Fast and Reliable Tag Search in Large-Scale RFID Systems: A Probabilistic Tree-based Approach

Jihong Yu*, Wei Gong*, Jiangchuan Liu*, Lin Chen†
*School of Computing Science, Simon Fraser University, Canada. Email: {jihongy, jcliu, gongweig}@sfu.ca
†LRI-CNRS UMR 8623, University of Paris-Sud, France. Email: chen@lri.fr

***Abstract*—Searching for a particular group of tags in an RFID system is a key service in such important Internet-of-Things applications as inventory management. When the system scale is large with a massive number of tags, deterministic search can be prohibitively expensive, and probabilistic search has been advocated, seeking a balance between reliability and time efficiency. Given a failure probability $\frac{1}{\mathcal{O}(K)}$, where K is the number of tags, state-of-the-art solutions have achieved a time cost of $\mathcal{O}(K \log K)$ through multi-round hashing and verification. Further improvement however faces a critical bottleneck of repetitively verifying each individual target tag in each round. In this paper, we present a novel Tree-based Tag Search (TTS) that approaches $\mathcal{O}(K)$ through batched verification. TTS smartly hashes multiple tags into each internal tree node and adaptively controls the node degrees. It conducts bottom-up search to verify tags group by group with the number of groups decreasing rapidly. We derive the optimal hash code length and node degrees to accommodate hash collisions, and demonstrate the superiority of TTS through both theoretical analysis and extensive simulations. In particular, we show that, with increasing reliability demand and system size, TTS achieves an even higher performance gain, making it a highly scalable solution.**

I. Introduction

Radio frequency identification (RFID) technology [11] has recently been attracting remarkable attentions from both the academic and the industrial communities. A typical RFID system comprises two types of devices. One type called *the reader* transmits a high power RF signal. The other type called *the tag* is attached on a physical object and is able to capture the energy in the RF signal of a nearby reader and modulates this RF signal by adjusting the impedance match on its antenna such that a message of zeros and ones can be sent back to the reader. With the development of RFID technology, new generations of tags are armed with abilities of sensing, and computing and become programmable, e.g., WISP tag [1]. These capabilities and low manufacturing cost make them widely deployed in various applications ranging from inventory control [2] and supply chain management [13], to object localization [8] and human-computer interaction [23].

This paper focuses on the fundamental tag search problem in a large-scale RFID system which is formally defined as: *given a set of wanted tags, the target is to search in the system to confirm which wanted tags are currently present within interrogation areas of the readers*. For example, suppose some defective products from a manufacturer have been delivered to multiple warehouses, the manufacturer wants to know which defective products exist in which warehouse to further recall and fix them in time. To this end, the manufacturer provides the IDs of the tags attached to these products to warehouse administrators and asks for tag search service [4]. Obviously, fast and reliable tag search is desirable in this scenario to reduce financial loss and even avoid potential safety problems.

Despite its importance, the efficiency of tag search has yet to be further optimised. Deterministic protocols introduced in [4] can solve the tag search problem. While they are time-consuming for transmitting a large number of tag IDs. For example, when searching for $20,000$ tags in a system of $50,000$ tags, they require 19.4s and 130s, which are 4.4 times and 29.6 times the searching time of the probabilistic method with the failure probability 0.001 in [4]. Therefore, a series of probabilistic search protocols [25] [4] [16] [28] are proposed to accelerate tag search with a guaranteed failure probability.

Albeit the previous schemes employing Bloom filter [25] [4] or filtering vectors [16] [28] can filter out non-wanted tags effectively without transmission of tag IDs, they waste a large amount of time verifying the found wanted tags individually. For instance, suppose $5,000$ tags of a wanted tag set exist in a system of $20,000$ tags, to achieve a failure probability 10^{-4}, E-STEP [16] needs to execute 21 rounds. In fact, after the first 7 rounds, there are only 69 ineligible tags, indicating that the main purpose of the remaining 14 rounds is to verify the correctness of each individual target tag rather than further filter out ineligible tags. In this context, the existing methods that cannot verify target tags in a batch are not efficient anymore. Moreover, PLAT [28] assumes that the reader has all tag IDs in the system besides those of wanted tags; therefore, it cannot work in the scenario with unknown tags in [25] [4] [16].

In this paper, we propose a fast and reliable Tree-based Tag Search (TTS) that enables batched verification. TTS builds a tree of adaptive depth and node degrees by smartly hashing multiple tags into each internal tree node. It then executes two hashing-based functions on top of the tree, namely verification and refinement functions. Specifically, TTS first verifies tags group by group from the bottom of the tree to the up with the number of groups decreasing rapidly. Only when the verification function finds the existence of ineligible tags is the refinement function executed to refine this group. We perform theoretical analysis for determining optimal hash code length as well as depth and node degrees of the tree to guarantee reliability demand and minimise time cost. We would like to emphasize that given a failure probability $\frac{1}{\mathcal{O}(K)}$ where K is relative to the number of tags, TTS achieves a time cost

978-1-5386-4128-6/18/$31.00 ©2018 IEEE

of $\mathcal{O}(K\log^{(d)}K)$[1] with tree depth d, providing a significant improvement over prior work $\mathcal{O}(K\log K)$.

Note that a small d can reduce $\log^{(d)}K$ to a constant quickly. For example, for a large $K{=}2^{96}$, we know $\log K{=}96$, while setting $d{=}4$ yields $\log^{(4)}K{=}1.4$. That said, the time cost of TTS approaches $\mathcal{O}(K)$. Its superiority is also confirmed by extensive simulations. In particular, with increasing reliability demand and system size, TTS achieves an even higher performance gain, making it a highly scalable solution.

II. Problem Formulation and Motivation

We study an RFID system that consists of a backend server, one or multiple readers and a large number of tags. The backend server that has powerful computing and storage capability coordinates the readers and is responsible for the data storage and information processing. The readers, connected via high-speed channels with the backend server, transmit commands to the tags and report their responses to the server. When the server synchronizes the readers, we can logically consider them as a whole [25] [4] [16]. Consequently, we regard the server and the readers as a single entity that is denoted by the reader for simplicity. The tags, each having a unique 96-bit ID, are able to communicate with the reader wirelessly and implement the commands with lightweight hash functions [1].

The communication between the reader and tags follows the Listen-before-talk mechanism [6]: the reader initiates communication first by sending commands and broadcasting the parameters to tags, such as frame size and random seed. Each tag uses a hash function and the received seed to map its ID to one slot in the frame and replies to the reader in this slot. Consider an arbitrary time slot, if no tag replies in this slot, it is called an empty slot; if one or multiple tags reply in this slot, it is called a busy slot.

A. Problem Formulation

Let $Y=\{y_1,y_2,\cdots,y_m\}$ denote the set of m tags attached on the products that are currently covered by the RFID system. These tags are referred to as *present tags*. Due to the dynamics of the RFID system, e.g., unknown (i.e., new) tags/products move in and/or known ones leave from the warehouse, the reader does not know the tags covered by the system, that is, it has no knowledge of IDs of the present tags in Y. While it is a common assumption [25] [4] [16] that the reader knows the cardinality of Y from the estimation through the tag counting schemes analysed in [27].

Given a set of n wanted tags $X{=}\{x_1,x_2,\cdots,x_n\}$, we are interested in finding which wanted tags in X are currently present in the coverage area of the RFID system, i.e., finding the set of the tags $Z{=}X{\cap}Y$. For clearness, the tags in $X{\cap}Y$ are referred to as *target tags*, and the others are called *ineligible tags* containing $X-Y$ (*non-target tags*) and $Y-X$ (*non-wanted tags*). It is of great importance for a tag search scheme to have high reliability, which is required in realistic applications. In this case, a question arises naturally: how to achieve high reliability while keeping time cost as small as possible? Therefore, we devote this paper to designing a fast and reliable tag search scheme.

[1]Throughout the paper, we use log to denote the logarithm to the base 2 and define an iterated logarithm function $\log^{(i)}k$ with the following properties: 1) $\log^{(0)}k=k$; 2) for an integer $i\geq 1$, $\log^{(i)}k=\max\{1,\log(\log^{(i-1)}k)\}$.

TABLE I
Main Notations

Symbols	Descriptions
X	The set of the wanted tags: $\lvert X\rvert=n$
Y	The set of the present tags in the system: $\lvert Y\rvert=m$
Z	The set of the target tags: $X\cap Y$
λ	The intersection ratio: $\lvert Z\rvert/\min\{\lvert X\rvert,\lvert Y\rvert\}$
Z^*	The set of the tags in the final search result
P_{fail}	The required failure probability
P^*_{fail}	The achieved failure probability by TTS
K	$\max\{m,n\}$
d	The depth of the tree and # of rounds in TTS
δ_i	The degree of a node at the level i
L_i	The set of the nodes at the level i
p_i	The allowed failure probability in the round i
r_i, ℓ_i	The length of hash codes used in the round i
$\log^{(i)}k$	1) if $i=0$, $\log^{(0)}k=k$ 2) if $i\geq 1$, $\log^{(i)}k=\max\{1,\log(\log^{(i-1)}k)\}$

Denote by Z^* the set of tags in the final search result and let P_{fail} be the probability that the final search result is unequal to the ground truth, the RFID tag search problem is formally defined as follows:

Definition 1 (Tag search problem)**.** *Given X and Y, the tag search problem is to devise a protocol to find $Z=X\cap Y$ with the probability at least $Pr\{Z^*=Z\}\geq 1-P_{fail}$ within minimum time. In this problem, P_{fail} should be relative to the number of tags [4], so we set*

$$P_{fail}=\frac{1}{\mathcal{O}(K^a)},$$

with $K{=}\max\{m,n\}$ and a constant $a{>}0$, such that the expected number of failure events $K{\cdot}P_{fail}$ among K runs could approach 0 for a large K.

In the prior work [25] [4] [16], the cardinality of the set Z is set to $\lambda\cdot\min\{|X|,|Y|\}$ where λ called intersection ratio is a constant. In the analysis, we assume that $|X|$ and $|Y|$ and $|Z|$ are of the same order of magnitude, i.e., $\mathcal{O}(K)$. Table I summaries main notations used in the paper.

B. Motivation

1) Prior art: Multi-round filters. It is desirable for a tag search scheme to have high reliability and time efficiency. The existing works [25] [4] [16], however, experience a significant degradation of time efficiency as the reliability demand increases. Specifically, the reader either receives a filtering vector from tags in Y or sends one to them, and detects ineligible tags by observing differences between the received vector and the supposed responses of tags in X. The existing schemes essentially execute multiple rounds each with a constant failure probability to satisfy a required failure probability. As a result, they must operate for $\mathcal{O}(\log K)$ rounds (or $\mathcal{O}(\log K)$ hash functions should be used in [25]) each consuming time $\mathcal{O}(K)$ in order to achieve the required failure

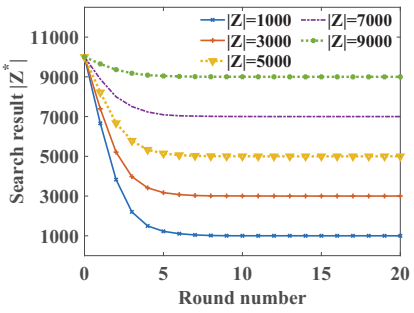


Fig. 1. Exemplify [16]: $P_{fail} = 10^{-4}$, $|X| = 10,000$, $|Y| = 20,000$.

probability $\mathcal{O}(\frac{1}{K^a})$ with K=max$\{m, n\}$ and a constant a, which results in the overall time cost $\mathcal{O}(K \log K)$.

2) Observations. We observe from the previous work [25] [4] [28] [16] that a lager amount of time is wasted verifying each individual tag repetitively. Take [16] as an example. Suppose $|X| = 10,000$, $|Y| = 20,000$, $|Z| = 1,000 : 2,000 : 9,000$ and $P_{fail} = 10^{-4}$, the number of rounds is equal to 21 [16]. As shown in Fig. 1, after the first 7 rounds, there are 79 non-target tags when $|Z| = 1,000$, and only 14 non-target tags left when $|Z| = 9,000$, indicating that the remaining 14 rounds in nature are repeated to verify the correctness of search result. For clearness, look at Fig. 2 where almost the whole filtering vector consists of responses from target tags. As only one ineligible tag exists, all slots here are in fact used to check target tags individually. Moreover, after this round, all target tags have been found, but the existing schemes still run round after round until the required failure probability is achieved, leading to the waste of a large amount of time. Therefore, if we can design a compact structure to verify tags in batches with a low failure probability, the overall time cost will be reduced significantly.

This motivates us to wonder: *can we design a scheme that achieves a failure probability $\mathcal{O}(\frac{1}{K^a})$ while reducing the prior time cost $\mathcal{O}(K \log K)$ towards $\mathcal{O}(K)$?*

3) Design overview. Motivated by the observations above, the design of our scheme follows the guidelines below:

- First, we should verify tags in batches instead of individual verification in previous work, and refine search result only when the verification result is false.
- Second, the number of runs should be reduced significantly compared to $\mathcal{O}(\log K)$ in the previous work, suggesting a better scalability to reliability requirement.

Based on these guidelines, we propose a fast and reliable Tree-based Tag Search (TTS) that exploits an adaptive tree to map tags into multiple groups. TTS operates in multiple rounds each consisting of two phases: 1. Batched verification: the reader verifies correctness of tags group by group. 2. Refinement: if the verification result is false, we further refine this group by examining tags individually. As we will demonstrate in Sec. IV, our scheme is able to achieve the failure probability $\frac{1}{\mathcal{O}(K^a)}$ with a time cost of $\mathcal{O}(K \log^{(d)} K)$, which is significantly superior to the previous $\mathcal{O}(K \log K)$.

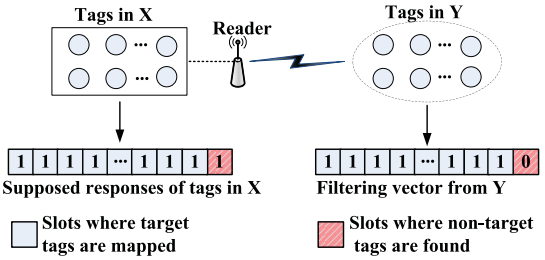


Fig. 2. Filtering vector in prior work. 0 and 1 mean zero and at least one response in the slot. Ineligible tags can be found in a slot of which the state is different between supposed and received filtering vectors.

III. TTS Design

In what follows, we first illustrate the basic idea of TTS with a simple example, and elaborate its design and theoretical performance analyse, subsequently.

A. TTS: Basic Idea

In this subsection, we introduce the basic idea of TTS with Example 1. Note that a non-leaf node and a leaf node are referred to as *node* and *leaf*, respectively for clearness. The height of a node means its distance from leaves, and the level i is the layer where nodes with the hight i locate.

Example 1. Given $n = 4$ wanted tags: X=$\{x_0, x_1, x_2, x_3\}$ and there are m=4 present tags: Y=$\{y_0, y_1, y_2, y_3\}$ in the RFID system. Suppose $x_1 = y_1$ and $x_2 = y_2$, we have the target tag set $Z = \{x_1, x_2\}$.

Before executing TTS, we first build a tree of the depth 2, as shown in Fig. 3. Specifically, we use a hash function h to hash the tags in X and Y into $K = 4$ values in $\{0, 1, 2, 3\}$. Suppose $h(x_0)$=$h(x_1)$=$h(y_1)$=0, $h(y_0)$=1, $h(x_2)$=$h(y_2)$=2, and $h(x_3)$=$h(y_3)$= 3. Then we use these 4 values as the leaves of the tree, i.e., leaves 0 to 3. Each leaf can be interpreted as a set of tags assigned to it from X and Y. Let each node at the level 1 have $\log K$=2 children (i.e., leaves), and let the node at the level 2, i.e., root, have $\frac{K}{\log K}$=2 children, we can obtain the tree shown in Fig. 3. By the tree, we divide the tag sets X and Y into different groups, i.e., tags are assigned to different leaves and nodes.

Obviously, two same tags, i.e., target tags, will be assigned to the same leaf, e.g., x_1 and y_1, x_2 and y_2, due to the fact that $h(x)$=$h(y)$ if the tag ID x=y. While two different tags may also map to the same leaf, e.g., x_0 and y_1, x_3 and y_3, because of hash collisions. Two questions thus arise: 1. How to know whether only the same tags map to a leaf? 2. If different tags from X and Y map to the same leaf, how to filter out ineligible tags, e.g., non-target tags x_0 and x_3 and non-wanted tags y_0 and y_3? To address the challenges above, TTS proceeds in 2 rounds from the bottom of the tree to the up.

The first round: TTS operates at the level 0, as shown in Fig. 4(a), where each leaf can be interpreted as a slot. The reader first requests present tags assigned to the leaf (slot) 0 to reply and a new hash function is used by tags here. Since only the tag y_1 qualifies, it sends a 1-bit hash code at this slot, assumed to be 1. With the same hash function, the reader has the hash codes for the tags x_0, x_1, assumed to be 0 and 1. As

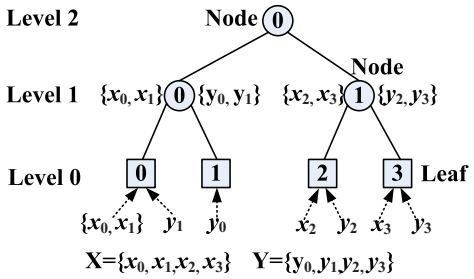


Fig. 3. Tree for Example 1.

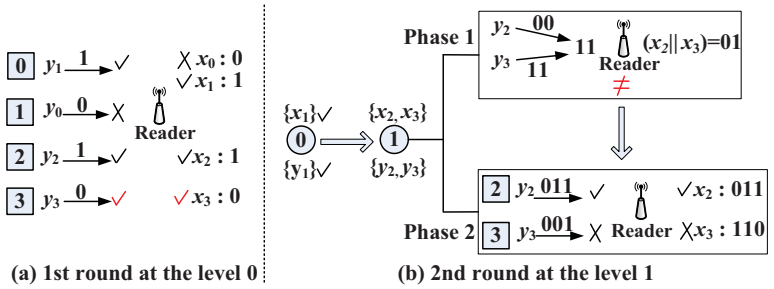


Fig. 4. Illustrate TTS on top of the tree in Example 1.

only the codes of x_1 and y_1 are equal, the reader considers x_0 to be non-target definitely, and temporarily regards x_1 as a target tag while keeping y_1 active for further verification.

The reader repeats these operations for the leaves (slots) 1 to 3. As no tag of X maps to the slot 1, y_0 is found non-wanted and will keep silent until TTS finishes. Moreover, as $x_2 = y_2$, the reader regards x_2 as a target tag temporarily. While for the leaf 3, though different, x_3 and y_3 still have the same code 0 due to the hash collision, and will keep active for further verification. After this round, the reader holds an updated candidate target tag set $\{x_1, x_2, x_3\}$, and y_1, y_2, y_3 are still active in the system.

The second round: TTS operates at the level 1*, as shown in Fig. 4(b)*, where each node can be regarded as a slot and new hash functions are used. The reader executes Phase 1: batched verification function. It first requires each of active tags mapped to the leaves in the subtree of the node 0, i.e., leaves 0 and 1, to reply with its new 2-bit hash code together. As only y_1 qualifies and x_1=y_1, their codes are equal, x_1 is regarded as a target tag with higher probability than the first round. Subsequently, the tags y_2, y_3 assigned to the leaves 2, 3 of the node 1 reply with their new hash codes, 00 and 11. For the concurrent transmission, the reader receives an aggregation of two physical-layer signals, assumed to be 11. From the hash codes of x_2 and x_3, i.e., 00 and 01, the reader calculates their combination, supposed to be 01. As the received value is different from the calculated one, there is at least one ineligible tag in X and Y mapped to the leaves 2 or 3.

To filter out ineligible tags, the reader further queries y_2 and y_3, and the one with the small leaf number responds first with a 3-bit hash code. The reader first receives 011 from y_2, which is the same as x_2, and regards x_2 as a target tag. Similarly, as the codes of y_3 and x_3 are unequal, the reader finds x_3 ineligible. After this round, the reader has the final result Z^*=$\{x_1, x_2\}$ that is equal to the ground truth Z.

Note that we will formally present how to configure the hash code sizes used in TTS in Sec. IV such that the required failure probability and time cost can be guaranteed.

B. Tree Architecture

As mentioned in Sec. II-B, an efficient tag search should be able to verify tags in batches as well as should have limited rounds. In this subsection, we show how to group tags by a tree of depth d, where d is the number of rounds in TTS. The challenge lies in that we need to carefully design the relationship between the tree depth and node degrees, which decides the performance of TTS.

Given the wanted tag set X and the present tag set Y, the reader has K=$\max\{m, n\}$. Then, the reader constructs a tree following the rules [3] below: First, it maps each wanted tag ID into one of values in $[K]$=$\{0, 1, 2, \cdots, K-1\}$, referred to as buckets, by a uniform hash function. Second, it builds a tree with these K buckets as its leaves.

Step one: hash tags into buckets. Since a tag ID is 96-bit, there would be 2^{96} RFID tags at most. Suppose the universe is U, we define h: $U \to [K]$ as a hash function that can uniformly map each tag into $[K]$. We present a hash value in decimal and the hash code length is $\log K$. For each $j \in [K]$, we define a set $X_j = \{x \in X | h(x) = j\}$ representing the tags of X that are mapped to the bucket j. With h, each tag of Y is also hashed to $[K]$, which, however, is unknown to the reader. We define $Y_j = \{y \in Y | h(y) = j\}$ for Y.

Step two: build the tree. Let $\mathcal{T}$ denote the tree of the depth d. Define the set of nodes with the height $0 \le i \le d$ as L_i. We build $\mathcal{T}$, as depicted in Fig. 5 following the rules below:

1) We make the K buckets obtained in the step one as the K leaves. That is, each leaf j stands for a set of the tags that are mapped to its corresponding hash value, i.e., j. We denote by $\mathcal{A}(j)$ and $\mathcal{B}(j)$ the set of tags of X and Y assigned to the leave j, respectively.
2) Denote by δ_i the degree of each node with the height i. That is, each node v at the level i has δ_i children. For $i = 1$, let $\delta_1 = \log^{(d-1)} K$; and let $\delta_i = \frac{\log^{(d-i)} K}{\log^{(d-i+1)} K}$ for $2 \le i \le d$.

We can extract two pieces of information from the tree. First, tags are assigned to different groups. Each leaf j stands for tags mapped to it. For each node v at the level $0<i<d$, tags of all leaves in its subtree can be regarded as those assigned to it, forming a bigger group than leaves, e.g., blue and red rectangles in Fig 5. As a result, the reader can query at a leaf or a node a group of tags, enabling batched verification. Second, $\mathcal{A}(j)$ at each leaf j is actually a candidate target tag set due to the fact that if there exists at least one target tag in $\mathcal{A}(j)$, i.e., $\mathcal{A}(j) \cap Z \neq \emptyset$, then at least one tag of Y is also mapped to the leaf j, i.e., $\mathcal{A}(j) \cap \mathcal{B}(j) \neq \emptyset$. These candidate tag sets can be regarded as initial input of TTS. Denote by $\mathcal{A}(j)^{-1}$=$\mathcal{A}(j)$ and $\mathcal{B}(j)^{-1}$=$\mathcal{B}(j)$ the initial input for every leaf j.

For a node $v \in \mathcal{T}$, let $\Theta(v)$ be the set of all leaves in the subtree of v. We further denote the initial candidate target tag

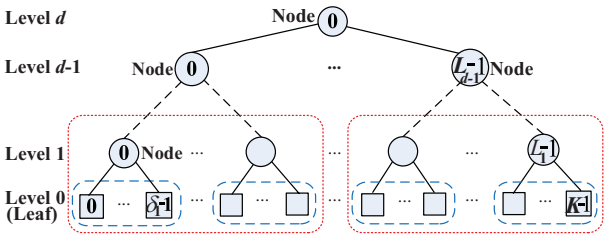


Fig. 5. Exemplify the built tree. Each leaf is assigned a group of tags, and each internal node contains tags of all leaves in its subtree, which can be interpreted as a bigger group, e.g., blue rectangles for nodes at the level 1, and red rectangles for nodes at the level $d-1$.

set for every node v by $\mathcal{A}_v^{-1}{=}\cup_{j\in\Theta(v)}\mathcal{A}(j)^{-1}$ and $\mathcal{B}_v^{-1} = \cup_{j\in\Theta(v)}\mathcal{B}(j)^{-1}$, and for the tree by $\mathcal{A}^{-1}{=}(\mathcal{A}(j)^{-1})_{0\le j\le K-1}$ and $\mathcal{B}^{-1}{=}(\mathcal{B}(j)^{-1})_{0\le j\le K-1}$.

C. General Search Process

Having described the tree design, we start formally presenting our tag search scheme TTS that operates on the tree for d rounds. The execution of d rounds starts from the level 0 where the leaves locate and terminates after the level $d{-}1$. Each round i for $1{\le}i{\le}d{-}1$ consists of two phases where two hashing-based functions are executed: 1) verifying the correctness of the candidate tag set for each node v; 2) refining candidate tag sets that are proven incorrect in the first phase. In the round $i=0$, the reader can verify and refine candidate sets in one phase. Specifically, TTS works as follows:

In the first round $i=0$, the reader first queries the tags of Y in the system with the command containing the frame size K, the hash code size r_0, and a random seed. On receiving the query, each tag uses the hash function h to map its ID to one slot of the frame, i.e., a leaf in the tree, and then transmits in its chosen slot an r_0-bit hash code generated by a hash function h_1. Here the existing techniques [24] [10] [20] are used to decode collisions when multiple tags respond in the same slot, which will be discussed later. In each slot, after obtaining hash codes from tags, the reader compares them with those of the wanted tags in X selecting this slot. If they match, the reader tentatively believes them to be the same tags, i.e., the target tags. Otherwise, they are ineligible tags, i.e, non-target tags in X and non-wanted tags in Y. Then the reader ACKs the found ineligible tags to silence them until TTS ends, while the others will keep active. Therefore, after this round, for each $0 \le j < K$, the reader deducts the non-target tags from the initial input $A^{-1}(j) = A(j)$ and gets an updated candidate set, denoted by $A^0(j)$.

Each of the remaining $d{-}1$ rounds, i.e., the levels 1 to $d{-}1$ of the tree, has two phases. Consider an arbitrary round i for $1{\le}i{\le}d-1$, with the candidate set $A^{i-1}(j)$ of the leaf j from the round $i{-}1$, TTS proceeds as follows:

Phase 1: batched verification. The reader further queries tags in Y with the parameters: the frame size K, the node degree δ_i at the level i, the hash code size r_i and ℓ_i, and a random seed. Each tag still picks the same slot as the round 0 by h such that the structure of the tree does not change. While in the slot v, i.e., node v at the level i, a tag whose chosen slot number is between $v{\cdot}\delta_i$ and $(v+1){\cdot}\delta_i-1$ replies with a r_i-bit hash code outputted from h_1. That said, tags assigned to the node v can be verified together. Moreover, executing L_i slots in this frame, i.e., the number of nodes at the level i, is enough to cover all K leaves. As more tags are scheduled to respond in one slot, the methods [24] [10] [20] separating the collided transmission may not work effectively, but fortunately, in this phase, we just need to check whether the tags in Y selecting this slot, accordingly those of Y mapped to the leaves $v{\cdot}\delta_i$ to $(v+1){\cdot}\delta_i{-}1$, are target tags. To this end, the reader measures the channel and aggregates physical-layer symbols from multiple tags, as implemented in [24] [26] [5]. If the hash codes of the responsive tags and the wanted tags are the same, their aggregated values should be the same. In this case, all responsive tags in this slot are temporarily regarded as target tags and will keep active, and the reader will start the next slot; otherwise, Phase 2 will be executed.

Phase 2: refinement. Because Phase 1 finds unequal hash codes in the node v (slot v), TTS refines candidate tag sets of all leaves in the subtree of the node v one by one. To this end, each of the tags mapped under h to the leaves between $v{\cdot}\delta_i$ and $(v+1){\cdot}\delta_i{-}1$, sends an ℓ_i-bit hash code by hash function h_2 in the order of their leaf numbers, e.g., from leaf 2 to 3 in Fig. 4 in Example 1. TTS then proceeds similarly as the round 0. After Phase 2, TTS starts to search in a new slot.

After the current round, for each leaf j, the reader deducts the found ineligible tags from the candidate set $A^{i-1}(j)$ and obtains an updated set $A^i(j)$ that will be used as the input for the next round. TTS then starts the new round, which is identical except that the founded non-wanted tags in Y will keep silent and the used parameters are different. The above process repeats round after round until the number of the executed rounds exceeds d when the reader is able to obtain the final candidate sets for all leaves such that their union set induced by the reader at the root of $\mathcal{T}$ is exactly $Z^* = X \cap Y = Z$ with a high probability.

IV. Performance Analysis

From Sec. III-A and III-C, we know that tree structure and success probability of two hashing-based operations, namely the verification and refinement functions, play important roles in the performance of TTS. Therefore, we next formally study how to design these parameters such that TTS can achieve $\frac{1}{\mathcal{O}(K^a)}$ failure probability and $\mathcal{O}(K\log^{(d)}K)$ time cost.

A. The failure probability of TTS

We first analyse success probability of verification function. As r_i-bit hash codes are used at each node v in each round $1{\le}i{\le}d{-}1$ to compare aggregated hash values of its candidate tags, i.e., tags in the set $\mathcal{A}_v^{j-1}{=}\cup_{j\in\Theta(v)}\mathcal{A}(j)^{j-1}$ and $\mathcal{B}_v^{j-1}{=}\cup_{j\in\Theta(v)}\mathcal{B}(j)^{j-1}$. As stated in Lemma 2 in Appendix, if two sets are nonidentical, the verification function can output $\mathcal{A}_v^{i-1}{\neq}\mathcal{B}_v^{i-1}$ with probability at least $1{-}\frac{1}{2^{r_i}}$. For the refinement function, its outputs in each round $0{\le}i{\le}d{-}1$ are the updated candidate tag sets $\mathcal{A}(j)^i$ and $\mathcal{B}(j)^i$ for a leaf j with the input of $\mathcal{A}(j)^{i-1}$ and $\mathcal{B}(j)^{i-1}$. As described in Sec. III-C, there are $\mathcal{A}(j)^{i-1} + \mathcal{B}(j)^{i-1}$ tags at the leaf j

each generating ℓ_i-bit (r_0-bit in the round 0) hash code. According to Lemma 3 in Appendix, we set such $\ell_i = \mathcal{O}(b\log(|\mathcal{A}(j)^{i-1}| + |\mathcal{B}(j)^{i-1}|))$, similarly for r_0, that the refinement function can succeed for each leaf with probability as least $1 - \frac{1}{(|\mathcal{A}(j)^{i-1}|+|\mathcal{B}(j)^{i-1}|)^b}$.

We make the failure probability of the verification function and the refinement function equal to the same value p_i:

$$\frac{1}{2^{r_i}} = \frac{1}{(|\mathcal{A}(j)^{i-1}| + |\mathcal{B}(j)^{i-1}|)^b} = p_i \tag{1}$$

such that after the round i for every leaf j it holds that $\mathcal{A}(j)^i = \mathcal{B}(j)^i$ with the probability at least $1 - p_i$ if $\mathcal{A}(j)^i \cap \mathcal{B}(j)^i \neq \emptyset$. The rationale lies in that in each round i, if j is in the subtree of a node v that passes verification at level i, we know $\mathcal{A}_v^{i-1}{=}\mathcal{B}_v^{i-1}$ and thus $\mathcal{A}(j)^i = \mathcal{B}(j)^i$ with success probability at least $1 - p_i$. Otherwise, j is in the subtree of a failed node v at level i. In this case, the refinement function is executed for j with success probability at least $1 - p_i$.

Note that as TTS operates, it needs to achieve the increasing success probability. To this end, in this paper, we configure p_i for round $0 \leq i \leq d-1$ as

$$p_i = \frac{1}{(\log^{(d-i+\alpha)} K)^\beta}, \tag{2}$$

where α and β are two constants and we will investigate how to configure them shortly. It is easy to check that the success probability $1 - p_i$ is proportional to the round number i. In order to achieve p_i, recall (1), we have the hash code sizes as

$$r_i = \begin{cases} \beta \log^{(d-i+\alpha+1)} K & \text{if } 1 \leq i \leq d-1 \\ \beta \log^{(d+\alpha+1)} K & \text{if } i = 0, \end{cases} \tag{3}$$

$$\ell_i = \beta \log^{(d-i+\alpha+1)} K \qquad \text{for } 1 \leq i \leq d-1. \tag{4}$$

Given p_i, as each node v at level $i{>}0$ has $\Theta(v){=}\log^{(d-i)} K$ leaves in its subtree (c.f. (9) in Appendix) each succeeding with probability $1{-}p_i$, after round i the success (i.e., $\mathcal{A}_v^i{=}\mathcal{B}_v^i$) probability for each node v, denoted by q_v, can be derived as

$$q_v \geq 1 - \Theta(v)p_i \geq 1 - \frac{\log^{(d-i)} K}{\left(\log^{(d-i+\alpha)} K\right)^\beta}$$

from the union bound over all its leaves.

Iteratively, after round $i{=}d{-}1$, TTS reaches the top of the tree, i.e., the root, and its success probability ($\mathcal{A}^{d-1}{=}\mathcal{B}^{d-1}$) is thus at least $1 - \frac{\log K}{\left(\log^{(1+\alpha)} K\right)^\beta}$. That is, the achieved failure probability by TTS, denoted by P_{fail}^*, satisfies

$$P_{fail}^* \leq \frac{\log K}{\left(\log^{(1+\alpha)} K\right)^\beta}. \tag{5}$$

B. Time cost of TTS

Recall the execution of TTS, its overall expected time cost consists of two parts: one for the verification and the other for the refinement. Next, we start to study the first part.

As TTS executes the verification function from round $i{=}1$ to the round $d{-}1$ at each node of level i with hash code size r_i, the overhead for the verification, denoted by T_1, is

$$T_1 = \sum_{i=1}^{d-1} |L_i| \cdot r_i = \sum_{i=1}^{d-1} \frac{\beta K \log^{(d-i+\alpha+1)} K}{\log^{(d-i)} K},$$

where $|L_i|$ is formulated in (8) in Appendix.

For the second part, TTS conducts the refinement function once in the round 0 but probabilistically in the other $d{-}1$ rounds. Specifically, at level i, i.e., the round i, the reader will carry out the refinement function on each leaf of node v if this node fails to pass verification. This would happen as long as node v has one incorrect child. For a leaf j let $V_i(j)$ denote its unique predecessor node at the level i and v is a child of $V_i(j)$. The probability of executing the refinement function on the leaf j in the round i can be calculated as

$$Pr\{V_i(j) \text{ does not pass the verification}\} \leq \delta_i \cdot (1 - q_v),$$

where the inequality holds by a union bound. Moreover, as there are d rounds and K leaves, we can compute the expected time cost for the refinement function, denoted by T_2, as

$$T_2 \leq \sum_{j=1}^{K} \left(r_0 + \sum_{i=1}^{d-1} \delta_i \cdot (1 - q_v) \cdot r_i \right).$$

Denote by T the overall time cost of TTS, we thus have

$$T \leq \sum_{i=1}^{d-1} \frac{\beta K \log^{(d-i+\alpha+1)} K}{\log^{(d-i)} K} + \mathcal{O}\big(\beta K \log^{(d+\alpha+1)} K\big) + \sum_{j=1}^{K} \sum_{i=1}^{d-1} \frac{\beta \delta_i (\log^{(d-i)} K) \log^{(d-i+\alpha+1)} K}{\left(\log^{(d-i+\alpha)} K\right)^\beta}. \tag{6}$$

With the general formulations of P_{fail}^* and T, we now configure α and β and d such that the failure probability is at most $\frac{1}{\mathcal{O}(K^a)}$ and the overhead is at most $\mathcal{O}(K \log^{(d)} K)$.

Recall (2), we can observe that $d - i + \alpha$ cannot be smaller than zero for all $i \in [0, d-1]$, requiring that $\alpha \geq -1$. Now, let $\alpha = -1$ and $\beta \geq 2$ and substitute them into (5), we have

$$P_{fail}^* \leq \frac{\log K}{K^\beta} \leq \frac{1}{K^{\beta-1}}, \tag{7}$$

for a large K. Furthermore, substitute them into (6) yields

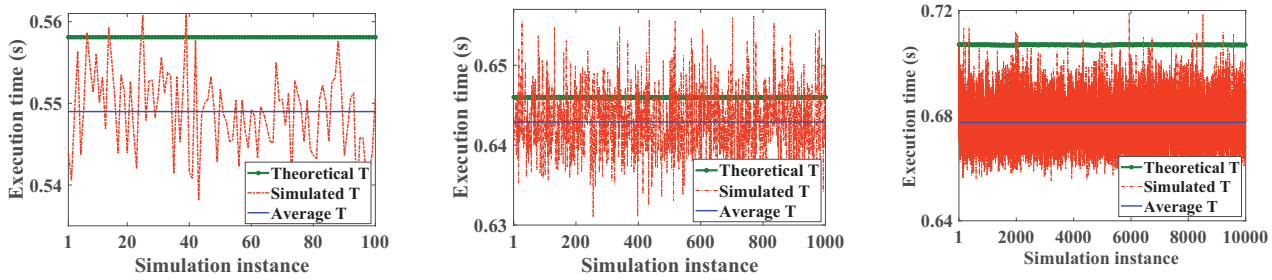
$$T = \mathcal{O}(K \log^{(d)} K),$$

which confirms our claim on the performance of TTS. Besides, in this setting, we have the hash code size r_i and ℓ_i used in round $1 \leq i < d$ are in the same order of magnitude as the number $\Theta(v)$ of leaves of a node v, i.e., $\mathcal{O}(\log^{(d-i)} K)$.

Next, we would like to show that TTS can also meet the user-defined requirement on the failure probability. Recall (5), given the required failure probability P_{fail}, we have

$$P_{fail}^* \leq P_{fail} \Rightarrow \beta \geq \frac{\log^{(2)} K - \log(P_{fail})}{\log^{(2+\alpha)} K}.$$

In this case, we should study how to set α, β and d such that the overall time cost of TTS is minimized. Having known $\alpha{\geq}{-1}$ from the previous analysis, we now determine the upper bound for α. From the definition of the arithmetic operation $\log$, we know that the allowed maximum value of α is bounded by the constraint that $\log^{(\alpha+2)} K{\geq}1$. For d, its minimum value is 1 and its maximum value is the one satisfying $\log^{(d)} K{\geq}1$. Therefore, the optimal parameter collection, denoted by $\{\alpha^*, \beta^*, d^*\}$, can be obtained by solving $\min_{\alpha,\beta,d} T$ subjected to the constraints above. Note that given a large $K{=}2^{96}$, then $\log^{(2)} K{=}6.6$ and $\log^{(4)} K{=}1.4$. As the feasible solution space is small, we could directly search for the optimum with which the tree structure will be fixed.



(a) $P_{fail} = 10^{-2}$, # of runs: 100 (b) $P_{fail} = 10^{-3}$, # of runs: $1,000$ (c) $P_{fail} = 10^{-4}$, # of runs: $10,000$

Fig. 6. Relationship between theoretical and simulation results under different P_{fail}. Parameter setting: $|X| = 5,000$, $|Y| = 10,000$, $\lambda = 0.5$.

Discussion on the assumption. In this paper, the existing techniques [24] [10] [20] are used to decode the collision when multiple tags respond in the same slot. It has been proven in their implementations that the reader is able to decode the signals from concurrent transmission of up to 16 tags. Theoretically, at most $\mathcal{O}(\frac{\log K}{\log \log K})$ tags [18] select the same slot when the number of tags is equal to the frame size K, so even $K = 2^{32}$, there will be at most 7 tags in a slot and just one tag in a slot on average. We thus assume that these methods operate successfully.

V. Performance Evaluation

A. Simulation Setup and Performance Metrics

In the simulation, we use the communication parameters specified in the EPC global C1G2 standard [6]. In our experiments, we set tag-to-reader transmission rate and reader-to-tag transmission rate to 100kbps as in [4], accordingly, the time cost for one-bit transmission is 10^{-5} sec.. The ratio of the target tags is defined as $\lambda = \frac{|X \cap Y|}{\min\{|X|,|Y|\}}$ where X is the set of wanted tags and Y is the set of tags currently present in the system. Moreover, the parameters used in TTS are set according to our theoretical analysis. Besides, we also set up E-STEP with its optimal parameter configuration [16].

The reliability is the paramount metric. In the simulation, a protocol needs to find all target tags with the required failure probability: $Pr\{Z^* = Z\} \geq 1 - P_{fail}$. As a result, to show the reliability of TTS, it runs N times if the required failure probability is $\frac{1}{N}$ in Sec.V-B1. Another important metric is the time it takes to meet a particular reliability requirement, which reflects the protocol efficiency. This is regarded as the primary metric in Sec.V-B2.

B. Simulation Results

1) TTS investigation: We demonstrate that TTS provides reliable tag search within bounded average time theoretically established in our analysis. To this end, we conduct a series of three experiments varying failure probability P_{fail} from 10^{-2} to 10^{-3} to 10^{-4}, respectively, while fixing the other parameters as follows: $|X|$=$5,000$, $|Y|$=$10,000$, and the target tag ratio λ=0.5. Moreover, TTS is executed for 100 times, $1,000$ times and $10,000$ times in the three experiments, respectively. After each experiment, we record the number of times of $Z^* \neq Z$, and a protocol fails to guarantee the required failure probability if this number exceeds one.

TABLE II
Performance Investigation of TTS. Parameter setting: $|X| = 5,000$, $|Y| = 10,000$, $\lambda = 0.5$

Required P_{fail}	Achieved P^*_{fail}	d	α	β
10^{-2}	0%	2	0	3
10^{-3}	0%	3	0	4
10^{-4}	0%	2	-1	2

Table II lists the failure probability of TTS and the derived optimal parameters used in TTS. It can be observed that the failure probability of TTS is zero in all three experiments, that is, TTS is able to find all target tags with the required failure probability. Moreover, with the increase of the required P_{fail}, α becomes from 0 to -1. This verifies our theoretical analysis in (7) that TTS is able to achieve the failure probability at most $\frac{1}{K^{\beta-1}}$ when $\alpha = -1$. Exactly, here $\beta = 2$ and $K = 10^4$, so we obtain $\frac{1}{K^{\beta-1}} = 10^{-4}$. In addition, when P_{fail} becomes from 10^{-2} to 10^{-3}, β increases by one while α keeps constant, suggesting the ability of TTS of adjusting its parameters to the user-defined requirement. In addition, depth of the tree d is also tunable to minimise time cost given the required P_{fail}.

We also record execution time of TTS in each run and obtain its average value over all runs, which are depicted in Fig. 6(a), Fig. 6(b) and Fig. 6(c). One thing worth noting is that, the theoretical T is the upper bound of the expected execution time. As illustrated in the three figures, the average execution time calculated from the simulation, referred to as average T, is upper bounded by the theoretical T derived from our analysis. And the relative error between them is less than 5%. The results also suggest that the execution time increases slowly compared to the significant change of the required failure probability. Specifically, TTS just consumes extra 20% time to reduce the failure probability from 0.01 to 0.0001.

2) Performance comparison: In this section, we start comparing performance of TTS with the state-of-the-art probabilistic tag search E-STEP [16] and the deterministic approach in [4] that broadcasts IDs of the wanted tags one by one.

First, we compare the time efficiency of three protocols under different wanted tag population $|X|$. Given the particular failure probability $P_{fail} = 10^{-5}$, we set the number of the present tags $|Y| = 20,000$ and the target tag ratio $\lambda = 0.5$ while changing $|X|$ from $10,000$ to $30,000$ by the step of $5,000$. Fig. 7 depicts the execution time used by each of three approaches to fulfill the tag search. As shown in the figure,

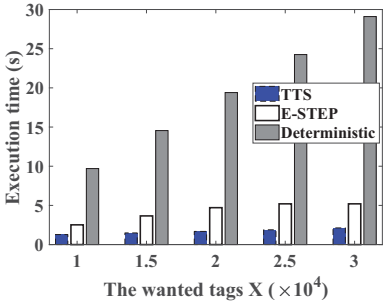


Fig. 7. Performance comparison with different $|X|$: $P_{fail} = 10^{-5}$, $|Y| = 20,000$, $\lambda = 0.5$.

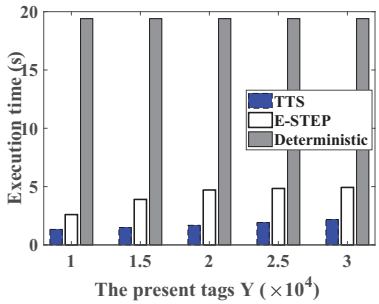


Fig. 8. Performance comparison with different $|Y|$: $P_{fail} = 10^{-5}$, $|X| = 20,000$, $\lambda = 0.5$.

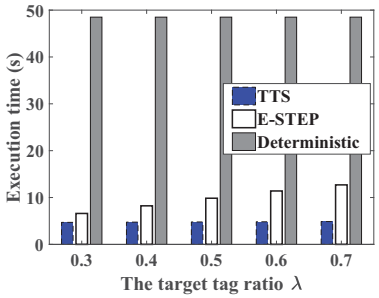


Fig. 9. Performance comparison with different λ: $P_{fail} = 10^{-4}$, $|X| = 50,000$, $|Y| = 60,000$.

TABLE III
PERFORMANCE COMPARISON OF TTS, E-STEP AND DETERMINISTIC PROTOCOL: $|X| = 20,000$, $|Y| = 50,000$, $\lambda = 0.5$

P_{fail}	10^{-4}	10^{-6}	10^{-8}	10^{-10}	10^{-12}	10^{-16}
TTS	3.4	3.5	3.5	4.7	4.7	5.6
E-STEP	4.3	6.0	7.8	9.6	11.3	14.9
Deterministic	19.4	19.4	19.4	19.4	19.4	19.4

TABLE IV
PERFORMANCE COMPARISON OF TTS, E-STEP AND DETERMINISTIC PROTOCOL: $|X| = 100,000$, $|Y| = 50,000$, $\lambda = 0.5$

P_{fail}	10^{-4}	10^{-6}	10^{-8}	10^{-10}	10^{-12}	10^{-16}
TTS	7.4	7.5	7.5	9.8	9.8	12.3
E-STEP	9.7	15.3	18.5	23.2	27.4	40.5
Deterministic	97	97	97	97	97	97

TTS is the most time-efficient and the change of the wanted tag population size has no significant impact on its execution time. Specifically, with the same P_{fail}, the execution time of E-STEP is at least twice as much as that of TTS among all five cases. From a different point of view, this result also suggests that given a certain amount of searching time, the failure probability of TTS will be much smaller than E-STEP.

Second, we compare the time efficiency of three approaches under different present tag population $|Y|$. Given the particular $P_{fail} = 10^{-5}$, we set the number of the wanted tags $|X| = 20,000$ and the target tag ratio $\lambda = 0.5$ while changing $|Y|$ from $10,000$ to $30,000$ by the step of $5,000$. From Fig. 8, we can observe the similar results that the time efficiency of TTS is significantly superior to E-STEP.

Third, we compare the time efficiency of three protocols under different target tag ratios λ. Given the particular $P_{fail} = 10^{-4}$, we use the following setting: $|X| = 50,000$, $|Y| = 60,000$, and $\lambda = 0.3 : 0.1 : 0.7$. The simulation results are exhibited in Fig. 9. As shown in the figure, TTS still remarkably outperforms E-STEP, specifically, with the performance gain of up to 62%. Moreover, E-STEP experiences a significant increase in the execution time with the increase of the target tag ratio. In contrast, TTS performs more stably.

Besides evaluating the impact of $|X|$, $|Y|$ and λ, we further compare the performance of three approaches with diverse failure probabilities P_{fail}. To this end, we fix $|X| = 20,000$, $|Y| = 50,000$ and $\lambda = 0.5$ while varying P_{fail} from 10^{-4} to 10^{-12}. Table III summarizes the time spent by each of the three approaches. As illustrated in Table III, TTS achieves the required failure probability within the least time. Especially under the smaller P_{fail}, TTS only consumes less than the half of the execution time of E-STEP. For a comprehensive comparison, we conduct another experiment where the setting above does not change except that $|X|$ is set to $100,000$. From the results listed in Table IV, we can draw the similar conclusion that TTS is of greater scalability to reliability requirement than E-STEP. As a matter of fact, given the required failure probability at most $\frac{1}{\mathcal{O}(K^a)}$, as TTS consumes $\mathcal{O}(K \log^{(d)} K)$ time compared to the $\mathcal{O}(K \log K)$ of E-STEP, the performance gain of TTS over S-STEP will be rather larger when the requirement on the reliability scales up.

VI. RELATED WORK

Various protocols have been proposed to collect tag IDs in RFID systems. Existing identification protocols generally can be classified into two categories: ALOHA-based protocols and Tree-based protocols. In ALOHA-based identification protocols [22] [12], each tag randomly selects one slot to transmit its ID. If there is collision, the tag will continue participating in the next frame until its ID is received successfully. In tree-based identification protocols [21] [9], the reader encodes all tag IDs as leaves of a tree and requires tags with matching masks to transmit their IDs. Although these schemes can be borrowed to search for tags, they spend too much time sending tag IDs and are thus inefficient in large-scale systems [4].

Many research efforts have also been devoted to monitoring missing tag event and unknown tag event. Missing tag monitoring protocols aim at probabilistically [17] [5] or exactly [14] [26] finding out the tags that should exist in the system but actually are absent. On the other hand, unknown tag detection [7] and identification [15] are to probabilistically detect and deterministically identify the tags whose IDs are not recorded by the system. Opposite to missing and unknown tag monitoring, the tag search problem focuses on finding a particular set of tags in interrogation areas.

Several probabilistic approaches have been proposed to address the tag search problem. The works [25] [4] employ Bloom filter to encode tag IDs, accelerating the tag search task. The former [25], named CATS, works on the hypothesis that the cardinality of the wanted tag set is smaller than that of the present tag set in the system. It thus may not work when the assumption fails [4]. Instead of transmitting a long Bloom filter

in CATS, ITSP [4] decomposes it into multiple short filtering vectors and uses them to filter out ineligible tags iteratively, which reduces time cost. To accelerate search process, two most recent works called E-STEP [16] and PLAT [28] exploit testing slots and non-testing slots to filter out ineligible tags. While PLAT [28] assumes that the reader knows IDs of all tags in the system besides that of wanted tags; therefore, it cannot work in the scenario in the presence of unknown tags in [25] [4] [16] and this paper. In essence, the existing works require time cost $\mathcal{O}(K \log K)$ to achieve failure probability $\mathcal{O}(\frac{1}{K^a})$. Differently, this paper introduces a tree-based tag search scheme that reduces the time cost to $\mathcal{O}(K \log^{(d)} K)$.

VII. Conclusion

This paper studied the tag search problem in large-scale RFID systems. We designed a fast and reliable Tree-based Tag Search (TTS) that exploits an adaptive tree to map tags into its internal nodes. TTS enables batched verification by verifying tags at each node from the bottom to the up with the number of groups decreasing rapidly. We theoretically demonstrated that TTS can achieve time cost $\mathcal{O}(K \log^{(d)} K)$ while guaranteeing required failure probability $\frac{1}{\mathcal{O}(K^a)}$, providing a significant improvement over prior $\mathcal{O}(K \log K)$. RFID tags are of massive amount, which are generally orders of magnitude higher than conventional network devices. Our tree-based design sheds lights in effectively searching the tags. For the future work, we are implementing it on the state-of-the-art tags and intend to further improve its efficiency. Moreover, we are planning to design a Bloom filter on top of the tree, which may also be applied in other large-scale networked systems, e.g., stealthy network activity detection in the Internet.

VIII. Acknowledgement

This work was supported by a Canada Technology Demonstration Program (TDP) Grant, a Canada NSERC Discovery Grant, and an NSERC E.W.R. Steacie Memorial Fellowship. L. Chen's research is supported in part by the CNRS Projet Exploratoire Premier Soutien on Objects communicants: algorithms, achitectures et application under the project MIRFID: Management of Infrastructural RFID systems: algorithm design and implementation.

References

[1] WISP platform. Available: http://wisp.wikispaces.com/.
[2] Barcoding Inc. How RFID works for inventory control in the warehouse. Available: http://www.barcoding.com/rfid/inventory-control.shtml.
[3] J. Brody *et al*. Beyond set disjointness: the communication complexity of finding the intersection. In *ACM PODC*, pages 106–113, 2014.
[4] M. Chen *et al*. An efficient tag search protocol in large-scale rfid systems. In *IEEE INFOCOM*, pages 899–907, 2013.
[5] M. Chen *et al*. Dbf: A general framework for anomaly detection in rfid systems. In *IEEE INFOCOM*, 2017.
[6] EPCglobal Inc. Class-1 generation-2 UHF RFID protocol for communications at 860 mhz - 960 mhz, 2005. Available: http://www.gs1.org.
[7] W. Gong *et al*. Efficient unknown tag detection in large-scale rfid systems with unreliable channels. *IEEE/ACM TON*, 2017.
[8] J. Han *et al*. Twins: Device-free object tracking using passive tags. *IEEE/ACM TON*, 24(3):1605–1617, 2016.
[9] Y. Hou and Y. Zheng. Phy assisted tree-based rfid identification. In *IEEE INFOCOM*, 2017.
[10] J. Kaitovic and M. Rupp. Improved physical layer collision recovery receivers for rfid readers. In *IEEE RFID*, pages 103–109, 2014.
[11] F. Klaus. *RFID Handbook: Radio-frecuency Identification: Fundamentals and Applications*. John wiley and Sons, 1999.
[12] L. Kong, L. He, Y. Gu, M.-Y. Wu, and T. He. A parallel identification protocol for rfid systems. In *IEEE INFOCOM*, pages 154–162. IEEE.
[13] C.-H. Lee and C.-W. Chung. Efficient storage scheme and query processing for supply chain management using RFID. In *ACM SIGMOD*, pages 291–302, 2008.
[14] T. Li, S. Chen, and Y. Ling. Identifying the missing tags in a large RFID system. In *ACM MobiHoc*, pages 1–10, 2010.
[15] X. Liu *et al*. Efficient unknown tag identification protocols in large-scale rfid systems. *IEEE TPDS*, 25(12):3145–3155, 2014.
[16] X. Liu *et al*. Step: A time-efficient tag searching protocol in large rfid systems. *IEEE TC*, 64(11):3265–3277, 2015.
[17] W. Luo *et al*. Missing-tag detection and energy–time tradeoff in large-scale rfid systems with unreliable channels. *IEEE/ACM TON*, 22(4):1079–1091, 2014.
[18] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge university press, 2017.
[19] M. Mitzenmacher *et al*. Why simple hash functions work: exploiting the entropy in a data stream. In *ACM-SIAM SODA*, pages 746–755, 2008.
[20] J. Ou, M. Li, and Y. Zheng. Come and be served: Parallel decoding for cots rfid tags. In *ACM MobiCom*, pages 500–511, 2015.
[21] M. Shahzad and A. X. Liu. Probabilistic optimal tree hopping for rfid identification. *ACM SIGMETRICS*, 41(1):293–304, 2013.
[22] B. Sheng, Q. Li, and W. Mao. Efficient continuous scanning in rfid systems. In *IEEE INFOCOM*, pages 1–9, 2010.
[23] J. Wang, D. Vasisht, and D. Katabi. RF-IDraw: virtual touch screen in the air using RF signals. In *ACM SIGCOMM*, pages 235–246, 2014.
[24] J. Wang *et al*. Efficient and reliable low-power backscatter networks. In *ACM SIGCOMM*, pages 61–72, 2012.
[25] Y. Zheng and M. Li. Fast tag searching protocol for large-scale rfid systems. *IEEE/ACM TON*, 21(3):924–934, 2013.
[26] Y. Zheng and M. Li. P-mti: Physical-layer missing tag identification via compressive sensing. *IEEE/ACM TON*, 23(4):1356–1366, 2015.
[27] Z. Zhou, B. Chen, and H. Yu. Understanding rfid counting protocols. *IEEE/ACM TON*, 24(1):312–327, 2016.
[28] F. Zhu *et al*. Plat: A physical-layer tag searching protocol in large rfid systems. In *IEEE SECON*, pages 1–9, 2016.

Appendix

Lemma 1. *Given $\mathcal{T}$, let L_i denote the set of nodes of the height $0 \le i \le d$. For an arbitrary node $v \in \mathcal{T}$, let $\Theta(v)$ denote the set of all leaves in the subtree of v. It holds for i-level L_i and $\Theta(v)$ that:*

$$|L_i| = \begin{cases} K, & \text{if } i = 0 \\ \frac{K}{\log^{(d-i)} K}, & \text{if } 1 \le i \le d, \end{cases} \tag{8}$$

$$|\Theta(v)| = \begin{cases} 1, & \text{if } i = 0 \\ \log^{(d-i)} K, & \text{if } 1 \le i \le d. \end{cases} \tag{9}$$

Proof. For L_i it holds when i=0 and d since $\mathcal{T}$ has one root and K leaves. As the node degree at the level $1 \le i \le d-1$ equals δ_i, the number of the nodes with height i can be computed from the top down as $\prod_{i'=i+1}^{d} \delta_i = \frac{K}{\log^{(d-i)} K}$. For $\Theta(v)$, v is a leaf when $i = 0$, so $|\Theta(v)| = 1$. When $1 \le i \le d$, as each node has δ_i children, $|\Theta(v)|$ can be obtained from $\prod_{i'=1}^{i} \delta_i$. □

Lemma 2. *[19] [3] [18] Given a hash function into r bits, if sets A and B are identical, the aggregated hash value of A is equal to that of B with probability 1. Otherwise, their aggregated values are unequal with probability at least $1 - \frac{1}{2^r}$.*

Lemma 3. *[18] For any set of u elements, a hash function into $\mathcal{O}(b \log u)$ bits for any $b > 0$ has no hash code collision for all u elements in this set with probability at least $1 - \frac{1}{u^b}$.*