On Efficient Tree-Based Tag Search in Large-Scale RFID Systems

Jihong Yu, Wei Gong, *Member, IEEE*, Jiangchuan Liu, *Fellow, IEEE*, Lin Chen, *Member, IEEE*, and Kehao Wang

***Abstract*— Tag search, which is to find a particular set of tags in a radio frequency identification (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 an efficient tree-based tag search (TTS) that approaches $\mathcal{O}(K)$ through batched verification. The key novelty of TTS is to smartly hash multiple tags into each internal tree node and adaptively control the node degrees. It conducts bottom–up search to verify tags group by group with the number of groups decreasing rapidly. Furthermore, we design an enhanced tag search scheme, referred to as TTS+, to overcome the negative impact of asymmetric tag set sizes on time efficiency of TTS. TTS+ first rules out partial ineligible tags with a filtering vector and feeds the shrunk tag sets into TTS. We derive the optimal hash code length and node degrees in TTS to accommodate hash collisions and the optimal filtering vector size to minimize the time cost of TTS+. The superiority of TTS and TTS+ over the state-of-the-art solution is demonstrated through both theoretical analysis and extensive simulations. Specifically, as reliability demand on scales, the time efficiency of TTS+ reaches nearly 2 times at most that of TTS.**

***Index Terms*— RFID, IoT, tag search, time efficiency.**

Manuscript received February 1, 2018; revised August 6, 2018; accepted October 22, 2018; approved by the IEEE/ACM TRANSACTIONS ON NETWORKING Editor Y. Guan. Date of publication December 3, 2018; date of current version February 14, 2019. This work was supported in part by the Beijing Institute of Technology Research Fund Program for Young Scholars, in part by the Canada NSERC Discovery Grant, and in part by the NSERC E.W.R. Steacie Memorial Fellowship. The work of K. Wang and L. Chen was supported in part by NSF of China under Grant 61672395 and in part by the CNRS PEPS Project MIRFID. *(Corresponding author: Jiangchuan Liu.)*

J. Yu was with the School of Computer Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada. He is now with the School of Information and Electronics, Beijing Institute of Technology, Beijing 100081, China (e-mail: jihong-yu@hotmail.com).

W. Gong was with the School of Computer Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada. He is now with the School of Computer Science and Technology, University of Science and Technology, Hefei 230026, China (e-mail: weigong@ustc.edu.cn).

J. Liu is with the School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada (e-mail: jcliu@sfu.ca).

L. Chen is with the Laboratoire de Recherche en Informatique, Université Paris-Sud, 91400 Orsay, France, and also with the Institut Universitaire de France, 75231 Paris, France (e-mail: chen@lri.fr).

K. Wang is with the Key Laboratory of Fiber Optic Sensing Technology and Information Processing, School of Information Engineering, Wuhan University of Technology, Wuhan 430070, China (e-mail: kehao.wang@whut.edu.cn).

Digital Object Identifier 10.1109/TNET.2018.2879979

I. Introduction

RECENT years have witnessed an unprecedented development of the radio frequency identification (RFID) technology [13]. In RFID systems, an attachable *tag* that stores information of a physical object does not expend its own energy on data communication while it is able to capture the energy in the RF signal of a nearby RFID *reader* and modulate this signal by adjusting the impedance match on its antenna such that a message of zeros and ones can be sent back to the reader. In addition to simple tags, programmable tags are produced and armed with abilities of sensing and computing, e.g., WISP tag [1]. The distinct advantages of RFID, such as low manufacture cost, wireless non-line-of-sight communication and parallel tag identification, make it widely deployed in various applications ranging from inventory control [2] and supply chain management [15], to object localization [9] and human-computer interaction [27].

In this paper, we study 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 [5]. Obviously, efficient tag search is desirable in this scenario to reduce financial loss and even avoid potential safety problems.

While deterministic schemes [5] can exactly pinpoint the wanted tags that are covered by the current RFID system, they are time-consuming for transmitting a large number of tag IDs. Instead, finding a set of wanted tags that locate within the coverage range with desired accuracy and probability is adequate in many RFID applications, where it is impossible for deterministic schemes to achieve the acceptable efficiency due to the overwhelming large volume of objects, e.g., RFID-enabled ports [4], [21]. For example, when searching for $20,000$ tags from the present tag set of $50,000$ tags, they require 19.4s and 130s, which are 4.4 and 29.6 times the searching time of the probabilistic scheme with the failure probability 0.001 [5]. Hence, it is necessary to improve the search efficiency to facilitate management of large-scale RFID systems, especially those with stringent time requirements.

To this end, several probabilistic search schemes [5], [19], [30], [32] are proposed to accelerate tag search with a guaranteed failure probability. Although these schemes employing Bloom filter [5], [30] or filtering vectors [19], [32] 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 [19] 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 schemes that cannot verify target tags in a batch are not efficient anymore. Moreover, PLAT [32] cannot work in the scenario with unknown tags in [5], [19], and [30] because it requires that the reader must have all IDs of tags present in the system besides those of wanted tags.

In this paper, we fundamentally shift from the traditional design paradigm to a new route. Specifically, 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. More 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 in TTS. Importantly, we prove that given a required failure probability $\frac{1}{\mathcal{O}(K)}$ where K is relative to the number of tags, TTS achieves a time cost 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, e.g., for a large $K{=}2^{96}$, with $d=4$ we have $\log^{(4)}K{=}1.4$. That is to say, the time cost of TTS approaches $\mathcal{O}(K)$.

We further improve the performance of TTS in the case that the wanted and the present tag sets have significantly different sizes with a two-step tag search scheme, named TTS+. Since the search efficiency is dominated by the bigger set size, the key idea of TTS+ is to shrink the size asymmetry first and use the reduced size to build a search tree. Specifically, we employ a filtering vector built from the smaller set to rule out partial ineligible tags of the bigger set in the first step. In the second step, we start accurate search among the remaining tags by executing TTS with the shrunk set size used in the tree construction. We derive the optimum parameters for TTS+ which are able to guarantee reliability demand and minimize time cost. Its superiority is confirmed by extensive simulations. In particular, as reliability demand scales, the time efficiency of TTS+ reaches nearly 2 times at most that of TTS.

[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 $i \geq 1$, $\log^{(i)}k = \max\{1, \log(\log^{(i-1)}k)\}$.

II. Background and Motivation

A. System Model

A typical RFID system consists of three parts: tags, a back-end server and one/multiple readers. The tags, each having a unique ID, can be either read-only or read/written by the reader wirelessly and implement the commands with lightweight hash functions [1]. The back-end 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 back-end server, transmit commands to the tags and report their responses to the server. When the back-end server synchronizes the readers, we can logically consider them as a whole [5], [19], [30]. We will denote the back-end server and the readers by the reader for simplicity.

The reader communicates with tags in a Listen-before-talk way [7]: the reader first queries tags with a command containing the parameters, such as frame size and random seed, to initiate communication. 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 at least one tag responds in this slot, it is called a busy slot; otherwise, it is called an empty slot.

B. The Tag Search Problem

Suppose that we have a known wanted tag set $X{=}\{x_1, x_2, \cdots, x_n\}$, and an unknown to-be-queried tag set $Y{=}\{y_1, y_2, \cdots, y_m\}$ containing m tags attached on the products currently within the coverage area of the RFID system. The tags of X and Y are referred to as *wanted tags* and *present tags*, respectively. 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, i.e., without IDs of the present tags in Y. While it is a common assumption [5], [19], [30] that the reader knows the cardinality of Y from the estimation through the tag counting schemes analyzed in [31].

Given the wanted tag set X, this paper intends to find which wanted tags in X are currently present in the coverage area of the RFID system, i.e., finding the intersection tag set $Z{=}X{\cap}Y$. 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? This paper is devoted to answering this question.

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 tag search problem is defined as follows.

Definition 1 (Tag search problem): Given X and Y, the tag search problem is to find $Z = X \cap Y$ with the success probability $Pr\{Z^* = Z\} \geq 1 - P_{fail}$ within minimum time. In this problem, P_{fail} should be relative to the number of tags [5], so we set $P_{fail}{=}\frac{1}{\mathcal{O}(K^a)}$, where $K{=}\max\{m, n\}$ and a constant $a{\geq}1$, so that the expected number of failure events $K{\cdot}P_{fail}$ among K runs could approach 0 for a large K.

C. Prior Art and Limitations

In this section, we briefly summarize the existing work related to the tag search problem in RFID systems. Various schemes have been proposed to collect tag IDs in RFID systems. In ALOHA-based identification schemes [14], [26], 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 schemes [10], [25], 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 [5].

Many research efforts have also been devoted to monitoring missing tag and unknown tag event. Missing tag monitoring aims at probabilistically [6], [20] or exactly [16], [17], [29] finding out the tags that should exist in the system but actually are absent. Unknown tag detection [8] and identification [18] are to probabilistically detect and deterministically identify the tags whose IDs have not been recorded. Opposite to missing and unknown tag monitoring, the tag search problem focuses on finding a particular set of tags in interrogation areas.

There are several probabilistic schemes designed to deal with the tag search problem. The works [5], [30] employ Bloom filter to encode tag IDs, accelerating the tag search task. The former [30], 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 [5]. Instead of transmitting a long Bloom filter in CATS, ITSP [5] decomposes it into multiple short filtering vectors and uses them to filter out ineligible tags iteratively, which reduces time cost. Two most recent works called E-STEP [19] and PLAT [32] exploit testing slots and non-testing slots to filter out ineligible tags. While PLAT [32] assumes that the reader knows IDs of all tags in the system besides those of wanted tags; therefore, it cannot work in the scenario in the presence of unknown tags in [5], [19], [30], and this paper.

In these works, 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 the tags in X. They 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 [30]) each consuming time $\mathcal{O}(K)$ in order to achieve the required failure probability $\mathcal{O}(\frac{1}{K^a})$ with $K{=}\max\{m, n\}$ and a constant a, which results in the overall time cost $\mathcal{O}(K \log K)$. These works set the cardinality of the set Z 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., $\mathcal{O}(K)$.

D. Motivation

It is desirable for a tag search scheme to have high reliability and time efficiency. The existing probabilistic tag search schemes, however, experience a significant degradation of time efficiency as the reliability demand increases. Specifically, we observe from these works that a large amount of time is wasted verifying each individual tag repetitively. Take [19] 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 [19]. 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 $\mathcal{O}(K)$?* Motivated by the observations, our design 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.

Following 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. III-D, 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 $\mathcal{O}(K \log K)$.

III. TTS: Tree-Based Tag Search

In this section, we show the basic idea of TTS with a simple example, and then detail its design and performance analysis.

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.

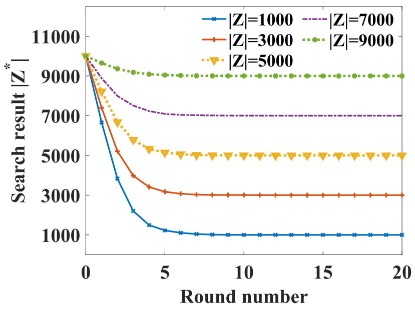
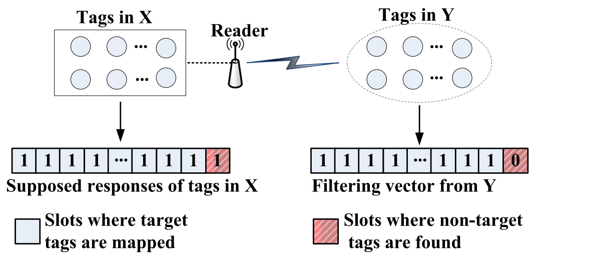


Fig. 1. Exemplify [19]: $P_{fail} = 10^{-4}$, $|X| = 10,000$, $|Y| = 20,000$.



Tags in X
Reader
Tags in Y
Supposed responses of tags in X
Filtering vector from Y
Slots where target tags are mapped
Slots where non-target tags are found

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.

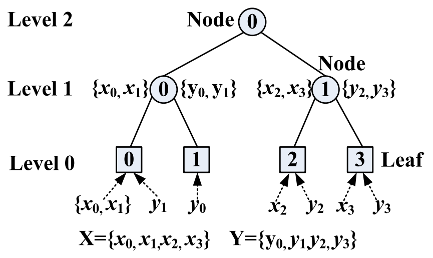


Fig. 3. Tree for Example 1.

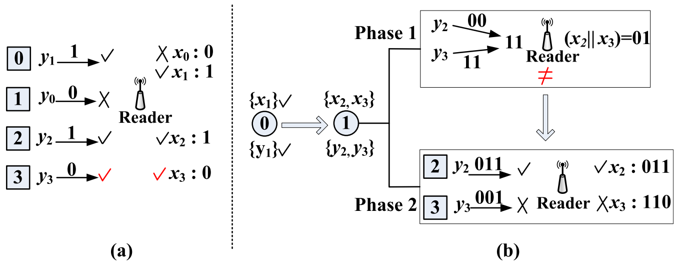


Fig. 4. Illustrate TTS on top of the tree in Example 1. (a) 1st round at the level 0. (b) 2nd round at the level 1.

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.[2] 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 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 a 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 smaller 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

[2]The tree here is binary, but is multiway generally as presented later.

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. III-D so that the required failure probability and time cost can be guaranteed.

B. Tree Architecture

As mentioned in Sec. II-D, 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: $U\rightarrow[K]$ 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, i.e., 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 because 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. Define $\mathcal{A}(j)^{-1}=\mathcal{A}(j)$ and $\mathcal{B}(j)^{-1}=\mathcal{B}(j)$ as the initial input for 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 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 $\mathcal{B}^{-1}=(\mathcal{B}(j)^{-1})_{0\le j\le K-1}$.

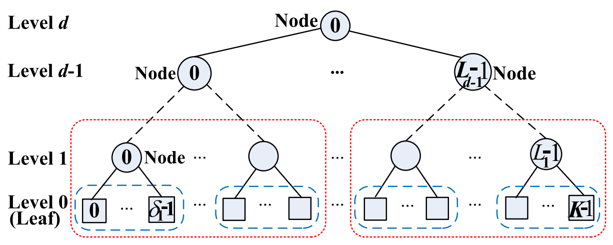


Fig. 5. Exemplify the built tree. Each leaf is assigned a group of tags, and each internal node contains tags of all leaves in its subtree, which can be interpreted as a bigger group, e.g., blue rectangles for nodes at the level 1, and red rectangles for nodes at the level $d-1$.

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 [12], [24], [28] 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 is to say, 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 [12], [24], [28] 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 [6], [28], and [29]. 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.

D. Performance Analysis

Since tree structure and success probability of two hashing-based operations, namely the verification and refinement functions, play important roles in the performance of TTS, we next study how to design the parameters such that TTS achieves $\frac{1}{\mathcal{O}(K^a)}$ failure probability and $\mathcal{O}(K\log^{(d)} K)$ time cost.

1) The failure probability of TTS. We first analyze 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 a 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 a 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 $0 \le i \le d-1$ 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

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

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

Given p_i, as each node v at level $i{>}0$ has $\Theta(v){=}\log^{(d-i)} K$ leaves in its subtree (c.f. (27) 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

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

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

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

2) Time cost of TTS. The overall expected time cost of TTS 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 (26) 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. As there are d rounds, the expected number of times the refinement function is run at each leaf is smaller than

$$1+\sum_{i=1}^{d-1}\delta_i \cdot (1-q_v) = 1+\sum_{i=1}^{d-1}\frac{(\log^{(d-i)}K)^2}{\log^{(d-i+1)}K\Big(\log^{(d-i+\alpha)}K\Big)^{\beta}}$$

which is $\mathcal{O}(1)$ when $\alpha=-1$ and $\beta=2$. Moreover, as there exist K leaves, we can compute the expected time cost for the refinement function, denoted by T_2, as

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

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

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

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 $\frac{1}{\mathcal{O}(K^{\alpha})}$ 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

$$P^*_{fail}\leq\frac{\log K}{K^{\beta}}\leq\frac{1}{K^{\beta-1}}, \quad (7)$$

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., $\mathcal{O}(\log^{(d-i)}K)$.

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}, it is enough to have

$$P^*_{fail}=P_{fail}\Rightarrow\beta=\frac{\log^{(2)}K-\log(P_{fail})}{\log^{(2+\alpha)}K}. \quad (8)$$

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 $\log^{(\alpha+2)}K\geq1$. For d, its minimum value is 2 and its maximum value is the one satisfying $\log^{(d+\alpha+1)}K\geq1$. Therefore, the optimal parameter collection, denoted by $\{\alpha^*,\beta^*,d^*\}$, can be obtained by solving $\min\limits_{\alpha,\beta,d}T$ 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.

Discussion on the assumption. In this paper, the existing techniques [12], [24], [28] 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 [22] 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.

IV. TTS+: Search With Asymmetric Tag Sets

With TTS we have achieved the failure probability $\mathcal{O}(\frac{1}{K^{\alpha}})$ at the time cost of $\mathcal{O}(K\log^{(d)}K)$ instead of $\mathcal{O}(K\log K)$ in prior work. On top of this result, we propose TTS+ to further enhance the time efficiency of TTS when the sizes of X and Y are remarkably different with two-step operations: asymmetry shrinkage and accurate tag search.

A. Motivation

1) Observations. From the analysis in Sec. III, we know that the time cost of TTS is $T=\mathcal{O}(K\log^{(d)}K)$, which is proportional to $K=\max\{|X|,|Y|\}$. That is to say, the bigger one between X and Y dominates the time cost of TTS. This makes sense when the set size of X and Y are close to each other. Yet it is unreasonable in the case that the two sets have remarkably asymmetric sizes. Look at a toy example: $|X|=100,000$ and $|Y|=10,000$. In this case TTS builds a search tree with $K=|X|$, i.e., it maps the tags of Y to K leaves, making $(1-\frac{1}{K})^{|Y|}\approx 90.5\%$ of K leaves empty for Y. It is worth noting that the reader can distinguish the ineligibility of the tags of X which are mapped to these empty leaves but cannot verify the tags of Y from these empty leaves. That is to say, TTS spends most of time combating with the interference of the non-target tags of X, which limits its time efficiency.

If we can first reduce the asymmetry between $|X|$ and $|Y|$, e.g., shrinking $|X|$ to $20,000$, even to $10,000$, and use the shrunk set size to build the tree, the rate of empty leaves for Y will decrease to 60.7% and 36.8%, and the time efficiency of TTS can thus be improved significantly.

2) Design guideline. According to the observations, a promising scheme should work in two-step pattern: First, we should screen out and suppress ineligible tags of the bigger set to relieve their interference while keeping eligible tags active for further accurate search. Second, the remaining tags after the first step will be fed into TTS to conduct the accurate tag search. Following this guideline, we design an extended tag search scheme on the top of TTS for higher time efficiency, named TTS+, which consists of two steps:

1) Asymmetry shrinkage: we build a filtering vector from the smaller tag set and use it to rule out the ineligible tags of the bigger set in this step.
2) Accurate tag search: we input the tags passing filtering test which comprise all target tags and partial ineligible tags into TTS for the further tag search.

The key challenge of TTS+ lies in configuring the parameters to guarantee the required failure probability while optimizing time efficiency. On one hand, a natural problem we need to tackle is how to determine the execution time of the first step, i.e., the filtering vector length, as a bigger filtering vector can rule out more ineligible tags but leads to a higher time cost. On the other hand, we need to tune parameters in the second step so that TTS+ using the shrunk set size in the second step outputs the same failure probability as TTS using the original size K. In what follows, we first describe the TTS+ design and then show how we address the challenge.

B. TTS+ Description

In this subsection, we start to formally introduce TTS+ that consists of two steps: asymmetry shrinkage and accurate tag search. Without loss of generality, we assume here $|X| \geq |Y|$ for clearness and will discuss the case of $|X| < |Y|$ shortly.

Step 1: asymmetry shrinkage. In the first step, we use a filtering vector to rule out the non-target tags in X. Specifically, the reader first issues a *query command* containing a random seed, the frame size f, the number of reply slots of a tag J. The parameter configuration will be studied in Sec.IV-C. Upon receiving the command, each tag in Y employs hash functions and the seed to map its ID to J slots, and replies to the reader in the corresponding slots. In each slot, the reader sends a *slot start command* and wait for responses. For each tag, if one of its reply slot numbers is equal to the current slot number, it will respond immediately. Otherwise, it will keep silent. There are thus two types of slots: empty slots and busy slots.

After the execution of f slots, i.e., one frame, the reader obtains all responses and can encode an f-bit filtering vector from these responses as follows: Initially, the filtering vector is null until the reader starts feeding it with the responses. For a position w in the filtering vector, i.e., the slot w in the frame, if this slot is busy, the reader sets the element in the position w of the filtering vector to '1', otherwise '0'. Repeating the operations for all positions, the reader obtains a filtering vector and can use it to prune non-target tags in X.

To test the tags in X, the reader maps each of them to J positions as the tags in Y, and checks these J positions in the filtering vector. If the elements in these J positions are all '1's, this tag is eligible. Otherwise, this tag is ineligible and will not participate into the subsequent operations.

Step 2: accurate tag search. Let X' define the set of the tags in X which pass the filtering vector test. In the second step, we execute TTS for the tag search with $K' = \max\{|X'|, |Y|\}$. Note that K' plays the same role as K in Sec.III-B. We will investigatet the parameter configuration in Sec.IV-C.

We would like to explain that TTS+ can be directly extended to the case of $|X| < |Y|$: the reader builds a filtering vector from X and broadcasts it to the tags in Y in the first step instead of constructing one from responses of the tags in Y. We here focus on the case of $|X| \geq |Y|$ in the analysis, but we consider both cases in the simulation.

C. Performance Analysis

In this subsection, we study how the parameters in TTS+ are configured to ensure the same failure probability while reducing time cost in comparison with TTS.

1) Parameters tuning in Step 1: the objective is to maximize the filtering success probability and formulate the execution time of this step. Since the reader and the tags in Y use the same hash functions and the seed, the target tags in X that belong to the intersection $Z = X \cap Y$ will only map to '1' positions in the filtering vector and deterministically pass the test. Yet there exist false positives, that is to say, some non-target tags in X may also pass the test. We now compute the false positive probability that is denoted by P_{fp}.

Recalling Step 1, we know that each tag in Y selects J slots in a frame, so the probability that a slot is still empty, i.e., the element in the corresponding position of the filtering vector is '0', after the responses of all tags in Y is equal to $(1-\frac{1}{f})^{J|Y|} \approx e^{-\frac{J|Y|}{f}}$.

Meanwhile, during the test, if all J positions a non-target tag in X maps to are '1's, it can pass the test and the reader cannot know its ineligibility. Hence, the false positive probability is related to the probability that an element in the filtering vector is '1', which can be calculated as $P_{fp} = \left(1 - e^{-\frac{J|Y|}{f}}\right)^J$.

In order to maximize the filtering efficiency, we need to minimize the false positive probability with respect to J with f fixed. To this end, we compute the derivative of P_{fp}:

$$\frac{\mathrm{d}P_{fp}}{\mathrm{d}J} = \left[\ln(1-e^{-\frac{J|Y|}{f}}) + \frac{\frac{J|Y|}{f}}{e^{\frac{J|Y|}{f}}-1}\right] \cdot \left(1-e^{-\frac{J|Y|}{f}}\right)^J.$$

If let this derivative equal zero, we get $J = \frac{f \ln 2}{|Y|}$ when the false positive probability is minimum, which is

$$P^*_{fp} = \left(\frac{1}{2}\right)^J. \tag{9}$$

This is because $\frac{\mathrm{d}P_{fp}}{\mathrm{d}J} \geq 0$ for $J \geq \frac{f \ln 2}{|Y|}$ while $\frac{\mathrm{d}P_{fp}}{\mathrm{d}J} < 0$ for $J < \frac{f \ln 2}{|Y|}$. We thus know the frame size

$$f = \frac{|Y|J}{\ln 2}, \tag{10}$$

which stands for the execution time of Step 1 and can be determined once the value of J is fixed. We will show how to configure J later.

Since X' represents the tags in X which pass the test, it consists of two parts: the tags in X belong to $X \cap Y$ and the mistaken non-target tags in X. Its size can be computed as

$$\begin{aligned} |X'| &\approx (|X| - |X \cap Y|) \cdot P^*_{fp} + |X \cap Y| \\ &\leq |Y| + (|X| - |Y|) \cdot P^*_{fp} \qquad (11) \\ &\geq (|X| - |Y|) \cdot P^*_{fp} \qquad (12) \end{aligned}$$

for $0 \leq |X \cap Y| \leq |Y|$.

2) Parameters tuning in Step 2: the objective is to formulate the execution time of this step and the failure probability. Recall that we input Y and X' to TTS in the second step of TTS+. Since the intersection size $|X \cap Y|$ cannot be known a prior, we use the upper bound of $|X'|$ in the computation of the execution time: $\overline{K'} = |Y| + (|X| - |Y|) \cdot P^*_{fp}$, while using the lower bound of $|X'|$ in the computation of the failure probability: $\underline{K'} = \max\{|Y|, (|X| - |Y|) \cdot P^*_{fp}\}$, which can guarantee all properties of TTS with the exact $|X'|$. Therefore, referring to (6), the execution time of Step 2, defined as T_{S2}, can be expressed as

$$\begin{aligned} T_{S2} \leq &\sum_{i=1}^{d-1} \frac{\beta \overline{K'} \log^{(d-i+\alpha+1)} \overline{K'}}{\log^{(d-i)} \overline{K'}} + \beta \overline{K'} \log^{(d+\alpha+1)} \overline{K'} \\ &+ \sum_{j=1}^{\overline{K'}} \sum_{i=1}^{d-1} \frac{\beta \delta_i (\log^{(d-i)} \overline{K'}) \log^{(d-i+\alpha+1)} \overline{K'}}{\left(\log^{(d-i+\alpha)} \overline{K'}\right)^{\beta}}. \end{aligned} \tag{13}$$

We use $\overline{T_{S2}}$ to express the right side of the inequality.

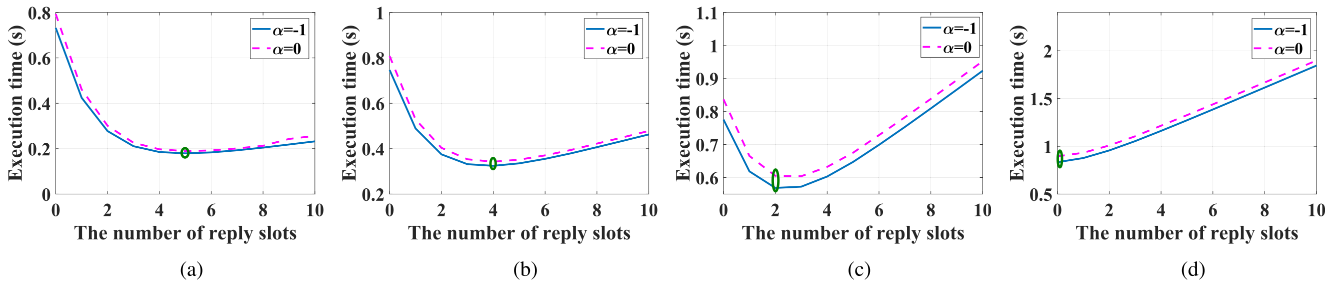


Fig. 6. Different numbers of reply slots vs. execution time under diverse $|Y|$: $d = 2$, $|X| = 10000$, $P_{fail} = 10^{-4}$, transmission rate 100kbps. (a) $|Y| = 1000$. (b) $|Y| = 2000$. (c) $|Y| = 4000$. (d) $|Y| = 8000$.

We proceed to prove that TTS+ can achieve the same failure probability $\frac{1}{K^a}$ as TTS in Sec.III. Recalling (5), we know that the failure probability is inversely proportional to K', so setting K' to $\underline{K'}$ yields

$$P^*_{fail} \leq \frac{\log \underline{K'}}{\left(\log^{(1+\alpha)} \underline{K'}\right)^{\beta}} \leq \frac{1}{(\underline{K'})^{\beta-1}}, \tag{14}$$

where the second inequality holds for $\alpha = -1$ and $\beta > 1$. For the required failure probability $\frac{1}{K^a}$, we have

$$\frac{1}{(\underline{K'})^{\beta-1}} = \frac{1}{K^a} \Rightarrow \beta = 1 + \frac{a \log K}{\log \underline{K'}}. \tag{15}$$

Similarly, given an arbitrary requirement on the failure probability P_{fail}, it is enough to have

$$P^*_{fail} = P_{fail} \Rightarrow \beta = \frac{\log^{(2)} \underline{K'} - \log(P_{fail})}{\log^{(2+\alpha)} \underline{K'}}. \tag{16}$$

β is a function of J for $\underline{K'} = \max\{|Y|, (|X| - |Y|) \cdot (\frac{1}{2})^J\}$.

3) Overall time cost optimization for TTS+. From the analysis above, we can obtain the overall time cost of TTS+, denoted by T_I, as follows:[3]

$$T_I = f + T_{S2}. \tag{17}$$

The results shown in (10) and (13) implies that a bigger J enlarges f but lessens T_{S2}, suggesting the existence of an optimum of J. Specifically, the frame size increases with J leading to a longer Step 1, but as a result, the set X' of the tags that pass Step 1 will become smaller and thus the time cost of Step 2 will be reduced. Note that if the optimum of J is equal to zero, it means that TTS+ will not run the first step and degrade to the original TTS.

Let us look at an illustrative example showing the impact of different J on the time cost T_I. The settings are: $d = 2$; $|X| = 10,000$; $P_{fail} = 10^{-4}$; $\alpha = -1$ and 0; and the symmetric transmission rate 100 kbps. By varying $|Y|$ from $1,000$ to $8,000$, we intend to illustrate how the optimum of J changes across diverse $\frac{|Y|}{|X|}$. As shown in Fig. 6, the optimum of J decreases from 5 to 0 as $|Y|$ closes to $|X|$, and T_I is convex with respect to J. The numerical results confirm our judgement on the relationship between T_I and J.

Searching for the exactly minimum T_I. Since T_I is too complicated to directly derive its exact minimum value, we plan to solve this problem by searching in the feasible region. We have obtained the constrains on d, α in Sec.III-D and β in (16), so the key here is to fix the range of J. To this end, we prove in **Lemma 4** in Appendix that T_I does not decrease with J after J is over a threshold J_{th}. Therefore, minimizing the time cost T_I can be formulated as the following optimization problem that can be solved via direct searching:

$$\text{obj.: } \min\ T_I \tag{18}$$

$$\text{s.t.: } \alpha \geq -1 \text{ and } \log^{(\alpha+2)} \underline{K'} \geq 1 \tag{19}$$

$$d \geq 2 \text{ and } \log^{(d+\alpha+1)} \overline{K'} \geq 1 \tag{20}$$

$$J \in [0, J_{th}] \text{ and } (16) \tag{21}$$

The analysis above shows that we can search the feasible region for the optimum J that minimizes T_I, the theoretical characterizes of T_I with respect to J has yet to be studied. To understand better behavior of T_I, in what follows, we proceed to derive the relationship between T_I and J theoretically.

Theoretical analysis on T_I.[4] We require the failure probability $P_{fail} = \frac{1}{K^a}$ where $a \geq 1$, as stated in Definition 1. To make the analysis feasible, we will further amplify $\overline{T_{S2}}$ and study this loosened upper bound of T_I.

To guarantee $P_{fail} = \frac{1}{K^a}$, α should be -1 when we have

$$\overline{T_{S2}} < 2\beta(d-1)\overline{K'} + \beta\overline{K'}\log^{(d)} \overline{K'}$$
$$< \left(2(d-1) + \log^{(d)} \overline{K'}\right)\left(1 + \frac{a \log K}{\log \underline{K'}}\right)\overline{K'} \tag{22}$$

following from (15). We will proceed in two cases for $\underline{K'} = \max\{|Y|, (|X| - |Y|) \cdot (\frac{1}{2})^J\}$.

Case 1: $\underline{K'} = (|X| - |Y|) \cdot (\frac{1}{2})^J$ when $J \leq \log \frac{|X|-|Y|}{|Y|}$. Substituting this $\underline{K'}$ and (22) to (17) yields

$$T_I = \frac{|Y|J}{\ln 2} + \left(2(d-1) + \log^{(d)} \overline{K'}\right)\overline{K'}$$
$$\cdot \left(1 + \frac{a \log K}{\log(|X| - |Y|) \cdot (\frac{1}{2})^J}\right). \tag{23}$$

We observe by studying the derivative of T_I that its derivative is negative, meaning the decrease in the value of T_I with J when $J \leq \log \frac{|X|-|Y|}{|Y|}$. The technical proof is detailed in **Lemma 5** in Appendix.

Case 2: $\underline{K'} = |Y|$ when $J \geq \log \frac{|X|-|Y|}{|Y|}$. In this case, we can obtain (24) from (22). Since $\log^{(d)}(\cdot)$ will converge to a small number, the amplification here is compact:

$$\overline{T_{S2}} < \left(2(d-1) + \log^{(d)}(2|Y|)\right)\left(1 + \frac{a \log K}{\log |Y|}\right)\overline{K'}. \tag{24}$$

[3]We provide the exact formula of T_I in (17), but we set $T_{S2} = \overline{T_{S2}}$ by default in the subsequent analysis. That is to say, we try to optimize the upper bound of T_I.

[4]T_I here is greater than that in (17) as we loosen $\overline{T_{S2}}$ in order to obtain the closed-form optimum J.

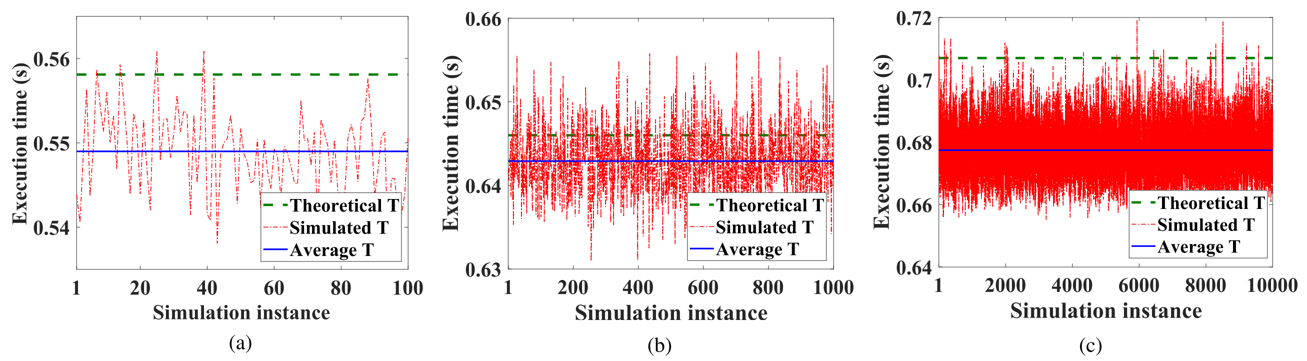


Fig. 7. TTS: Relationship between theoretical and simulation results under different P_{fail}. Parameter setting: $|X| = 5,000$, $|Y| = 10,000$, $\lambda = 0.5$. (a) $P_{fail} = 10^{-2}$, # of runs: 100. (b) $P_{fail} = 10^{-3}$, # of runs: $1,000$. (c) $P_{fail} = 10^{-4}$, # of runs: $10,000$.

Similarly, we can express T_I in this case as

$$T_I = \frac{|Y|J}{\ln 2} + \left(2(d-1) + \log^{(d)} 2|Y|\right)\left(1 + \frac{a \log K}{\log |Y|}\right) \cdot \left(|Y| + (|X| - |Y|)\left(\frac{1}{2}\right)^J\right). \quad (25)$$

We analyze the derivative of T_I in this case and prove in **Lemma 6** in Appendix that T_I is convex with respect to J and reaches its optimal value when $J = \log \frac{\left[2d-2+\log^{(d)}(2|Y|)\right]\left(1+\frac{a \log K}{\log |Y|}\right)}{|Y|/[(|X|-|Y|)(\ln 2)^2]} \triangleq \hat{J}$.

From the analysis in Case 1 and Case 2, we know: Because $\log \frac{|X|-|Y|}{|Y|} < \hat{J}$, T_I decreases with J when $J \leq \hat{J}$. Otherwise, it increases with J. Therefore, $\hat{J}$ is the optimum of J that minimizes T_I. Since $\hat{J}$ is derived from the upper bound of time cost and must be an integer, we can set $\hat{J}=\lfloor \hat{J} \rfloor$ in practice.

Given the system and the other parameters, we can obtain the value of $\hat{J}$. For example, recall the optimum of J for $\alpha = -1$ are 5, 4, 2, 0 in Fig. 6, we can drive that $\hat{J}$= 5.8, 4.6, 3.1, 0.5 in the same settings. By rounding down $\hat{J}$, we have 5, 4, 3, 0, most of which are equal to the searched optimal value except $3 > 2$. To reduce this probability, we can search around $\hat{J}$, e.g., comparing the time cost at $\hat{J} - 1$ to that at $\hat{J}$ and finally using the one that minimizes the time cost.

V. Performance Evaluation

In the simulation, we use the communication parameters specified in the EPC global C1G2 standard [7]. In our experiments, we set tag-to-reader transmission rate and reader-to-tag transmission rate to 100kbps as in [5], 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 and TTS+ are set according to our theoretical analysis. Besides, we also set up E-STEP with its optimal parameter configuration [19].

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

TABLE I
Performance Investigation of TTS+ and TTS: $|X| = 5,000$, $|Y| = 10,000$, $\lambda = 0.5$

Required P_{fail}	Achieved P^*_{fail}	d	α	β	J
10^{-2}	(0,0)	(2,2)	(0,0)	(3,3)	(1,–)
10^{-3}	(0,0)	(3,3)	(0,0)	(4,4)	(2,–)
10^{-4}	(0,0)	(2,2)	(-1,-1)	(2,2)	(2,–)

A. Performance Verification

We demonstrate that TTS and 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 and TTS+ are 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 report the failure of the tag search task if this number exceeds one.

Table I lists the failure probability achieved by TTS and TTS+ and their optimal parameters, where the two elements in $(\cdot,\cdot)$ represent the value in TTS+ and TTS respectively and "–" means not applicable. It can be observed that TTS+ and TTS achieve the zero failure probability in all three experiments, that is to say, they are 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 (8) and (16) that TTS and TTS+ are able to achieve the failure probability at most $\frac{1}{K}$ when $\alpha = -1$. In addition, when P_{fail} changes from 10^{-2} to 10^{-3}, β increases by one while α keeps constant, and J in TTS+ rises to 2 reducing the increment in the time cost of the second step resulted from the higher requirement. These results suggest the ability of TTS and TTS+ of adjusting the parameters to the user-defined requirement.

We also record the execution time of TTS and TTS+ in each run and obtain the average value over all runs, which are depicted in Fig. 7 and Fig. 8, respectively. One thing worth noting is that the theoretical T and T_I are the upper bound of the expected execution time. In the figures, the average execution time calculated from the simulation, referred to as average T and T_I, is upper bounded by the theoretical T

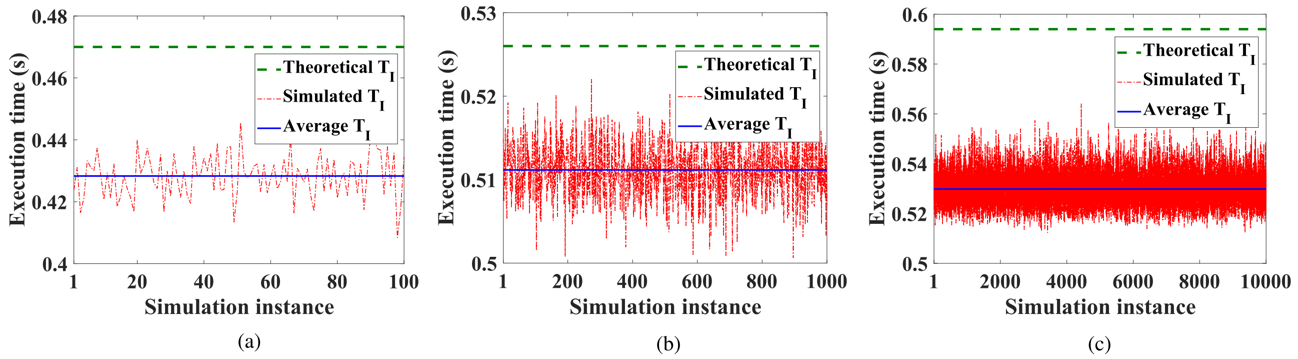


Fig. 8. TTS+: Relationship between theoretical and simulation results under different P_{fail}. Parameter setting: $|X| = 5,000$, $|Y| = 10,000$, $\lambda = 0.5$. (a) $P_{fail} = 10^{-2}$, # of runs: 100. (b) $P_{fail} = 10^{-3}$, # of runs: 1,000. (c) $P_{fail} = 10^{-4}$, # of runs: 10,000.

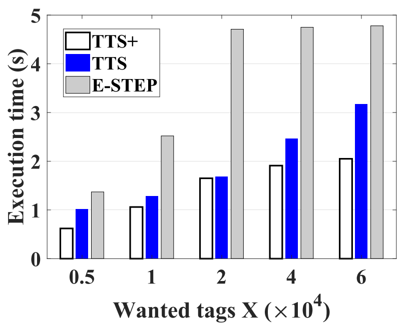


Fig. 9. Performance comparison with different $|X|$: $P_{fail} = 10^{-5}$, $|Y| = 20,000$, $\lambda = 0.5$.

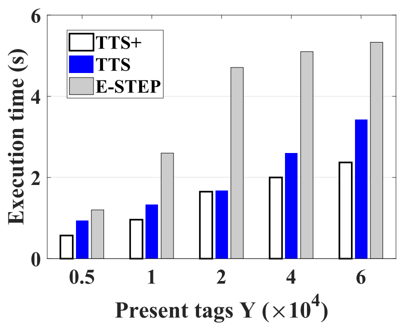


Fig. 10. Performance comparison with different $|Y|$: $P_{fail} = 10^{-5}$, $|X| = 20,000$, $\lambda = 0.5$.

and T_I derived from our analysis. The results indicate that the execution time varies slowly compared to the significant change of the required failure probability. Specifically, TTS and TTS+ just consume extra 20% and 24% time to reduce the failure probability from 0.01 to 0.0001.

B. Performance Comparison

We here start comparing performance of TTS and TTS+ with the state-of-the-art probabilistic tag search E-STEP [19].

First, we compare the time efficiency of three approaches 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 5,000 to 60,000. Fig. 9 depicts their execution time to fulfill the tag search task.

As shown in the figure, TTS+ is the most time-efficient and TTS performs better than E-STEP. Specifically, with the same P_{fail}, the execution time of E-STEP is up to two and three times as much as that of TTS and TTS+. From a different point of view, this result also suggests that given a certain amount of searching time, the failure probability of TTS and 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 5,000 to 60,000. From Fig. 10, we can observe the similar results that the time efficiency of TTS and TTS+ is significantly superior to E-STEP, and TTS+ can save time up to 40% compared with TTS.

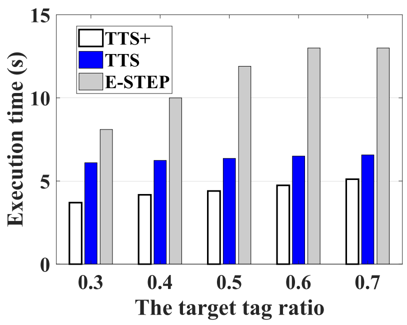


Fig. 11. Performance comparison with different λ: $P_{fail} = 10^{-5}$, $|X| = 100,000$, $|Y| = 50,000$.

Third, we compare the time efficiency of three approaches under different target tag ratios λ. Given the particular $P_{fail} = 10^{-5}$, we use the following setting: $|X| = 100,000$, $|Y| = 50,000$, and $\lambda = 0.3 : 0.1 : 0.7$. The simulation results are exhibited in Fig. 11. As shown in the figure, TTS and TTS+ still remarkably outperform E-STEP, specifically, with the performance gain of up to 100% and 176%. Moreover, E-STEP experiences a significant increase in the execution time with the increase of the target tag ratio. In contrast, TTS and TTS+ perform more stably.

After 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 II summarizes the execution time of

TABLE II

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

P_{fail}	10^{-4}	10^{-6}	10^{-8}	10^{-10}	10^{-12}	10^{-16}
TTS+	2.3	2.3	3.0	3.0	3.5	3.5
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

TABLE III

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

P_{fail}	10^{-4}	10^{-6}	10^{-8}	10^{-10}	10^{-12}	10^{-16}
TTS+	4.8	4.8	4.8	6.0	6.0	6.5
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

the three approaches. As listed in Table II, TTS and TTS+ spend less time achieving the required failure probability than E-STEP, especially under the smaller P_{fail}. Specifically, when P_{fail}=10^{-16}, TTS consumes less than the half of the execution time of E-STEP while TTS+ just needs one quarter.

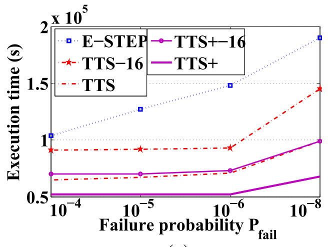
For a comprehensive comparison, we conduct another set of experiments where we keep the settings in Table II and just set $|X|$ to $100,000$. From the results listed in Table III, we can draw the similar conclusion that TTS+ is of the greatest scalability to reliability requirement. For example, the execution time of TTS+ that is half of E-STEP when P_{fail}=10^{-4} becomes to one sixth when P_{fail}=10^{-16}. In fact, given the required failure probability $\frac{1}{\mathcal{O}(K^a)}$, TTS and TTS+ consume $\mathcal{O}(K \log^{(d)} K)$ time compared to $\mathcal{O}(K \log K)$ of E-STEP, their performance gain over E-STEP will be rather larger when the requirement on the reliability scales up. Besides, we observe that TTS spends nearly $2\times$ time achieving the failure probability of 10^{-16} in comparison with TTS+.

VI. PRACTICAL ISSUES IN IMPLEMENTATION

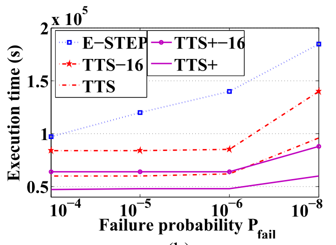
In this section, we discuss some practical issues arising in the parallel decoding techniques and scheme implementation.

Parallel decoding: overhead and reliability. In our work, we assume that r-bit sequences can be decoded, while the existing methods [11], [12], [24], [28] usually decode RN16 by default which is a random 16-bit sequence. To more accurately evaluate time cost of our schemes, we set the hash values sent by tags to be 16-bit long at least. Consider a pair of settings: $|X|$=$5,000$, $|Y|$=$10,000$; $|X|$=$10,000$, $|Y|$=$5,000$; we set $\lambda = 0.5$, and change P_{fail} from 10^{-4} to 10^{-8}. The results are depicted in Fig. 12 where TTS-16 and TTS+-16 are their individual original schemes with 16-bit configuration. It can be observed that TTS-16 and TTS+-16 need another 46.5% and 46.7% time at most compared with their original schemes, yet they are still more time-efficient than E-STEP.

To further evaluate the impact of parallel decoding technologies on our schemes, given the same settings as above, we fix $P_{fail} = 10^{-4}$ and vary decoding reliability with which the decoding is successful. As shown in Fig. 13, the number of found target tags reduces with decrease of the decoding reliability. Moreover, TTS+ outperforms TTS when $|X| < |Y|$ but performs worse in the other case. This can be interpreted as: TTS+ shrinks the asymmetry of $|X|$ and $|Y|$ to nearly 1 : 1.1. Then the number of leaves is set to the shrunk size when the probability of multiple tags mapped to one leaf is 0.23. In contrast, TTS sets the number of leaves to $\max\{|X|, |Y|\}$, i.e., $10,000$ here, so that the collision probabilities in two cases are 0.26 and 0.09, respectively. Thus, TTS depends on the decoding reliability stronger than TTS+ in the first case but weaklier in the second case.

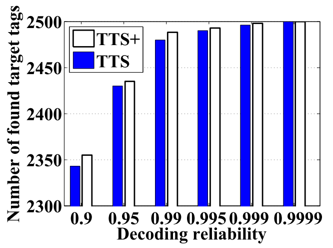


(a)

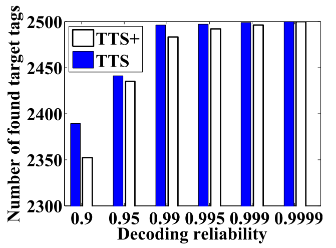


(b)

Fig. 12. Execution time under diverse required failure probability P_{fail}. (a) $|X| = 5,000$, $|Y| = 10,000$. (b) $|X| = 10,000$, $|Y| = 5,000$.



(a)



(b)

Fig. 13. The number of found target tags under diverse decoding reliability. (a) $|X| = 5,000$, $|Y| = 10,000$. (b) $|X| = 10,000$, $|Y| = 5,000$.

Potential implementation. Consider the implementation of the proposed schemes, programmable tags, such as those based on WISP hardware, and a USRP-based Software-Defined RFID reader are needed. In order to achieve hashing functionality, a 128-bit (or 256-bit) hash value that is long enough is pre-stored in each tag, which is supported by WISP 4.1. In the scheme implementation, three commands need to be added: 1) TRANSIV that is used to transmit a filtering vector; 2) DECORES that is used to initiate a slot and measure and decode responses from tags. The reader broadcasts DECORES command along with other parameters, e.g., the hash value size r_i and l_i, random seeds, the number of response slots J; 3) REPLHV that can guide tags to reply with specified hash value at specified slots. When a programmable tag receives DECORES, it replies with r_i-bit (or l_i-bit) hash value randomly selected from the pre-stored 128-bit hash value at the slots corresponding to the J positions in the stored 128-bit hash value which are indicated by random seeds.

VII. CONCLUSION

In this paper, we have studied the tag search problem in large-scale RFID systems. With the observation that prior work wastes much time verifying each individual target tag repeatedly, we have designed an efficient 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 have theoretically demonstrated that TTS can achieve the failure probability $\frac{1}{\mathcal{O}(K^{a})}$ at the time cost of $\mathcal{O}(K\log^{(d)}K)$, providing a significant improvement over prior $\mathcal{O}(K\log K)$. Furthermore, on the top of TTS, an extended two-step tag search scheme, namely TTS+, has been proposed to combat asymmetric tag sets in order to improve time efficiency of TTS. TTS+ exploits filtering technique to shrink set asymmetry first and then feeds the remaining tags into TTS. The simulation results have shown that the time efficiency of TTS+ could reach about two times that of TTS.

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:

$$|L_i| = \begin{cases} K, & \text{if } i = 0 \\ \frac{K}{\log^{(d-i)} K}, & \text{if } 1 \le i \le d, \end{cases} \tag{26}$$

$$|\Theta(v)| = \begin{cases} 1, & \text{if } i = 0 \\ \log^{(d-i)} K, & \text{if } 1 \le i \le d. \end{cases} \tag{27}$$

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 a node has δ_i children, $|\Theta(v)|$ is got from $\prod_{i'=1}^{i}\delta_i$. ■

Lemma 2: [3], [22], [23] Given a hash function into r bits, if two sets A and B are identical, their aggregated hash values are equal with probability 1. Otherwise, their aggregated values are unequal with a probability at least $1{-}\frac{1}{2^r}$.

Lemma 3: [22] 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 a probability at least $1 - \frac{1}{u^b}$.

Lemma 4: Given d, α, to find the minimum T_I, it is enough to probe every integer J in the range $[0, J_{th}]$ where $J_{th} = 2\,J^*$ and J^* is the solution to the following equation:

$$J^* = \frac{\overline{T_{S2}}(J^*)\ln 2}{|Y|}. \tag{28}$$

Proof: Given d and α, since β is relative to J, T_{S2} and T_I are functions of J, which can be expressed as T_{S2} and $T_I(J)$, respectively. To prove the lemma, it is enough to show that for any $J{>}2J^*$ it holds that $T_I(J){>}T_I(J^*)$. The constant J^* will be studied below. Recalling (13) and (17), we have

$$T_I(J) > \frac{|Y|J}{\ln 2} \tag{29}$$

$$T_I(J) \le \frac{|Y|J}{\ln 2} + \overline{T_{S2}}(J) \tag{30}$$

Setting J^* to the solution to $\frac{|Y|J}{\ln 2} = \overline{T_{S2}}(J)$, we can have $T_I(J) > \frac{|Y|2J^*}{\ln 2} \ge T_I(J^*)$ for any $J > 2J^*$. Thus, $T_I(J)$ does not decrease with J once its value is over $2J^*$. ■

Lemma 5: T_I is a monotonously decreasing function with respect to J for $J \le \log\left(\frac{|X|-|Y|}{|Y|}\right)$.

Proof: We derive the derivative of T_I with respect to J. We use $u{=}(|X| - |Y|)(\frac{1}{2})^J$ and $u^*{=}u{+}|Y|$ for simplification.

$$\begin{aligned}\frac{\mathrm{d}T_I}{\mathrm{d}J} &= \frac{|Y|}{\ln 2} - \frac{ua\ln 2\cdot\log K}{(\log u)^2}\Bigg[2(d-1) + \log^{(d)} u^* \\ &\quad + \prod_{c=1}^{d-1}\frac{1}{\log^{(c)} u^*}\Bigg]\cdot\left(\log u + \frac{(\log u)^2}{a\log K}\right) \\ &\quad + \frac{ua\ln 2\cdot\log K}{(\log u)^2}\Big[2(d-1) + \log^{(d)} u^*\Big]\left(1 + \frac{Y}{u}\right) \\ &< \frac{\ln 2\cdot\log K}{(\log u)^2/(ua)}\Bigg[\frac{3(\log u)^2}{a\log K} - \Big[\log^{(d)} u^* + 2(d-1)\Big] \\ &\quad \cdot\left(\log u + \frac{(\log u)^2}{a\log K}\right) + 4(d-1) + 2\log^{(d)} u^*\Bigg] < 0.\end{aligned}$$

Hence, the lemma follows from the negative derivative. ■

Lemma 6: For $J \ge \log\frac{|X|-|Y|}{|Y|}$, T_I is convex and its minimum exists when $J = \log\frac{\left[2d-2+\log^{(d)}(2|Y|)\right]\left(1+\frac{a\log K}{\log|Y|}\right)}{|Y|/[(|X|-|Y|)(\ln 2)^2]}$.

Proof: We first calculate the derivative of T_I:

$$\begin{aligned}\frac{\mathrm{d}T_I}{\mathrm{d}J} &= \frac{|Y|}{\ln 2} - \left(2(d-1) + \log^{(d)} 2|Y|\right)\left(1 + \frac{a\log K}{\log|Y|}\right) \\ &\quad \cdot(|X| - |Y|)\left(\frac{1}{2}\right)^J \ln 2.\end{aligned}$$

We then derive the second order derivative and have

$$\begin{aligned}\frac{\mathrm{d}^2\,T_I}{\mathrm{d}J^2} &= \left(2(d-1) + \log^{(d)} 2|Y|\right)\left(1 + \frac{a\log K}{\log|Y|}\right) \\ &\quad \cdot(|X| - |Y|)\left(\frac{1}{2}\right)^J (\ln 2)^2 \quad \ge 0.\end{aligned}$$

Hence, T_I is a convex function of J. Let $\frac{\mathrm{d}T_I}{\mathrm{d}J} = 0$, we have

$$J = \log\frac{\left[2d - 2 + \log^{(d)}(2|Y|)\right]\left(1 + \frac{a\log K}{\log|Y|}\right)}{|Y|/[(|X| - |Y|)(\ln 2)^2]} \tag{31}$$

which minimizes T_I. ■

References

[1] *WISP Platform*. [Online]. Available: http://wisp.wikispaces.com/

[2] Barcoding Inc. *How RFID Works for Inventory Control in the Warehouse*. [Online]. Available: http://www.barcoding.com/rfid/inventory-control.shtml

[3] J. Brody, A. Chakrabarti, R. Kondapally, D. P. Woodruff, and G. Yaroslavtsev, "Beyond set disjointness: The communication complexity of finding the intersection," in *Proc. ACM PODC*, 2014, pp. 106–113.

[4] M. Cash. *Third Business Mission Focuses on Cargo-Tracking Technology*. [Online]. Available: https://www.winnipegfreepress.com/business/centreportheading-back-to-china-147708545.html

[5] M. Chen, W. Luo, Z. Mo, S. Chen, and Y. Fang, "An efficient tag search protocol in large-scale RFID systems," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 899–907.

[6] M. Chen, J. Liu, S. Chen, Y. Qiao, and Y. Zheng, "DBF: A general framework for anomaly detection in RFID systems," in *Proc. IEEE INFOCOM*, May 2017, pp. 1–9.

[7] EPCglobal Inc. (2005). *Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz–960 MHz*. [Online]. Available: http://www.gs1.org

[8] W. Gong, J. Liu, and Z. Yang, "Efficient unknown tag detection in large-scale RFID systems with unreliable channels," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, pp. 2528–2539, Aug. 2017.

[9] J. Han *et al.*, "Twins: Device-free object tracking using passive tags," *IEEE/ACM Trans. Netw.*, vol. 24, no. 3, pp. 1605–1617, Jun. 2016.

[10] Y. Hou and Y. Zheng, "PHY assisted tree-based RFID identification," in *Proc. IEEE INFOCOM*, 2017, pp. 1–9.

[11] M. Jin, Y. He, X. Meng, Y. Zheng, D. Fang, and X. Chen, "Fliptracer: Practical parallel decoding for backscatter communication," in *Proc. ACM MobiCom*, 2017, pp. 275–287.

[12] J. Kaitovic and M. Rupp, "Improved physical layer collision recovery receivers for RFID readers," in *Proc. IEEE RFID*, Apr. 2014, pp. 103–109.
[13] F. Klaus, *RFID Handbook: Radio-frecuency Identification: Fundamentals and Applications*. Hoboken, NJ, USA: Wiley, 1999.
[14] L. Kong, L. He, Y. Gu, M.-Y. Wu, and T. He, "A parallel identification protocol for RFID systems," in *Proc. IEEE INFOCOM*, Apr./May 2014, pp. 154–162.
[15] C.-H. Lee and C-W. Chung, "Efficient storage scheme and query processing for supply chain management using RFID," in *Proc. ACM SIGMOD*, 2008, pp. 291–302.
[16] T. Li, S. Chen, and Y. Ling, "Identifying the missing tags in a large RFID system," in *Proc. ACM MobiHoc*, 2010, pp. 1–10.
[17] X. Liu, K. Li, G. Min, Y. Shen, A. X. Liu, and W. Qu, "Completely pinpointing the missing RFID tags in a time-efficient way," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 87–96, Jan. 2015.
[18] X. Liu *et al.*, "Efficient unknown tag identification protocols in large-scale RFID systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 12, pp. 3145–3155, Dec. 2014.
[19] X. Liu, B. Xiao, S. Zhang, K. Bu, and A. Chan, "STEP: A time-efficient tag searching protocol in large RFID systems," *IEEE Trans. Comput.*, vol. 64, no. 11, pp. 3265–3277, Nov. 2015.
[20] W. Luo, S. Chen, Y. Qiao, and T. Li, "Missing-tag detection and energy–time tradeoff in large-scale RFID systems with unreliable channels," *IEEE/ACM Trans. Netw.*, vol. 22, no. 4, pp. 1079–1091, Aug. 2014.
[21] M. McNeill. *Officials Hope Tags Help Future Sales*. [Online]. Available: https://www.winnipegfreepress.com/business/manitoba-fish-off-tochina-180669521.html
[22] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge, U.K.: Cambridge Univ. Press, 2017.
[23] M. Mitzenmacher and S. Vadhan, "Why simple hash functions work: Exploiting the entropy in a data stream," in *Proc. ACM SODA*, 2008, pp. 746–755.
[24] J. Ou, M. Li, and Y. Zheng, "Come and be served: Parallel decoding for cots RFID tags," in *Proc. ACM MobiCom*, 2015, pp. 500–511.
[25] M. Shahzad and A. X. Liu, "Probabilistic optimal tree hopping for RFID identification," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 41, no. 1, pp. 293–304, 2013.
[26] B. Sheng, Q. Li, and W. Mao, "Efficient continuous scanning in RFID systems," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.
[27] J. Wang, D. Vasisht, and D. Katabi, "RF-IDraw: Virtual touch screen in the air using RF signals," in *Proc. ACM SIGCOMM*, 2014, pp. 235–246.
[28] J. Wang, H. Hassanieh, D. Katabi, and P. Indyk, "Efficient and reliable low-power backscatter networks," in *Proc. ACM SIGCOMM*, 2012, pp. 61–72.
[29] Y. Zheng and M. Li, "P-MTI: Physical-layer missing tag identification via compressive sensing," *IEEE/ACM Trans. Netw.*, vol. 23, no. 4, pp. 1356–1366, Aug. 2015.
[30] Y. Zheng and M. Li, "Fast tag searching protocol for large-scale RFID systems," *IEEE/ACM Trans. Netw.*, vol. 21, no. 3, pp. 924–934, Jun. 2013.
[31] Z. Zhou, B. Chen, and H. Yu, "Understanding RFID counting protocols," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, pp. 312–327, Feb. 2016.
[32] F. Zhu, B. Xiao, J. Liu, X. Liu, and L.-J. Chen, "PLAT: A physical-layer tag searching protocol in large RFID systems," in *Proc. IEEE SECON*, Jun. 2016, pp. 1–9.

Jihong Yu received the B.E. degree in communication engineering and the M.E. degree in communication and information systems from the Chongqing University of Posts and Telecommunications, Chongqing, China, in 2010 and 2013, respectively, and the Ph.D. degree in computer science from the University of Paris-Sud, Orsay, France, in 2016. He was a Research Fellow with the School of Computing Science, Simon Fraser University. He is currently an Associate Professor with the School of Information and Electronics, Beijing Institute of Technology. His research interests include radio frequency identification technologies, wireless communications, and Internet of Things.

Wei Gong (M'14) received the B.S. degree from the Department of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China, in 2003, the M.S. degree from the School of Software, Tsinghua University, Beijing, China, in 2007, and the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, in 2012. He was a Research Fellow with the School of Computing Science, Simon Fraser University. He is currently a Professor with the School of Computer Science and Technology, University of Science and Technology. His research interests include backscatter networks, mobile computing, and Internet of Things.

Jiangchuan Liu (S'01–M'03–SM'08–F'17) received B.Eng. degree *(cum laude)* from Tsinghua University, Beijing, China, in 1999, and the Ph.D. degree from The Hong Kong University of Science and Technology in 2003. He is currently a Full Professor (with University Professorship) with the School of Computing Science, Simon Fraser University, BC, Canada. He is an IEEE Fellow and an NSERC E.W.R. Steacie Memorial Fellow.

Prof. Liu is a Steering Committee Member of the IEEE TRANSACTIONS ON MOBILE COMPUTING. He was a co-recipient of the Test of Time Paper Award of IEEE INFOCOM in 2015, the ACM TOMCCAP Nicolas D. Georganas Best Paper Award in 2013, and the ACM Multimedia Best Paper Award in 2012. He is an Associate Editor of the IEEE/ACM TRANSACTIONS ON NETWORKING and the IEEE TRANSACTIONS ON MULTIMEDIA.

Lin Chen (S'07–M'10) received the B.E. degree in radio engineering from Southeast University, China, in 2002, the Dipl.-Ing. degree from Telecom ParisTech, Paris, in 2005, and the M.S. degree in networking from the University of Paris 6. He is currently an Associate Professor with the Department of Computer Science, University of Paris-Sud. His main research interests include modeling and control for wireless networks, distributed algorithm design, and game theory. He serves as the Chair for the IEEE Special Interest Group on Green and Sustainable Networking and Computing with Cognition and Cooperation, IEEE Technical Committee on Green Communications and Computing.

Kehao Wang received the B.S. degree in electrical engineering and the M.S. degree in communication and information system from the Wuhan University of Technology, China, in 2003 and 2006, respectively, and Ph.D. degree from the Department of Computer Science, University of Paris-Sud, France, in 2012. He is currently an Associate Professor with the Department of Information Engineering, Wuhan University of Technology. His research interests are cognitive radio networks and resource management.