Identifying Key Tag Distribution in Large-Scale RFID Systems

Yanyan Wang[†‡] Jia Liu[§] Shen-Huan Lyu[†‡] Zhihao Qu[†‡*] Bin Tang[†‡] Baoliu Ye[§]
[†]Key Laboratory of Water Big Data Technology of Ministry of Water Resources, Hohai University, China
[‡]College of Computer Science and Software Engineering, Hohai University, China
[§]State Key Laboratory for Novel Software Technology, Nanjing University, China
{yanyan.wang, lvsh, quzhihao, cstb}@hhu.edu.cn, {jialiu, yebl}@nju.edu.cn

***Abstract*—With the proliferation of RFID-enabled applications, multiple readers are required for the complete coverage of numerous tags in a large-scale RFID system. In this scenario, we sometimes pay more attention to a subset of tags instead of all, which are referred to as key tags. In this paper, we study an under-investigated problem *key tag distribution identification*, which aims to identify which key tags are beneath which readers. This is crucial for efficiently managing specific items of interest, which can quickly pinpoint key tags and help RFID readers covering these tags collaborate to improve the tag inventory efficiency. Since key tags typically make up a small part of all tags, it is time consuming to deal with all tags in the traditional way. We propose a protocol called Kadept that identifies the key tag distribution by using a sophisticatedly designed filter that teases out key tags as well as assigns each of them a singleton slot for response. With this design, a great number of trivial (non-key) tags will keep silent and free up bandwidth resources for key tags, and each key tag is sorted in a collision-free way and can be identified with only 1-bit response, which significantly improves the time efficiency. We theoretically analyze how can we optimize protocol parameters of Kadept and conduct extensive simulations under different tag distribution scenarios. Compared with the state-of-the-art, Kadept can improve the time efficiency by a factor of 3.7×, when the ratio of key tags to all tags is 0.1.**

I. Introduction

Radio Frequency IDentification (RFID) technology has brought great benefits to a variety of applications, including but not limited to warehouse inventory [1], [2], [3], [4], human-machine interface [5], [6], [7], [8], object tracking and authentication [9], [10], [11], [12], [13]. In these applications, each object is attached with an RFID tag that has a unique ID to indicate the tagged object. The tag can communication with an RFID reader through backscatter, which is free of batteries. Since the communication distance between a reader and a tag is limited to a few meters, with the proliferation of RFID-enabled applications, multiple RFID readers must be used to ensure the full coverage of desired inventory zones.

In a large-scale RFID system with multiple readers, we sometimes focus only on a subset of tags rather than all, which are referred to as key tags. For example, in a shopping mall with tens of thousands of goods, the staff might need to conduct inventory over a specific category of goods, where these specific tags are treated as key tags and the left are trivial tags out of interest. In another example, given a limited bandwidth resource, the shop owner would like to pay more attention to valuable goods (key tags) rather than cheap ones in a retail store [14]; a frequent check over key tags is necessary. In such cases, fast identifying key tags' distribution allows only readers that cover key tags to zoom into the corresponding regions to communicate with them, ensuring efficient key tag management. For clarity, we present a toy example shown in Fig. 1. If the distribution of key tags k_{1-8} is known, we can schedule readers $R1$ and $R3$ to work concurrently, with each of them handling only the key tags that it covers. If we do not have this distribution knowledge, all readers R_{1-4} need to work in a round-robin scheduling, with each reader querying key tags multiple times.

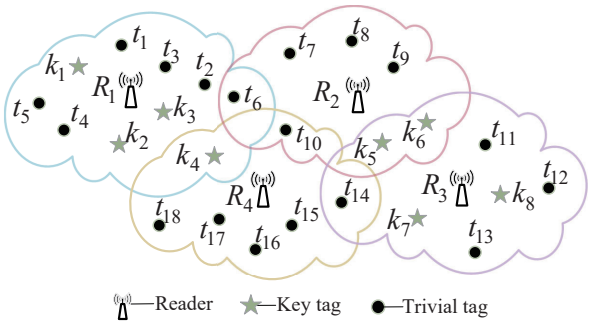


Fig. 1: The system model of key tag distribution.

In this paper, we study the problem of key tag distribution identification that quickly explores which key tags are beneath which reader's coverage, with the knowledge of all tag IDs a priori. This information plays a crucial role in various inventory operations. For example, it can help detect missing key tags [15] without the assumption that each reader knows its local tag distribution. It can also facilitate key tags information collection [16] by appropriately scheduling multiple readers that cover key tags.

An intuitive solution to this problem is to identify the distribution of all tags by running the tag identification protocols [17], [18] or the tag distribution identification protocol IB [19]. However, these protocols suffer from long time delays because they require all tags (including a great number of trivial tags) to transmit data, which is a waste of bandwidth resources. Besides global identification, tag polling is a more efficient solution: each reader just in turn broadcasts each key tag's

*Corresponding author: Zhihao Qu

ID to activate the key tag (if exist), which avoids the tag competition caused by trivial tags. However, this solution need to transmit tag IDs of key tags and breach the privacy, which is forbidden in some privacy-sensitive RFID systems [20]. IPP, proposed in [21], is an advanced polling protocol that achieves remarkable efficiency by reducing the polling vector from 96 bits to only 1.6 bits. However, in our problem, each reader still needs to poll all tags to identify the key tag information.

In light of this, we propose a key tag distribution identification protocol (*Kadept*), which targets at quickly identifying the distribution of key tags, without transmitting any tag IDs. Kadept consists of four operations: 1) Filtering. It constructs a Cuckoo filter to tease out key tags and avoid the interference of trivial tags. 2) Assigning. Kadept assigns a unique, continuous slot for each key tag to make key tags free of collision and respond to the reader's query without time waste. 3) Separating. Kadept leverages flag technology to let each tag know whether it is a key tag or not so that key tags can be appropriately managed in subsequent inventory operations. 4) Identifying. Kadept identifies key tag distribution: non-conflicting tags are identified in parallel, while conflicting tags are scheduled collision-free. We theoretically analyze how can we optimize the protocol parameters of Kadept and evaluate its performance via extensive simulations. The results show that Kadept is far superior to the baseline IB [19] and the sate-of-the-art polling protocol IPP [21]. For example, in a multi-reader RFID system with 10,000 tags (9000 trival tags and 1000 key tags), Kadept reduces the execution time from 19.2s to 4.09s, producing a 3.7× performance gain.

The remainder of this paper is organized as follows. Section II gives the preliminary. Section III details the Kadept protocol. Section IV presents the theoretical analysis of Kadept. Section V evaluates the protocol performance. Section VI introduces the related work. Section VII concludes this paper.

II. Preliminary

A. System Model

We consider a large-scale RFID system comprising a back-end server, multiple readers, and a great number of tags. Each tag has a unique ID that exclusively indicates the item it is attaches to. The tags use backscattering to communicate with readers, limiting the communication distance between a reader and a tag to a few meters. To cover all tags in the surveillance regions, multiple readers are deployed with many overlapped interrogation regions and interference regions. Each reader covers a portion of the tags, while each tag may be covered by one or more readers. The server maintains an item list of all tag IDs but has no information about the tag distribution.

The communication between the readers and tags follows *Reader Talks First* mode as defined in C1G2 standard [22]. In this mode, one reader initiates a query with a continuous RF waveform (CW), energizing the tags it covers. The tags then transmit their information by reflecting the modulated CW back to the reader. We use t_{ID} and t_s to represent the lengths of two types of time slots: the former is for transmitting tag ID, and the latter is for transmitting 1-bit information. Similar to [23], the readers broadcast ACK_l to label tags. The time for transmitting an ACK is referred to as t_{ack}.

B. Problem Definition

Consider a multi-reader RFID system. The set of readers in the system is denoted as $\mathcal{R} = \{R_1, \ldots, R_m\}$. The topology of readers can be depicted as a graph $\mathcal{G} = (\mathcal{R}, \mathcal{E})$, where $\mathcal{E}$ is the set of edges each connecting two readers which have overlapped monitor areas; these two readers are neighbors. For reader $R_i (R_i \in \mathcal{R}, 1 \leq i \leq m)$, its neighbor and non-neighbor reader sets are denoted as $\Gamma(R_i)$ and $\Upsilon(R_i)$, respectively. The tags are divided into two sets: the key tag set $K = \{k_1, \ldots, k_x\}$ and the trivial tag set $T = \{t_1, \ldots, t_y\}$. Symbol K_i represents the set of key tags under the coverage region of R_i. Since each contentious key tag in the overlapped region is covered by two or more readers, we have $\bigcup_{i=1}^{m} K_i = K$ and $|K_i \bigcap K_j| \geq 0 \ (i \neq j)$. With the knowledge of reader topology and tag IDs, this paper is to identify which key tags are beneath which readers, i.e., identifying K_i for R_i.

III. Kadept Protocol

A. Overview

The idea of Kadept is to quickly identify the distribution of key tags by eliminating the interference of trivial tags and assigning slots for key tags to transmit information. To achieve this, Kadept elaborately constructs a new Cuckoo filter, which can assign each inserted key tag a unique slot to transmit information besides filtering trivial tags. Due to the instinct of false positives in Cuckoo filters, some trivial tags may pass the filter and simultaneously reply to a reader with key tags, which causes the failure to identify the distribution of these collided key tags. Therefore, to identify the distribution of all key tags, Kadept may require multiple rounds of execution.

In each round, Kadept generally consists of three phases: *constructing phase* that constructs a Cuckoo filter based on key tags; *filtering-assigning-separating phase* that utilizes the Cuckoo filter to filter trivial tags, assign each key tag a unique and continuous slot, and separate key tags from trivial tags; *identifying phase* that identifies which key tag is under which reader's coverage by checking the status of each assigned slot. In Kadept, if a reader is sure about whether a key tag is under its coverage or not, we say that this key tag is identified by the reader, or unidentified otherwise. Details are given below.

B. Constructing Phase

Consider an arbitrary reader R_i. Initially, R_i has no knowledge of its local key tags and thus uses all key tags to construct a Cuckoo filter, denoted by $\mathcal{CF}$. The basic unit of $\mathcal{CF}$ is called a bucket, and $\mathcal{CF}$ consists of an array of buckets. $\mathcal{CF}$ stores key tags' fingerprints to reduce the filter size as well as avoid leaking key tag IDs. In the construction process, however, $\mathcal{CF}$ temporarily stores tag IDs, to restore and rehash an original key tag in a bucket to find its alternate location. Next, we design a new construction scheme to extend $\mathcal{CF}$ beyond filtering to perform slot assignment operations. By

doing this, Kadept ensures the inserted key tags can respond to the reader one after another without empty or collision slots.

For a key tag $k_j (k_j \in K, 1 \leq j \leq x)$, the reader calculates its constant-sized fingerprint F_j and the indexes $H_j^{\hbar} (1 \leq \hbar \leq h)$ of the h candidate buckets as follows:

$$\begin{aligned} F_j &= H(ID_{k_j}, s_f) \mod 2^d, \\ H_j^{\hbar} &= H(ID_{k_j}, s_{\hbar}) \mod f_c, \end{aligned} \tag{1}$$

where $H(\cdot)$ is hash function. Since each tag actually holds one hash function, which is shared by readers and tags, the reader broadcasts h hash seeds to tags to mimic h hash functions. s_f and $s_{\hbar} (1 \leq \hbar \leq h)$ are the hash seeds used to calculate the fingerprints and candidate buckets, respectively; f_c and d are the lengths of $\mathcal{CF}$ and its fingerprint, respectively. Let $\mathcal{H}_j$ be the candidate bucket set of k_j, and we have $\mathcal{H}_j = \{H_j^1, \cdots, H_j^h\}$. Before inserting k_j into $\mathcal{CF}$, we classify the fingerprints of all key tags and then insert k_j following this classification. According to Eq. (1), the fingerprints in the set $\{0, \ldots, 2^d\}$ can be divided into three categories: *useless fingerprints* mapped by no key tags, *singleton fingerprints* mapped by exactly one key tag, and *collision fingerprints* mapped by more than one key tag. We refer to the key tags with the same fingerprint as the same group. The reader R_i inserts k_j into $\mathcal{CF}$ in terms of the status of F_j, and the specific steps are as follows.

F_j is a singleton fingerprint. If only some candidate h buckets are assigned tags, R_i randomly selects an empty bucket and inserts k_j by putting ID_{k_j} into it. Otherwise, R_i picks one of the h buckets at random and inserts k_j into it, replacing a key tag previously in this bucket.

F_j is a collision fingerprint. As described above, key tags with the same fingerprint belong to the same group. With the h hash functions, R_i can calculate all possible buckets that are selected by the other tags except k_j from the same group. Let $\mathcal{G}_j$ represent the set of these possible buckets. If $\mathcal{H}_j \subseteq \mathcal{G}_j$, the insertion failures, then we will reconstruct $\mathcal{CF}$. Otherwise, R_i removes the same bucket(s) in the two sets from $\mathcal{H}_j$, and the candidate bucket set of k_j changes to $\mathcal{H}_j - \mathcal{G}_j$; where $1 \leq |\mathcal{H}_j - \mathcal{G}_j| \leq h$ and $|\cdot|$ represents the number of items in a set. Then R_i inserts k_j into one of its candidate buckets according to the insertion steps described when F_j is a singleton fingerprint.

For the previously existing key tag that makes room for the new one, the reader uses the same insertion steps to relocate it. The reinsertion process will continue until the last key tag that is kicked out successfully maps to an empty bucket within the given maximum loops or fails to insert into $\mathcal{CF}$. If the last key tag to be inserted can map to an empty bucket within the given maximum loops, the reader puts each key tag's fingerprint into its bucket to replace of ID. Otherwise, we update f_c to $f_c + n_l$ and reconstruct $\mathcal{CF}$ using the new filter size, where n_l is the number of tags that fail to insert $\mathcal{CF}$.

For empty buckets, the reader fills them with one useless fingerprint. If there are no useless fingerprints, the reader inserts a singleton fingerprint that maps to other buckets for each empty bucket. So far, $\mathcal{CF}$ has been successfully constructed. We will analyze the optimal parameters of $\mathcal{CF}$ in Section IV.

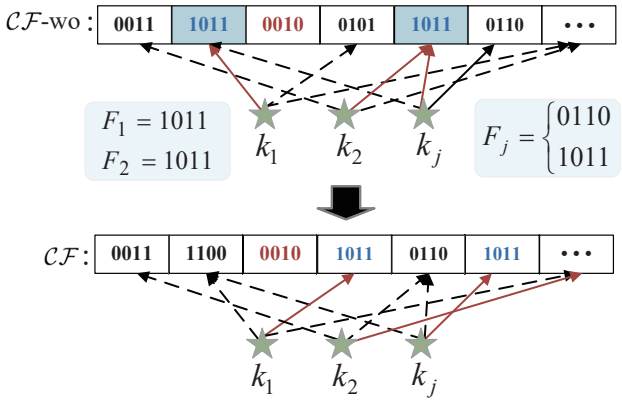


Fig. 2: The construction scheme with and without considering the interferences of collision fingerprints.

Our proposed protocol, Kadept, shines in that it is able to filter interference tags and assign each key tag a unique and continuous slot simultaneously by designing a new construction scheme for the Cuckoo filter. However, traditional filters, such as the widely used Bloom filer, only function as a filter. After filtering, the reader has to issue a slotted frame to check each key tag's response in its randomly selected slot, which retards the identification process caused by useless (empty and collided) slots.

C. Filtering-assigning-separating Phase

Filtering. After getting $\mathcal{CF}$, the reader broadcasts it and the construction parameters $\langle f_c, s_{1-h}, d, s_f \rangle$, and the useless fingerprint F_u (if it exists) to tags. Upon receiving this request, each tag first calculates its fingerprint and h candidate buckets according to Eq. (1). Then these h buckets are read: if the fingerprint in only one bucket matches, the tag passes $\mathcal{CF}$ and keeps active; otherwise, it keeps silent.

Assigning. Since the Cuckoo filter has no false negatives, all key tags will keep active. Consider the key tag k_j. Let SI_j be the slot index of k_j. We denote the number of buckets that store useful fingerprints preceding the matched bucket of k_j as b_i. According to $\mathcal{CF}$, k_j can get b_j, and set $SI_j = b_j + 1$. Note that each active tag only has one slot index because each key tag has one matched bucket; the false positive trivial tags that map to more than one matched bucket will keep silent.

Separating. To achieve the precise management of key tags without the interference of trivial tags, Kadept separates key tags from trivial tags by setting a flag for each tag. Specifically, Kadept lets each tag keep a flag with a value that is initially set to 0. A tag that cannot pass $\mathcal{CF}$ updates its flag to 1 before keeping silent, indicating that it is a trivial tag. After that, the reader broadcasts the flag 0 and then queries tags; only key tags will respond to the reader when the flag is 0.

Fig. 2 illustrates how collision fingerprints affect k_j's slot assignment. $\mathcal{CF}$-wo is constructed without considering the interference of collision fingerprints, that is, all key tags are inserted into $\mathcal{CF}$-wo in the same way as key tags with singleton fingerprints. Additionally, the useless fingerprint 0010 is

used to fill empty buckets. When F_j is the singleton fingerprint 0110, after receiving $\mathcal{CF}$-wo, k_j can find its matched bucket and get $b_j = 4$. Then it sets its slot index to $SI_j = b_j + 1 = 5$. However, when $F_j = F_1 = F_2$ is the collision fingerprint 1011, and k_1, k_2 map to two candidate buckets of k_j, k_j is unsure whether SI_j should be set to 2 or 4. Even if k_j responds in either the second, the fourth, or both slots, the reader cannot identify which key tag caused the response when two or more key tags respond in the same slot. In contrast, in $\mathcal{CF}$, k_j is inserted into one of its candidate buckets that are not mapped by the other tags from its group. The proposed construction scheme enables $\mathcal{CF}$ to assign a unique slot for each inserted tag in addition to filtering interference tags.

D. Identifying Phase

This phase describes how readers identify noncontentious key tags in parallel first and then identify contentious key tags based on a schedule.

1) Identifying Noncontentious Key Tags: In practice, overlapped zones are usually smaller than interrogation zones, and the majority of tags are noncontentious tags. To accelerate the identification process, readers first concurrently identify noncontentious key tags; only noncontentious tags can listen to readers and participate in this step due to reader collisions.

Consider the reader R_i. It identifies the noncontentious key tags that it covers by constructing two arrays E_i and A_i of length l_i. The first array E_i is constructed by the expected responses from the tags that pass $\mathcal{CF}$. From the filtering step in III-C, we know that the key tags used to construct $\mathcal{CF}$ and the active false positive trivial tags will pass $\mathcal{CF}$. According to tags' responses, the slots can be divided into three categories: *empty slots* picked by no tags, *singleton slots* picked by one tag, and *collision slots* picked by two or more tags. They are symbolized by '0', '1', and 'c', respectively. Since R_i has prior knowledge of all active tags, it can predict the expected slot category information of tags that pass $\mathcal{CF}$, and accordingly generates E_i: each element in E_i records each slot's expected category information. Note that R_i virtually constructs E_i without taking any communication overhead.

The second array A_i is built on tags' actual responses. Since each tag gives a 1-bit response, the actual slots follow into two categories: *empty slots* picked by no tags, *non-empty slots* picked by one or more tags. They are symbolized by '0' and '1', respectively. Recall that each tag that passes $\mathcal{CF}$ gets its slot index and keeps active. Then, R_i constructs A_i: each element in A_i records each slot's actual category information.

Comparing E_i and A_i, R_i can identify whether a tag is or not its local tag. Two slots with the same index in E_i and A_i are called a slot pair. For a slot pair $\langle a_1, a_2 \rangle$, a_1 and a_2 denote the slot categories in E_i and A_i, respectively. By checking each slot pair, R_i updates three tag sets that it stores: the local noncontentious key tag set $K_i^{L_n}$, the unidentified noncontentious key tag set $K_i^{U_n}$, and the active noncontentious trivial tag set $T_i^{A_n}$. These sets initially contain zero key tags, all key tags, and all false positive trivial tags, respectively. By analyzing l_i slot pairs, R_i acts as follows.

- $\langle 1, 1 \rangle$. This slot pair indicates that the key tag that is expectedly in this slot replies. The reader sends ACK_l to label it, puts it into $K_i^{L_n}$, and removes it from $K_i^{U_n}$. The labeled key tags keep silent after this round.
- $\langle 1, 0 \rangle$ and $\langle c, 0 \rangle$. These two kinds of slot pairs can be used to identify the noncontentious key tags that are not in the coverage regions of R_i. R_i removes the corresponding key tags from $K_i^{U_n}$ and the trivial tags from $T_i^{A_n}$.
- $\langle c, 1 \rangle$. This slot pair cannot be served as the vehicle to identify key tags. Because the reader R_i cannot infer the response(s) comes from which tag(s).

After the first round, most of the noncandidate key tags and the false positive trivial tags are removed from $K_i^{U_n}$ and $T_i^{A_n}$, respectively. With the updated $K_i^{L_n}$, $K_i^{U_n}$, and $T_i^{A_n}$, R_i executes the above three phases to identify the unidentified key tags. The process is repeated for several rounds and finishes when there are no $\langle c, 1 \rangle$ slot pairs in one round. After that, the noncontentious key tags are all identified.

2) Identifying Contentious Key Tags: Readers can identify contentious key tags by working in different time windows with a specific scheduling algorithm, such as Colorwave [24]. In this scenario, each identification round consists of multiple schedules, and the readers inside the same schedule work in parallel. Consider the reader R_i. In one schedule, R_i maintains another three tag sets: the local contentious key tag set $K_i^{L_c}$, the unidentified contentious key tag set $K_i^{U_c}$, and the active contentious trivial tag set $T_i^{A_c}$. They are initialized to empty set, $K - \sum_{i=1}^{m} K_i^{L_n}$, and $\bigcup_{\theta=1}^{|\Gamma(R_i)|+1} T_\theta^{A_n}$, respectively; where $R_\theta \in \{\Gamma(R_i) \bigcup R_i\}$. Similar to the method used for identifying noncontentious key tags, R_i identifies contentious key tags by constructing and comparing two arrays. In addition, the labeled contentious key tags will keep silent after this round while not this schedule. In this way, each reader can identify all the contentious key tags it covers. Furthermore, by sharing information with other readers, each reader can know that each of its local contentious key tags is covered by which readers.

E. Case Study

For clarity, we now use the scale-down RFID system shown in Fig. 1 to illustrate the execution process of one round in Kadept. We here only consider the case that the readers R_{1-4} concurrently identify the noncontentious key tags k_{1-3} and k_{7-8}, and we detail the identification process of R_1, as seen in Fig. 3. In the first round, R_1 constructs the Cuckoo filter $\mathcal{CF}$ using all key tags k_{1-8}, since it has no prior knowledge about tag distribution. $K_1^{L_n}$, $K_1^{U_n}$, and $T_1^{A_n}$ are initiated to $\emptyset$, $\{k_1, k_2, \cdots, k_8\}$, and $\{t_2, t_4, t_9, t_{13}, t_{18}\}$, respectively. R_1 inserts k_{1-8} into $\mathcal{CF}$ with three specified hash functions first and then replaces the IDs of k_{1-8} with their fingerprints. Besides, R_1 fills the empty bucket with the useless fingerprint '1010'. After getting $\mathcal{CF}$, R_1 broadcasts it and the construction parameters in the system. With the received information, the tags k_{1-3} and t_{1-5} that can listen to R_1 check whether they can pass $\mathcal{CF}$ or not. The key tags k_{1-3} and the false positive trivial tag t_2 find their own fingerprint in one of their three candidate buckets, besides, their fingerprints are not identical

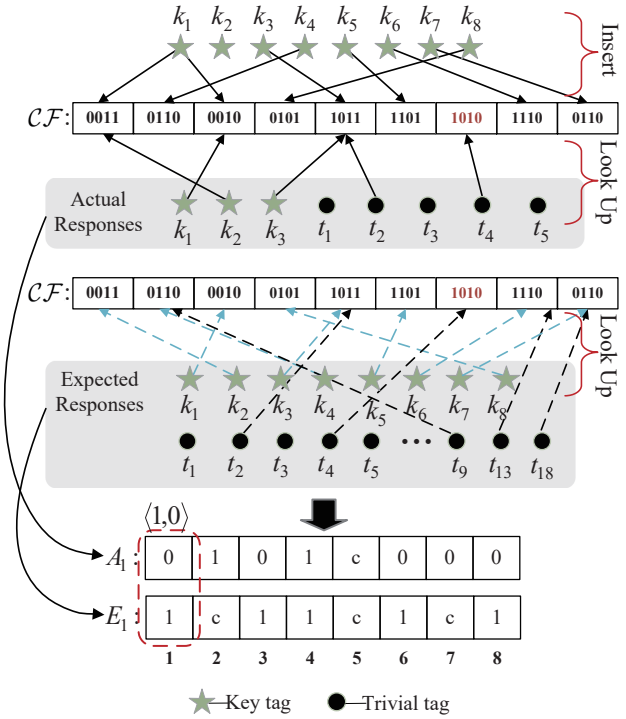


Fig. 3: An illustration of the execution of Kadept.

with '1010', then they pass $\mathcal{CF}$. On this basis, each tag that passes the filter sets its slot index. For k_1, it sets its slot index to 3 on account of the observation that it maps to the third non-empty bucket of $\mathcal{CF}$, and replies R_1 in the third slot. Similarly, k_2 and k_3 reply R_1 in the first and fifth slot, respectively. According to the actual slot status, R_1 constructs A_1. With the information of key tags k_{1-8} and trivial tags t_{1-18}, R_1 constructs E_1. Comparing E_1 and A_1, R_1 achieves eight slot pairs, based on which it labels tags and updates tag sets. For example, according to the first slot pair $\langle 1,1\rangle$, R_1 infers k_2 that expectedly maps to this slot is under its nonoverlapped region. It transmits ACK$_l$ to label k_2, adds k_2 to $K_1^{L_n}$, and removes k_2 from $K_1^{U_n}$. After this round, $K_1^{L_n}$, $K_1^{U_n}$, and $T_1^{A_n}$ update to $\{k_1, k_2\}$, $\{k_3, k_4\}$, and $\{t_2, t_{18}\}$, respectively.

IV. Performance Analysis

We first analyze the execution time of Kadept, denoted by T_{Kadept}, and then give the optimized parameters that can meet the given accuracy requirement and minimize the average time cost. From the global view of Kadept, T_{Kadept} consists of two parts: 1) The time T_n for readers to identify noncontentious key tags in parallel, and 2) the time T_c for readers to identify contentious key tags based on the schedule algorithm Colorwave. According to Kadept, we have $T_n = \max\{T_i\}$ $(1 \le i \le m)$ and $T_c = \sum_{j=1}^{C} \max\{T_i^j\}$, where C is the number of schedules. Consider the reader R_i. In the τth $(1 \le \tau \le w_i)$ round, the time for R_i transmitting $\mathcal{CF}_\tau$ is $\lceil \frac{f_\tau \times d_\tau}{96} \rceil t_{ID}$, and the time for active tags transmitting one bit is $n_\tau^k \times t_s$; where w_i is the number of rounds, f_τ is the length of $\mathcal{CF}_\tau$, and n_τ^k is the number of key tags used for constructing $\mathcal{CF}_\tau$. Besides, in each slot, the reader transmits an ACK to instruct the next action of tags in the corresponding slot. Hence, we have $T_i = \sum_{\tau=1}^{w_i} \{\lceil \frac{f_\tau \times d_\tau}{96} \rceil t_{ID} + n_\tau^k \times (t_s + t_{ack})\}$. We can achieve T_{Kadept} by adding T_n and T_c. Because each reader identifies noncontentious and contentious key tags in the same way, without loss of generality, we only analyze the time for R_i identifying noncontentious key tags.

A. Performance Efficiency

Due to the false positive of Cuckoo filter, some trivial tags may keep active and participate in the identification process, which causes the reader to fail to identify the key tags that map to collided buckets. We consider the worst case that each false positive trivial tag results in a key tag not being identified, and let the number of false negative key tags meet the following constraint:

$$I_{fn} = \frac{n_{fp}}{n_k} \le \alpha, \tag{2}$$

where I_{fn} is the false negative indicator, n_{fp} and n_k are the numbers of false positive trivial tags and unidentified key tags. The term α is the given indicator threshold, which is determined by the application requirement.

In one round, we aim to optimize the number of identified key tags per unit time to minimize the total identification time. Recall that key tags can be identified by the slot pairs $\langle 1,1\rangle$, $\langle 1,0\rangle$, and $\langle c,0\rangle$. We denote E_{11}, E_{10}, and E_{c0} as the numbers of tags in $\langle 1,1\rangle$, $\langle 1,0\rangle$, and $\langle c,0\rangle$, respectively. Let t_i be the execution time. The number of identified key tags per unit time, denoted by θ_i, can thus be represented as:

$$\theta_i = \frac{E_{11} + E_{10} + E_{c0}}{t_i}, \tag{3}$$

where $t_i = \{\lceil \frac{f_c \times d}{96} \rceil t_{ID} + n_k \times (t_s + t_{ack})\}$.

We take the following lemmas to analyze θ_i. Lemma 1 gives the expected values of E_{11}, E_{10}, E_{c0}, and p_f, which is the false positive ratio of $\mathcal{CF}$.

***Lemma** 1:* The expected values of E_{11}, E_{10}, E_{c0}, and p_f are:

$$\begin{aligned}
E_{11} &= n_k^l (1 - \frac{p_f}{f})^{n_t} \\
E_{10} &= (n_k - n_k^l)(1 - \frac{p_f}{f})^{n_t} \\
E_{c0} &= (n_k - n_k^l)\{(n_t - n_t^l)\frac{p_f}{f} \\
&\quad + \sum_{j=1}^{n_t - n_t^l} \binom{n_t - n_t^l}{j} (\frac{p_f}{f})^j (1 - \frac{p_f}{f})^{n_t - n_t^l - j}\} \\
p_f &= \frac{h}{2^d}(1 - \frac{h-1}{2^d}),
\end{aligned} \tag{4}$$

where n_k^l, n_t^l, and n_t are the numbers of local key tags, local trivial tags, and trivial tags, respectively.

Proof: In this round, the reader constructs $\mathcal{CF}$ with the n_k unidentified key tags and n_t active trivial tags. The slot pair $\langle 1,1\rangle$ occurs when one bucket of $\mathcal{CF}$ is inserted by local key tag, and no active local trivial tag(s) maps to this bucket. Let p_{11} be the probability that $\langle 1,1\rangle$ occurs and we have:

$$p_{11} = (1 - \frac{p_f}{f})^{n_t} \times \frac{n_k^l}{f}. \tag{5}$$

For a bucket, the probability that a tag that is not inserted into this bucket and returns a false-positive successful match is at most $1/2^d$. After making h such comparisons, the probability that only one successful match is:

$$p_f = \frac{h}{2^d}(1-\frac{1}{2^d})^{h-1} \approx \frac{h}{2^d}(1-\frac{h-1}{2^d}). \tag{6}$$

Hence, we have the expected number E_{11} of identified key tags in slot pairs $\langle 1,1\rangle$:

$$E_{11} = p_{11} \times f = n_k^l(1-\frac{p_f}{f})^{n_t}, \tag{7}$$

we will give a new cardinality estimation method to estimate the values of n_k^l and n_t^l shortly.

The slot pair $\langle 1,0\rangle$ occurs when one bucket is inserted by only one nonlocal key tag. The probability that $\langle 1,0\rangle$ occurs, denoted by p_{10}, can be derived:

$$p_{10} = \frac{n_k - n_k^l}{f}(1-\frac{p_f}{f})^{n_t}. \tag{8}$$

With p_{10}, we can calculate the expected value of E_{10}:

$$E_{10} = (n_k - n_k^l)(1-\frac{p_f}{f})^{n_t}. \tag{9}$$

The slot pair $\langle c,0\rangle$ occurs when one bucket is inserted by one nonexclusive key tag and mapped by one or more false positive trivial tags in $T_i^{A_n}$. Similarly, we can derive the probability that $\langle c,0\rangle$ occurs, denoted by p_{c0}:

$$p_{c0} = \frac{n_k - n_k^l}{f}\sum_{j=1}^{n_t-n_t^l}\binom{n_t-n_t^l}{j}(\frac{p_f}{f})^j(1-\frac{p_f}{f})^{n_t-n_t^l-j}. \tag{10}$$

Since there is one key tag and one or more trivial tags in $\langle c,0\rangle$, we can derive the expected values of E_{c0}:

$$\begin{aligned} E_{c0} &= f\frac{n_k-n_k^l}{f}\sum_{j=1}^{n_t-n_t^l}\binom{n_t-n_t^l}{j}(j+1)(\frac{p_f}{f})^j(1-\frac{p_f}{f})^{n_t-n_t^l-j} \\ &= (n_k-n_k^l)\{(n_t-n_t^l)\frac{p_f}{f} \\ &+ \sum_{j=1}^{n_t-n_t^l}\binom{n_t-n_t^l}{j}(\frac{p_f}{f})^j(1-\frac{p_f}{f})^{n_t-n_t^l-j}\}. \end{aligned} \tag{11}$$

■

B. The Parameters and Reconstruction of Cuckoo Filter

1) The Parameters of Cuckoo Filter: Lemma 2 gives the optimized construction parameters of the Cuckoo filter $\mathcal{CF}$ that ensures the maximal θ_i in Eq. (3).

***Lemma** 2:* Given the false negative indicator α, the values of h, f, and d are:

$$\begin{aligned} h &= \lceil \ln n_k \rceil \\ f_c &= n_k + 1 \\ d &= \max(\theta_i) \quad s.t. \quad \frac{h}{2^d}(1-\frac{h-1}{2^d}) \le \frac{\alpha n_k}{n_t}, \end{aligned} \tag{12}$$

where f_c is the initial size of $\mathcal{CF}$.

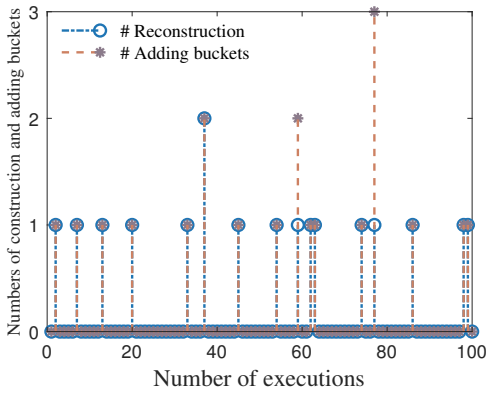


Fig. 4: An illustration of the reconstruction of Cuckoo filter.

Proof: To find the f_c, h, and d that maximizes θ_i, we analyze the constrain conditions of these three parameters. To ensure all n_k key tags can be successfully inserted into $\mathcal{CF}$, the minimal value of f_c is n_k, that is, $f_c \ge n_k$. Further, we derive the requirement that h needs to satisfy to insert n_k key tag into f_c buckets. For one bucket of $\mathcal{CF}$, the probability that no tag is mapped to it by using h hash functions is $(1-\frac{1}{f_c})^{n_k h}$, and the number of empty buckets is $f_c(1-\frac{1}{f_c})^{n_k h}$. Let $f_c(1-\frac{1}{f_c})^{n_k h} < 1$, we have the second constrain condition $h > \frac{f_c \ln f_c}{n_k}$. Besides, with Eq. (6), we can derive the number of false positive tags n_{fp}:

$$n_{fp} = n_t \times p_{fp} = \frac{n_t h}{2^d}(1-\frac{h-1}{2^d}). \tag{13}$$

From Eq. (2) and Eq. (13), we know that the length d of the fingerprint needs to satisfy $\frac{h}{2^{\hat{d}}}(1-\frac{h-1}{2^{\hat{d}}}) \le \frac{\alpha n_k}{n_t}$. With the above three constrain conditions, we can use one optimization function, such as fmincon, to solve the optimal solution of Eq. (3). The result is $f_c = n_k + 1$, $h = \lceil \ln n_k \rceil$, and $d = \min \hat{d}$, where $\hat{d}$ meets $\frac{h}{2^{\hat{d}}}(1-\frac{h-1}{2^{\hat{d}}}) \le \frac{\alpha n_k}{n_t}$. Moreover, the optimal solution is not influenced by the initial values of f_c, h, and d. Note that, the initial value of d is set to $log_2(n_k)$ for readers have no knowledge about the numbers of their own local key tags and local trivial tags. ■

2) The Reconstruction of Cuckoo Filter: With the optimal construction parameter settings, we study the construction of Cuckoo filter $\mathcal{CF}$ since one reader may take more than once to construct the Cuckoo filter successfully. Fig. 4 plots the reconstruction of $\mathcal{CF}$ in 100 independent simulations, where n_k, n_t, n_k^l, and n_t^l are set to 500, 2000, 50 and 200, respectively. We can see that the number of reconstructions is zero in most cases, that is, one reader just needs to execute the construction once to successfully construct the Cuckoo filter. Moreover, in the cases where the Cuckoo filter needs to be reconstructed, the numbers of reconstructions and corresponding adding buckets are small. Hence, the reader can efficiently construct $\mathcal{CF}$ with the optimal parameter settings.

C. Cardinality Estimation

To achieve the optimal Cuckoo filter construction parameters in Lemma 2, one reader needs to estimate the cardinalities n_k^l and n_t^l for the sets of local key tags and still active trivial

tags. However, utilizing the separate tag cardinality estimation protocols will increase the execution time. We propose an estimation scheme without extra time consumption by using the information identified. Consider any arbitrary round and the reader R_i. After this round, R_i precisely counts the number of slot pairs $\langle 1,1\rangle$, $\langle 1,0\rangle$, and $\langle c,0\rangle$, which are denoted as n_{11}, n_{10}, and n_{c0}^t, respectively. With n_{11} and Eq. (7), we have:

$$n_k^l = n_{11} e^{\frac{p_f}{f} n_t}. \tag{14}$$

Based on this, we derive the expected number of trivial tags in slot pairs $\langle c,0\rangle$, denoted by E_{c0}^t:

$$\begin{aligned} E_{c0}^t &= f\frac{n_k - n_k^l}{f} \sum_{j=1}^{n_t - n_t^l} \binom{n_t - n_t^l}{j} j (\frac{p_f}{f})^j (1 - \frac{p_f}{f})^{n_t - n_t^l - j} \\ &= (n_k - n_k^l)(n_t - n_t^l)\frac{p_f}{f}. \end{aligned} \tag{15}$$

With n_k^l, n_{c0}^t, and E_{c0}^t, we can calculate n_t^l:

$$n_t^l = n_t - \frac{n_{c0}^t f}{p_f (n_k - n_k^l)}. \tag{16}$$

According to the obtained parameters, the reader can update the parameters used in the next round to identify the left key tags. Lemma 3 gives how to achieve the updated n_k, n_t, n_k^l, and n_t^l, which are denoted as $\hat{n_k}$, $\hat{n_t}$, $\hat{n_k^l}$, and $\hat{n_t^l}$, respectively.

***Lemma** 3:* The expected values of $\hat{n_k}$, $\hat{n_t}$, $\hat{n_k^l}$, and $\hat{n_t^l}$ are:

$$\begin{aligned} \hat{n_k} &= n_k - n_{11} - n_{10} - n_{c0}^k \\ \hat{n_t} &= n_{fp} - n_{c0}^t \\ \hat{n_k^l} &= n_k^l - n_{11} \\ \hat{n_t^l} &= n_t^l, \end{aligned} \tag{17}$$

Proof: Since each of the slot pairs $\langle 1,1\rangle$, $\langle 1,0\rangle$, and $\langle c,0\rangle$ corresponds to an identified key tag, the number of key tags that are removed from $K_i^{U_n}$ is $n_{11} + n_{10} + n_{c0}^k$. Therefore, $\hat{n_k}$ is equal to $n_k - n_{11} - n_{10} - n_{c0}^k$. Moreover, after filtering, the n_{fp} trivial tags that are still active are further identified by slot pairs $\langle c,0\rangle$ and removed from $T_i^{A_n}$. Hence, the number $\hat{n_t}$ of active trivial tags is $n_{fp} - n_{c0}^t$. Remember that the local key tags that map to $\langle 1,1\rangle$ will be labeled and do not participate in the following identification. Therefore, the expected value of $\hat{n_k^l}$ is $n_k^l - n_{11}$. Since the reader cannot infer the information of local trivial tags from $\langle c,1\rangle$, all the local trivial tags keep active, we have $\hat{n_t^l} = n_t^l$. ■

V. Performance Evaluation

A. Simulation Configurations

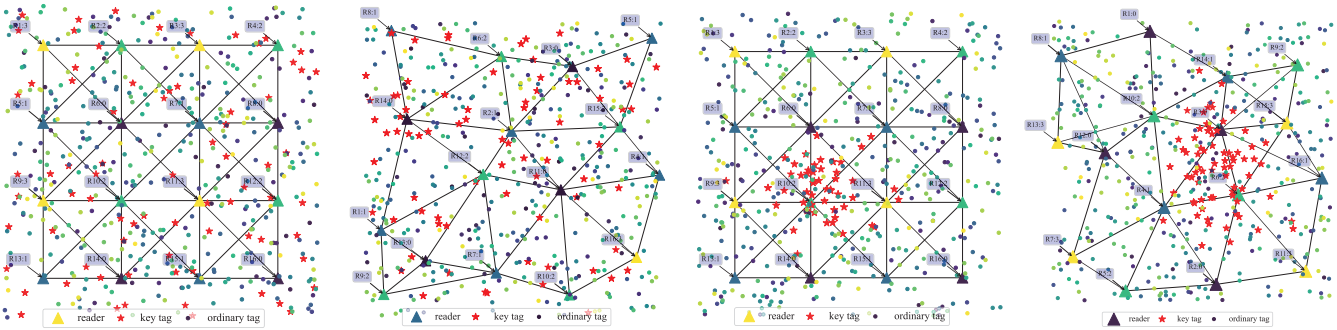
1) Parameter Setting: Similar to the existing work [25], [26], [16], [27], the parameter settings follow the specifications of the C1G2 standard [22]. In C1G2, the tag-to-reader transmission rate ranges from 40kbps to 640kbps when FM0 is used and 5kbps to 320kbps when Miller is used. We take the lower bound 40kbps in the intersection set 40kbps to 320kbps as the tag-to-reader data rate. Similarly, the reader-to-tag data rate varies from 26.7kbps to 128kbps and we adopt 26.7kbps. Any consecutive communications between the reader and tags are separated by different time intervals. When the reader transmits commands, tags wait for time $T1$ before replying. After receiving replies from the tags, the reader waits for time $T2$ before further communication. In our simulation, we set $T1 = 100\mu s$ and $T2 = 50\mu s$, which comply with the C1G2 standard. Then we have $t_{ID} = 37.45 \times 96 = 3595.2\mu s$, $t_s = (25 \times 1 + 150) = 175\mu s$, and $t_{ack} = 37.45 \times 2 = 74.9\mu s$. Note that, it has only one time interval between a tag transmits 1 bit to the reader and the reader transmits an ACK to this tag. We implement a simulator with Matlab to compare the performance of our protocol with the baseline IB [19], and the state-of-the-art polling method IPP [21].

2) Readers & Tags Deployment Strategy: We conduct our study on RFID systems with both regular and random reader deployment, where tags are arranged in a random pattern. Following prior approaches [19], [16], we leverage an undirected graph to represent the potential reader collision, with edges connecting neighboring readers. Fig. 5 illustrates the simulation systems. Fig. 5(a) and Fig. 5(c) depict the regular reader deployment scenario, where readers are deployed in a grid pattern; Fig. 5(b) and Fig. 5(d) depict the random reader deployment scenario with readers randomly distributed throughout the system. For tags, we first consider the scenario where they are uniformly distributed in the system, as shown in Fig. 5(a) and Fig. 5(b). Moreover, in view of the most typical scenario where key tags represent valuable goods of the same kind or belong to the same merchant, we simulate the corresponding systems by employing the Gaussian distribution to position the key tags. For simplicity, we refer to these systems as Gaussian or Uniform distribution systems, depending on the distribution of key tags.

B. Time Efficiency

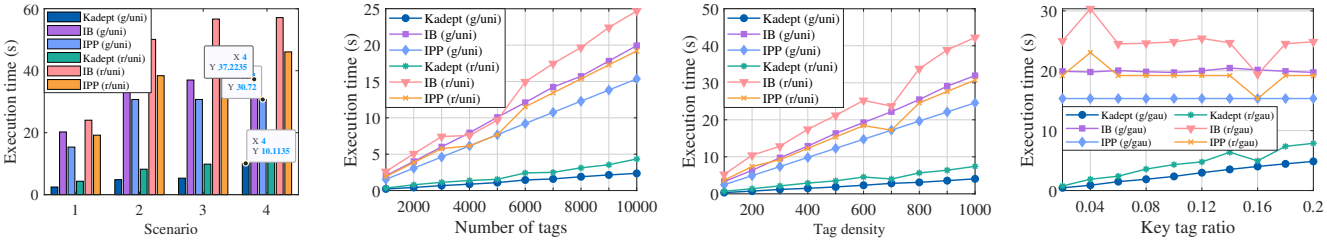
In the following simulations, we evaluate the time efficiency of Kadept in various RFID systems with different parameter settings. In Fig. 6 and Fig. 7, we compare the execution time of Kadept with the IB and IPP under the Uniform distribution systems depicted in Fig. 5(a) and Fig. 5(b) and the Gaussian distribution systems depicted in Fig. 5(c) and Fig. 5(d). In addition, to identify key tags correctly and completely, we eliminate the negative interference and clock difference caused by wave superpose [27], which is exploited in IB. Hence, in IB, we let tags respond to readers with one bit in a slot according to the specifications of C1G2 standard.

In Fig. 6(a) and Fig. 7(a), we provide a comprehensive comparison of the execution time between Kadept, IB, and IPP under four distinct scenarios. For clarity, we use the symbols m_g (m_r) to denote the number of readers in a regular (random) reader deployment scenario. Let ρ and κ represent the tag density and the key tag ratio, respectively. Recall that x, y are the numbers of key tags and trivial tags in the system. Naturally, $\rho = \frac{x+y}{m_g(m_r)}$ and $\kappa = \frac{x}{x+y}$. In scenario 1, we set $x + y = 10,000, \kappa = 0.1, m_g = 16, m_r \approx 23$. Without changing other parameters, we double the tag size $x + y$ in



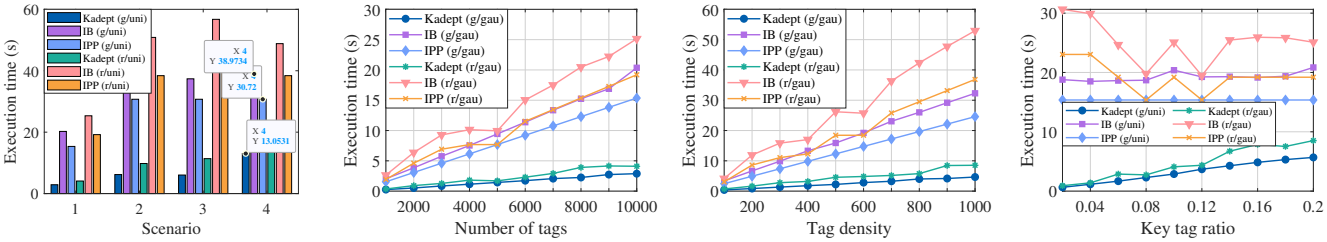
(a) Regular readers & random tags (b) Random readers & random tags (c) Regular readers & the Uniform distribution (trivial tags) and the Gaussian distribution (key tags) (d) Random readers & the Uniform distribution (trivial tags) and the Gaussian distribution (key tags)

Fig. 5: Deployment strategy of readers and tags.



(a) (b) (c) (d)

Fig. 6: Execution time under the Uniform distribution.



(a) (b) (c) (d)

Fig. 7: Execution time under the Gaussian distribution.

scenario 2, i.e., $x + y = 20,000, \kappa = 0.1, m_g = 16, m_r \approx 23$. In scenario 3, by varying the communication radius of readers, we set $m_g = 36, m_r \approx 48$. In turn, we double κ in scenario 4. To examine the execution time of our protocol Kadept, we take a closer look at scenario 4 in Fig. 6(a) and Fig. 7(a). In Fig. 6(a), IB takes 37.2s (57.1s) when readers are deployed in a grid (random) pattern and key tags follow the Uniform distribution. IPP reduces the execution time to 30.7s (46s), and our Kadept further shortens the execution time to 10.1s (18.9s), by filtering the trivial tags as well as assigning a unique slot to each key tag for replying. In Fig. 7(a), IB takes 38.9s (48.8s) with readers deployed in a grid (random) pattern and key tags following the Gaussian distribution. IPP reduces the execution time to 30.7s (38.4s), and Kadept further reduces the execution time to 13s (18.5s). A similar conclusion can also be drawn in the other three scenarios: Kadept performs the best, IPP follows, and IB is the least efficient.

Now, we study the impact of system scale, tag density, and key tag ratio on the execution time of Kadept, IB, and IPP in both the Uniform and Gaussian distribution systems.

The Uniform Distribution System. First, we investigate the impact of the system scale on the execution time of different approaches:Kadept, IB, and IPP. In Fig. 6(b), we fix $\kappa = 0.1$ and vary $x+y$ from 1000 to 10,000 with a step-length of 1000; the minimum number of readers needed to cover all tags may differ in random reader deployment scenarios. The execution time of Kadept, IB, and IPP experiences an increasing trend in general as $x + y$ increases and fluctuates with the systems. The reason is that the time taken by Kadept, IB, and IPP is related to not only the number of tags to be handled but also the parallelism of readers. As $x + y$ increases, all three approaches need to identify more tags, which naturally takes more time. Furthermore, as the number of schedules increases, the parallelism of readers decreases, leading to more time. For example, in random scenarios, when $x + y$ is set to 3000 and 4000, the numbers of schedules are 5 and 4, respectively. Consequently, Kadept, IB, and IPP consume more time with 3000 tags compared to the system with 4000 tags.

In Fig. 6(c), we show how the tag density influences the execution time of Kadept, IB, and IPP. In this simulation, we fix the number of readers to 16 and vary $x + y$ from 1600 to 16,000. That is, we vary ρ from 100 to 1000 with a step-

length of 100. Similarly, the execution time of Kadept, IB, and IPP increases as ρ increases and fluctuates with the systems in which readers are randomly distributed. This is due to the fact that when ρ increases, each reader takes more time to deal with more tags. The reason for the fluctuation is the parallelism of the readers, depending on the locations of the readers.

In Fig. 6(d), we study the execution time of Kadept, IB, and IPP with respect to the key tag ratio κ. We vary κ from 0.02 to 0.2 and fix $x + y = 10,000$. The time of Kadept experiences an increasing trend with κ, while the time of IB and IPP remains stable. This is intuitive as Kadept needs to identify more key tags when κ increases. Moreover, when $\kappa \leq 0.2$, Kadept outperforms IB and IPP significantly.

The Gaussian Distribution System. Fig. 7(b) demonstrates the execution time of Kadept, IB, and IPP in relation to the system scale. We vary $x+y$ from 1000 to 10,000 with a step-length of 1000. Similar to Fig. 6(b), the execution time of both Kadept, IB, and IPP increases with $x + y$; however, in most cases, Kadept and IB take more time in the Gaussian distribution systems. The disparity is due to the Gaussian distribution of key tags, leading to an uneven load on each reader. As a result, the readers with fewer tags covered will finish the key tag identification earlier than other readers (with more tags). Moreover, these readers have to wait for all the readers until they are done. This is a waste of time. For IPP, its execution time remains stable in regular reader deployment systems due to fixed reader deployment. In random scenarios, IPP's time is only affected by the parallelism of readers.

In Fig. 7(c), we vary the tag density ρ from 100 to 1000 with a step-length of 100. The execution time of Kadept, IB, and IPP increases as ρ increases. IPP's time remains stable in regular reader deployment systems. In random scenarios, the execution times of Kadept, IB, and IPP are influenced by the parallelism of readers and idividual reader load. For instance, when $\rho = 600$, the parallelism of readers is the same in both uniform and Gaussian scenarios, but the execution time of Kadept and IB 4.55s and 25.22s in the Uniform scenario, and 4.83s and 25.71 in the Gaussian scenario, respectively.

The parameter settings in Fig. 7(d) are the same as in Fig. 6(d). The execution time of Kadept, IB, and IPP varies with random reader deployment, while IB and IPP remain stable with uniform deployment. Despite experiencing an increasing trend with the parameter κ, Kadept still outperforms IB and IPP, particularly when the key tag ratio is small. For instance, in the reader random deployment scenario with $\kappa = 0.1$, IB takes 25.08s to identify key tags, IPP takes 19.2s, and Kadept reduces the time to 4.09s, resulting in a 3.7× performance gain. Kadept demonstrates outstanding performance in scenarios where the proportion of key tags is relatively small compared to the total number of tags. This makes Kadept well-suited for various practical situations that require effective monitoring of critical tags.

VI. Related Work

Tag identification is a fundamental problem for RFID systems. Its objective is to collect all the tags' IDs under the reader's coverage in an efficient way. Since the tags can only communicate with the reader and cannot self-regulate their radio transmissions, the key issue for tag identification is to avoid tag-to-tag collisions. Prior work on the tag identification falls into two categories: Aloha-based [18], [28] and tree-based [29], [17]. The former is to let each tag select a slot to transmit its ID. The tag ID can be successfully transmitted to the reader when a slot is selected by only one tag. In the tree-based protocols, the collided tags are continuously divided into two subsets until only at most one tag in each set.

In recent years, the research has shifted to collecting functional RFID data. For instance, tag searching [25], [30] tries to find a group of interested tags from the existing tag set; key tag tracking [26] is to track the number of key tags in multi-tenant systems; missing key tag identification [14], [15] aims to identify whether and which key tags are absent; information collecting [16], [27] targets at collecting the tagged product's information or the sensing data of sensor-augmented tags.

The key tag distribution identification can facilitate the collection of functional data of key tags, which plays a crucial role in various inventory operations. Intuitively, the tag identification protocols [17], [18] can address the key tag distribution problem. However, transmitting long tag IDs may leak the privacy of key tags, which is forbidden in some privacy-sensitive RFID systems [20]. While the tag distribution identification protocol IB [19] and the tag polling protocol IPP [21] can identify the distribution of key tags without leaking privacy, they both suffer from long time delays in dealing with all tags. We therefore propose Kadept that identifies key tag distribution with eliminating the interference of trivial tags and without leaking ID information of key tags.

VII. Conclusion

In this paper, we study the problem of key tag distribution identification in large-scale RFID systems. A protocol, Kadept, is proposed to fast identify the distribution of all key tags and separate key tags from trivial tags. By ingeniously constructing the Cuckoo filter, Kadept is able to assign a unique, continuous slot for each key tag when it filters trivial tags. By checking the statuses of assigned slots, readers can identify which key tags are under their own coverage. Extensive simulations show good performance of Kadept, which accelerates the identification process by up to 3.7× when the key tag ratio is 0.1, in comparison to the state-of-the-art.

Acknowledgment

This research is financially supported by the National Natural Science Foundation of China (Nos. 62102079, 62072231, 62306104, and 62102131), the Fundamental Research Funds for the Central Universities (B240201062), the Natural Science Foundation of Jiangsu Province under Grant (BK20210361), the China Postdoctoral Science Foundation (2023TQ0104), the Jiangsu Excellent Postdoctoral Program (2023ZB140), and the Collaborative Innovation Center of Novel Software Technology and Industrialization.

REFERENCES

[1] S. Li, S. Li, M. Chen, C. Song, and L. Lu, "Frequency scaling meets intermittency: Optimizing task rate for RFID-scale computing devices," *IEEE Transactions on Mobile Computing*, vol. 23, no. 2, pp. 1689–1700, 2023.

[2] X. Liu, Y. Huang, Z. Xi, J. Luo, and S. Zhang, "An efficient RFID tag search protocol based on historical information reasoning for intelligent farm management," *ACM Transactions on Sensor Networks*, 2023.

[3] J. Yu, W. Gong, J. Liu, L. Chen, K. Wang, and R. Zhang, "Missing tag identification in cots RFID systems: Bridging the gap between theory and practice," *IEEE Transactions on Mobile Computing*, vol. 19, no. 1, pp. 130–141, 2020.

[4] A. D. Smith, A. A. Smith, and D. L. Baker, "Inventory management shrinkage and employee anti-theft approaches," *International Journal of Electronic Finance*, vol. 5, no. 3, pp. 209–234, 2011.

[5] V. C. Maia, K. M. de Oliveira, C. Kolski, and G. H. Travassos, "Using RFID in the engineering of interactive software systems: A systematic mapping," *Proceedings of the ACM on Human-Computer Interaction*, vol. 7, no. EICS, pp. 1–37, 2023.

[6] J. Liu, X. Chen, S. Chen, X. Liu, Y. Wang, and L. Chen, "TagSheet: Sleeping Posture Recognition with an Unobtrusive Passive Tag Matrix," in *Proc. of IEEE INFOCOM*, 2019, pp. 874–882.

[7] Y. Wang and Y. Zheng, "TagBreathe: Monitor Breathing with Commodity RFID Systems," *IEEE Transactions on Mobile Computing*, vol. 19, no. 4, pp. 969–981, 2020.

[8] L. Shangguan and K. Jamieson, "The design and implementation of a mobile RFID tag sorting robot," in *Proc. of MobiSys*, 2016, pp. 31–42.

[9] B. Liang, P. Wang, R. Zhao, H. Guo, P. Zhang, J. Guo, S. Zhu, H. H. Liu, X. Zhang, and C. Xu, "Rf-chord: Towards deployable RFID localization system for logistic networks," in *Proc. of USENIX NSDI*, 2023, pp. 1783–1799.

[10] I. Kurt, F. Alagöz, and M. Akgün, "Scalable RFID authentication protocol based on physically unclonable functions," *Computer Networks*, vol. 230, p. 109793, 2023.

[11] X. Liu, J. Zhang, S. Jiang, Y. Yang, K. Li, J. Cao, and J. Liu, "Accurate localization of tagged objects using mobile RFID-augmented robots," *IEEE Transactions on Mobile Computing*, vol. 20, no. 4, pp. 1273–1284, 2021.

[12] G. Wang, C. Qian, L. Shangguan, H. Ding, J. Han, K. Cui, W. Xi, and J. Zhao, "Corrections to "hmo: Ordering RFID tags with static devices in mobile environments"," *IEEE Transactions on Mobile Computing*, vol. 20, no. 04, pp. 1746–1746, 2021.

[13] Z. An, Q. Lin, P. Li, and L. Yang, "General-purpose deep tracking platform across protocols for the internet of things," in *Proc. of MobiSys*, 2020, pp. 94–106.

[14] J. Yu, W. Gong, J. Liu, L. Chen, F. Wang, and H. Pang, "Practical key tag monitoring in RFID systems," in *Proc. of IWQoS*, 2018, pp. 1–2.

[15] H. Chen, Z. Wang, F. Xia, Y. Li, and L. Shi, "Efficiently and completely identifying missing key tags for anonymous RFID systems," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2915–2926, 2017.

[16] X. Liu, J. Yin, S. Zhang, B. Xiao, and B. Ou, "Time-efficient target tags information collection in large-scale RFID systems," *IEEE Transactions on Mobile Computing*, vol. 20, no. 9, pp. 2891–2905, 2020.

[17] L. Zhang, W. Xiang, and X. Tang, "An efficient bit-detecting protocol for continuous tag recognition in mobile RFID systems," *IEEE Transactions on Mobile Computing*, vol. 17, no. 3, pp. 503–516, 2018.

[18] J. Su, A. X. Liu, Z. Sheng, and Y. Chen, "A partitioning approach to RFID identification," *IEEE/ACM Transactions on Networking*, vol. 28, no. 5, pp. 2160–2173, 2020.

[19] F. Zhu, B. Xiao, J. Liu, B. Wang, Q. Pan, and L.-J. Chen, "Exploring tag distribution in multi-reader RFID systems," *IEEE Transactions on Mobile Computing*, vol. 16, no. 5, pp. 1300–1314, 2017.

[20] K. Bu, M. Xu, X. Liu, J. Luo, S. Zhang, and M. Weng, "Deterministic detection of cloning attacks for anonymous RFID systems," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 6, pp. 1255–1266, 2015.

[21] J. Liu, B. Xiao, X. Liu, K. Bu, L. Chen, and C. Nie, "Efficient polling-based information collection in RFID systems," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 948–961, 2019.

[22] "EPC radio-frequency identity protocols generation-2 UHF RFID standard," GS1, ISO/IEC 18000-63, Jul. 2018, https://www.gs1.org/standards/epc-rfid/uhf-air-interface-protocol.

[23] B. Sheng, Q. Li, and W. Mao, "Efficient continuous scanning in RFID systems," in *Proc. of IEEE INFOCOM*, 2010, pp. 1–9.

[24] J. Waldrop, D. W. Engels, and S. E. Sarma, "Colorwave: an anticollision algorithm for the reader collision problem," in *Proc. of IEEE ICC*, vol. 2, 2003, pp. 1206–1210.

[25] S. Zhang, X. Liu, S. Guo, A. Y. Zomaya, and J. Wang, "Why queue up? fast parallel search of RFID tags for multiple users," in *Proc. of ACM Mobihoc*, 2020, pp. 211–220.

[26] X. Liu, X. Xie, K. Li, B. Xiao, J. Wu, H. Qi, and D. Lu, "Fast tracking the population of key tags in large-scale anonymous RFID systems," *IEEE/ACM Transactions on Networking*, vol. 25, no. 1, pp. 278–291, 2016.

[27] J. Liu, S. Chen, Q. Xiao, M. Chen, B. Xiao, and L. Chen, "Efficient information sampling in multi-category RFID systems," *IEEE/ACM Transactions on Networking*, vol. 27, no. 1, pp. 159–172, 2018.

[28] S.-R. Lee, S.-D. Joo, and C.-W. Lee, "An enhanced dynamic framed slotted aloha algorithm for RFID tag identification," in *Proc. of MobiQuitous*, 2005, pp. 166–172.

[29] J. Su, Z. Sheng, C. Huang, G. Li, A. X. Liu, and Z. Fu, "Identifying RFID tags in collisions," *IEEE/ACM Transactions on Networking*, vol. 31, no. 4, pp. 1507 – 1520, 2022.

[30] J. Yu, W. Gong, J. Liu, L. Chen, and K. Wang, "On efficient tree-based tag search in large-scale RFID systems," *IEEE/ACM Transactions on Networking*, vol. 27, no. 1, pp. 42–55, 2018.