

Joined Q-ary Tree Anti-Collision for Massive Tag Movement Distribution

Prapassara Pupunwiwat

Bela Stantic

Institute for Integrated and Intelligent Systems
Griffith University, Queensland, Australia
Email: {p.pupunwiwat, b.stantic}@griffith.edu.au

Abstract

Radio-Frequency Identification (RFID) systems consist of tags and networked electromagnetic readers. Despite the emergence of RFID technology, the problem of identifying multiple tags, due to the *Collisions* is still a major problem. The problem can be solved by using anti-collision methods such as *ALOHA-based* approaches and *Tree-based* approaches. ALOHA-based approaches suffer from tag starvation, which causes that not all tags can be identified. The tree-based approaches suffer from too long identification delay caused by lengthy queries during identification process. In this paper, we propose a tree-based anti-collision method called “Joined Q-ary Tree”, which adaptively adjusts tree branches according to tag movement behavior and number of tags within an interrogation zone. In this empirical study, we demonstrate that the proposed method is suitable for numerous scenarios. It requires less queries issued per complete identification than existing approaches while ensuring identification of all tags within the interrogation zone.

Keywords: Radio Frequency Identification - RFID, Anti-Collision

1 Introduction

RFID technology uses radio-frequency waves to automatically identify people or objects. Currently, RFID technology is used in different systems such as: transportation, distribution, retail and consumer packaging, security and access control, monitoring and sensing, library system, defence and military, health care, and baggage and passenger tracing at the airports. RFID reader retrieves information from tags and sends that information back to host computer via middleware. RFID data, which is captured by readers, can be accumulated very fast and does not carry much information as it is raw. These data are inaccurate and they need to be filtered in order to improve database management.

The main issue that usually occurs in RFID data streams is data errors. The most common errors are Missed reads, which are caused by collisions where Radio Frequency (RF) signals interfere with each other, preventing the reader from identifying tags. RF collision problem can be solved by using anti-collision techniques to prevent two or

more tags from responding to a reader at the same time. Tag anti-collision algorithms can be categorised into ALOHA-based algorithms and Tree-based algorithms. ALOHA-based algorithms suffer from Tag starvation problems such that not all tags can be identified. On the other hand, Tree-based algorithms make trees while performing the tag identification procedure using a unique ID of each tag, which lead to lengthy queries issued by reader that causes long identification delay.

In this paper, we present a “Joined Q-ary Tree”, which is based on Memoryless Q-ary Tree. We performed extensive experimental study in order to prove the efficiency of the proposed technique. The results and analysis of the experiments indicate that “Joined Q-ary Tree” can more effectively reduce queries length and improve the system efficiency than the Naive Q-ary Trees.

The remainder of this paper is organised as follows: In Section 2, some general background are provided on RFID and information related to Tag collision and Massive Tag Movement. In Section 3, we discuss related works on our proposed method and their limitations, which include ALOHA-based and Tree-based anti-collision, and Query tree protocols. In Section 4, we present a new technique, the Joined Q-ary Tree including methodology and scenarios. In Section 5, we present experimental evaluation, results, analysis and discussions; and finally in Section 6 we provide our conclusion and future work.

2 RFID Background

RFID technology is an automatic identification technology of contactless method that identifies electronic tags attached to items. There are several methods of identification but the most common is to store a serial number that identifies a person or object such as Electronic Product Code (EPC). RFID may only consist of a tag and a reader but a complete RFID system involves many other technologies, such as computer, network, Internet, and software such as middleware and user applications.

2.1 Electronic Product Code (EPC)

EPC Class 1 or passive EPC tag is widely used in Ultra High Frequency (UHF) range for communications at 860-960MHz, where the standards have been created by EPCGlobal (EPCGlobal 2006). The most common encoding scheme currently widely used includes: General Identifier (GID-96), Serialised Global Trade Item Number (SGTIN-96), Serialised Shipping Container Code (SSCC-96), Serialised Global Location Number (SGLN-96), Global Returnable Asset Identifier (GRAI-96), Global Individual Asset Identifier (GIAI-96), and DoD Identifier (DoD-96).

In this paper, we only focus on **General Identifier 96-bits** type to evaluate our approach. The implementation and experiment will be determined by the impact of *EPC Pattern* and *Massive tag movement*. Therefore at present, only one type of encoding is necessary.

| GID-96 | Bit | Max.Decimal/Binary |
|----------------------|-----|--------------------|
| Header | 8 | 0011 0101 |
| GMN* | 28 | 268,435,455 |
| Object Class | 24 | 16,777,215 |
| Serial Number | 36 | 68,719,476,735 |

Table 1: The General Identifier (GID-96) includes three fields in addition to the *Header* with a total of 96-bits binary value (*GMN = General Manager Number).

The general structure of EPC tag encodings is a string of bits, consisting of a fixed length (8-bits) *Header* followed by a series of numeric fields whose overall length, structure, and function is completely determined by the *Header* value. Table 1 shows an example of GID-96 EPC passive tag encoding scheme. Only *Header* is shown in binary; the rest are shown in decimal number.

2.2 Massive RFID Tags Movement

Items tend to move and stay together through different locations especially in a large warehouse (Gonzalez, Han & Li 2006), (Gonzalez, Han, Li & Klabjan 2006). For example, 4 pallets - with 24 cases of crystal wine glasses, plates, bowls, and jugs - each may be ready to leave the warehouse and deploy to different retailer. At this point, 4 pallets move along a conveyor belt through dock doors mounted with RFID readers. We can, for example, use the assumption that many RFID objects stay or move together, especially at the early stage of distribution. These EPC data will be very similar since the first few bits of encoding will determine the type of *Encoding Scheme* (Header), *Company Prefixes* (GMN) and *Object Class*.

Currently, for example in warehouse distribution environment where RFID systems are deployed, the UHF range of RF waves and passive tags are used for long distance identification. In order to manage and monitor the traffic of RFID data effectively, *EPC pattern* is usually used to keep unique ID on each item arranged within a specific range. *EPC pattern* does not represent a single tag encoding, but rather refers to a set of tag encodings. For example, there are 20 cases of wine-glasses packed together in the same pallet. *EPC pattern* ranging between 1 to 20 may be used so that a foreign tag, e.g. an EPC tag with ID 47, can be filtered out easily if accidentally captured by the reader.

2.3 RFID Data Stream Errors

Due to the low-power and low-cost constraints of RFID tags, reliability of RFID readings is of concern in many circumstances (Brusey et al. 2003), (Vogt 2002). There are four typical errors: *Unreliable reads*, *Noise*, *Missed reads*, and *Duplication*. Several techniques for filtering RFID data have been proposed in literatures (Bai et al. 2006), (Jeffery et al. 2005), (Jeffery, Garofalakis & Franklin 2006), (Jeffery, Alonso, Franklin, Hong & Widom 2006), (Carbunar et al. 2005), (Fishkin et al. 2004). However, these techniques only filter specific kind of errors. A research

on *Noise* and *Duplication* data filtering has been done very well previously; nonetheless, the Unreliable reads can be prevented only at some point. This depends on the deployment of readers, tags, and an environment.

Missed Reads are very common in RFID applications, which are caused by collisions of RF signals that interfere with each other preventing the reader from identifying tags. RF collisions can be solved by performing anti-collision at the edge to prevent two or more tags from responding to a reader at the same time. RF collision can happen at two levels: Collision at reader level and Collision at tag level.

2.4 Tag and Reader Collision

Simultaneous transmissions in RFID systems lead to collisions as the readers and tags typically operate on the same channel. Three types of collisions are possible: Reader-Reader collision, Reader-Tag collision, and Tag-Tag collision.

- **Reader-to-Reader:** Interference occurs when one reader transmits a signal that interferes with the operation of another reader; and prevents the second reader from communicating with tags in its interrogation zone (Jain & Das 2006). Reader-to-Reader collision can be easily avoided by determining the appropriate reader's deployment that prevents direct signal interference between two or more readers.
- **Reader-to-Tag:** Interference occurs when one tag is simultaneously located in the interrogation zone of two or more readers, where more than one reader attempts to communicate with that tag at the same time (Jain & Das 2006).
- **Tag-to-Tag:** Tag collision in RFID systems, also known as Multi-Access, happens when multiple tags are energised by the RFID tag reader simultaneously, and reflect their respective signals back to the reader at the same time. This problem is often seen whenever a large volume of tags must be read together in the same reader zone because the reader is unable to differentiate these signals.

3 Related Works

Tag collision or Multi-Access problem is more complex than those within Reader collision categories. Therefore, in this paper we only focus on tag anti-collision approaches.

3.1 Tag Anti-Collision Approaches

The various types of tag anti-collision methods for Multi-access/Tag collision can be reduced to two basic types: ALOHA-based method and Tree-based method. In an ALOHA-based method, tags respond at randomly generated times. If a collision occurs, colliding tags will have to identify themselves again after waiting a random period of time. This technique is faster than Tree-based but suffers from Tag starvation problem where not all tags can be identified due to the random nature of chosen time.

The Tree-based method starts by asking for the first number of the tag (Query Tree algorithm) until it matches the tags; then it continues to ask for additional characters until all tags within the region are found. This method is slow and introduces a long Identification delay but leads to fewer collisions, and have 100 percent successful identification rate.

3.1.1 ALOHA-Based Methods

The ALOHA-based methods usually refers to “Slot ALOHA” (Quan et al. 2006), which introduces discrete time-slots for tags to be identified by reader at the specific time. The principle of Slotted-ALOHA techniques is based on the “Pure ALOHA” introduced in early 1970s (Abramson 1970), where each tag is identified randomly. To improve the performance and throughput rate, a “Frame Slotted ALOHA” (Shin et al. 2007) anti-collision algorithm has been proposed, where each frame is formed of specific number of slots that is used for communication between the readers and the tags. Each tag in the interrogation zone arbitrarily selects a slot for transmitting the tag’s information. Several researchers (Wang et al. 2007), (Lee et al. 2005), (Lee et al. 2008), (Cho et al. 2007) have attempted to improve the throughput rates by implementing a Frame Estimation Tool. However, the ALOHA-based method can only be improved to a very high throughput rate but they still cannot achieve a hundred percent tag identification.

3.1.2 Tree-Based Methods

Such Tree-based methods can be classified into a Memory-based algorithm and a Memoryless-based algorithm. In the Memory-based algorithm, which can be grouped into a splitting tree algorithm such as an “Adaptive Splitting Algorithm” and a “Bit Arbitration Algorithm,” the reader’s inquiries and the responses of the tags are stored and managed in the tag memory, resulting in an equipment cost increase especially for RFID tags. In contrast for the Memoryless-based algorithm, the responses of the tags are not determined by the reader’s previous inquiries. The tags’ responses and the reader’s present inquiries are determined only by the present reader’s inquiries so that the cost for the tags can be minimised. Memoryless-based algorithms include a “Query Tree Algorithm”, a “Collision Tracking Tree Algorithm”, and a “Tree Walking Algorithm.”

In this paper, we will focus on Memoryless Query Tree based protocols since it is the most popular and is effective anti-collision technique for passive UHF tags. However, there are other improved anti-collision methods based on Query Tree such as an “Adaptive Query Splitting” (AQS) proposed by Myung & Lee (2006b), Myung & Lee (2006a); and a “Hybrid Query Tree” (HQT) proposed by Ryu et al. (2007). AQS keeps information, which is acquired during the last identification process in order to shorten the collision period. This technique requires tags to support both the transmission and reception at the same time, thereby making it difficult to apply to low-cost passive RFID systems. HQT uses a 4-ary query tree instead of a binary query tree, which increased too many *Idle cycles* despite reducing *Collision cycles*; while extra memory needed also increases as an identification process gets longer, because each query increases the prefixes by 2-bits instead of 1-bit. Accordingly, the Query Tree algorithm, adopted at present as the anti-collision protocol in EPC Class 1, may be limited to the tree based anti-collision protocol, which can be implemented (Choi et al. 2008).

3.2 Query Tree Based Protocols

The Query tree is a data structure for representing prefixes which is sent by the reader in Query tree protocols. A reader identifies tags through an uninterrupted communication with tags. The Query tree protocols consist of loops, and in each loop, the reader transmits a query with specific prefixes, and the tags respond with their IDs. Only tags with IDs

that match the prefixes, respond. When only one tag responds to reader, the reader successfully recognises the tag. When more than one tag tries to respond to reader’s query, tag collision occurs and the reader cannot get any information about the tags. The reader, however, can recognise the existence of tags to have ID that matches the query. For identifying tags that lead to the collision, the reader tries to query with 1-bit longer prefixes in next loops. By extending the prefixes, the reader can recognise all the tags.

Depending on the number of tags that respond to the interrogator, there are three cycles of communication between tag and reader.

- **Collision cycle:** Number of tags that respond to the reader is more than one. The reader cannot identify the ID of tags.
- **Idle cycle:** No response from any tag. It is a waste that should be reduced.
- **Success cycle:** Exactly one tag responds to the reader. The reader can identify the ID of the tag.

The delay in identification of tags is mostly affected by the *Collision cycles*, *Idle cycles*, and *Massive Tag Movement*. Therefore, minimising the number of *Collision cycles* and eliminating *Idle cycles* can improve the identification ability of the reader because *Bits Length* required by the reader can be reduced.

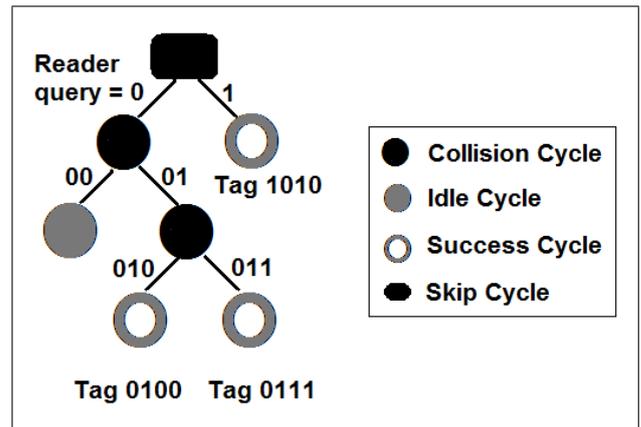


Figure 1: Tree-based anti-collision protocols: Memoryless Query Tree.

Figure 1 displays an example of a QT procedure. An identification process starts at Level one of tree, where QT uses tag IDs to split a tag set. It can be seen that *Tag 1010* is successfully identified in the first round because from all three tags, only *Tag 1010* has ‘1’ for the first bit of string. In the second round of identification, *Idle cycle* was created as there was no tag starting with ‘00’ for the first two bits. In the third round of identification, the other two tags, *Tag 0100* and *Tag 0111*, are successfully identified.

To overcome shortcomings of existing methods for anti-collision, we propose a “Joined Q-ary Tree” based on Memoryless Q-ary Tree, which adaptively adjust their tree branches according to Number of tags. We focus on analysing the impact of *Massive Tag Movement* within warehouse, therefore, we only considered static tags where tags have no mobility.

4 Methodology

In order to minimise the length of queries issued by a reader, a “Joined Q-ary Tree” or a combination of

two Q-ary trees is proposed. The Joined Q-ary Tree is based on a Memoryless Q-ary Tree anti-collision protocol, where no other memory is required beside a tag's IDs. This section will describe the scenarios where massive EPC tag movement exists and *EPC pattern* are used; a classification of *Splitting Fitness*; the *Joined Q-ary Tree* with Sample Tag Splitting; and the Tags Prediction and Classification for *Unique Bits* of EPC.

4.1 Warehouse Distribution Scenarios

In this work, a specific scenario is examined based on *Massive tag movement*. The paper focus's is on Crystal warehouse scenario, which can be classified into four different scenarios, as follows:

- **Scenario One:** All items have the same Encoding Schemes, same Company Prefixes, same Object Class, but different Serial Number (Figure 2a).
- **Scenario Two:** All items have the same Encoding Schemes and same Company Prefixes; but different Object Class and different Serial Number (Figure 2b).
- **Scenario Three:** All items have the same Encoding Schemes; but different Company Prefixes, different Object Class, and different Serial Number (Figure 2c).
- **Scenario four:** All items have different Encoding Schemes, different Company Prefixes, different Object Class, and different Serial Number (Figure 2d).

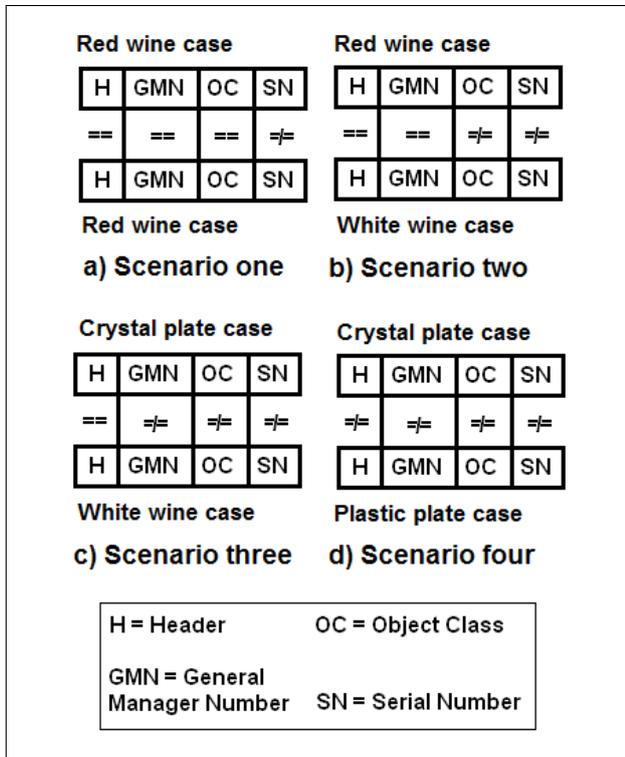


Figure 2: Crystal Warehouse Scenario: a) Two cases of Red-wine, b) White-wine case and Red-wine case, c) White-wine case and Crystal plate case, and d) Crystal plate case and Plastic plate case.

Out of the four scenarios, Scenario one and Scenario two will most likely occur in a large warehouse

distribution. For example, there are 5 pallets of: Red-wine glasses; White-wine glasses; Champagne glasses; Beer glasses; and Tumbler; with 12 cases each that are ready to leave the warehouse to a smaller retailer. If all of 5 pallets move into an interrogation zone at the same time, we will have the data with five different *Object Class* (OC) and sixty different *Serial Numbers* (SN). Nevertheless, since all items are from the same company that produced crystal-ware, they will all have the same *Header* and *General Manager Number*. When tags from the same pallet collide, we will have two or more tags with the same *Header*, same *General Manager Number*, same *Object Class*, and unique *Serial Number* which are related to Scenario one. On the other hand, when tags from different pallets collide, we will have two or more tags with the same *Header*, same *General Manager Number*, different *Object Class*, and different *Serial Number*, which are related to Scenario two.

In addition, a data management in a large warehouse distribution can be improved by using *EPC pattern*. As explained earlier, *EPC pattern* is referred to as a set of tag encodings. A sample of *EPC pattern* is: 25.1545.[3456-3478].[778-795] in decimal, which later will be encoded to binary and embedded onto tags. Thus, within this sample pattern, *Header* is fixed to 25 and *Company Prefixes* is 1545; while *Object Class* can be any number between 3456 and 3478; and *Serial Number* can be anything between 778 and 795.

In this paper, we will focus on data captured according to Scenario one and Scenario two, and *EPC pattern* will be used to create different set of tag encodings.

4.2 Splitting Fitness

Splitting fitness can be classified into *Worst-case splitting*, *Perfect splitting*, and *Random splitting*. All three cases are discussed below:

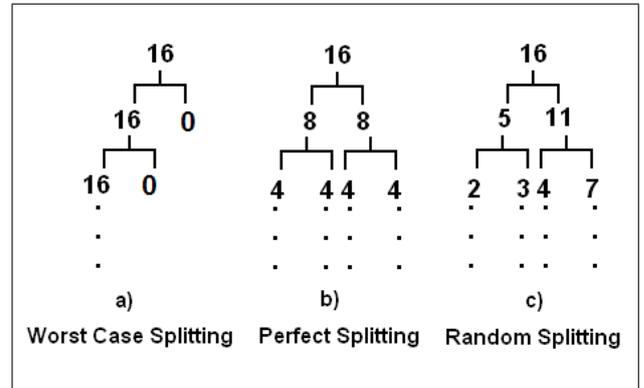


Figure 3: Splitting Fitness: a) Worst-Case Splitting, b) Perfect Splitting, and c) Random Splitting.

4.2.1 Worst-Case Splitting

Worst-Case splitting is when tags spliced into an unbalanced tree, where one child node has no further node in a binary tree case. Figure 3a) shows that there are 16 tags at Level 0 tree; then at Level 1, tags spliced into 16 tag on the left-hand node and no tag on the right-hand node. No further splitting is necessary on the right-hand node, since there is no tag left. This case of splitting will likely happen for the first few bits of EPC identification in real world warehouse environment because most items have *Massive tag movement* and usually belong to the same *EPC pattern* with

similar ID. The worst case splitting caused more *Idle cycles* especially for higher level trees (4-ary, 8-ary, and 16-ary) because all tags will be travelling down to only one side of the tree, which results in further collision.

4.2.2 Perfect Splitting

Perfect splitting happens when a set of tags spliced to the left and right child node equally. Figure 3b) shows that there are 16 tags at Level 0 tree; then at Level 1, tags spliced equally into 8 nodes. Further splitting is required for both left-hand and right-hand nodes until only one tag is left. This case of splitting is impossible in real world scenario but will be the closest case to the latter stages (bits) of EPC identification within warehouse environment because most items belong to the same group of *EPC pattern*. For example, 1 pallet of White-wine glasses containing 20 cases move into one interrogation zone. All items from the pallet will have the same *Object Class* and will travel along the same side of child node at earlier Levels; resulting in *Worst Case Splitting*. However, the remaining few bits will be unique for each EPC since they belong to *Serial Number*. These remaining bits encoded within the same *EPC pattern* will split almost equally to the left and right child nodes. Both child nodes of left-hand side and right-hand side of binary tree will not be exactly equal since data captured are not always even. Therefore, we can call this situation as *Almost-Perfect splitting*.

4.2.3 Random Splitting

Random splitting happens when a set of tags spliced to the left and right child node randomly and splitting pattern cannot be found. Figure 3c) shows that there are 16 tags at Level 0 tree; then at Level 1, tags spliced into 5 and 11. Further at Level 2, tags spliced into 2, 3, 4, and 7 where no specific splitting pattern exists; thus, this situation is called *Random Splitting*. This case of splitting will likely happen in retail distribution environments since all items usually come from different locations. Thus, this splitting case will not be further discussed as this research will only focus on warehouse environment.

4.3 Joined Q-ary Tree

Query Tree uses each bit of tag ID to split a tag set, Q-ary tree uses every 2-, 3-, or 4-bits of tag ID to split a tag set. Q-ary tree increases the child node of tree from ‘2’ to ‘4’, ‘8’ or ‘16’ nodes and so on. This way, we can reduce collisions but at the same time, the *Idle cycles* will increase, along with the *Bits Length* needed in each query. The best way to solve this problem is to employ the right combination of Q-ary trees for each specific scenario. This will depend on the specific Number of tags within an interrogation zone, *Massive Tag Movement*, and *Splitting Fitness* based on *EPC pattern*.

The Joined approach is a combined Q-ary trees, specifically 2-ary Tree and 4-ary Tree, which have been identified to be the best Q-ary trees in (Pupun-wiwat & Stantic 2009). The Joined approach will be applied on each collided tags EPC, which will be split using every 1 or 2-bits of tag ID for the first few queries; and then at one point, every 1 or 2-bits will be queried. With the fact that most items from warehouse have massive movement, first few bits of EPC will be identical and the remaining bits will be very similar. In order to optimise the performance of Joined Q-ary Tree, the right Separating Point (SP) between the two Q-ary trees needs to be configured.

The objective of Joined Q-ary Tree is to reduce the *Bits Length* queried by a reader so that Identification time can be minimised. In this paper, we will investigate and compare the “Naive Q-ary Tree” approach and our newly proposed “Joined Q-ary Tree”.

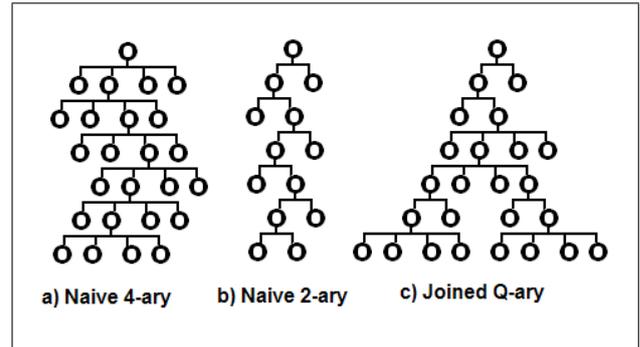


Figure 4: A sample of: a) a Naive 4-ary Tree, b) a Naive 2-ary Tree, and c) a Joined Q-ary Tree.

Figure 4 shows the example of a) Naive 2-ary, b) Naive 4-ary, and c) Joined Q-ary Tree. Joined Q-ary Tree bonded both 2-ary and 4-ary tree together and apply to specific bits of EPC depending on how Identical or Unique they are. The classification of *Identical bits* and *Unique bits* will be explained further within this section.

4.3.1 EPC Bits Prediction and Classification

In warehouse distribution environment according to Scenario one and Scenario two, it is known that the first 36-bits of EPC (Header and GMN) are definitely Identical. However, 24-bits of *Object Class* can be both Identical and Unique for all tags, depending on how many pallets existed within one interrogation zone. For example, if there are 5 pallets of 12 cases each in the interrogation zone, there will be five different *Object Class* and sixty unique *Serial Numbers* for all sixty items (cases).

Since *Object Class* involved 24-bits of EPC (allow 16,777,215 unique tags) but only 5 unique OC is needed, we must calculate a certain number of *Unique bits* needed in order to apply the right Q-ary tree. This also applied to *Serial Number* that contains 36-bits of string. Assuming that *EPC pattern* is used, not all 36-bits of these strings will be Unique.

Table 2 shows a formal structure for bits classification of GID-96 bits EPC. It can be seen that the *Identical bits* of EPC always equal to 36-bits for the first 36-bits of EPC. This includes 8-bits of *Header* and 28-bits of *GMN*, which are always the same for all tags. For *Object Class*, 24-bits are available where *Unique bits* within *Object Class* (UOC) can be predicted using Equation (1). In addition, *Serial Number* with 36-bits can also be predicted using the same Equation.

| | Length | Identical | Unique |
|----------------------|--------|-----------|--------|
| Header | 8 | 8 | 0 |
| GMN | 28 | 28 | 0 |
| Object Class | 24 | 24 - UOC | UOC* |
| Serial Number | 36 | 36 - USN | USN** |

Table 2: Formal structure of bits classification of EPC GID-96 bits. *UOC is number of Unique bits within *Object Class* and **USN is number of Unique bits within *Serial Number*.

Our method is executed based on the assumption that the approximate Number of tags (pallets, cases)

is known prior to the identification process. This information is needed for *Unique bits* calculation: UOC, and USN from table 2. However, in most circumstances, Number of tags is usually unknown until the first query issued by the reader. Therefore, UOC and USN of Joined Q-ary Tree can be initially set to zero and after the first round of identification, these two parameters can be calculated.

Joined Q-ary Tree adaptively adjusts their tree branches at specific Separating Point. These SP is configured according to *Identical bits* and *Unique bits* within an EPC data. In order to calculate the estimated number of *Unique bits* within an EPC, we need the average Number of tags within an interrogation zone, and then to apply the equation below:

$$B = \log_2(N) \quad (1)$$

Where N = Number of tags, B = *Unique Bits* of EPC.

Figure 5 illustrates tag splitting behaviour of massive tag within a warehouse. The first 36-bits of EPC belongs to *Header* and *GMN*, therefore 2-ary Tree is applied to these bits since it is simplified to be the most suitable tree for *Identical bits* of EPC. UOC and USN on the other hand, can be split using 4-ary Tree since it is proven to be the most suitable tree for *Unique bits* of EPC.

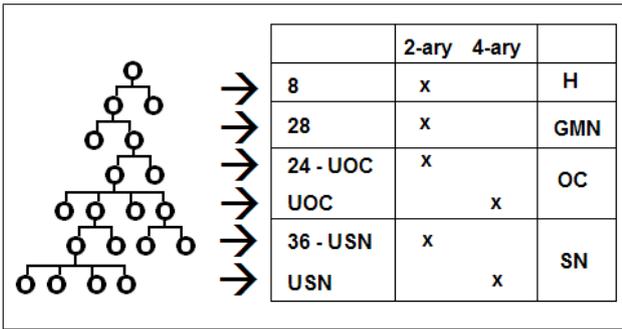


Figure 5: Joined Q-ary Tree structure for GID-96 bits EPC.

4.3.2 Sample Bits Prediction

To demonstrate a calculation of *Unique bits* of EPC, we examine a *Massive tag movement* of 720 tags within 12 pallets (OC). By using Equation (1), UOC and USN can be calculated as follows:

$$UOC = \log_2(N) = \frac{\log_{10}(N)}{\log_{10}(2)} = \frac{\log_{10}(12)}{\log_{10}(2)} \approx 4$$

$$USN = \log_2(N) = \frac{\log_{10}(N)}{\log_{10}(2)} = \frac{\log_{10}(60)}{\log_{10}(2)} \approx 6$$

Therefore, number of *Unique bits* required to cover all unique *Object Class* is approximately 4-bits and approximately 6-bits for *Serial Number*.

Table 3 shows a sample structure for bits classification of 720 tags with GID-96 bits EPC encoding scheme. Corresponding with figure 6, we can see that the *Identical bits* of EPC always equal to 36-bits for the first 36-bits of EPC. 2-ary Tree is applied to these bits. *Identical bits* of *Object Class* (20-bits) and *Serial Number* (30-bits) are also spliced by 2-ary Tree. UOC (4-bits) and USN (6-bits) on the other hand, can be split using 4-ary Tree.

| | Length | Identical | Unique |
|----------------------|-----------|-----------|--------|
| Header | 8 | 8 | 0 |
| GMN | 28 | 28 | 0 |
| Object Class | 24 | 20 | 4 |
| Serial Number | 36 | 30 | 6 |

Table 3: Sample bits classification of EPC GID-96 bits, where *Object Class* = 12 and *Serial Number* = 60 (Total of 720 tags).

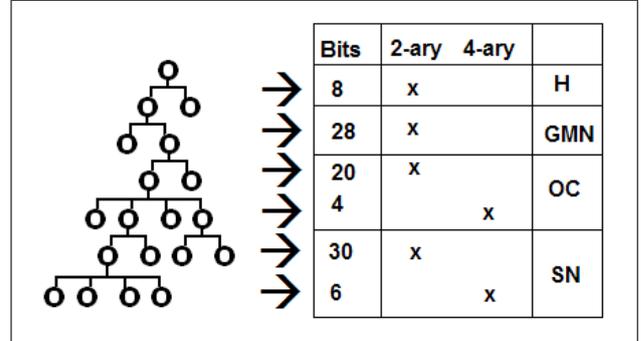


Figure 6: Joined Q-ary Tree structure for GID-96 bits EPC with 36 *Identical bits* Header and GMN, 20 *Identical bits* OC, 4 *Unique bits* OC, 30 *Identical bits* SN, and 6 *Unique bits* SN.

4.3.3 Sample Tags Splitting

We are now initiating a sample comparison between the performance of Naive 2-ary tree, Naive 2-ary, and Joined Q-ary Tree. Table 4 shows 10 sample EPC of 36-bits tags with 24-bits *Identical* and 8-bits *Unique*. We assumed that all *Identical bits* of EPC belong to *Header* and *General Manager Number*, while *Unique bits* of EPC belong to *Object Class* and *Serial Number*.

| Identical bits | Unique bits |
|-------------------------------|-------------|
| 0011 0101 1010 1101 0000 1010 | 1011 0101 |
| 0011 0101 1010 1101 0000 1010 | 0100 1011 |
| 0011 0101 1010 1101 0000 1010 | 1101 1100 |
| 0011 0101 1010 1101 0000 1010 | 0010 0100 |
| 0011 0101 1010 1101 0000 1010 | 1100 1000 |
| 0011 0101 1010 1101 0000 1010 | 0001 0100 |
| 0011 0101 1010 1101 0000 1010 | 0011 0011 |
| 0011 0101 1010 1101 0000 1010 | 1110 1010 |
| 0011 0101 1010 1101 0000 1010 | 1111 0010 |
| 0011 0101 1010 1101 0000 1010 | 0000 0110 |

Table 4: Sample 36 bits tags with 24 bits *Identical* and 8 bits *Unique*.

Sample Tag Identification After applying different trees on these tags, we can see that the Naive 2-ary tree has the best performance on *Identical bits* of EPC by using smallest number of queries (*Bits Length*) as shown in table 5. On the other hand, 4-ary tree performed better on *Unique bits* than 2-ary tree.

Sample Bits Length Calculation In order to calculate a *Total Bits Length* required for the whole identification process for 2-ary Tree, 4-ary Tree, and Joined Q-ary Tree, information on Number of Child Nodes (NCN) for each level of tree and Number of Bits per Query (NBQ) for that specific level, is

| Performances on Identical bits | | | | | |
|--------------------------------|-------------|------------------|-------------------|--------------|-------------|
| Q-ary Tree | Idle Cycles | Collision Cycles | Successful Cycles | Total Cycles | Bits Length |
| 2-ary | 24 | 24 | 0 | 48 | 600 |
| 4-ary | 36 | 12 | 0 | 48 | 624 |
| Performances on Unique bits | | | | | |
| Q-ary Tree | Idle Cycles | Collision Cycles | Successful Cycles | Total Cycles | Bits Length |
| 2-ary | 0 | 8 | 10 | 18 | 36 |
| 4-ary | 2 | 0 | 10 | 12 | 32 |

Table 5: Identification process of 2-ary Tree and 4-ary Tree on *Identical bits* and *Unique bits* of EPC.

needed. Number of bits per Level (NBL) can be calculated as follows:

$$NBL = NCN \times NBQ \quad (2)$$

After calculating the NBL for each level of tree, the *Total Bits Length* (NBLs) required can be found by doing the summation of all NBL. After adding all NBL together, the NBLs of 828-bits, 848-bits, and 728-bits are shown in table 6 respectively for 2-ary Tree, 4-ary Tree, and Joined Q-ary Tree.

| Level | 2-ary | 4-ary | Joined |
|--------------|------------|------------|------------|
| 1-12 | 156 | 624 | 156 |
| 13-14 | 54 | 224 | 54 |
| 15-24 | 390 | 0 | 390 |
| 25-26 | 104 | 0 | 128 |
| 27-28 | 124 | 0 | 0 |
| NBLs | 828 | 848 | 728 |

Table 6: Calculation of *Total Bits Length* required for two Naive Q-ary Trees and a Joined Q-ary Tree. NBLs shows the *Bits Length* required for the specific Naive/Joined Q-ary Tree.

Table 7 shows the overall calculation of *Bits Length* queried on *Identical bit* and *Unique bits* of EPC. We can see that the Joined Q-ary Tree required the least *Total Bits Length* compared to the two Naive Q-ary Trees. Joined Q-ary Tree reduced *Bits Length* by almost 15% compared to Naive 4-ary Tree and by 12% compared to Naive 2-ary Tree.

5 Experimental Evaluation

To study the proposed ‘‘Joined Q-ary Tree’’ and compare with the performance of Naive approaches, experiments are performed according to a Crystal warehouse scenario. The experiment is assumed to be set

| Different Q-ary Tree on Sample Tags | | | | | | | |
|-------------------------------------|-------------|------------------|-------------------|--------------|----------------|-------------|-------------------|
| Q-ary Tree | Idle Cycles | Collision Cycles | Successful Cycles | Total Cycles | Identical Bits | Unique Bits | Total Bits Length |
| 2-ary | 24 | 32 | 10 | 66 | 600 | 228 | 828 |
| 4-ary | 38 | 12 | 10 | 60 | 624 | 224 | 848 |
| Joined | 26 | 24 | 10 | 60 | 600 | 128 | 728 |

Table 7: Performance Analysis of 2-ary, 4-ary, and Joined Q-ary on set of 10 sample tags.

up in a well controlled environment where there is no metal or water nearby. A UHF RFID reader is used and mounted on a dock door at the end of a conveyor belt. Passive RFID tags are attached to each case of crystal ware. Each pallet of wine glasses, plates, and bowls are moved along this conveyor belt. At this stage, we assume that all three pallets move-in and move-out at the same time to an interrogation zone and no Arriving tag or Leaving tags are present during each identification round.

Specification: An Intel Core 2 CPU with 2.40GHz processor and 3GB RAM is used for testing. A Microsoft Window XP professional with Service Pack 3 is installed on the computer. Algorithms for data simulation are implemented using Java JCreator.

5.1 Data Set

In this study, we conducted an experiment using three different tag sets: 288 tags, 576 tags, and 864 tags. The impact of different Number of tags in an interrogation zone and performances of difference approaches are to be evaluated. Data sets using *EPC pattern* from table 8 are used:

| | EPC Pattern |
|------------|------------------------------------|
| H | 0011 0101 |
| GMN | 104,426,055 |
| OC | [9,872,273 - 9,872,308] |
| SN | [26,292,755,245 - 26,292,755,268] |

Table 8: The *EPC Pattern* is shown in Decimal for GMN, OC, and SN, but will be encoded in Binary on to each tag attached to the item.

- DATA SET ONE: 12 pallets, 24 cases each
- DATA SET TWO: 24 pallets, 24 cases each
- DATA SET THREE: 36 pallets, 24 cases each

5.1.1 Theoretical Bits Prediction

Assuming that the existence of tags are known before identification process, by using Equation (1), UOC and USN of data set one, two, and three can be calculated as follows:

Data Set one:

$$UOC = \log_2(12) \approx 4, USN = \log_2(24) \approx 5$$

Data Set two:

$$UOC = \log_2(24) \approx 5, USN = \log_2(24) \approx 5$$

Data Set three:

$$UOC = \log_2(36) \approx 6, USN = \log_2(24) \approx 5$$

Therefore, number of *Unique bits* required to cover all unique *Object Class* and *Serial Number* are as shown in table 9.

| | 288 tags | | 576 tags | | 864 tags | |
|------------|----------|---|----------|---|----------|---|
| | I | U | I | U | I | U |
| H | 8 | 0 | 8 | 0 | 8 | 0 |
| GMN | 28 | 0 | 28 | 0 | 28 | 0 |
| OC | 20 | 4 | 19 | 5 | 18 | 6 |
| SN | 31 | 5 | 31 | 5 | 31 | 5 |

Table 9: Identical and Unique Bits classification of EPC GID-96 bits for Data set one, two and three. I = Identical bits, U = Unique bits.

5.1.2 Actual Separating Point Configuration

After theoretical bits estimation, we assumed that the actual encoding of *Unique bits* may be 1 to 2-bits longer than predicted. Therefore, we added 2-bits to the predicted *Unique bits* of each data set. If the predicted bits added up as odd number, 1 more bit is further attached. For example, UOC of data set two (576 tags) is predicted to be 5-bits long. By affixing additional 2-bits, total of 7-bits are applied to UOC. However, this added up as odd number, therefore, 1 more bit is further added (Total 8-bits). Table 10 shows an actual SP for each data set. At specific SP, the Joined Q-ary Tree will adaptively change its branch to 2-ary or 4-ary. Since 2-ary Tree is applied from SP1 to SP3, the Joined Q-ary Tree only needs to adjust its branch at SP4, SP5, and SP6.

| | SP | 288 tags | | 576 tags | | 864 tags | |
|--------------|----------|----------|----|----------|----|----------|----|
| | | 2 | 4 | 2 | 4 | 2 | 4 |
| H | 1 | 0 | - | 0 | - | 0 | - |
| GMN | 2 | 0 | - | 0 | - | 0 | - |
| OC(I) | 3 | 37 | - | 37 | - | 37 | - |
| OC(U) | 4 | - | 55 | - | 53 | - | 53 |
| SN(I) | 5 | 61 | - | 61 | - | 61 | - |
| SN(U) | 6 | - | 89 | - | 89 | - | 89 |

Table 10: Actual Separating Point for Data set one, two, and three. At a specific SP, Joined Q-ary Tree will adjust its branch to either 2-ary or 4-ary tree.

5.2 Results

Based on the experiment results shown in table 11 and figure 7, the Joined Q-ary Tree always performed the best out of the three approaches considered, while 4-ary Tree has the worst performance, regardless of number of tags within an interrogation zone.

Table 11 demonstrates that 2-ary tree has a better performance than 4-ary tree by about 1 percent, while Joined Q-ary Tree's performance is approximately 12 percent better than the 2-ary tree. The 4-ary Tree has the worst performance out of the three approaches considered. Figure 8 illustrates the improvement in percentage of Joined Q-ary Tree compared to the 2-ary Tree and the 4-ary Tree. The percentage of improvement increases more slowly once the number of tags within the interrogation zone gets higher.

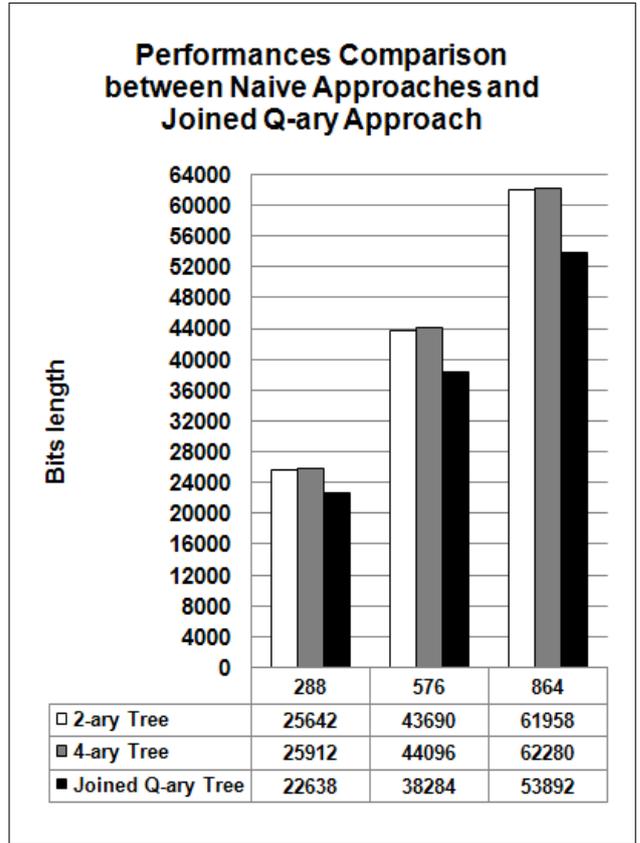


Figure 7: Performances comparison between Naive approaches and Joined Q-ary approach.

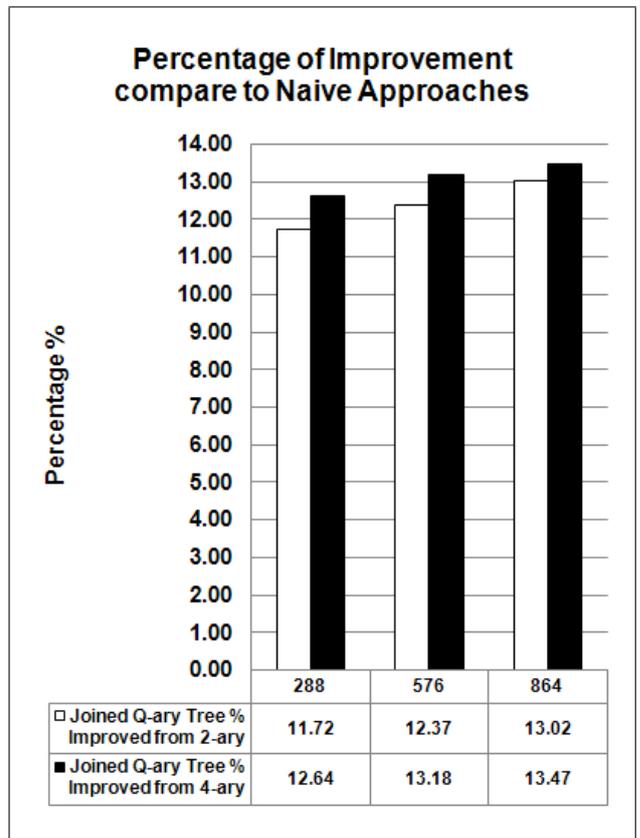


Figure 8: Percentage of improvement of Joined Q-ary Tree compared to Naive 2-ary Tree and Naive 4-ary Tree.

| | 2-ary | | 4-ary | | Joined | |
|-----|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| | % improved from 2-ary | % improved from 4-ary | % improved from 2-ary | % improved from 4-ary | % improved from 2-ary | % improved from 4-ary |
| 288 | 0 | 1.04 | -1.05 | 0 | 11.72 | 12.64 |
| 576 | 0 | 0.92 | -0.93 | 0 | 12.37 | 13.18 |
| 864 | 0 | 0.52 | -0.52 | 0 | 13.02 | 13.47 |

Table 11: Results of 2-ary Tree, 4-ary Tree, and Joined Q-ary Tree on three different data sets.

5.3 Analysis and Discussion

To further analyse and compare performances of Naive Q-ary approaches and Joined Q-ary approach, table 12 shows *Accumulative Bits Length* of each approach until all tags were identified. We can see that the number of *Bits Length* accumulates faster when the identification reached SP4. This is when EPC data became more Unique and larger *Bits Length* issued by reader were required. All three approaches have the same pattern of increment, where number of tags in an interrogation zone have no impact on the performances when EPC data were Identical. This pattern can be seen in figure 9, where *Accumulative Bits Length* of the Joined Q-ary Tree approach is displayed. There are no or little differences between SP1 to SP4 on all data sets, however, number of *Bits Length* started to increase rapidly once it reaches SP5 and SP6.

| | 2-ary Tree | | |
|-----|-------------------|----------|----------|
| | 288 tags | 576 tags | 864 tags |
| SP1 | 72 | 72 | 72 |
| SP2 | 1332 | 1332 | 1332 |
| SP3 | 2970 | 2756 | 2756 |
| SP4 | 3682 | 3730 | 3926 |
| SP5 | 17178 | 27642 | 38310 |
| SP6 | 25642 | 43690 | 61958 |
| | 4-ary Tree | | |
| | 288 tags | 576 tags | 864 tags |
| SP1 | 80 | 80 | 80 |
| SP2 | 1368 | 1368 | 1368 |
| SP3 | 3024 | 2808 | 2808 |
| SP4 | 3736 | 3776 | 3816 |
| SP5 | 17400 | 27968 | 38536 |
| SP6 | 25912 | 44096 | 62280 |
| | Joined Q-ary Tree | | |
| | 288 tags | 576 tags | 864 tags |
| SP1 | 72 | 72 | 72 |
| SP2 | 1332 | 1332 | 1332 |
| SP3 | 2970 | 2756 | 2756 |
| SP4 | 3358 | 3308 | 3348 |
| SP5 | 17246 | 27948 | 38628 |
| SP6 | 22638 | 38284 | 53892 |

Table 12: *Accumulative Bits Length* of three approaches: Naive 2-ary Tree, Naive 4-ary Tree, and Joined Q-ary Tree.

Out of the three approaches, the Joined Q-ary Tree has the slowest incremental rate between SP4 and SP6 for all data sets. The incremental *Bits Length* also slows down once number of tags within the interrogation zone gets bigger. Thus, the performance of Joined Q-ary Tree will increase for larger set of tags compared to the Naive approaches.

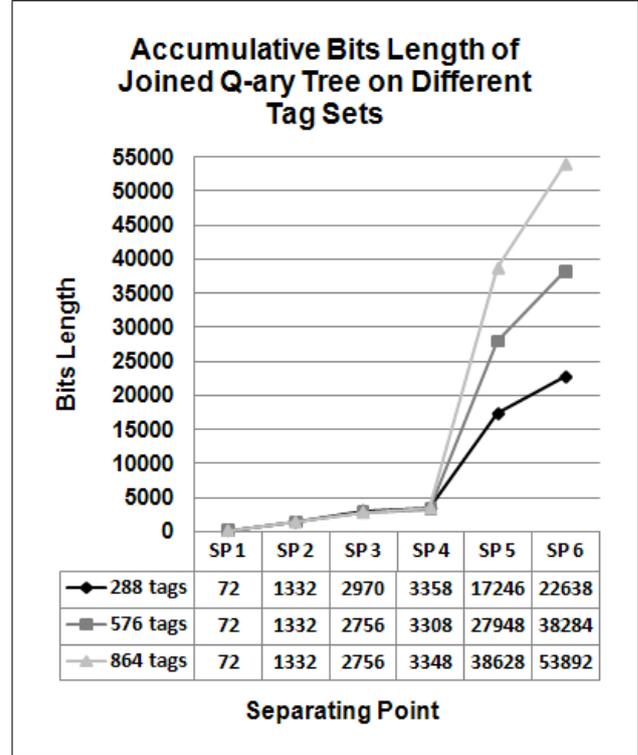


Figure 9: *Accumulative Bits Length* of three approaches: Naive 2-ary Tree, Naive 4-ary Tree, and Joined Q-ary Tree.

Since the Joined Q-ary Tree accumulated least *Bits Length* out of the three approaches, it can now be concluded that the best performing approach is Joined Q-ary Tree. Also, the number of tags within the interrogation zone have no impact on *Identical bits* of EPC data. Additionally, by applying the right Q-ary tree on specific bits of EPC, performance of Joined Q-ary Tree is improved.

6 Conclusion

In this study, we identified the significance of RFID tags anti-collisions and developed efficient method to minimise the *Bits Length* issued by a reader; and at the same time to ensure that 100 percent of RFID tags are identified. We proposed a “Joined Q-ary Tree”, which adaptively adjust its tree branches according to number of tags, in order to minimise *Bits Length* queried. In the experimental evaluation, we showed that the proposed method has better performance when compared to existing methods. Specifically, it requires about 12 percent less queries issued per complete identification than the existing approaches and at the same time guarantees identification of all tags.

In terms of future work, we intend to test a performance of “Joined Q-ary Tree” on different *Encoding Scheme* other than GID-96. Different pallet sizes simulation will also be inspected to determine the impact of packaging and density on quality of captures data. In addition, power consumption of readers and impact of capture effect will be observed.

Acknowledgements

This research is partly supported by ARC (Australian Research Council) grant no DP0557303.

References

- Abramson, N. (1970), The ALOHA System - Another Alternative for Computer Communications, *in* 'Proceedings of Fall Joint Computer Conference, AFIPS Conference', Houston, Texas, pp. 281–285.
- Bai, Y., Wang, F. & Liu, P. (2006), Efficiently Filtering RFID Data Streams, *in* 'CleanDB Workshop', pp. 50–57.
- Brusey, J., Floerkemeier, C., Harrison, M. & Fletcher, M. (2003), Reasoning About Uncertainty in Location Identification with RFID, *in* 'Workshop on Reasoning with Uncertainty in Robotics at IJCAI', Acapulco, Mexico.
- Carbunar, B., Ramanathan, M. K., Koyuturk, M., Hoffmann, C. & Grama, A. (2005), Redundant Reader Elimination in RFID Systems, *in* '2nd Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON'05)', pp. 176–184.
- Cho, H., Lee, W. & Baek, Y. (2007), LDFSA: A Learning-Based Dynamic Framed Slotted ALOHA for Collision Arbitration in Active RFID Systems, *in* 'Advances in Grid and Pervasive Computing Second International Conference', Vol. 4459, Springer Berlin/Heidelberg, Paris, France, pp. 655–665.
- Choi, J. H., Lee, H. J., Lee, D., Lee, H. S., Youn, Y. & Kim, J. (2008), 'Query Tree Based Tag Identification Method in RFID Systems'. www.freshpatents.com/Query-tree-based-tag-identification-method-in-rfid-systems-dt20080508ptan20080106383.php.
- EPCGlobal (2006), 'EPCGlobal Tag Data Standards Version 1.3: Ratified Specification'. <http://www.epcglobalinc.org/standards/tds/>.
- Fishkin, K. P., Jiang, B., Philipose, M. & Roy, S. (2004), I Sense a Disturbance in the Force: Unobtrusive Detection of Interactions with RFID-tagged Objects, *in* 'UbiComp 2004: Ubiquitous Computing', Seattle, Washington, USA, pp. 268–282.
- Gonzalez, H., Han, J. & Li, X. (2006), Mining Compressed Commodity Workflows from Massive RFID Data Sets, *in* 'CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management', ACM Press, New York, NY, USA, pp. 162–171.
- Gonzalez, H., Han, J., Li, X. & Klabjan, D. (2006), Warehousing and Analyzing Massive RFID Data Sets, *in* 'ICDE '06: Proceedings of the 22nd International Conference on Data Engineering', IEEE Computer Society, Washington, DC, USA, p. 83.
- Jain, S. & Das, S. R. (2006), Collision avoidance in a dense RFID network, *in* 'WiNTECH '06: Proceedings of the 1st international workshop on Wireless network testbeds, experimental evaluation & characterization', ACM, New York, NY, USA, pp. 49–56.
- Jeffery, S. R., Alonso, G., Franklin, M. J., Hong, W. & Widom, J. (2005), A Pipelined Framework for Online Cleaning of Sensor Data Streams, Technical Report UCB/CSD-05-1413, EECS Department, University of California, Berkeley.
- Jeffery, S. R., Alonso, G., Franklin, M. J., Hong, W. & Widom, J. (2006), Declarative Support for Sensor Data Cleaning, *in* 'Pervasive Computing', Springer Berlin/Heidelberg, Dublin, Ireland, pp. 83–100.
- Jeffery, S. R., Garofalakis, M. & Franklin, M. J. (2006), Adaptive cleaning for RFID data streams, *in* 'VLDB'2006: Proceedings of the 32nd international conference on Very large data bases', VLDB Endowment, Seoul, Korea, pp. 163–174.
- Lee, C. W., Cho, H. & Kim, S. W. (2008), 'An Adaptive RFID Anti-Collision Algorithm Based on Dynamic Framed ALOHA', *IEICE Transactions* **91-B**(2), 641–645.
- Lee, S. R., Joo, S. D. & Lee, C. W. (2005), An enhanced dynamic framed slotted aloha algorithm for rfid tag identification, *in* 'MOBIQUITOUS '05: Proceedings of the The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services', IEEE Computer Society, Washington, DC, USA, pp. 166–174.
- Myung, J. & Lee, W. (2006a), 'Adaptive binary splitting: a RFID tag collision arbitration protocol for tag identification', *Mob. Netw. Appl.* **11**(5), 711–722.
- Myung, J. & Lee, W. (2006b), Adaptive splitting protocols for RFID tag collision arbitration, *in* 'MobiHoc '06: Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing', ACM, New York, NY, USA, pp. 202–213.
- Pupunwiwat, P. & Stantic, B. (2009), Unified Query Tree for RFID Tag Anti-Collision Resolution, *in* A. Bouguettaya & X. Lin, eds, 'Twentieth Australasian Database Conference (ADC 2009)', Vol. 92 of *CRPIT*, ACS, Wellington, New Zealand, pp. 49–58.
- Quan, C. H., Hong, W. K. & Kim, H. C. (2006), Performance Analysis of Tag Anti-collision Algorithms for RFID Systems, *in* 'Emerging Directions in Embedded and Ubiquitous Computing', Vol. 4097, Springer Berlin/Heidelberg, Seoul, Korea, pp. 382–391.
- Ryu, J., Lee, H., Seok, Y., Kwon, T. & Choi, Y. (2007), A Hybrid Query Tree Protocol for Tag Collision Arbitration in RFID systems, *in* 'MobiHoc '06: Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing', IEEE Computer Society, Glasgow, UK, pp. 5981–5986.
- Shin, J. D., Yeo, S. S., Kim, T. H. & Kim, S. K. (2007), Hybrid Tag Anti-collision Algorithms in RFID Systems, *in* 'Computational Science ICCS 2007', Vol. 4490, Springer Berlin/Heidelberg, Beijing, China, pp. 693–700.
- Vogt, H. (2002), Efficient Object Identification with Passive RFID Tags, *in* 'Pervasive '02: Proceedings of the First International Conference on Pervasive Computing', Springer-Verlag, London, UK, pp. 98–113.
- Wang, Z., Liu, D., Zhou, X., Tan, X., Wang, J. & Min, H. (2007), 'Anti-collision Scheme Analysis of RFID System', *Auto-ID Labs White Paper*. <http://www.autoidlabs.org/single-view/dir/article/6/281/page.html>.