Towards Optimal Tradeoff Between Data Freshness and Update Cost in Information-update Systems

Zhongdong Liu, Bin Li, Zizhan Zheng, Y. Thomas Hou, and Bo Ji

***Abstract*—In this paper, we consider a discrete-time information-update system, where a service provider can proactively retrieve information from the information source to update its data and users query the data at the service provider. One example is crowdsensing-based applications. In order to keep users satisfied, the application desires to provide users with fresh data, where the freshness is measured by the *Age-of-Information (AoI)*. However, maintaining fresh data requires the application to update its database frequently, which incurs an update cost (e.g., incentive payment). Hence, there exists a natural tradeoff between the AoI and the update cost at the service provider who needs to make update decisions. To capture this tradeoff, we formulate an optimization problem with the objective of minimizing the total cost, which is the sum of the staleness cost (which is a function of the AoI) and the update cost. Then, we provide two useful guidelines for the design of efficient update policies. Following these guidelines and assuming that the aggregated request arrival process is Bernoulli, we prove that there exists a threshold-based policy that is optimal among all online policies and thus focus on the class of threshold-based policies. Furthermore, we derive the closed-form formula for computing the long-term average cost under any threshold-based policy and obtain the optimal threshold. Finally, we perform extensive simulations using both synthetic data and real traces to verify our theoretical results and demonstrate the superior performance of the optimal threshold-based policy compared with several baseline policies.**

***Index Terms*—Data freshness, update cost, MDP, threshold-based policy, Age-of-Information.**

I. Introduction

With the remarkable development of communication networks and smart portable devices in recent years, we have witnessed significant advances in crowdsensing-based applications (e.g., Google Waze [1] and GasBuddy [2]). These applications provide services to users by resorting to the community to sense and send back real-time information (e.g., traffic conditions and gas prices) [3]. To satisfy the diverse needs of users, such applications need to maintain their knowledge of a set of distributed points of interest (PoI). For example, GasBuddy monitors gasoline prices at a large number of scattered gas stations in a certain area. In order to quickly and accurately respond to users' requests, the applications need to keep their data fresh. However, given the dynamic changes of the data, maintaining the freshness of data introduces a natural tradeoff between data freshness and update cost. On the one hand, users are unsatisfied if the responses to their requests are outdated; on the other hand, there is a cost for the applications to update their data because updating data relies on user feedback and often requires monetary payment to incentivize users [2], [3].

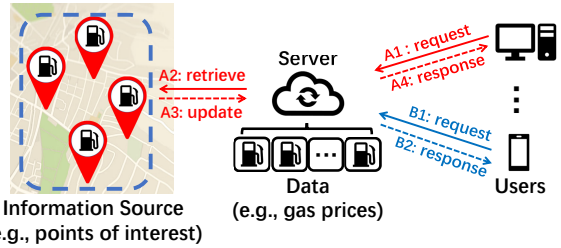


Fig. 1: An illustration of our system model. Upon receiving a request from the users, the server can either first update the data and then reply (red path: A1-A4) or simply reply with local data (blue path: B1-B2).

In fact, the tradeoff between data freshness and update cost does not only exist in crowdsensing-based applications, but also in a wide variety of time-sensitive data-driven applications that require timely information updates [4]–[8]. For example, in stock analysis applications, the server keeps track of the prices of a large number of stocks and generates different versions of the analysis reports for the stocks at certain times, and users send requests to query these analysis reports [8]. To ensure that users receive the real-time analysis of the stock they are trading with, the server needs to retrieve timely information (e.g., various stock market indexes) from the stock market, which incurs an update cost (e.g., bandwidth resources). Similar applications also include news feeds, weather updates, and flight aggregators.

The aforementioned applications have two notable characteristics: First, the server can proactively retrieve information from the information source to update its data, and users need to query the server to obtain the data (i.e., the "Pull" model [5], [9]); Second, the responses to users' requests (e.g., gas prices) typically do not require significant processing, and the packet size is usually small, making the packet transmission time negligible. However, retrieving the data from the information

The work of Zhongdong Liu and Bo Ji was supported in part by the NSF under Grants CNS-2112694 and CNS-2106427. The work of Y. T. Hou was supported in part by ONR MURI grant N00014-19-1-2621, Virginia Commonwealth Cyber Initiative (CCI), and Virginia Tech Institute for Critical Technology and Applied Science (ICTAS).

Zhongdong Liu (zhongdong@vt.edu) and Bo Ji (boji@vt.edu) are with the Department of Computer Science, Virginia Tech, Blacksburg, VA. Bin Li (binli@psu.edu) is with the Department of Electrical Engineering, the Pennsylvania State University, State College, PA. Zizhan Zheng (zzheng3@tulane.edu) is with the Department of Computer Science, Tulane University, New Orleans, LA. Y. Thomas Hou (thou@vt.edu) is with the Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA.

source often requires certain resources and introduces costs. These two characteristics not only distinguish such applications from other ones whose update costs mainly come from service and communication delays [10]–[12] but also lead to the tradeoff between the data freshness and the update cost.

To that end, in this work, we aim to optimize the tradeoff between data freshness and update cost. Specifically, we consider a discrete-time system in the setting where a service provider can proactively retrieve information from the information source and users obtain the data at the service provider by sending requests (see Fig. 1). The freshness of the data received by users is measured by a popular timeliness metric called *Age-of-Information (AoI)* [4], which is defined as the time elapsed since the most recent update occurred. To represent the dissatisfaction of users receiving stale data, we introduce the *staleness cost*, which is a non-decreasing function of the AoI (see formal definition in Section III). Clearly, one needs to account for both the update cost and the staleness cost when designing an online update policy.

We summarize our main contributions as follows.

First, we study the tradeoff between the data freshness and the update cost by formulating an optimization problem to minimize the sum of the staleness cost (which is a function of the AoI) and the update cost.

Second, we provide two useful guidelines for the design of optimal update policies. These guidelines suggest that 1) the service provider should update the data only at a point when it receives a request, and 2) the server should perform an update when the staleness cost is no smaller than the update cost.

Third, following these guidelines and assuming that the request arrival process is Bernoulli, we reformulate our problem as a Markov decision process (MDP) and show that there exists a threshold-based policy that is optimal among all online policies, which motivates us to focus on the class of threshold-based policies. Furthermore, we derive the closed-form expression of the average cost under any threshold-based policy and obtain the optimal threshold.

Finally, we perform extensive simulations using both synthetic data and real traces to verify our theoretical results and evaluate the performance of our proposed policy compared with several baseline policies. Our simulation results show that the threshold-based policy outperforms the baselines in more general settings (e.g., when the request arrival process is non-Bernoulli).

The remainder of this paper is organized as follows. We first discuss related work in Section II. The system model is described in Section III. Two guidelines for designing update policies are provided in Section IV. Then, we prove that our MDP formulation admits an optimal threshold-based policy and derive the optimal threshold in Sections V and VI, respectively. Finally, we present the numerical results in Section VII and conclude our paper in Section VIII.

II. Related Work

Ever since the concept of AoI was introduced in [4], the study on the AoI has attracted a lot of research interest. There is a large body of work that provides detailed analyses on the AoI performance of information-update systems under different queueing models (M/M/1, M/D/1, etc.) and scheduling policies (FCFS, LCFS, etc.) [13], [14].

Another important line of research focuses on AoI minimization. One specific type of optimization problem, which is similar to our work, is the joint minimization of AoI and certain costs [15]–[17]. In [15], the authors consider a discrete-time system where an information source is monitored over a communication channel with a transmission cost. They investigate the optimal policy for minimizing the sum of transmission cost and the inaccuracy of the state information at the monitor. It turns out that the optimal policies also have a threshold-based structure. Note that in their model, they assume that the source is governed by a random walk process and there are no users, which is different from ours. A similar source monitoring problem is considered in [16], where the goal is to minimize the sum of transmission costs and an unknown, time-varying penalty function of the AoI. They consider both single-source and multi-source scenarios and propose online learning algorithms with provable regret. In [17], the authors consider a source-monitor pair with stochastic arrival of updates at the source. The source pays a transmission cost to send the update, and its goal is to minimize the weighted sum of AoI and transmission costs. Under the assumption that the update arrival process is Poisson, they propose an optimal threshold-based policy. Their work differs from ours in their continuous-time setting and no user involvement.

Along this line, researchers have also considered AoI minimization with constraints (see a survey in [18]). The considered constraints can be viewed as a special type of update cost. For example, in wireless networks, the update of data consumes wireless channel resources. Therefore, the number of packets that can be transmitted depends on the interference model [19]. Similarly, for caching services, the cache server can only update certain contents at a time due to the capacity constraint [20]–[23]. Another example is the energy constraint [24]–[27], which is common in energy-constrained IoT systems. In these models, the update cost is usually imposed as a constraint of the optimization problem.

While the tradeoff between AoI and costs has been studied, most of them fall into the category where the costs primarily come from service (e.g., CPU cycle and storage) and/or communication (e.g., channel resources, delay, and energy consumption) caused by large packet size, which is different from our model, where the packet size is usually small and the update costs come from the retrieval of data from the information source. In addition, we also emphasize users' perspective in our model (i.e., the "Pull" model [5], [9]), where users can proactively query the server to obtain the data.

III. System Model and Problem Formulation

We consider a discrete-time information-update system that consists of an information source, a service provider (or server for short), and multiple users (see Fig. 1). The server can communicate with the information source and update its data

with the latest information. The users need to query the server to obtain the data.

We consider an aggregated arrival process formed by the requests from all the users (which will be assumed as Bernoulli process in Section V for further analysis). The requests arrive at the beginning of the time-slot, and the server replies to the requests with the most recently updated data at the end of a time-slot. We use the metric *Age-of-Information (AoI)* to measure the freshness of data, which is defined as the time elapsed since the most recent update. For ease of exhibition, we assume that the AoI drops to 0 after the update at the end of a time-slot[1]. The evolution of $\Delta(t)$ is as follows:

$$\Delta(t) = \begin{cases} \Delta(t-1)+1, & \text{if } u(t)=0; \\ 0, & \text{if } u(t)=1, \end{cases} \tag{1}$$

where $u(t)$ indicates whether the server updates the data at time-slot t. We assume that the time-slot is indexed from 1 and the initial AoI also equals 1, i.e., $\Delta(1)=1$. Let u_i denote the i-th update time. Then, an update policy π can be denoted by the update times: $\pi \triangleq \{u_i^\pi\}_{i=1}^{\infty}$. An illustration of a typical AoI evolution is shown in Fig. 2. To reflect the dissatisfaction level of the users when they receive stale data, we also introduce a *staleness cost* for each response of the server. Specifically, the staleness cost is defined as a penalty function $f(\Delta)$ of the AoI Δ, where the function $f : [0,\infty) \mapsto [0,\infty)$ is assumed to be measurable, non-negative, and non-decreasing. For simplicity, we let $f(0)=0$.

At the beginning of each time-slot, the server can decide whether to update the data or not. If it does, it needs to pay a constant update cost p and receives the latest data from the information source at the end of time-slot. To avoid the staleness cost, the server can first update the data and then reply to the request with the latest data. Let r_j be the arrival time of the j-th request. After the server receives the request at r_j, if the server chooses to update the data before replying to the request, its AoI drops to 0 after the update and its staleness cost becomes $f(0)=0$. In such a case, the server needs to pay an update cost p though. Otherwise, if the server does not update and replies with the current local data, the server needs to pay a staleness cost $f(\Delta(r_j))$.

Assume that the number of updates during the process of serving N requests under policy π is $U^\pi(N)$, i.e.,

$$U^\pi(N) \triangleq \max\{i | u_i^\pi \le r_N\}. \tag{2}$$

Then, the total cost of serving N requests, which is the sum of update costs and the staleness costs, is defined as

$$C^\pi(N) \triangleq \sum_{j=1}^{N} f(\Delta(r_j)) + pU^\pi(N). \tag{3}$$

The objective is to find an update policy π that minimizes *the long-term average expected cost per request (or average cost for short)*, which is defined as

$$\bar{C}^\pi \triangleq \lim_{N\to\infty} \frac{\mathbb{E}[C^\pi(N)]}{N}, \tag{4}$$

[1]Some work also assumes that the AoI drops to 1 [16], [28]. We assume that the AoI drops to 0 to make the discussion concise and clear.

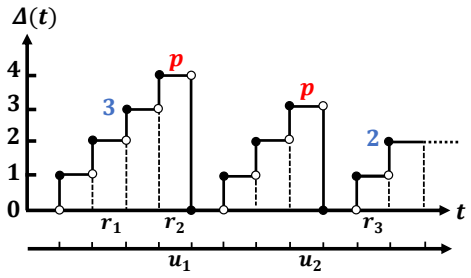


Fig. 2: An illustration of the AoI evolution at the server. There are two updates (in time-slots u_1 and u_2) during the process of serving three requests (in time-slots r_1, r_2, and r_3).

where the expectation is taken over the randomness in the arrival process and the update policy. Here, we assume that the limit of average cost under policy π exists. We focus on the set of online policies, denoted by Π, under which the information available at time t for making update decisions includes the update history, the arrival times of requests that arrive until t, and the update cost p. Then, we can formulate the following optimization problem:

$$\min_{\pi\in\Pi} \bar{C}^\pi. \tag{5}$$

IV. Guidelines for Algorithm Design

In this section, we provide two useful guidelines for the design of efficient update policies. Through a sample-path-dominance argument, we show that policies following these guidelines can achieve a lower total cost than those that do not. Therefore, we can reduce the search space of problem (5) to a certain class of online policies.

A. Reactive Policies

In this subsection, we present our first guideline for the design of update policies. As described in Section III, the server can update the data at any time. However, we show that to achieve a lower total cost, it is sufficient for the update policy to just consider updating the data immediately upon receiving a new request. We call such policies *Reactive Policies* as the server does not need to update the data when there is no request. We use Π^R to denote the set of reactive policies:

$$\Pi^R \triangleq \{\pi \in \Pi \mid u_k^\pi \in \{r_j\}_{j=1}^{\infty} \text{ for all } k\}. \tag{6}$$

Next, we show that restricting to reactive policies does not incur any performance loss.

Lemma 1. *For any policy $\pi \in \Pi$, there exists a reactive policy $\pi' \in \Pi^R$ that achieves an average cost no larger than that of policy π, i.e., $\bar{C}^{\pi'} \le \bar{C}^\pi$.*

Due to space limitations, we omit the proof but explain the key ideas in the following. Intuitively, postponing the update until a request arrives does not increase the total cost because the total number of updates remains the same, but doing this achieves a lower staleness cost since the update time is closer to the request arrival time. Therefore, reactive policies can

achieve a smaller total cost than those non-reactive policies. We provide the detailed proof in our technical report [29].

Lemma 1 implies that the search space of Problem (5) can be further reduced from the set of online policies Π to the set of reactive policies Π^R. Now, consider any reactive policy π. Upon receiving a request, the server needs to decide whether to update the data or not before responding. Therefore, we use I_j^π to denote the decision made by the server upon receiving the j-th request for the data at time r_j:

$$I_j^\pi \triangleq \begin{cases} 1, & \text{if the server updates the data at time } r_j; \\ 0, & \text{otherwise.} \end{cases}$$

B. Capped Reactive Policies

In this subsection, we present the second guideline for the design of update policies. In Section IV-A, we show that the reactive policies achieve a smaller or equal average cost by postponing the update until a request arrives. In fact, after the server receives the request, if the staleness cost is no smaller than the update cost, it is better for the server to update the data to avoid a larger staleness cost. Doing so not only leads to a smaller cost for this request but also benefits the next few requests. We use Π^{R+} to denote the set of reactive policies that satisfy the above guideline:

$$\Pi^{R+} \triangleq \{\pi \in \Pi^R \mid I_j^\pi = 1 \text{ for all } j \text{ when } f(\Delta(r_j)) \geq p\}.$$

That is, for any policy $\pi \in \Pi^{R+}$, it must update the data when the staleness cost is no smaller than the update cost; otherwise, it can choose to update the data or not. We call such policies *Capped Reactive Policies* because the staleness cost of such policies is capped by the update cost. Fig. 3 illustrates the relationship between Π^R and Π^{R+}. Note that the condition $f(\Delta(r_j)) \geq p$ can also be expressed as $\Delta(r_j) \geq \Delta^*$, where Δ^* is the smallest AoI such that the staleness cost is no smaller than the update cost, i.e.,

$$\Delta^* \triangleq \min\{\Delta | f(\Delta) \geq p\}. \tag{7}$$

In the following, we show that restricting to capped reactive policies does not incur any performance loss.

Lemma 2. For any policy $\pi \in \Pi^R$, there exists a capped reactive policy $\pi' \in \Pi^{R+}$ that achieves an average cost no larger than that of policy π, i.e., $\bar{C}^{\pi'} \leq \bar{C}^{\pi}$.

Due to space limitations, we omit the proof but explain the key ideas in the follow. Upon receiving a request, policy π' performs an update if the staleness cost is no smaller than the update cost. Compared to policy π that does not make such an update, doing so incurs an update cost for policy π', but it avoids a larger staleness cost. Besides, it also reduces the staleness cost for the requests that arrive thereafter. Therefore, policy π' can achieve a total cost no larger than that of policy π. We provide the detailed proof in our technical report [29].

By Lemma 2, we can further reduce the search space to the class of capped reactive policies Π^{R+}. Therefore, Problem (5) can be further reduced to the following:

$$\min_{\pi \in \Pi^{R+}} \ \bar{C}^{\pi}. \tag{8}$$

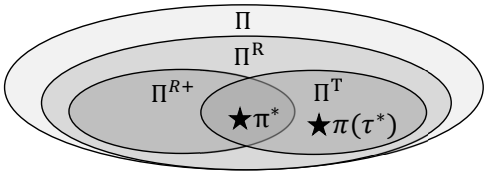


Fig. 3: The relationship between online policies Π, reactive policies Π^R, capped reactive policies Π^{R+}, threshold-based policies Π^T, an overall optimal policy π^* (see Theorem 1), and an optimal threshold-based policy $\pi(\tau^*)$ (see Corollary 1). Note that policy $\pi(\tau^*)$ is also an overall optimal policy and could be the same as policy π^* in some cases.

Till this point, we do not make any assumption on the request arrival process. The aforementioned guidelines can be applied to general request arrival processes. In the following, unless otherwise specified, we focus on the capped reactive policies Π^{R+}. *This capped property plays an important role in characterizing the threshold-based structure of an optimal policy for solving the MDP formulation in Section V.*

V. MDP Formulation and Threshold structure

Under a capped reactive policy, the server makes update decisions upon receiving requests and pays a cost (an update cost or a staleness cost) based on the decision. Naturally, this sequential decision process can be modeled as an MDP. In this section, we assume that the request arrival process is Bernoulli with rate $\lambda \in (0, 1)$, denoted by Bernoulli(λ), i.e., the probability that a request arrives in a time-slot is λ. Then, we reformulate Problem (8) as a discrete-time MDP and show that there exists a stationary threshold-based policy that solves the Bellman equation of the considered MDP and is thus optimal among all online policies.

The MDP formulation has the following key components: $\{\mathcal{N}, \mathcal{S}, \mathcal{A}_s, p(\cdot \mid s, a), c(s, a) : n \in \mathcal{N}, s \in \mathcal{S}, a \in \mathcal{A}_s\}$, where

1) $\mathcal{N} = \{1, 2, \cdots\}$ is the set of decision epochs. Under a capped reactive policy, the n-th decision epoch is the time-slot when the n-th request arrives.
2) $\mathcal{S} = \{0, 1, \cdots\}$ is the set of system states (which are all possible values of the AoI). We use s_n to denote the AoI value when the n-th request arrives.
3) $\mathcal{A}_s$ is the set of actions when the system state is s. Let $a \in \mathcal{A}_s$ denote the possible actions, where $a = 1$ means updating the data and $a = 0$ means not. Under a capped reactive policy, there are two sets of actions depending on the state s: when the staleness cost $f(s)$ is no smaller than the update cost p, the only available action is to update, i.e., $\mathcal{A}_{\{s:f(s)\geq p\}} = \{1\}$; otherwise, the system can either update or not, i.e., $\mathcal{A}_{\{s:f(s)<p\}} = \{0, 1\}$.
4) The transition probability can be calculated as

$$p(z \mid s, a) = \begin{cases} (1-\lambda)^{z-1}\lambda, & \text{if } z \geq 1 \text{ and } a = 1; \\ (1-\lambda)^{z-s-1}\lambda, & \text{if } z > s, f(s) < p, \text{ and } a = 0; \\ 0, & \text{otherwise.} \end{cases}$$

That is, when the system is in state s, if the server updates the data, the system will enter state z $(z \geq 1)$ with probability $(1-\lambda)^{z-1}\lambda$ because the request arrival process follows Bernoulli(λ); otherwise, if the server does not update, under a capped reactive policy, the system will enter state z $(z > s)$ with probability $(1-\lambda)^{z-s-1}\lambda$ only when the staleness cost $f(s)$ is smaller than the update cost p.

5) The cost at each decision epoch can be expressed as

$$c(s,a) = \begin{cases} p, & \text{if } a = 1; \\ f(s), & \text{if } a = 0. \end{cases} \tag{9}$$

That is, when the system is in state s, updating the data incurs an update cost of p; otherwise, there is a staleness cost of $f(s)$. Note that under a capped reactive policy, we always have $c(s,a) \leq p$.

The objective of the MDP is to find a stationary capped reactive stationary update policy that minimizes the long-term average expected cost, i.e.,

$$\min_{\pi \in \Pi^{R+}} \lim_{N \to \infty} \frac{\mathbb{E}_\pi \left[\sum_{n=1}^{N} c(s_n, a_n) | s_1 = s \right]}{N}, \tag{10}$$

where $\mathbb{E}_\pi[\cdot]$ represents the conditional expectation, given that policy π is employed; s_n and a_n are the state and action taken at decision epoch n, respectively; and s is the initial state. We emphasize that, unlike traditional MDP formulations that mainly focus on optimization over time, we optimize our objective over users' requests. This allows us to reduce the state space and thus simplify the analysis. Note that the objective in Problem (10) is the same as that in Problem (8) except that we specify the initial state s in Problem (10). In other words, an optimal policy for Problem (10) is also an optimal policy for Problem (8). Next, we show that there exists an optimal policy for Problem (10) that has a threshold-based structure, which enables us to search for an optimal policy in the class of threshold-based policies (see Section VI). We state this result in Theorem 1.

Theorem 1. *There exists an optimal stationary capped reactive policy $\pi^* \in \Pi^{R+}$ that has a threshold-based structure.*

We provide the detailed proof in Appendix A and present an outline of the proof in the following. First, we study a discounted MDP and derive its optimal value function. Second, based on the optimal value function, we derive the Bellman equation of the expected total average cost and show that the Bellman equation has a threshold-based structure. Specifically, the server needs to update the data when the current AoI value (i.e., the state) is no smaller than a certain fixed threshold s^* (see definition in Eq. (20)); otherwise, it does not. Now consider a stationary capped reactive policy $\pi^* \in \Pi^{R+}$ that makes update decisions based on threshold s^*. Apparently, policy π^* minimizes the Bellman equation for any state, thus it is an overall optimal policy [30, Chapter V, Theorem 2.1]. The threshold structure of the optimal policy π^* indicates that among all threshold-based policies, there is an overall optimal policy (see Fig. 3). This motivates us to search for the optimal threshold-based policy in Section VI.

VI. Optimal Threshold-based Policy

Theorem 1 tells us that we can further reduce the search space from the set of capped reactive policies to the set of capped reactive threshold-based policies. In this section, we formally define threshold-based policies and derive the closed-form expression of the average cost of the threshold-based policies. Using the closed-form expression, we can find the optimal threshold-based policy. Furthermore, we show that the optimal threshold-based policy is also an optimal policy among all online policies.

We begin with the definition of threshold-based policies.

Definition 1 (Threshold-based Policies). *A policy in Π^R is called a threshold-based policy if it performs updates according to the following rule with a predetermined positive integer threshold τ: for the request arriving at time r_j, we have*

$$I_j = \begin{cases} 1, & \Delta(r_j) \geq \tau; \\ 0, & \Delta(r_j) < \tau. \end{cases}$$

That is, the server updates the data at r_j before replying if the AoI at r_j is no smaller than threshold τ; otherwise, the server simply replies with the current local data.

We consider an integer threshold because the values of the AoI are integers. Let $\pi(\tau)$ be the threshold-based policy with threshold τ, and let Π^T be the set of all threshold-based policies. Fig. 3 shows the relationship of Π^R, Π^{R+}, and Π^T.

Assume that the request arrival process is Bernoulli, we can derive the closed-form expression of the average cost under any threshold-based policy. We state this result in Theorem 2.

Theorem 2. *Assume that the request arrival process is Bernoulli(λ), the staleness cost function is $f(\Delta)$, and the update cost is p. Then, for any policy $\pi(\tau) \in \Pi^T$ with a positive integer threshold τ, the average expected cost can be computed as follows:*

$$\bar{C}^{\pi(\tau)} = \frac{\lambda \sum_{t=1}^{\tau-1} f(t) + p}{\lambda(\tau - 1) + 1}. \tag{11}$$

We provide the detailed proof in Appendix B and present an outline of the proof in the following. Since the request arrival process is Bernoulli, under a threshold-based update policy, the lengths of update intervals are independent and identically distributed (*i.i.d.*). Thus, the update process is a renewal process. Due to the ergodicity of the process, the expected average cost can be computed as $\bar{C}^{\pi(\tau)} = \mathbb{E}[C_k]/\mathbb{E}[N_k]$, where $\mathbb{E}[C_k]$ and $\mathbb{E}[N_k]$ are the expected total cost and the expected number of requests in the k-th update interval, respectively. In addition, by exploiting the properties of Bernoulli arrival process, we can further derive the closed-form expressions of $\mathbb{E}[C_k]$ and $\mathbb{E}[N_k]$, which are shown in the numerator and denominator of Eq. (11), respectively.

With the result in Theorem 2, we can easily compute the optimal threshold τ^*. In fact, this optimal threshold-based policy $\pi(\tau^*)$ is also an overall optimal policy, which is shown in Corollary 1.

Corollary 1. *Assume that the request arrival process is Bernoulli(λ). Then, the threshold of the optimal threshold-based policy $\pi(\tau^*) \in \Pi^T$ is the following:*

$$\tau^* = \left\{ \underset{\tau \in \{\lfloor \tau' \rfloor, \lceil \tau' \rceil\}}{\arg\min} \bar{C}^{\pi(\tau)} \right\}, \tag{12}$$

where τ' is the real number that achieves the smallest expected average cost (i.e., $\tau' = \arg\min_{\tau>0} \bar{C}^{\pi(\tau)}$). Furthermore, the optimal threshold-based policy $\pi(\tau^*) \in \Pi^T$ is also an overall optimal policy among all online policies.

Proof. Theorem 1 states that there exists an overall optimal capped reactive threshold-based policy $\pi^* \in \Pi^{R+} \cap \Pi^T$. For the optimal threshold-based policy $\pi(\tau^*) \in \Pi^T$, we have $\bar{C}^{\pi(\tau^*)} \leq \bar{C}^{\pi^*}$. On the other hand, since policy π^* is an overall optimal policy, we also have $\bar{C}^{\pi(\tau^*)} \geq \bar{C}^{\pi^*}$. Therefore, we have $\bar{C}^{\pi(\tau^*)} = \bar{C}^{\pi^*}$. This implies that the optimal threshold-based policy $\pi(\tau^*) \in \Pi^T$ is also an overall optimal policy among all online policies. □

Here, the real number τ' in Eq. (12) can either be theoretically calculated if the expression of $\sum_{t=1}^{\tau-1} f(t)$ in $\bar{C}^{\pi(\tau)}$ is known (see below for two examples) or be numerically calculated otherwise. The optimal threshold τ^* may not be unique, depending on the staleness cost function f. Also, policy $\pi(\tau^*)$ could be the same as policy π^* in some cases.

In the following, we provide the average expected cost and optimal threshold when the staleness cost is a linear function and a square function of the AoI, respectively.

Example 1 (A linear staleness cost function). *Assume that the request arrival process is Bernoulli(λ), $f(\Delta) = \Delta$, and the update cost is p. Then, for any policy $\pi(\tau) \in \Pi^T$ with a positive integer threshold τ, we have*

$$\bar{C}^{\pi(\tau)} = \frac{\lambda\tau(\tau-1)/2 + p}{\lambda(\tau-1)+1}, \tag{13}$$

and the optimal threshold $\tau^* = \left\{\arg\min_{\tau \in \{\lfloor \tau' \rfloor, \lceil \tau' \rceil\}} \bar{C}^{\pi(\tau)}\right\}$, where $\tau' = (\sqrt{2p\lambda - \lambda + 1} + \lambda - 1)/\lambda$.

Example 2 (A quadratic staleness cost function). *Assume that the request arrival process is Bernoulli(λ), $f(\Delta) = \Delta^2$, and the update cost is p. Then, for any policy $\pi(\tau) \in \Pi^T$ with a positive integer threshold τ, we have*

$$\bar{C}^{\pi(\tau)} = \frac{\lambda\left[(\tau-1)^3/3 + (\tau-1)^2/2 + (\tau-1)/6\right] + p}{\lambda(\tau-1)+1},$$

and the optimal threshold $\tau^* = \left\{\arg\min_{\tau \in \{\lfloor \tau' \rfloor, \lceil \tau' \rceil\}} \bar{C}^{\pi(\tau)}\right\}$, where τ' is the solution of

$$1 - 6p - 6\tau + 6\tau^2 + \lambda(4\tau - 1)(\tau - 1)^2 = 0.$$

VII. Numerical and Experimental Results

In this section, we perform extensive simulations and experiments to verify our theoretical results and compare the performance of the optimal threshold-based policy with several baseline policies using both synthetic data and real traces. Throughout this section, we assume that the staleness cost is the AoI itself (i.e., $f(\Delta) = \Delta$).

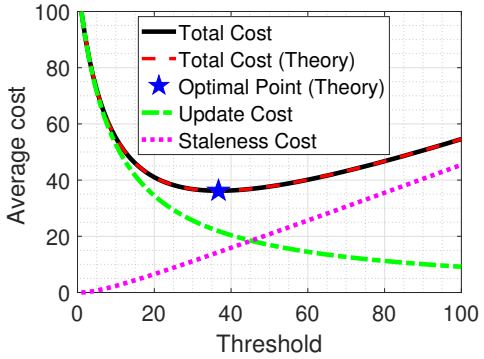
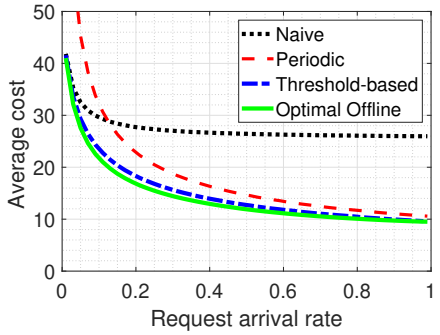


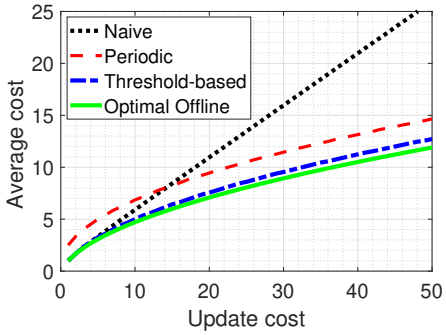
Fig. 4: Average cost under threshold-based policies with different thresholds, where $\lambda = 0.1$ and $p = 100$.

We first evaluate the performance of threshold-based policies with different thresholds in Fig. 4. The setting of the simulations is as follows. The request arrival process is Bernoulli with rate $\lambda = 0.1$, and the update cost is $p = 100$. The simulation results are the average of 100 simulation runs, where each run consists of $N = 10^4$ requests (which is our default setting for the synthetic simulations). We also include a breakdown of the results in terms of average staleness cost $\bar{a} \triangleq \sum_{j=1}^{N} f(\Delta(r_j))/N$ and average update cost $\bar{p} \triangleq pU^{\pi}(N)/N$. We observe that the simulation results of average total cost under threshold-based policies perfectly match the theoretical results in Example 1. Clearly, as the threshold increases, the update cost decreases, but the staleness cost increases. This is as expected because a higher threshold leads to less frequent updates, which results in a smaller update cost but a larger staleness cost. As a result, the average total cost, which is the sum of the two, first decreases and then increases. The optimal cost $C^{\pi(\tau^*)} \approx 36.22$ is achieved at $\tau^* = \{\arg\min_{\{\lfloor \tau' \rfloor, \lceil \tau' \rceil\}} C^{\pi(\tau)}\} = 37$, where $\tau' = (\sqrt{2p\lambda - \lambda + 1} + \lambda - 1)/\lambda \approx 36.72$.

Next, we compare the performance of the optimal threshold-based policy with several baselines. We consider three baselines: (i) a naive policy, (ii) periodic policies, and (iii) the optimal offline policy. The naive policy is a capped reactive threshold-based policy with a threshold being equal to $\Delta^* = \lceil p \rceil$. A periodic policy has a positive integer period d and updates the data every d time-slots, i.e., $u_i = id$ for $i = 1, 2, \ldots$. Note that a periodic policy is not a reactive policy. Following a similar argument in the proof of Theorem 2, we can show that the average cost under a periodic policy with period d is $(p + \lambda d(d-1)/2)/\lambda d$. In the comparisons, we only consider the optimal periodic policy with $d^* = \left\lceil \sqrt{2p/\lambda} \right\rceil$. The optimal offline policy has the exact knowledge of all the request arrival times and is obtained based on the dynamic programming approach. Hence, the average cost under an optimal offline policy can be viewed as a lower bound of all online policies.



(a) Fixed update cost $p = 50$



(b) Fixed request arrival rate $\lambda = 0.5$

Fig. 5: Performance comparisons of different policies with different request arrival rate λ and different update cost p, respectively.

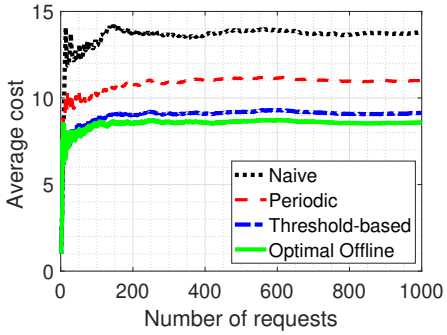


Fig. 6: Performance comparisons of different policies using trace dataset, where update cost $p = 25$ and request arrival rate $\lambda = 0.4$.

Fig. 5a shows the results for the setting with a fixed update cost $p = 50$ and a varying request arrival rate λ. We observe that the optimal threshold-based policy outperforms all the other online policies and is very close to the optimal offline policy. When the request arrival rate is small, the optimal periodic policy performs poorly. This is because Bernoulli process with a small rate λ results in a larger mean (i.e., $1/\lambda$) and a larger variance (i.e., $(1-\lambda)/\lambda^2$) of the inter-arrival time of the requests. Hence, the inter-arrival time of the requests is usually larger and more random. In this case, a periodic policy that updates the data at fixed time instants resulting in a larger staleness cost. On the other hand, the interarrival time of requests is large when the rate λ approaches 0, resulting in a large AoI (and thus a high staleness cost) when the requests arrive. All the capped reactive policies as well as the offline optimal policy would make an update decision for each request, making their average cost close to the update cost p. This aligns well with our theoretical result when we let $\lambda = 0$ in Eq. (13). When the request arrival rate becomes larger, the performance of the optimal periodic policy improves and is close to that of the optimal threshold-based policy. This is because a large request arrival rate λ leads to a small period d^*. In this case, the requests arrive more frequently, and the updates also occur more frequently. Hence, the staleness cost becomes small, and the update cost becomes dominant. We also observe that the naive policy performs poorly compared to the other policies when λ is large. This is because the naive policy is agnostic about the request arrival rate. The update period under a naive policy is roughly equal to p regardless of the request arrival rate. However, when λ becomes large, there could be many more requests arriving during an update interval of length p, which results in a large staleness cost. In addition, when the request arrival rate λ approaches 1, the optimal threshold-based policy and the optimal periodic policy have the same optimal threshold/period (i.e., $\tau^* = d^* = \lceil\sqrt{2p}\rceil$) as well as the same average cost (i.e., $\sqrt{2p} - 1/2$), and their knowledge about the future request arrival process is almost the same as the optimal offline policy (i.e., there is a request arriving in every time-slot), so their performance is also very close.

Fig. 5b shows the results for the setting with a fixed request arrival rate $\lambda = 0.5$ and a varying update cost p. We observe that the optimal threshold-based policy again outperforms all the other online policies and performs closely to the optimal offline policy. The optimal periodic policy performs poorly because it is not a reactive policy, some of its updates may be unnecessary, which results in a higher update cost. We also observe that as the update cost increases, the naive policy performs much worse compared to the other policies. This is because the average cost under a naive policy increases at a rate of $O(p)$, while the average cost under the other policies increases at a rate of $O(\sqrt{p})$.

In Fig. 6, we also compare the performance of different policies using the real trace dataset [31]. The trace dataset collects around 700 billion user requests (each contains a timestamp, anonymized key, operation, etc.; see details in [31]) from 54 Twemcache clusters, which are the in-memory caching used by Twitter. In order to simplify the analysis and presentation, we focus on the user request arrival times at the cache of the 26th Twemcache cluster, whose request arrival rate is about 0.4. The request arrival process is no longer Bernoulli. In Fig. 6, we assume that the update cost is $p = 25$ and show the performances of different policies using the trace dataset in the first 10^3 requests. For the optimal offline policy, we still apply the dynamic programming to obtain the optimal update times. For the naive policy, we simply let its threshold equal the update cost of 25. For the threshold-based policy, we obtain the optimal threshold $\tau^* = 9$ by plugging $\lambda = 0.4$ into Eq. (12). Similarly, we obtain the optimal period $d^* = 11$ for the periodic policy. We observe that the optimal threshold-based policy still outperforms the other online policies even though the request arrival process is now non-Bernoulli.

VIII. Conclusion

In this paper, we considered a fundamental tradeoff between the data freshness and the update cost in a time-sensitive information-update system. We provided useful guidelines for the design of update policies. Assuming Bernoulli request arrival process, we also proposed a threshold-based update

policy and proved its optimality. Our simulations based on both synthetic data and real traces corroborated the theoretical results and showed that the optimal threshold-based policy outperforms the baseline policies. For future work, one interesting direction would be to consider more general settings where users are interested in multiple contents.

REFERENCES

[1] Waze mobile app. [Online]. Available: https://www.waze.com/

[2] Gasbuddy mobile app. [Online]. Available: https://www.gasbuddy.com/

[3] B. Li and J. Liu, "Achieving information freshness with selfish and rational users in mobile crowd-learning," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 5, pp. 1266–1276, 2021.

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

[5] F. Li, Y. Sang, Z. Liu, B. Li, H. Wu, and B. Ji, "Waiting but not aging: Optimizing information freshness under the pull model," *IEEE/ACM Transactions on Networking*, vol. 29, no. 1, pp. 465–478, 2021.

[6] Z. Liu and B. Ji, "Towards the tradeoff between service performance and information freshness," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.

[7] Y. Ling and J. Mi, "An optimal trade-off between content freshness and refresh cost," *Journal of Applied Probability*, vol. 41, no. 3, p. 721–734, 2004.

[8] Market data from morgan stanley. [Online]. Available: https://us.etrade.com/l/f/disclosure-library/market-data

[9] Y. Sang, B. Li, and B. Ji, "The power of waiting for more than one response in minimizing the age-of-information," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–6.

[10] L. Huang and E. Modiano, "Optimizing age-of-information in a multi-class queueing system," *arXiv preprint arXiv:1504.05103*, 2015.

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

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

[13] M. Costa, M. Codreanu, and A. Ephremides, "Age of information with packet management," in *2014 IEEE International Symposium on Information Theory*, 2014, pp. 1583–1587.

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

[15] J. Yun, C. Joo, and A. Eryilmaz, "Optimal real-time monitoring of an information source under communication costs," in *2018 IEEE Conference on Decision and Control (CDC)*, 2018, pp. 4767–4772.

[16] V. Tripathi and E. Modiano, "An online learning approach to optimizing time-varying costs of aoi," in *Proceedings of the Twenty-second International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, 2021, pp. 241–250.

[17] K. Saurav and R. Vaze, "Minimizing the sum of age of information and transmission cost under stochastic arrival model," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[18] 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.

[19] N. Lu, B. Ji, and B. Li, "Age-based scheduling: Improving data freshness for wireless real-time traffic," in *Proceedings of the eighteenth ACM international symposium on mobile ad hoc networking and computing*, 2018, pp. 191–200.

[20] R. D. Yates, P. Ciblat, A. Yener, and M. Wigger, "Age-optimal constrained cache updating," in *2017 IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 141–145.

[21] J. Zhong, R. D. Yates, and E. Soljanin, "Two freshness metrics for local cache refresh," in *2018 IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 1924–1928.

[22] M. Bastopcu and S. Ulukus, "Cache freshness in information updating systems," in *2021 55th Annual Conference on Information Sciences and Systems (CISS)*, 2021, pp. 01–06.

[23] H. Tang, P. Ciblat, J. Wang, M. Wigger, and R. D. Yates, "Cache updating strategy minimizing the age of information with time-varying files' popularities," in *2020 IEEE Information Theory Workshop (ITW)*, 2021, pp. 1–5.

[24] A. Arafa, J. Yang, and S. Ulukus, "Age-minimal online policies for energy harvesting sensors with random battery recharges," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.

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

[26] B. T. Bacinoglu, Y. Sun, E. Uysal–Bivikoglu, and V. Mutlu, "Achieving the age-energy tradeoff with a finite-battery energy harvesting source," in *2018 IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 876–880.

[27] Y. Dong, P. Fan, and K. B. Letaief, "Energy harvesting powered sensing in iot: Timeliness versus distortion," *IEEE Internet of Things Journal*, vol. 7, no. 11, pp. 10 897–10 911, 2020.

[28] V. Tripathi, R. Talak, and E. Modiano, "Age of information for discrete time queues," *arXiv preprint arXiv:1901.10463*, 2019.

[29] Z. Liu, B. Li, Z. Zheng, Y. T. Hou, and B. Ji, "Towards Optimal Tradeoff Between Data Freshness and Update Cost in Information-update Systems," *arXiv e-prints*, p. arXiv:2204.14123, Apr. 2022.

[30] S. M. Ross, *Introduction to stochastic dynamic programming*. Academic press, 2014.

[31] J. Yang, Y. Yue, and K. V. Rashmi, "A large scale analysis of hundreds of in-memory cache clusters at twitter," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 191–208.

[32] S. M. Ross, *Introduction to probability models*. Academic press, 2014.

APPENDIX

A. Proof of Theorem 1

Proof. Our proof includes two steps: 1) We study a discounted MDP and show that the optimal value function of the discounted MDP is non-decreasing in the initial state s; 2) Based on the optimal value function of the discounted MDP, we derive the Bellman equation of the average expected cost and show that it has a threshold-based structure, where the threshold is based on the AoI. This implies that there exists an optimal threshold-based stationary capped reactive policy for the average expected cost.

Step 1): In general, the derivation and properties of the Bellman equation of the average expected cost are not easy to obtain, and we usually rely on the study of the discounted MDP to get some insights towards the design of an optimal policy [30].

The expected total α-discounted cost of a capped reactive policy $\pi \in \Pi^{R+}$ is defined as

$$C_\alpha^\pi(s) \triangleq \mathbb{E}_\pi \left[\sum_{n=1}^{\infty} \alpha^{n-1} c(s_n, a_n) | s_1 = s \right], \qquad (14)$$

where $0 \leq \alpha < 1$ is the discount factor. Here, $C_\alpha^\pi(s)$ is well defined, given that for any n, we have $\mathbb{E}_\pi[c(s_n, a_n)|s_1 = s] \leq p$ under a capped reactive policy π, and thus, we have

$$C_\alpha^\pi(s) \leq \sum_{n=1}^{\infty} \alpha^{n-1} p = \frac{p}{1-\alpha}. \qquad (15)$$

Let $C_\alpha(s) \triangleq \min_\pi C_\alpha^\pi(s)$ be the optimal value function. Then, we can obtain the Bellman equation of the α-discounted MDP with $C_\alpha(s)$ [30], which is

$$C_\alpha(s) = \min_{a \in \mathcal{A}_s} \left\{ c(s,a) + \alpha \sum_{z \in S} p(z \mid s, a) C_\alpha(z) \right\}. \qquad (16)$$

The Bellman equation Eq. (16) states that the value of the initial state s (i.e., $C_\alpha(s)$) equals the expected return of the best action, which is the discounted expected value of the next state (i.e., $\alpha \sum_{z\in S} p(z \mid s,a) C_\alpha(z)$), plus the immediate cost along the way (i.e., $c(s,a)$). In the following, we show that $C_\alpha(s)$ is non-decreasing in s. This property enables us to show that the Bellman equation of the average cost (i.e., Lemma 4) has a threshold-based structure.

Lemma 3. *The optimal value function $C_\alpha(s)$ is non-decreasing in the initial state s.*

Proof. Our proof idea is to construct a sequence $\{C_{\alpha,n}(s)\}$ that is non-decreasing in s for any n, where $C_{\alpha,n}(s)$ is the minimal expected discounted cost in an n-stages problem. Then, we show that $C_\alpha(s) = \lim_{n\to\infty} C_{\alpha,n}(s)$, which implies that $C_\alpha(s)$ is also non-decreasing in s.

First, we show how to construct the sequence $\{C_{\alpha,n}(s)\}$. Consider an n-stage problem of our α-discounted MDP. Denote the minimal expected discounted cost of this n-stage problem by

$$C_{\alpha,n}(s) \triangleq \min_{a\in A_s}\left\{c(s,a) + \alpha\sum_{z\in S} p(z\mid s,a) C_{\alpha,n-1}(z)\right\}$$
$$= \begin{cases} p + \alpha\sum\limits_{z=1}^{\infty}(1-\lambda)^{z-1}\lambda C_{\alpha,n-1}(z), \text{ if } s\geq\Delta^*; \\ \min\left\{p + \alpha\sum\limits_{z=1}^{\infty}(1-\lambda)^{z-1}\lambda C_{\alpha,n-1}(z),\right. \\ \left. f(s) + \alpha\sum\limits_{z=s+1}^{\infty}(1-\lambda)^{z-s-1}\lambda C_{\alpha,n-1}(z)\right\}, \text{ if } s<\Delta^*, \end{cases}$$

where the terminal cost is $C_{\alpha,1}(s) \triangleq \min\{p, f(s)\}$.

Then, we prove by induction that our constructed sequence $\{C_{\alpha,n}(s)\}$ is non-decreasing in s for any n. Obviously, $C_{\alpha,1}(s)$ is non-decreasing in s. We assume that $C_{\alpha,n-1}(s)$ is non-decreasing in s. Next, we show that $C_{\alpha,n}(s)$ is non-decreasing in s. When $s\geq\Delta^*$, $C_{\alpha,n}(s)$ is a constant and is independent of s. On the other hand, when $s<\Delta^*$, to better present our discussion, we denote

$$c \triangleq p + \alpha\sum_{z=1}^{\infty}(1-\lambda)^{z-1}\lambda C_{\alpha,n-1}(z)$$

and

$$C'_{\alpha,n-1}(s) \triangleq f(s) + \alpha\sum_{z=s+1}^{\infty}(1-\lambda)^{z-s-1}\lambda C_{\alpha,n-1}(z)$$
$$= f(s) + \alpha\sum_{k=1}^{\infty}(1-\lambda)^{k-1}\lambda C_{\alpha,n-1}(s+k),$$

where the last step follows by setting $k = z-s$. We want to show that $C'_{\alpha,n-1}(s)$ is non-decreasing in s. To see this, consider two states: $j\geq i>0$. Then, we have

$$C'_{\alpha,n}(j) - C'_{\alpha,n}(i)$$
$$= f(j) - f(i)$$
$$+ \alpha\sum_{k=1}^{\infty}(1-\lambda)^{k-1}\lambda(C_{\alpha,n-1}(j+k) - C_{\alpha,n-1}(i+k)).$$

Since $f(s)$ is non-decreasing in s, we have $f(j)\geq f(i)$. By inductive hypothesis that $C_{\alpha,n-1}(s)$ is non-decreasing in s, we have $C_{\alpha,n-1}(j+k)\geq C_{\alpha,n-1}(i+k)$ for any $k>0$. Therefore, $C'_{\alpha,n}(j) - C'_{\alpha,n}(i)\geq 0$, i.e., $C_{\alpha,n}(s)$ is non-decreasing in s. This, along with c being a constant, implies that $C_{\alpha,n}(s) = \min\{c, C'_{\alpha,n-1}(s)\}$ is also non-decreasing in s when $s<\Delta^*$. Till this point, we have shown that $C_{\alpha,n}(s)$ is non-decreasing in s when $s<\Delta^*$ and when $s\geq\Delta^*$, respectively. In addition, since $C_{\alpha,n}(s)$ achieves a smaller value when $s<\Delta^*$ (i.e., $\min\left\{c, C'_{\alpha,n-1}(s)\right\}$) compared to the case of $s\geq\Delta^*$ (i.e., c), this implies that $C_{\alpha,n}(s)$ is non-decreasing in s for any n.

Finally, given any non-negative integer s, by the definition of $C_{\alpha,n}(s)$ and the fact that the cost in each stage is non-negative, we know that $C_{\alpha,n}(s)$ is non-descreasing in n. In addition, $C_{\alpha,n}(s)$ is bounded. Indeed, the cost in each stage is bounded by the update cost p under our considered policy. As such, $C_{\alpha,n}(s)\leq\sum_{i=1}^{n}\alpha^{i-1}p = (1-\alpha^n)p/(1-\alpha)\leq p/(1-\alpha)$. Therefore, by monotone convergence theorem, we have

$$\lim_{n\to\infty} C_{\alpha,n}(s) = C_\alpha(s), \tag{17}$$

holding for any state $s=0,1,2,\ldots$. This also implies that $C_\alpha(s)$ is non-decreasing in s (see our detailed proof in [29]). □

Step 2): With the optimal value function of the α-discounted MDP (i.e., $C_\alpha(s)$), we can derive the Bellman equation of the average cost as follows.

Lemma 4. *Let $h(s) \triangleq \lim_{\alpha\to 1}[C_\alpha(s) - C_\alpha(1)]$ and $g \triangleq \lim_{\alpha\to 1}(1-\alpha)C_\alpha(1)$. Then, the Bellman equation of the average cost is given by the following:*

$$h(s) + g =$$
$$\begin{cases} p + \sum\limits_{z=1}^{\infty}(1-\lambda)^{z-1}\lambda h(z), \text{ if } s\geq\Delta^*; \\ \min\left\{p + \sum\limits_{z=1}^{\infty}(1-\lambda)^{z-1}\lambda h(z),\right. \\ \left. f(s) + \sum\limits_{z=s+1}^{\infty}(1-\lambda)^{z-s-1}\lambda h(z)\right\}, \text{ if } s<\Delta^*. \end{cases} \tag{18}$$

We provide the proof of Lemma 4 in our technical report [29] and explain the key ideas in the following. First, given the definitions of $h(s)$ and g, we show that Eq. (18) does hold. To this end, we define $h_\alpha(s) \triangleq C_\alpha(s) - C_\alpha(1)$ and substitute $h_\alpha(s)$ into the Bellman equation of the α-discounted MDP Eq. (16). Then, we prove that we can find a sequence $\{\alpha_m\} \to 1$ such that $\lim_{m\to\infty} h_{\alpha_m}(s) = h(s)$ for any s and $\lim_{m\to\infty}(1-\alpha_m)C_{\alpha_m}(1) = g$. Taking the limit $m\to\infty$ on both sides of the Bellman equation of $h_\alpha(s)$, we obtain Eq. (18). Second, we show that Eq. (18) is the Bellman equation for the average expected cost. This can be done by applying the same techniques used in [30, Chapter V, Theorem 2.1].

Next, we show that the Bellman equation Eq. (18) has a threshold structure, which guides us to find the optimal threshold-based stationary capped reactive policy.

Assume that the current state is s. Based on the Bellman equation Eq. (18), it is optimal to update when $s \geq \Delta^*$; and when $s < \Delta^*$, it is optimal to update if

$$\begin{aligned} f(s) + \alpha \sum_{z=s+1}^{\infty} (1-\lambda)^{z-s-1} \lambda h(z) \geq \\ p + \alpha \sum_{z=1}^{\infty} (1-\lambda)^{z-1} \lambda h(z), \end{aligned} \tag{19}$$

where the right hand side is a constant. It is easy to check that $h(s)$ is non-decreasing in s given that $C_\alpha(s)$ is non-decreasing in s. Hence, we can find s^* as follows:

$$\begin{aligned} s^* \triangleq \min \Bigg\{ s : f(s) + \alpha \sum_{z=s+1}^{\infty} (1-\lambda)^{z-s-1} \lambda h(z) \geq \\ p + \alpha \sum_{z=1}^{\infty} (1-\lambda)^{z-1} \lambda h(z) \Bigg\}. \end{aligned} \tag{20}$$

Set $s^* = \Delta^*$ when $s^* \geq \Delta^*$. Now consider a capped reactive policy $\pi^* \in \Pi^{R+}$: upon receiving a request, policy π^* updates the data if the current state is no smaller than s^*; otherwise, it does not update the data and replies with the current local data. Clearly, policy π^* is a stationary threshold-based policy. Besides, policy π^* selects the action that minimizes the right hand side of the Bellman equation Eq. (18) for any state. Thus, it is an optimal policy [30, Chapter V, Theorem 2.1]. □

B. Proof of Theorem 2

Proof. We start with some additional notations. For the k-th update interval $[u_{k-1}^\pi + 1, u_k^\pi]$, we use N_k and C_k to denote the number of requests that arrive in $[u_{k-1}^\pi + 1, u_k^\pi]$ and the total cost serving these N_k requests, respectively.

Since the request arrival process is Bernoulli, under a threshold-based update policy, the lengths of update intervals are *i.i.d.*, so the update process is a renewal process. By the ergodicity of the process, the average cost can be rewritten as

$$\bar{C}^{\pi(\tau)} = \mathbb{E}[C_k] / \mathbb{E}[N_k]. \tag{21}$$

To calculate $\mathbb{E}[N_k]$, we consider the requests that arrive in $[u_{k-1}^\pi + 1, u_k^\pi]$. Apparently, there is only one request in $[u_{k-1}^\pi + \tau, u_k^\pi]$, which arrives exactly at u_k^π because of the threshold-based policy. Besides, the expected number of requests arriving in $[u_{k-1}^\pi + 1, u_{k-1}^\pi + \tau - 1]$ is $\lambda(\tau - 1)$ according to the Bernoulli process. Therefore, we have

$$\mathbb{E}[N_k] = \lambda(\tau - 1) + 1. \tag{22}$$

The total cost in an update interval is composed of an update cost and some staleness costs, i.e.,

$$C_k = p + \sum_{n=1}^{N_k} f(\Delta(r_n)) \mathbb{1}_{\{N_k > 0\}}, \tag{23}$$

where $\mathbb{1}_{\{\cdot\}}$ is the indicator function. Here, we slightly abuse the notation of n and use it to denote the index of requests arriving in $[u_{k-1}^\pi + 1, u_k^\pi]$ (i.e., r_n is the arrival time of the n-th request in $[u_{k-1}^\pi + 1, u_k^\pi]$). The staleness costs can be rewritten as

$$\begin{aligned} \sum_{n=1}^{N_k} f(\Delta(r_n)) \mathbb{1}_{\{N_k > 0\}} &= \sum_{n=1}^{\infty} f(\Delta(r_n)) \mathbb{1}_{\{n \leq N_k\}} \\ &= \sum_{n=1}^{\infty} f(\Delta(r_n)) \mathbb{1}_{\{\Delta(r_n) \leq \tau - 1\}}. \end{aligned} \tag{24}$$

For the expected staleness cost of the n-th arrival of the requests, we have

$$\mathbb{E}[f(\Delta(r_n)) \mathbb{1}_{\{\Delta(r_n) \leq \tau - 1\}}] = \sum_{t=1}^{\tau-1} f(\Delta(t)) p_n(t), \tag{25}$$

where by slightly abusing the notation, we let t denote the index of time-slot after u_{k-1}^π, and let $p_n(t)$ be the probability mass function that the n-th request arrives at time-slot t. Here $p_n(t)$ follows the negative binomial distribution [32], i.e.,

$$p_n(t) = \begin{cases} \binom{t-1}{n-1} \lambda^n (1-\lambda)^{t-n}, & t = n, n+1, \cdots; \\ 0, & \text{otherwise.} \end{cases} \tag{26}$$

Plugging Eq. (26) into Eq. (25), we obtain

$$\begin{aligned} &\mathbb{E}[f(\Delta(r_n)) \mathbb{1}_{\{\Delta(r_n) \leq \tau - 1\}}] \\ &= \sum_{t=1}^{\tau-1} f(\Delta(t)) \binom{t-1}{n-1} \lambda^n (1-\lambda)^{t-n}, \end{aligned} \tag{27}$$

where the item in the summation equals 0 when $t < n$.

We rewrite the expected total cost in an update interval as

$$\begin{aligned} \mathbb{E}[C_k] &= p + \sum_{n=1}^{N_k} \mathbb{E}[f(\Delta(r_n)) \mathbb{1}_{\{N_k > 0\}}] \\ &= p + \sum_{n=1}^{\infty} \sum_{t=1}^{\tau-1} f(\Delta(t)) \binom{t-1}{n-1} \lambda^n (1-\lambda)^{t-n} \\ &\overset{(a)}{=} p + \sum_{t=1}^{\tau-1} \sum_{n=1}^{\infty} f(\Delta(t)) \binom{t-1}{n-1} \lambda^n (1-\lambda)^{t-n} \\ &\overset{(b)}{=} p + \sum_{t=1}^{\tau-1} \sum_{n=1}^{t} f(\Delta(t)) \binom{t-1}{n-1} \lambda^n (1-\lambda)^{t-n} \\ &= p + \sum_{t=1}^{\tau-1} f(\Delta(t)) \lambda \sum_{n=1}^{t} \binom{t-1}{n-1} \lambda^{n-1} (1-\lambda)^{t-n} \\ &\overset{(c)}{=} p + \sum_{t=1}^{\tau-1} f(\Delta(t)) \lambda (\lambda + (1-\lambda))^{t-1} \\ &= p + \sum_{t=1}^{\tau-1} f(\Delta(t)) \lambda \\ &\overset{(d)}{=} p + \sum_{t=1}^{\tau-1} f(t) \lambda, \end{aligned} \tag{28}$$

where we interchange the order of summation in (a) because the sum is finite, (b) is because the maximal number of requests cannot exceed the length of the update interval, (c) comes from the binomial theorem, and (d) is due to the definition of the AoI. Finally, plugging Eqs. (22) and (28) into Eq. (21) gives Eq. (11). □