

Tran Khanh Dang
Josef Küng
Tai M. Chung (Eds.)

Communications in Computer and Information Science

2708

Future Data and Security Engineering

12th International Conference, FDSE 2025
Ho Chi Minh City, Vietnam, November 27–29, 2025
Proceedings, Part I

Part 1

 Springer

FDSE 2025

CIDIX: Collaborative Intrusion Detection Through Human–AI Interaction
and Explainable Visual Analytics 126
Tran Tri Dang and Tam Van Nguyen

UABAC: A Flexible and Scalable ABAC Framework Unifying DAC,
MAC, and RBAC 142
Pham Thi Bach Hue and Minh-Triet Tran

Evaluating the Robustness of Graphs Against Link Prediction Attack
Threats 159
Tran Nguyen and Quang-Vinh Dang

Query Processing and Optimization

On Online Self-Organizing Linear Search 177
H. K. Dai, H. C. Su, and D. Ying

Improving RFID Network Planning Efficiency with Genetic Algorithm -
Hopfield Network Integrated Model 194
*Nguyen Van Tung, Nguyen Le Kim Thanh, Le Van Hoa,
and Vo Viet Minh Nhat*

A Physics-Informed Neural Network for Modeling Chloride Diffusion
in Reinforced Concrete Structure 208
Nguyen Dinh Nhat Minh, Bui Duc Vinh, and Tran Van Hoai

dLTQueue: A Non-blocking Distributed-Memory Multi-producer
Single-Consumer Queue 220
*Do Nguyen An Huy, Manh-Thin Nguyen, Thanh-Dang Diep,
and Nam Thoai*

An Improvement of Distinguished-Bit Tracking Knowledge-Based Query
Tree for RFID Tag Identification 235
*Phan-Nguyen-Bao Nguyen, Duc-Nhat-Quang Nguyen, Van-Hoa Le,
and Viet-Minh-Nhat Vo*

Data Analytics and Healthcare Systems

Task-Specific Ensemble Learning for Parkinson’s Disease Diagnosis
Using Acoustic Features 253
Sungwook Hur, Jieming Zhang, and Tai-Myoung Chung



An Improvement of Distinguished-Bit Tracking Knowledge-Based Query Tree for RFID Tag Identification

Phan-Nguyen-Bao Nguyen¹ , Duc-Nhat-Quang Nguyen¹ , Van-Hoa Le² ,
and Viet-Minh-Nhat Vo³

¹ University of Sciences, Hue University, Hue, Vietnam
ndnquang@hueuni.edu.vn

² School of Hospitality and Tourism, Hue University, Hue, Vietnam

³ Institute for Educational Testing and Quality Assurance - Hue University, Hue, Vietnam

Abstract. Collisions in tag identification have a detrimental effect on the performance of RFID systems. Repeated queries prolong the identification time, consume bandwidth, waste energy, and can also aggravate collisions. A range of approaches aimed at resolving tag collisions has been introduced to increase recognition performance, and the combination of a knowledge-based query tree and bit tracking has yielded impressive identification results. In the combination approach, the knowledge tree plays a vital role because it exploits the static knowledge of potential tag identifiers stored in the database, which has a decisive impact on identification efficiency. Building on this foundation, we propose an improved Distinguished-Bit Tracking Knowledge-based Query Tree (iDKQT) that, at each collision node, forms an additional set of distinguishable tags from the remaining tags. This addition increases the number of separable tags per split, shortens the knowledge tree, and enhances identification efficiency while preserving the worst-case complexity order of the original protocol. In simulations with up to 1,000 potential tags under identical conditions, the improved method reduces, on average, relative to the DKQT: collision slots by 15%, idle slots by 45%, and total slots by 21%; and relative to the BKQT: collision by 37%, idle by 94%, and total by 40%. It also shortens identification time by approximately 18%, with a one-time tree construction that is approximately twice as long in static-knowledge settings. These gains are supported by a mathematical analysis that shows the added set increases the number of separable tags per collision without altering the worst-case complexity order.

Keywords: RFID systems · knowledge-based query tree · anti-collision · bit tracking · distinguishing bit

1 Introduction

Radio Frequency Identification (RFID) is an identification technology that uses radio frequencies to wirelessly exchange data in two directions [1]. A basic RFID system comprises tags, readers, and a data management system (for example, a host). Tags can

be active, semi-active, or passive. Due to their small size and low cost, passive tags are the most commonly deployed. In operation, the reader sends a command to tags within its coverage area, and the tags reply with their data, such as their identifiers. The host processes and stores the received data [2].

Collisions are inevitable in an RFID system during data exchange between a reader and tags. There are two kinds of collisions: tag collision and reader collision. This paper considers the tag collision. Tag collision happens when multiple tags transmit simultaneously, preventing the reader from recognizing any of them.

Therefore, the reader must query multiple times before successfully identifying the tags. The algorithm to resolve collisions is called the anti-collision algorithm [3].

Anti-collision algorithms are generally classified into two primary categories: ALOHA-based [4–7] and tree-based [8–11]. The ALOHA-based algorithms are based on the random access principle, resulting in a high collision rate. Tree-based approaches are more computationally complex, but query performance improves significantly. Based on Manchester encoding, bit-tracking techniques [12–14] have significantly accelerated the identification speed of the tree-based anti-collision algorithm family. However, the additional complexity comes mainly from building the query tree. Some recent studies have incorporated prior knowledge of the tag database along with information on the quantity of tags present, a concept referred to as knowledge. The reader's prior classified knowledge into static and dynamic categories [16]. In the static case, the reader is aware of the universe of possible tag IDs; protocols that better leverage this database knowledge accelerate identification, particularly when the tag set varies significantly across sessions. In the dynamic case, the reader repeatedly interrogates a stable deployment and uses previous rounds to infer whether observed tags remain or have departed. This paper focuses on static-knowledge protocols. The models of combining a knowledge-based query tree and bit-tracking, such as Bit Tracking Knowledge-based Query Tree (BKQT) [15] and Distinguished-Bit Tracking Knowledge-based Query Tree (DKQT) [16], have also been introduced to improve tag identification performance.

The primary concept of the DKQT model is to generate the sets of distinguishable tags, called Dtags [16], at collision nodes while constructing the knowledge-based query tree. Specifically, at each collision node, tags distinguishable by a distinguishing bit are grouped into a set. The remaining tags, which are indistinguishable, are binary divided into two subsets, and the splitting process is repeated until each subset has only one readable tag or a Dtag. The constructed knowledge-based query tree, called d-tree [16], will allow simultaneous identification of multiple tags at each collision and thus increase the identification performance.

The efficiency of the DKQT model comes from the size of Dtags, where the larger the Dtags, the more compact the knowledge-based query tree is, and the higher the identification performance. Although DKQT has been verified to outperform other protocols significantly by simulation, it still needs to be verified by mathematical theory. Based on observation and analysis, it is possible to construct another Dtag, called Dtag+, from the remaining tags at a collision node after creating a Dtag. This paper exploits this finding and the improved DKQT called iDKQT.

The contributions of the paper include:

- proposing an improvement to the DKQT model by creating an additional set of distinguishable tags, Dtag +, at each collision node of the knowledge tree. This addition increases the size of Dtags at each collision node, shortening the knowledge tree and increasing the identification performance; and
- evaluating the performance of the improved DKQT based on mathematical analysis and simulation by comparing it with similar previous models, BKQT and DKQT.

This paper's structure is as follows: Sect. 2 analyzes and reviews related studies, focusing on models that combine query trees, knowledge, and bit-tracking. A detailed description of BKQT and DKQT is presented in Sect. 3, where an example of constructing an additional set of distinguishable tags is analyzed. A validation of the effectiveness of Dtags+ is also analyzed mathematically, and improved algorithms are also described. Section 4 explains the simulation, where experimental results are analyzed, compared, and evaluated. The conclusion is given in Sect. 5.

2 Related Works

The approaches for anti-collision tag identification can be summarized as follows.

The ALOHA family introduced the first anti-collision schemes: pure ALOHA (PA), slotted ALOHA (SA), framed slotted ALOHA (FSA), and dynamically framed slotted ALOHA (DFSA). In PA [4], tags retransmit after random backoff, which can still create partial overlaps and collisions. SA [5] divides time into slots; each tag randomly selects a slot and, after a collision, waits randomly before trying another slot until identified. FSA [6] groups slots into frames and limits each tag to one transmission per frame, improving performance over PA and SA but wasting slots when tags are few and causing heavy collisions when many, due to its fixed frame length. DFSA [7] counts idle, successful, and collision slots to estimate tag population and adapt the next frame size; its effectiveness depends on accurate estimation and frame-length adjustment.

In tree-based approaches, e.g., Binary Tree (BT) [8, 9] and Query Tree (QT) [10, 11], the operation of separating collision tags into subsets is repeatedly performed until there is only one tag left. In the BT approach, collisions are resolved by generating a binary-based random value, whereas QT distinguishes colliding tags through their unique identifiers. As a result, BT demands higher computational processing on the tag side than QT, while the distribution pattern of tag IDs influences the performance of QT.

Combining QT with other techniques, such as knowledge-based and bit tracking, is a recent approach to improve QT performance. Anti-collision models that incorporate knowledge include the Knowledge Query Tree (KQT) [17], Heuristic Query Tree (H-QT) [18], and Query Tree with Shortcutting and Couple Resolution (QTSC) [19]. With KQT, the number of tags is assumed to be known, and the range of tag identifiers is divided into segments with the expectation that each segment contains only one tag. The QT algorithm is subsequently invoked to identify the tags within each interval. H-QT constructs a decision tree of potential tags and continuously selects the most efficient bit for the following query by navigating the decision tree. QTSC builds a knowledge-based query tree from a given tag database. Based on the knowledge, QTSC also concatenates queries and creates shortcuts by skipping unnecessary queries.

Using Manchester encoding, the bit-tracking method allows the reader device to detect which bits experienced a collision within the collision slot. Collisions happen when two or more tags send different bits and cancel each other out. Different anti-collision algorithms have been introduced based on the position of the collision bit. Collision Tree Tracking Algorithm (CTTA) [12] divides a set of collision tags into two groups at the location of the first collision bit. Similarly, the Enhanced Anti-collision Algorithm (EAA) [13] divides the collision tags into two groups based on the position of the first collision bit. It adds a minimal feedback mechanism from the tag. Novel Enhanced Anti-collision Algorithm (NEAA) [14] improves EAA by separating the tags into groups of k bits “1”. It proposes a mechanism for identifying more than one tag in each collision slot.

The integration of knowledge and bit-tracking has also been proposed to enhance tag recognition efficiency. Bit-tracking Knowledge-based Query Tree (BKQT) [15] represents the earliest method combining knowledge with bit-tracking. In this approach, a knowledge tree (k-tree) is generated for all possible tags using data from the tag database. Bit-tracking is then employed to detect bit collision events, after which suitable actions are assigned to each node in order to query the tags present. Distinguished-Bit Tracking Knowledge-Based Query Tree (DKQT) [16] adopts a technique to identify the distinguishable tags with distinguishing bits. In its operation, DKQT constructs an n -ary tree where each node may contain a set of tags with distinct bits, allowing for the recognition of multiple tags involved in a collision during tree traversal. Compared with BKQT, DKQT achieves notable gains through fewer total slots and shorter tag recognition time.

The highlight of the DKQT model is to construct sets of distinguishable tags (Dtags) at collision nodes, where a distinguishable tag is uniquely identified based on the distinguishing bit (the bit location with a value of “0” or “1”). Obviously, the larger the Dtags size, the more tags can be identified simultaneously at each collision, and thus, the identification efficiency will be significantly improved. The following section describes how to increase the number of unique tags that can be distinguished.

3 Improvement of Distinguished-Bit Tracking Knowledge-Based Query Tree

3.1 Bit-Tracking Knowledge-Based Query Tree

BKQT in [15] is the earliest approach that merges knowledge with bit-tracking. It builds a knowledge tree, referred to as a k-tree, encompassing all possible tags, while also generating bit-collision scenarios and the corresponding actions for each collision node through bit-tracking. During execution, BKQT pinpoints the locations of collision bits to determine the possible tags. Its process is organized into four main stages: K-Tree Construction, Bit Collision Cases, Action Preparation, and Identification Operation. In the K-Tree Construction stage, a k-tree is created using prior information about all potential tags. The Bit Collision Cases phase identifies every possible scenario of bit collisions related to the collision nodes within the k-tree. The Action Preparation phase determines suitable operations corresponding to these identified collision scenarios. Finally, the Identification Operation phase implements the chosen operations for each relevant

collision scenario. BKQT has outperformed KQT, H-QT, and QTSC in terms of total slots and identification time, as demonstrated by experiments. However, exhaustively searching for bit collisions increases the computational complexity of BKQT. Furthermore, BKQT does not use information about the action corresponding to a collision to enumerate the following possible bit collisions. These drawbacks have affected the identification efficiency of BKQT.

3.2 Distinguished-Bit Tracking Knowledge-Based Query Tree

Within a static-knowledge category - where the entire potential tag set is known a priori and remains fixed - DKQT in [16] improves BKQT by introducing the concept of distinguishing bits and constructing a distinguishable tree (called d-tree), in which a collision node can contain a set of distinguishable tags (Dtags), which allows the identification of multiple tags simultaneously at a collision. In its operation, DKQT constructs an n-ary structure containing possible tags, allowing the formation of sets of distinguishable ones. During the recognition process, DKQT navigates the d-tree and rapidly detects collision tags through the distinguished-bit tracking technique. Since DKQT can detect the distinguishing bits of the tags in a collision slot, it will be able to identify them. Through experiments, DKQT has outperformed BKQT in terms of total slots and identification time. However, DKQT can be further improved by exploiting distinguishable tags in the remaining tags. Specifically, after creating Dtags as in [16], other distinguishable tags may exist from the remaining tags that can construct a new Dtags, Dtags+. Increasing the number of distinguishable tags will enhance identification efficiency, as illustrated in the following section.

3.3 From an Visual Example to Innovative Ideas

To illustrate the procedure, we construct a d-tree for a set of 13 potential tags with the following IDs: 0000101, 0001000, 0001100, 0001101, 0010111, 0101110, 1010110, 1011010, 1011110, 1101110, 1111000, 1111011, and 1111110. According to the d-tree construction algorithm described in [16] and implemented following the operation sequence in Table 1, these tags can be organized into an n-ary tree, as illustrated in Fig. 1. In the initial phase of construction, distinguishable tags that share the same prefix, "0", are identified. In this case, the tags 0001000, 0010111 and 0101110 share this prefix. They can be differentiated by examining the bits at positions 5, 3, and 2 respectively, which are therefore referred to as distinguishing bits. After the first separation step, we get a set of distinguishable tags (Dtag) as in [16]. The process is then repeated until the final step (Step 5, with bit sequence '111110'), as shown in Fig. 1 and Table 1, resulting in a complete query tree.

However, some tags still collide at certain nodes in the tree. In other words, the base criterion leaves several tags in the same equivalence class, so they are not uniquely identified. To address these cases, we examine additional bit positions beyond those initially used. Many of the remaining tags can be separated this way. For example, the tags $000\bar{0}101$ and $000110\bar{0}$ - which were not separated in the first pass - can be distinguished by examining the bit at position 4 for the former and position 7 for the latter. These two tags therefore form a supplementary set of distinguishable tags, which we

denote Dtag+. As summarized in Table 2, iteratively resolving collisions across multiple nodes yields a progressively refined structure: collision counts fall, average depth can decrease, and the share of uniquely identifiable tags rises. Figure 2 illustrates the final n-ary tree, where legacy branches capture decisions from the original procedure (forming Dtag) and supplementary branches reflect the additional splits (forming Dtag+). This combination enhances tag identification efficiency, improving both lookup speed and accuracy under complex collision patterns.

Table 1. Steps to construct the d-tree of DKQT with 13 given tags. Underlines mark the distinguishing bits.

Step	LCP	Dtags	Readable tags	Remaining potential tags (input for next step)
0	$\emptyset$	$\emptyset$		0000101, 0001 <u>0</u> 00, 0001100, 0001101, 0010 <u>1</u> 11, 0101110, 1010110, 1011010, 1011110, 1101110, 1111000, 1111011, 1111110
1	0	0001 <u>0</u> 00, 0010 <u>1</u> 11, 0101110		0000101, 000110 <u>0</u> , 0001101, 1010110, 1011010, 1011110, 1101110, 1111000, 1111011, 1111110
2	000	0000 <u>1</u> 01, 000110 <u>0</u>	0001101	1010110, 1011010, 1011110, 1101110, 1111000, 111101 <u>1</u> , 1111110
3	1	1010110, 1101110, 1111000, 1111011		1011010, 1011110, 1111110
4	1011	1011010, 1011110		1111110
5	11		1111110	

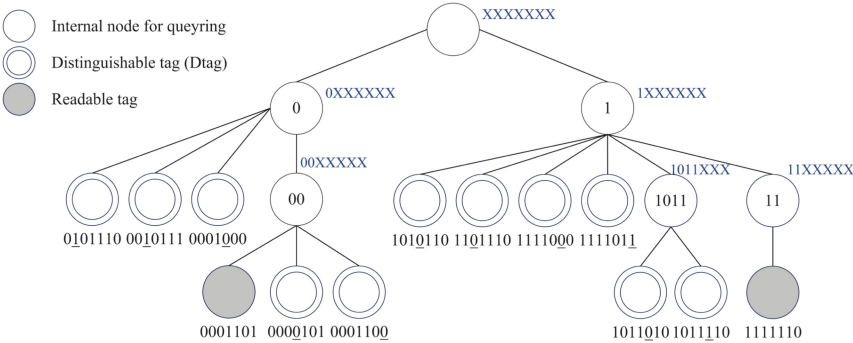


Fig. 1. The d-tree constructed by DKQT with 13 given tags.

Table 2. Steps to construct the d-tree of iDKQT with 13 given tags. Overlines mark the distinguishing bits for creating Dtag+.

Step	LCP	Dtags	Dtags+	Readable tags	Remaining potential tags (input for next step)
0	$\emptyset$	$\emptyset$	$\emptyset$		000 $\overline{0}$ 101, 0001 $\overline{000}$, 0001100, 000110 $\overline{1}$, 00 $\overline{10}$ 111, 0 $\overline{10}$ 1110, 10 $\overline{10}$ 110, $\overline{10}$ 11010, 1011110, 1101110, 1111000, 1111011, 1111110
1	0	0001 $\overline{000}$, 00 $\overline{10}$ 111, 0 $\overline{10}$ 1110	000 $\overline{0}$ 101, 000110 $\overline{0}$,	0001101	10 $\overline{10}$ 110, 1011 $\overline{0}$ 10, 10 $\overline{11}$ 110, 11 $\overline{01}$ 110, 1111 $\overline{00}$ 0, 11110 $\overline{1}$ 1, 1 $\overline{1}$ 11110
2	1	10 $\overline{10}$ 110, 11 $\overline{01}$ 110, 1111 $\overline{00}$ 0, 11 $\overline{11}$ 011	1011 $\overline{0}$ 10, 1 $\overline{1}$ 11110	1011110	

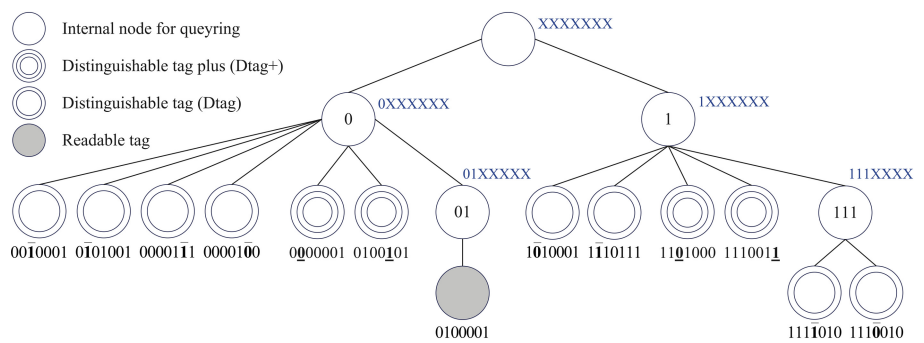


Fig. 2. The d-tree constructed by iDKQT with 13 given tags.

3.4 Mathematical Analysis

Consider a set of n tags with an ID length of d . If the set of n tags has all IDs, that is $n = 2^d$, the d-tree is a complete binary tree with no Dtags at each collision node. The number of collision nodes (n_c) equals $n - 1$. If at each collision node, there exists a Dtags with an average size of s_d ($s_d > 0$), the number of distinguishable tags at collision nodes is equal to $n_c \times s_d$, and there may be readable tags at the leaf position of d-tree. This means that $n > n_c \times s_d$. This equation suggests that for a given set of n tags, the more distinguishable tags identified at the collision nodes, the fewer the collision nodes of the d-tree will be. As a result, the identification will be faster. This paper proposes constructing an additional set of distinguishable tags, Dtags+, at the collision nodes,

which will undoubtedly lower collision occurrences, leading to shorter identification time.

3.5 Tree Construction and Identification Algorithms

With the proposed approach, the tree construction algorithm of iDKQT represents an enhancement of the original DKQT method described in [16]. The improvement is achieved by introducing additional pseudocode lines (Fig. 3), specifically lines 10 to 12, as shown Fig. 3. These modifications refine the tree-building process while preserving the overall algorithmic structure. Similarly, the identification algorithm in iDKQT (Fig. 4) is adapted from that of DKQT, incorporating additional pseudocode lines from lines 11 to 13, as illustrated in Fig. 4. These extensions are designed to improve operational efficiency and address specific limitations observed in the original method. Notably, the inclusion of these pseudocode segments does not increase the computational complexity of either the tree construction or the identification algorithms. However, the refined tree construction process in iDKQT does require slightly more execution time compared to DKQT, due to the additional processing steps involved in implementing the improvements.

Input : $G, G = \{ID_1, ID_2, \dots, ID_i\}$ is the group of potential tags where ID_i is the i -th ID in G

Output: node (root of knowledge tree)

```

1  function Construction(G)
2      if |G| == 0 then node = NULL
3      else if |G| == 1 then node = NewNode(), node.s = ID1
4      else
5          node = NewNode()
6          node.s = LongestCommonPrefix(G)
7          Dtags = Distinguish(G)
8          InsertD(node, Dtags)
9          G = G - Dtags
10         Dtags += Distinguish(G)
11         InsertD(node, Dtags+)
12         G = G - Dtags+
13         separate G into two subgroups G' and G'' according to IDi[|node.s|+1]
14         nodeL = Construction(G')
15         nodeR = Construction(G'')
16     end if
17 end if
18 return node

```

Fig. 3. iDKQT's knowledge tree construction algorithm.

3.6 Time Complexity

QTSC and BKQT [15, 19] already show they have better performance than other protocols belonging to the static-knowledge category. Because BKQT performs better than QTSC, herein we only discuss BKQT, DKQT, and its improved variant iDKQT in time complexity. BKQT visits each node in the k -tree during the Bit-collision Cases phase to prepare the bit-collision cases in each node. For a node with n possible tags, BKQT lists all combinations of these tag IDs and creates the bit-collision cases. Thus, the time complexity of creating the bit-collision cases for a node is $O(2^n)$ [15]. Since there are

n_c collision nodes in the k-tree, the time complexity of BKQT is $O(n_c \times 2^n)$, resulting in an exponential-time solution.

Transmit the command starting a frame

```

1   Initialize node to be the root of Tree
2   Push(Q, node)
3   while Q  $\neq \emptyset$  do
4       node = Pop(Q)
5       q = node.s
6       transmit q and receive the response r
7       if r is collision then
8           if hasDtag(r) then
9               | Get Dtags from DB according to q and each distinguished-bit's status in r
10          else if hasDtag+(r) then
11              | Get Dtags+ from DB according to q and each distinguished-bit's status in r
12          end if
13          if nodeL  $\neq \emptyset$  then Push(Q, nodeL)
14          if nodeR  $\neq \emptyset$  then Push(Q, nodeR)
15      else if r is readable then
16          | identify the tag
17      else if r is empty then
18          | do nothing
19      end if
20  end while

```

Fig. 4. iDKQT's tag identification algorithm.

For a node with n possible tags, our proposed protocol DKQT needs $O(b \times n)$ time to check each ID's suffix bits for the set of potential tag IDs, where b is the suffix length, i.e., $b = d - |\text{node.s}|$. Because $b \leq d$, we have $O(b \times n) \leq O(d \times n)$. Since there are n_c collision nodes in the d-tree, the time complexity of DKQT is $O(n_c \times d \times n) \leq O(n^2 \times d)$, resulting in a polynomial-time solution.

iDKQT follows the same procedure as DKQT, but when a collision occurs, it additionally exploits the richer Dtag+ information to prune the potential tag set further. Consequently, at a node, the work becomes $O(b^+ \times n^+)$ with $n^+ \leq n$ and $b^+ \leq b \leq d$. With n_c collision nodes in the d-tree, the overall time complexity of iDKQT is $O(n_c \times b^+ \times n^+) \leq O(n_c \times d \times n) \leq O(n^2 \times d)$. Hence, iDKQT has the same worst-case order as DKQT but is typically faster in practice due to fewer remaining tags and fewer visited nodes.

4 Simulation and Result Analysis

The simulation is programmed in Python 3.11.4 and executed on a PC equipped with a 12th-generation Intel Core i7 processor running at 2.10 GHz and 24 GB of RAM. We consider a set of 1000 potential tags with their IDs evenly distributed and their ID length of 48 bits. After the identification begins, the set of potential tags is unchangeable. That is, no other unknown tags enter the range of the reader. From the set of 1000 potential tags, 100, 200, ..., 900, and 1000 are randomly extracted to deploy as tags present in the operating range of the reader. Each appearing tag must exist in the potential tag set. We evaluate the performance of the proposed iDKQT against BKQT [15] and DKQT [16] under identical RFID conditions to ensure a fair comparison.

The metrics to evaluate the performance of models are:

Total slots: the overall slots necessary for recognizing every tag located in the reader's operating range.

Collisions (S_C): the overall slots in which multiple tags respond simultaneously and collisions occur. There are two cases: no tags are identified, called pure collision, and tags are identified using Dtags, called collision with multi-readables;

Idles (S_I): the number of slots where no tag responds;

Readables (S_R): the number of slots where only one tag replies;

Performance: the proportion of tags successfully detected relative to the total slots required;

Construction time (s): the duration needed to construct the knowledge tree;

Identification time (s): As proposed in [15] and [16], this value is the duration needed to identify all the appearing tags. Suppose b_R , b_C and b_I are the bit lengths of the readable, idle and collision slots, respectively. Since the idle slot is usually much shorter than the collision and readable slots, the length of b_I can be set to 3 bits, while b_R and b_C matching the 48-bit length of the tag ID. With transmission bandwidth 640 Kbps, the identification time is measured as

$$T_I = \frac{(S_R \times b_R) + (S_I \times b_I) + (S_C \times b_C)}{640}. \quad (1)$$

4.1 Total Slots, Collisions, Idles, and Readables

Regarding the total slots required to read all appearing tags, iDKQT achieves the fewest slots compared to those of DKQT and BKQT, as shown in Fig. 5. When the number of appearing tags is less than 500, the total slots increase rapidly as the number of appearing tags increases. However, from 500 tags onwards, the total slots stopped increasing and became saturated. When the number of appearing tags is small, few tags can be identified simultaneously using Dtags, so more slots are required to identify them all. However, when the number of appearing tags increases and approaches the number of potential tags, multiple appearing tags can be identified using Dtags. Thus, the total slots required become stable.

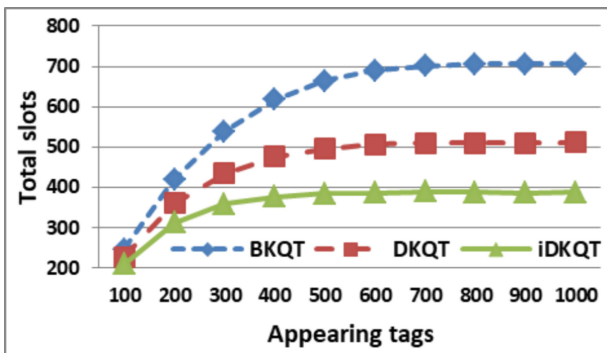


Fig. 5. A comparison of total slots between BKQT, DKQT and iDKQT.

The reduction in Total slots is due to the reduction in Collisions. As shown in Fig. 6, the Collisions of iDKQT and DKQT are significantly lower than those of BKQT. The decrease in the Collisions of iDKQT and DKQT is because multiple tags can be identified simultaneously using Dtags, reducing the number of reply tags and thus reducing collisions. With iDKQT, adding Dtags+ further reduces the number of reply tags, so iDKQT is better than DKQT in Collisions.

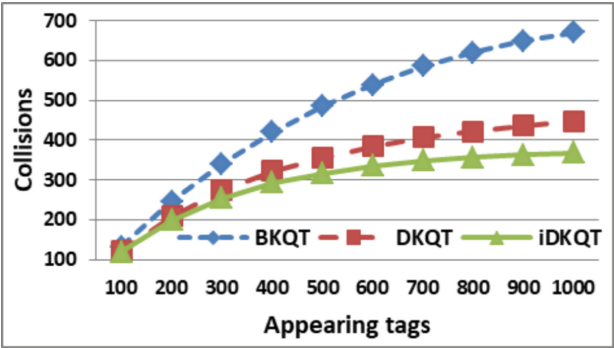


Fig. 6. A comparison of collisions between BKQT, DKQT and iDKQT.

Adding Dtags also reduces the number of idle slots (Idles). As shown in Fig. 7, the Idles of iDKQT and DKQT are significantly lower than those of BKQT. The ability to identify multiple tags at a collision slot reduces the number of remaining tags, so the Idles also decrease. With iDKQT and DKQT, the Idles initially increase slightly because the tags that appear are sparse in the space of potential tags, so there may be no tags to answer some queries. However, as the number of tags that appear increases, the Idles decrease rapidly, and this value approaches zero as the number of tags that appear approaches the number of potential tags.

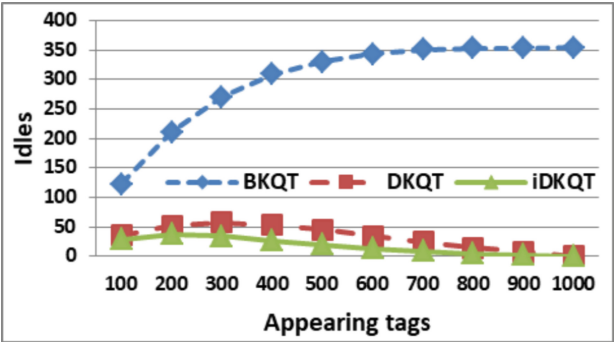


Fig. 7. A comparison of idles between BKQT, DKQT and iDKQT.

The addition of Dtags significantly reduces the readables of iDKQT and DKQT compared to that of BKQT, as shown in Fig. 8. This is reasonable because increasing the number of distinguishable tags in Dtags reduces the readable tags. The reduction in Readables contributes to the reduction in Total slots, as mentioned above, and increases the performance of tag identification, which will be analyzed in the following subsection.

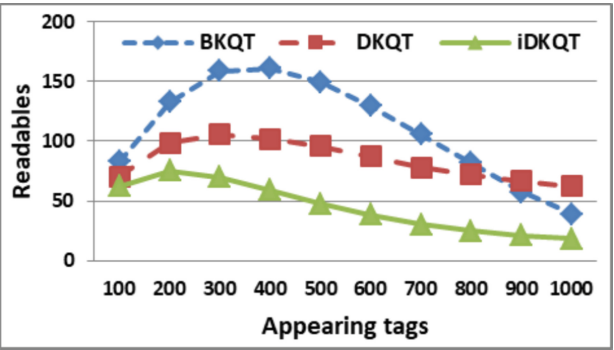


Fig. 8. A comparison of readables between BKQT, DKQT and iDKQT.

4.2 Performance

The reduction in total slots has a positive impact on the performance. As shown in Fig. 9, the difference in the performance of the models is insignificant when the number of appearing tags is low. However, it becomes clear that the number of appearing tags increases and approaches the potential tags. This illustrates the impact of adding Dtags+, which makes iDKQT the most effective.

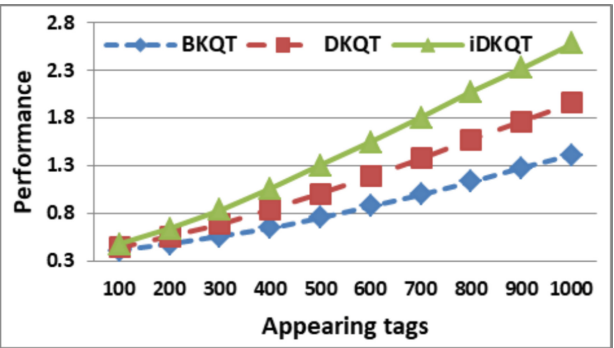


Fig. 9. A comparison of performance between BKQT, DKQT and iDKQT.

4.3 Construction and Identification Times

Although it has better performance, iDKQT suffers from an extensive knowledge tree construction time. As shown in Fig. 10, the construction time of IDKQT is significantly

higher than that of BKQT and DKQT. This is understandable because adding Dtags+ requires iDKQT to search and construct a new set of distinguishable tags at each collision node, which increases the time complexity. The construction time of iDKQT is twice that of DKQT.

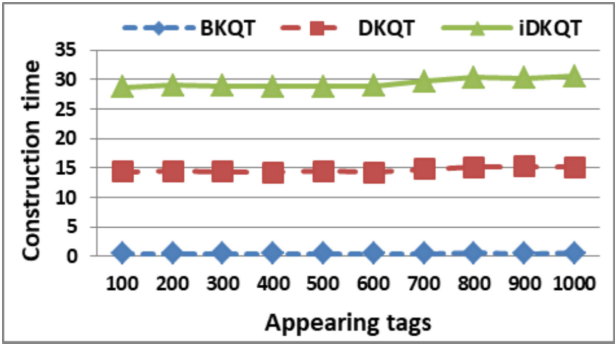


Fig. 10. A comparison in construction time between BKQT, DKQT and iDKQT.

However, the loss in construction time is compensated for by the reduction in identification time. As shown in Fig. 11, the identification time of iDKQT is diminished in comparison to DKQT. Although the average reduction in identification time is only about 18%, the performance of iDKQT in terms of time is significant compared to DKQT. The reason is that the construction of the knowledge tree from the database of potential tags is only done once initially. Then, the system can identify tags as many times as possible.

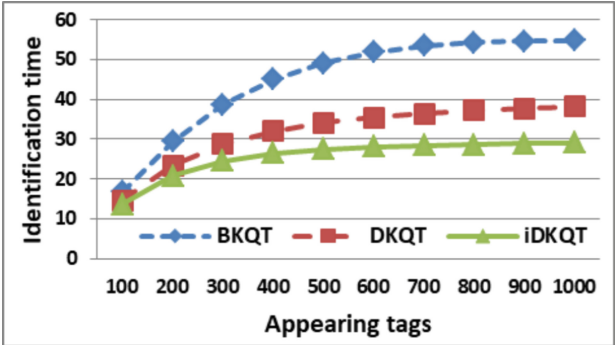


Fig. 11. A comparison of the identification time between BKQT, DKQT and iDKQT.

5 Conclusion

Collisions in RFID tag identification are inevitable and degrade performance by increasing both collision and idle slots. This paper presented iDKQT, a tree-based identification protocol that constructs a knowledge tree from a database of potential tag identifiers and then identifies tags against that prior knowledge. The principal contribution

is the introduction of an additional set of distinguishable tags, Dtag+, formed from the remaining tags at each collision node. This addition complements the original Dtag in DKQT, increases the number of separable tags per split, shortens the knowledge tree, and improves identification efficiency.

Across simulated scenarios with up to 1,000 potential tags under identical RFID conditions, iDKQT achieves substantial average improvements. Relative to DKQT, collision slots are reduced by 15%, idle slots by 45%, and total slots by 21%. Relative to BKQT, the corresponding average reductions are 37%, 94%, and 40%, respectively. In execution time, iDKQT shortens tag-identification time by approximately 18% compared with DKQT, while the knowledge-tree construction time is approximately twice as long. Because the paper focuses on static knowledge, the tree is built once at initialization and then reused for many identification rounds, making this one-time construction cost acceptable for large-scale deployments.

These empirical gains are verified by mathematical analysis: incorporating Dtag+ strictly increases the number of tags distinguishable at collision nodes. It reduces the expected tree depth under the same prior over tag identifiers, while preserving the same asymptotic worst-case complexity order as DKQT. This analytical guarantee explains - and supports - the observed reductions in collision, idle, and total slots.

Acknowledgments. Duc-Nhat-Quang Nguyen was funded by the Master, PhD Scholarship Programme of Vingroup Innovation Foundation (VINIF), code VINIF.2024.TS.072. The research was conducted within the framework of a Hue University scientific research project with code DHH2024-01-220.

References

1. Elbasani, E., Siriporn, P., Choi, J.S.: A Survey on RFID in Industry 4.0, , Ch. 1, pp. 1–16. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-32530-5_1
2. Zhang, J., Periaswamy, S.C., Mao, S., Patton, J.: Standards for passive UHF RFID. *GetMobile: Mobile Comput. Commun.* **23**(3), 10–15 (2020). <https://doi.org/10.1145/3379092.3379098>
3. Wang, L., Luo, Z., Guo, R., Li, Y.: A review of tags anti-collision identification methods used in RFID technology. *Electronics* **12**(17), 3644 (2023). <https://doi.org/10.3390/electronics12173644>
4. Burdet, L.A.: RFID Multiple Access Methods, seminar “Smart Environments”, ETH Zürich, Summer Semester 2004 (6 2004)
5. Klair, D., Chin, K.-W., Raad, R.: An investigation into the energy efficiency of pure and slotted aloha-based RFID anti-collision protocols. In: 2007 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, pp. 1–4 (2007). <https://doi.org/10.1109/WOWMOM.2007.4351749>
6. Vogt, H.: Efficient Object Identification with Passive RFID Tags, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Preface 2414, pp. 98–113 (2002). https://doi.org/10.1007/3-540-45866-2_9
7. Cha, J., Kim, J.-H.: Dynamic framed slotted ALOHA algorithms using fast tag estimation method for RFID system. In: 2006 3rd IEEE Consumer Communications and Networking Conference, vol. 2 (C), pp. 768–772 (2006). <https://doi.org/10.1109/CCNC.2006.1593143>

8. Myung, J., Lee, W.: Adaptive splitting protocols for RFID tag collision arbitration. In: International Symposium on Mobile Ad Hoc Networking and Computing, pp. 202–213 (2006). <https://doi.org/10.1145/1132905.1132928>
9. Lai, Y.-C., Hsiao, L.-Y.: Adaptive frame on binary splitting for radio frequency identification anticollision. *Int. J. Commun Syst* **27**(10), 1672–1685 (2014). <https://doi.org/10.1002/dac.2427>
10. Law, C., Lee, K., Siu, K.-Y.: Efficient memoryless protocol for tag identification (extended abstract). In: Proceedings of the 4th International Workshop on Discrete Algorithms, Methods of Mobile Computing, and Communication, pp. 75–84. ACM, New York, NY, USA (2000). <https://doi.org/10.1145/345848.345865>
11. Yang, C.-N., Hu, L.-J., Lai, J.-B.: Query tree algorithm for RFID tag with binary-coded decimal EPC. *IEEE Commun. Lett.* **16**(10), 1616–1619 (2012)
12. Zhou, F., Jin, D., Huang, C., Hao, M.: Optimize the power consumption of passive electronic tags for anti-collision schemes. In: IEEE International Symposium on Semi-conductor Manufacturing Conference Proceedings, vol. 2, pp. 1213–1217 (2003)
13. Chen, W.-C., Horng, S.-J., Fan, P.: An enhanced anti-collision algorithm in RFID based on counter and stack. In: 2007 Second International Conference on Systems, Networks, and Communications (ICSNC 2007), p. 21. IEEE (2007). <https://doi.org/10.1109/ICSNC.2007.16>
14. Chen, Y.-H., Horng, S.-J., Run, R.-S., Lai, J.-L., Chen, R.-J., Chen, W.-C., et al.: A novel anticollision algorithm in RFID systems for identifying passive tags. *IEEE Trans. Industr. Inf.* **6**(1), 105–121 (2010). <https://doi.org/10.1109/TII.2009.2033050>
15. Lai, Y.-C., Chen, S.-Y., Hailemariam, Z.L., Lin, C.-C.: A bit-tracking knowledge-based query tree for RFID tag identification in IoT systems. *Sensors* **22**(9), 3323 (2022). <https://doi.org/10.3390/s22093323>
16. Lin, C.-C., Lai, Y.-C., Hailemariam, Z.L.: A distinguished-bit tracking knowledge-based query tree for RFID tag identification. *Comput. Commun.* **218**(January), 166–175 (2024). <https://doi.org/10.1016/j.comcom.2024.01.017>
17. Bonuccelli, M.A., Lonetti, F., Martelli, F.: Exploiting ID knowledge for tag identification in RFID networks. In: Proceedings of the 4th ACM Workshop on Performance Evaluation in Wireless Ad Hoc, Sensor, and Ubiquitous Networks, pp. 70–77. ACM, New York, NY, USA (2007). <https://doi.org/10.1145/1298197.1298210>
18. Sung, J., Kim, D., Kim, T., Choi, J.: heuristic query tree protocol: use of known tags for RFID tag anti-collision. *IEICE Trans. Commun.* E95-B (2), 603–606 (2012). <https://doi.org/10.1587/transcom.E95.B.603>
19. Hailemariam, Z.L., Lai, Y.-C., Jayadi, R., Chen, Y.-H., Huang, S.-C.: A knowledge-based query tree with shortcutting and couple-resolution for RFID tag identification. *Comput. Commun.* **160**(January), 779–789 (2020). <https://doi.org/10.1016/j.comcom.2020.06.025>