is $\mathbf{x} = [x_0, x_1, x_2, \hat{x}]$, where x_0 is the age of the update being written; x_i, $i = 1, 2$, is the age of the source i update in memory; and $\hat{x} = \max(x_1, x_2)$. The discrete state is $\mathcal{Q} = \{0_1, 0_2, 1, 2\}$. At time t, the system is in state 0_i if the writer is idle and the oldest update belongs to source i. State $i \in \{1, 2\}$ corresponds to the writer writing source i update.

We now describe SHS transitions enumerated in the table in Fig. 3. For each collection of transitions, we focus on the age state components that change.

- $l = 1, 3, 4, 6$: In system idle states 0_1 and 0_2, the writer receives a new source update and initiates a new write mechanism. $x_0' = 0$ as the writer receives a fresh update, and $x_1', x_2', \hat{x}'$ are unchanged as the update is not yet written to the memory.
- $l = 2, 5$: The writer finishes writing and publishes a new source update.
  - $l = 2$: the writer publishes source 1 update: $x_1' = x_0$ as the age of source 1 update in the memory is reset to just written update. The source 2 update becomes the oldest update in the memory; hence, $\hat{x}' = x_2$.
  - $l = 5$: The writer publishes source 2 update: $x_2' = x_0$, the source 1 update becomes the oldest update, and $\hat{x}' = x_1$.

For the SHS analysis, we employ the normalized rates

$$\rho_1 = \lambda_1/\mu, \quad \rho_2 = \lambda_2/\mu. \tag{3}$$

We note that $\rho = \rho_1 + \rho_2$ is the total offered load of source updates being written to the memory. The Markov chain in Fig. 3 has stationary probabilities $\boldsymbol{\pi}$ with normalization constant C_π given by

$$\boldsymbol{\pi} = [\pi_{0_1} \ \pi_1 \ \pi_2 \ \pi_{0_2}] = C_\pi^{-1}[\rho_2/\rho \ \rho_1 \ \rho_2 \ \rho_1/\rho], \tag{4a}$$

$$C_\pi = 1 + \rho. \tag{4b}$$

With the shorthand notation

$$\lambda = \lambda_1 + \lambda_2, \tag{5}$$

we now use Theorem 1 to solve for

$$\bar{\mathbf{v}} = [\bar{\mathbf{v}}_{0_1} \ \bar{\mathbf{v}}_1 \ \bar{\mathbf{v}}_2 \ \bar{\mathbf{v}}_{0_2}], \tag{6}$$

where $\mathbf{v}_q = [v_{q0} \ v_{q1} \ v_{q2} \ v_{q3}], \forall q \in \mathcal{Q}$. This yields

$$\lambda\bar{\mathbf{v}}_{0_1} = \mathbf{1}\bar{\pi}_{0_1} + \mu\bar{\mathbf{v}}_2\mathbf{A}_5, \tag{7a}$$

$$\mu\bar{\mathbf{v}}_1 = \mathbf{1}\bar{\pi}_1 + \lambda_1\bar{\mathbf{v}}_{0_1}\mathbf{A}_1 + \lambda_1\bar{\mathbf{v}}_{0_2}\mathbf{A}_3, \tag{7b}$$

$$\mu\bar{\mathbf{v}}_2 = \mathbf{1}\bar{\pi}_2 + \lambda_2\bar{\mathbf{v}}_{0_1}\mathbf{A}_6 + \lambda_2\bar{\mathbf{v}}_{0_2}\mathbf{A}_4, \tag{7c}$$

$$\lambda\bar{\mathbf{v}}_{0_2} = \mathbf{1}\bar{\pi}_{0_2} + \mu\bar{\mathbf{v}}_1\mathbf{A}_2. \tag{7d}$$

We can now use Theorem 1 to calculate the AoI of source i update in the memory as $\mathrm{E}[x_i] = v_{0_1,i} + v_{0_2,i} + v_{1,i} + v_{2,i}$ for $i \in \{1, 2\}$ as well as $\mathrm{E}[\hat{x}] = v_{0_1,3} + v_{0_2,3} + v_{1,3} + v_{2,3}$. Some algebra yields the following theorem.

Theorem 2.

(a) Source i updates in the memory have average age

$$\mathrm{E}[x_i] = \frac{1}{\mu}\left(\frac{1+\rho}{\rho_i} + \frac{\rho}{1+\rho}\right). \tag{8}$$

(b) The max-age process $\hat{x}(t) = \max(x_1(t), x_2(t))$ in the memory has average age

$$\mathrm{E}[\hat{x}] = \frac{(1+\rho)^2(\rho_1^2 + \rho_1\rho_2 + \rho_2^2) + \rho^2\rho_1\rho_2}{\mu\rho(1+\rho)\rho_1\rho_2}. \tag{9}$$

Not surprisingly, the expected max-age $\mathrm{E}[\hat{x}]$ is symmetric in the load parameters ρ_1 and ρ_2. However, since the formula (9) is somewhat opaque, a plot of $\mathrm{E}[\hat{x}]$ appears in Fig. 4. A possibly non-obvious observation from the figure is that increasing the overall updating load ρ generally improves the average max-age because the writer queues no updates. The figure also reveals that the average max-age is penalized by asymmetry in the update rates of the individual sources. This is in part because a source that updates slowly will have high age and thus cause the max-age to be large. However, it is also true that with asymmetric loads, the high rate source will

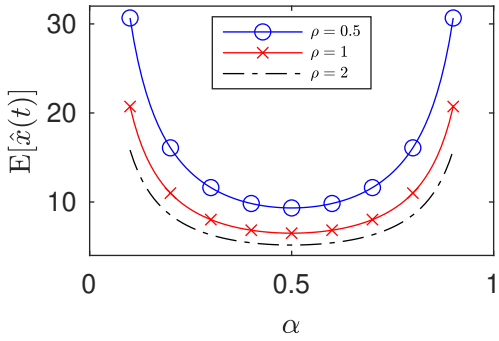


Fig. 4. Average age of max-age process $\hat{x}(t)$ in the memory. For a fixed updating load, we vary α with $\rho_1 = \alpha\rho$ and $\rho_2 = (1-\alpha)\rho$.

cause updates of the low rate source to be discarded at the writer. Because the writer is non-selective in offering service, it may be performing updates for the high rate source even when the age of that source is already low.

IV. Age of Decision Updates

In section II-C, we observed that the DP reader is *sampling* the source updates from the memory as a point process. In particular, we assume the inter-sample times $Y_1, Y_2, \ldots$ that are i.i.d continuous random variables identical to Y. In this case, the update sample times form a renewal process, and in the parlance of [20], the update age process $\hat{y}(t)$ is sampling the max-age update process $\hat{x}(t)$ in the shared memory.

A. Average Age at the Decision Process

At time t, the most recent read from memory occurred at time $t - Z(t)$. That is, $Z(t)$ is the age of the sampling renewal process. When the renewal process is in equilibrium, $Z(t)$ is stationary with first moment [30, Theorem 5.7.4]

$$\mathrm{E}[Z] = \frac{\mathrm{E}[Y^2]}{2\,\mathrm{E}[Y]}. \tag{10}$$

Next, following the approach in [20], we observe that the DP reader does not fetch any update in the interval $(t - Z(t), t]$. Hence, at time t, the update age $\hat{y}(t)$ satisfies

$$\hat{y}(t) = \hat{x}(t - Z(t)) + Z(t). \tag{11}$$

Further, $Z(t)$ is independent of $\hat{x}(t)$ because the inter-sample times Y_i are independent of the age processes in the shared memory. Thus stationarity of $\mathrm{E}[\hat{x}(t)]$ implies $\mathrm{E}[\hat{x}(t - Z(t)] = \mathrm{E}[\hat{x}(t)] = \mathrm{E}[\hat{x}]$. It then follows from (10) and (11) that $\hat{y}(t)$ has expected value[3]

$$\mathrm{E}[\hat{y}] = \mathrm{E}[\hat{x}] + \mathrm{E}[Z] = \mathrm{E}[\hat{x}] + \frac{\mathrm{E}[Y^2]}{2\,\mathrm{E}[Y]}. \tag{12}$$

[3]A stronger distributional result is derived in [20, Theorem 6] that is not needed for the average AoI analysis here.

B. Average Age at the Monitor: Lazy Sampling

When the DP reader samples the shared memory, the DP then computes a decision update based on this sample. On delivery of a decision update to the monitor, the update age $\hat{z}(t)$ is reset to the age of the oldest source update that was read and used to compute the decision update. This means that an arrival of decision update at the monitor at time t resets $\hat{z}(t)$ to $\hat{y}(t)$.

In this work, we assume that the decision computation times are i.i.d continuous random variables $T_1, T_2, \ldots$, each identically distributed to T. We will consider a DP that performs *lazy sampling*: after delivering the computation to the output monitor, the DP reader waits for a random time W before reading again. The alternative to being lazy is the *zero-wait* policy, a special case of lazy when $W = 0$.

Fig. 2 depicts the evolution of the max-age process $\hat{x}(t) = \max(x_1(t), x_2(t))$, the status-sampling process $\hat{y}(t)$, and the age at the monitor $\hat{z}(t)$. with i.i.d inter-sample intervals $Y_1, Y_2, \ldots$ such that samples are taken at times $\tau_i = \sum_{j=1}^{i} Y_j$.

Under lazy sampling, we admit the possibility that the ith computation time T_i and the ith waiting time W_i are correlated. However, in order for the $\hat{y}(t)$ process to be sampling the shared memory with independent inter-sample times $Y_i = T_i + W_i$, we require that the pairs $(T_1, W_1), (T_2, W_2), \ldots$ to be i.i.d., identical to (T, W). Under this assumption, it follows directly from (12) that the average update age at the input to the DP is

$$\mathrm{E}[\hat{y}] = \mathrm{E}[\hat{x}] + \frac{\mathrm{E}[(T+W)^2]}{2\,\mathrm{E}[T+W]}. \tag{13}$$

Curiously, (13) reveals that the problem of minimizing the average age at the input to DP appears to be isomorphic to the timely updating problem that was originally formulated in [16], [17], where the suboptimality of zero-wait policies was first identified. However, in this system, our objective is not to minimize $\mathrm{E}[\hat{y}]$ but rather to minimize the average age $\mathrm{E}[\hat{z}]$ at the monitor. Since $\hat{z}(t)$ is penalized by the waiting time W, choosing W to minimize $\mathrm{E}[\hat{y}]$ may not be good for $\mathrm{E}[\hat{z}]$. Fortunately, the following claim verifies this is not the case.

Theorem 3. *If $\hat{x}(t)$ is a stationary process, then for any waiting policy such that W_i depends only on T_i, the average age at the monitor satisfies*

$$\mathrm{E}[\hat{z}] = \mathrm{E}[\hat{y}] + \mathrm{E}[T]. \tag{14}$$

The proof appears in the Appendix. We observe that Theorem 3 can give one the mistaken impression that $\mathrm{E}[\hat{z}]$ is insensitive to the waiting time W. In fact, the theorem says that the waiting time W affects $\mathrm{E}[\hat{y}]$ and $\mathrm{E}[\hat{z}]$ identically. A hand-waving intuition is that $\hat{z}(t)$ lags $\hat{y}(t)$ only during the computation time T but, once the computation is complete, $\hat{z}(t) = \hat{y}(t)$ during any waiting period.

Combining (13) and (14), we obtain an end-to-end characterization of the average age in the system:

$$\mathrm{E}[\hat{z}] = \mathrm{E}[\hat{x}] + \frac{\mathrm{E}[(T+W)^2]}{2\,\mathrm{E}[T+W]} + \mathrm{E}[T]. \tag{15}$$

Since the computation time T is given, (15) shows that the choice of a waiting function W as a function of T is the same problem formulated in [16], [17]. Hence the solution is the same, namely the β-minimum waiting policy

$$W_i = (\beta - T_i)^+, \tag{16}$$

where the parameter β is chosen by numerical line search. With this policy, $T + W = \max(\beta, T)$ and it follows from (15) that the policy achieves end-to-end average AoI

$$\mathrm{E}[\hat{z}] = \mathrm{E}[\hat{x}] + \frac{\mathrm{E}[\max(\beta^2, T^2)]}{2\,\mathrm{E}[\max(\beta, T)]} + \mathrm{E}[T]. \tag{17}$$

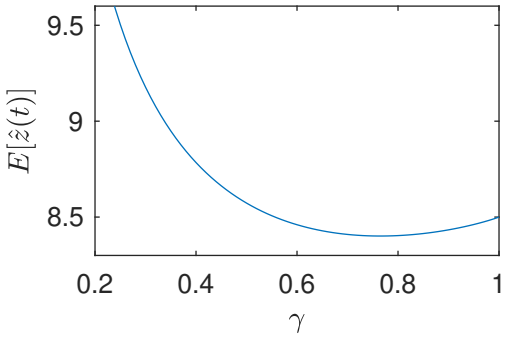
For completeness, the effectiveness of waiting is demonstrated in section V by some numerical evaluations of the lazy sampling policy. We will see that lazy sampling becomes important when the variance of the computation time T becomes large. Before presenting these results, we comment on the connection of this lazy sampling model to the lazy updating model in [16], [17].

In [16], [17], the random variable T represented the delivery time of a fresh update (say through a network) to the monitor. Fresh updates were generated at will and W represented the waiting time prior to generating the next fresh update. A key element of this system was the tight coupling of waiting and update generation. In this setting, the intuition behind β-minimum waiting was that if the prior delivery time was small, the age at the monitor would be small and it would be a waste of network resources to deliver an update when the age reduction afforded by the update would be small.

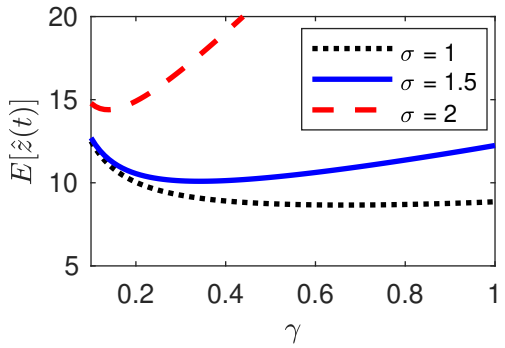
In this work, updates are generated by an exogenous process that is beyond the control of the DP. Moreover, because updates are disseminated through a shared memory publication process, the age processes of updates in shared memory are essentially uncoupled from the update sampling/processing policy implemented by the DP. In particular, any time the DP reader fetches a sample pair from the memory, the update age of that pair has expected value $\mathrm{E}[\hat{x}]$, which is just the average age in the shared memory. Nevertheless, even though DP operation is uncoupled from the age process in shared memory, the β-minimum waiting policy is effective. In particular, it reduces the expected value of $\hat{y}(t)$, the age process at the input to the DP. What is happening is that the waiting policy mitigates the deleterious effect of high-variance computation times T on the sampling policy at the DP reader. We note that Theorem 3 went unrecognized in [16], [17]. Specifically, Theorem 3 shows that no matter what policy is used, the output always lags the input by $\mathrm{E}[T]$ in terms of average age.

V. Numerical Evaluation

In this section, we examine some numerical examples of the performance β-minimum waiting policy, simply to remind the reader of the benefits of waiting. Fig. 5 illustrates age performance with respect to variance in the computation time with probability distributions $\exp(1)$ (Fig. 5(a)), and Log-Normal$(1, e^{\sigma^2} - 1)$ (Fig. 5(b)). The log-normal distributed computation times T has PDF [31],



(a) $T \sim \exp(1)$



(b) $T \sim$ Log-Normal$(1, e^{\sigma^2} - 1)$

Fig. 5. Average age at the monitor vs the sampling rate γ for the β-minimum policy for different distribution of computation time T. Total offered load by source updates is $\rho = 1$, with $\rho_1 = \rho_2 = 0.5$. Notice that $\gamma = 1$ is the zero-wait computation policy.

$$f_T(t) = \frac{e^{-(\ln(t)-b)^2/2\sigma^2}}{\sqrt{2\pi}\sigma t}, \qquad t > 0, \tag{18}$$

with free parameters b and $\sigma > 0$. In our numerical evaluations, we consider a given distribution on T such that the computation time is normalized to $\mathrm{E}[T] = 1$. In this regard, for Log-Normal distribution, for each σ, we set $b = -\sigma^2/2$ so that $\mathrm{E}[T] = 1$. By varying σ, we vary $\mathrm{Var}[T] = e^{\sigma^2} - 1$. These numerical results are largely similar to those in [16], [17]. In particular, the results remind the reader that zero-wait becomes increasingly sub-optimal when the computation time T has high variance. The choice of β specifies a sampling rate

$$\gamma = \frac{1}{\mathrm{E}[T] + \mathrm{E}[W]} = \frac{1}{\mathrm{E}[\max(\beta, T)]} \tag{19}$$

at the DP reader. We then plot the average age at the monitor as a function of γ. Because $\mathrm{E}[T] = 1$, the maximum update sampling rate is $\gamma = 1$, which corresponds to the zero-wait policy. As $\gamma \to 0$, the average age is increasingly dominated by the average inter-read time $1/\gamma$, because updates become too infrequent.

VI. Conclusion

In this work, we focused on the problem of timely processing of updates from multiple sources. Specifically, we

considered a model of a publish-subscribe system where a writer publishes updates from two independent sources in a shared memory and decision updates are derived by a decision process by reading from the memory. The decision processing is a subscriber that works independently of how the source updates are recorded in the memory. Even though the decision processing operates without knowledge of the ages of updates in the shared memory, its reading policy is still able to improve the end-to-end decision update timeliness.

We recognize that there could be other DP reading policies that exploit knowledge of the update age processes in shared memory to further reduce decision update age at the monitor; identifying such policies would be an interesting avenue for future research.

Another drawback of our system model is that when stale values are read from the shared memory, the DP will still perform its computations, even though the resulting DP update is not age-reducing. Effectively, the DP is waiting for a computation time before attempting to retrieve new updates from memory. Instead, it would be better for the DP to discard the stale updates and wait for an optimized time before reading the memory again. Analyzing such a model also remains as future work.

ACKNOWLEDGMENT

This work was supported by the US National Science Foundation under grant number CNS-2148104.

REFERENCES

[1] D. Raychaudhuri, I. Seskar, G. Zussman, T. Korakis, D. Kilper, T. Chen, J. Kolodziejski, M. Sherman, Z. Kostic, X. Gu, H. Krishnaswamy, S. Maheshwari, P. Skrimponis, and C. Gutterman, *Challenge: COSMOS: A City-Scale Programmable Testbed for Experimentation with Advanced Wireless*. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3372224.3380891

[2] A. Carzaniga, D. Rosenblum, and A. Wolf, "Design and evaluation of a wide-area event notification service," in *Foundations of Intrusion Tolerant Systems, 2003 [Organically Assured and Survivable Information Systems]*, 2003, pp. 283–334.

[3] "Kafka 3.4 documentation," https://kafka.apache.org/documentation.html#design_pull, accessed: 2023-05-04.

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

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

[6] R. D. Yates, Y. Sun, D. R. Brown, S. K. Kaul, E. Modiano, and S. Ulukus, "Age of information: An introduction and survey," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 5, pp. 1183–1210, 2021.

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

[8] R. Yates and S. Kaul, "Real-time status updating: Multiple sources," in *Proc. IEEE Int'l. Symp. Info. Theory (ISIT)*, Jul. 2012.

[9] L. Huang and E. Modiano, "Optimizing age-of-information in a multi-class queueing system," in *Proc. IEEE Int'l. Symp. Info. Theory (ISIT)*, Jun. 2015.

[10] J. Zhong, W. Zhang, R. Yates, A. Garnaev, and Y. Zhang, "Age-aware scheduling for asynchronous arriving jobs in edge applications," in *Infocom Workshop on Age of Information*, Apr. 2019.

[11] R. Yates, M. Tavan, Y. Hu, and D. Raychaudhuri, "Timely cloud gaming," in *Proc. INFOCOM*, May 2017, pp. 1–9.

[12] V. Ramani, J. Chen, and R. D. Yates, "Lock-based or lock-less: Which is fresh?" 2023.

[13] ——, "Timely mobile routing: An experimental study," 2023.

[14] B. Zhou and W. Saad, "Joint status sampling and updating for minimizing age of information in the internet of things," *CoRR*, vol. abs/1807.04356, 2018. [Online]. Available: http://arxiv.org/abs/1807.04356

[15] A. M. Bedewy, Y. Sun, and N. B. Shroff, "Age-optimal information updates in multihop networks," *CoRR*, vol. abs/1701.05711, 2017. [Online]. Available: http://arxiv.org/abs/1701.05711

[16] R. Yates, "Lazy is timely: Status updates by an energy harvesting source," in *Proc. IEEE Int'l. Symp. Info. Theory (ISIT)*, June 2015, pp. 3008–3012.

[17] Y. Sun, E. Uysal-Biyikoglu, R. D. Yates, C. E. Koksal, and N. B. Shroff, "Update or wait: How to keep your data fresh," *IEEE Trans. Info. Theory*, vol. 63, no. 11, pp. 7492–7508, Nov. 2017.

[18] P. E. Mckenney, J. Appavoo, A. Kleen, O. Krieger, O. Krieger, R. Russell, D. Sarma, and M. Soni, "Read-copy update," in *In Ottawa Linux Symposium*, 2001, pp. 338–367.

[19] P. E. McKenney, "What is rcu? – "read, copy, update"," [Online]. Available from: https://www.kernel.org/doc/html/latest/RCU/whatisRCU.html.

[20] R. D. Yates, "The age of information in networks: Moments, distributions, and sampling," *IEEE Transactions on Information Theory*, vol. 66, no. 9, pp. 5712–5728, 2020.

[21] J. P. Hespanha, "Modelling and analysis of stochastic hybrid systems," *IEE Proceedings-Control Theory and Applications*, vol. 153, no. 5, pp. 520–535, 2006.

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

[23] R. D. Yates, "Age of information in a network of preemptive servers," in *IEEE Conference on Computer Communications (INFOCOM) Workshops*, Apr. 2018, pp. 118–123, arXiv preprint arXiv:1803.07993.

[24] S. Farazi, A. G. Klein, and D. R. Brown, "Average age of information for status update systems with an energy harvesting server," in *IEEE Conference on Computer Communications (INFOCOM) Workshops*, April 2018, pp. 112–117.

[25] A. Maatouk, M. Assaad, and A. Ephremides, "Minimizing the age of information: NOMA or OMA?" in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019, pp. 102–108.

[26] S. Kaul and R. Yates, "Age of information: Updates with priority," in *Proc. IEEE Int'l. Symp. Info. Theory (ISIT)*, Jun. 2018, pp. 2644–2648.

[27] A. Maatouk, M. Assaad, and A. Ephremides, "On the age of information in a CSMA environment," *IEEE/ACM Transactions on Networking*, pp. 1–14, 2020.

[28] M. Moltafet, M. Leinonen, and M. Codreanu, "Moment generating function of the AoI in a two-source system with packet management," *IEEE Wireless Communications Letters*, vol. 10, no. 4, pp. 882–886, 2021.

[29] ——, "Source-aware packet management for computation-intensive status updating: MGF of the AoI," in *2021 17th International Symposium on Wireless Communication Systems (ISWCS)*, 2021, pp. 1–6.

[30] R. G. Gallager, *Stochastic processes: theory for applications*. Cambridge University Press, 2013.

[31] R. Yates and D. Goodman, *Probability and Stochastic Processes: A Friendly Introduction for Electrical and Computer Engineers*, ser. Probability and Stochastic Processes: A Friendly Introduction for Electrical and Computer Engineers. Wiley, 2014.

APPENDIX

Proof. (Theorem 3) Suppose $t \in (\tau_{i-1}, \tau_i]$, the ith inter-sample interval. We observe from Fig. 2 that $\hat{y}(\tau_{i-1}) = \hat{x}(\tau_{i-1})$ because the reader fetches update $i-1$ at that time. However, at that time, the monitor has only received the decision update based on update $i-2$, which had age $\hat{y}(\tau_{i-2})$ at time τ_{i-2} and now, at time $\tau_{i-1} = \tau_{i-2} + Y_{i-1}$, has age $\hat{y}(\tau_{i-2}) + Y_{i-1}$. Hence, at time τ_{i-1}, the monitor has age

$$\hat{z}(\tau_{i-1}) = \hat{y}(\tau_{i-2}) + Y_{i-1}. \tag{20}$$

Defining $H_i = \hat{y}(\tau_{i-2}) + Y_{i-1} - \hat{y}(\tau_{i-1})$, we can write

$$\hat{z}(\tau_{i-1}) = \hat{y}(\tau_{i-1}) + H_i. \tag{21}$$

Since $\hat{y}(\tau_i) = \hat{x}(\tau_i)$ for all i,

$$H_i = \hat{x}(\tau_{i-2}) + Y_{i-1} - \hat{x}(\tau_{i-1}). \tag{22}$$

It follows from stationarity of $\hat{x}(t)$ and independence of the sampling times τ_i and $\hat{x}(t)$ that

$$\begin{aligned} \mathrm{E}[H_i] &= \mathrm{E}[\hat{x}(\tau_{i-2})] + \mathrm{E}[Y_{i-1}] - \mathrm{E}[\hat{x}(\tau_{i-1})] \\ &= \mathrm{E}[Y_{i-1}] = \mathrm{E}[T] + \mathrm{E}[W]. \end{aligned} \tag{23}$$

At time τ_{i-1}, the ith busy period starts and both $\hat{y}(t)$ and $\hat{z}(t)$ grow linearly at rate 1 because neither process sees an update. Hence, $\hat{z}(t) = \hat{y}(t) + H_i$ during the busy period. Only when the busy period completes at time $\tau_{i-1}+T_i$ does $\hat{z}(t)$ drop and become equal to $\hat{y}(t)$. Let events B_t and I_t correspond to the decision process being busy and idle respectively, at time t. In this interval, the event B_t occurs while $\tau_{i-1} \le t \le \tau_{i-1} + T_i$; otherwise I_t occurs if $\tau_{i-1} + T_i \le t \le \tau_i$. With these events, we can write

$$\hat{z}(t) = \begin{cases} \hat{y}(t) + H_i, & \text{if } B_t, \\ \hat{y}(t), & \text{if } I_t. \end{cases} \tag{24}$$

For $t \ge \tau_{i-1}$, event B_t is independent of H_i, and it follows from the law of total expectation that

$$\begin{aligned} \mathrm{E}[\hat{z}(t)] &= \mathrm{E}[\hat{y}(t) + H_i | B] \, \mathrm{P}[B_t] + \mathrm{E}[\hat{y}(t) | I_t] \, \mathrm{P}[I_t] \\ &= \mathrm{E}[\hat{y}(t)] + \mathrm{E}[H_i] \, \mathrm{P}[B_t]. \end{aligned} \tag{25}$$

In each renewal period, the decision process is busy for time T and then idle for time W. By considering a renewal reward process in which a reward T is earned for the busy period, it follows that the limiting fraction of time spent in a busy state is given by

$$\mathrm{P}[B_t] = \frac{\mathrm{E}[T]}{\mathrm{E}[T] + \mathrm{E}[W]}. \tag{26}$$

Applying (23) and (26) to (25) yields the claim. □