Unified Q-ary Tree for RFID Tag Anti-Collision Resolution

Prapassara Pupunwiwat       Bela Stantic
Institute of Integrated and Intelligent Systems
Griffith University, Gold Coast Campus
Queensland 4222, Australia
Email: {p.pupunwiwat, b.stantic}@griffith.edu.au

Abstract

Radio Frequency Identification (RFID) technology uses radio-frequency waves to automatically identify people or objects. A large volume of data, resulting from the fast capturing RFID readers and a huge number of tags, poses challenges for data management. This is particularly the case when a reader simultaneously reads multiple tags and Radio Frequency (RF) collisions occur, causing RF signals to interfere with each other and therefore preventing the reader from identifying all tags. This problem is known as Missed reads, which can be solved by using anti-collision techniques to prevent two or more tags from responding to a reader at the same time. The current probabilistic anti-collision methods are suffering from Tag starvation problems so not all tags can be identified, while the deterministic methods suffer from too long Identification delay. In this paper, a “Unified Q-ary Tree Protocols” based on Query tree is presented. In empirical study compared with the Query tree and 4-ary tree, we show that the proposed method performs better, it requires less number of queries per complete identification, which results in less total identification time.

1 Introduction

RFID technology has gained significant momentum in the past few years. It has promised to improve the efficiency of business processes by providing the automatic identification and data capture. The core RFID technology is not new, and it can be traced back to World War II where it was used to distinguish between friendly and enemy aircrafts or known as friend-or-foe (Landt 2001). 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.

In warehouse distribution environment where RFID systems are deployed, the Ultra High Frequency (UHF) range of radio-frequency waves are used for long distance identification. UHF includes frequencies from 300 to 1000MHz, but only two frequency ranges, 433MHz and 860-960MHz, are used for UHF RFID systems. The 433MHz frequency is used for active tags, while the 860-960MHz range is used for passive tags or semi-passive tags. All protocols in the UHF range have some type of anti-collision capability, which allow multiple tags to be read simultaneously within the interrogator zone (Brown et al. 2007).

Despite significant improvement with respect to the quality of readers and tags, a significant percentage of captured data still has errors, which are particularly due to the Missed reads. To prevent these Missed reads, mostly caused by RF Collision, several techniques for the Edge anti-collision have been proposed in the literature. However, these approaches still suffer from either Tag starvation problem, or produced too many Collision cycles and Idle cycles, which causes Identification delay.

In this study, we propose a new anti-collision algorithm called “Unified Q-ary Tree Protocols”, which is a combination of Q-ary trees, particularly a binary tree, 4-ary tree, 8-ary tree, and 16-ary tree, to optimise the anti-collision in reading RFID tags. We focus on deterministic anti-collision protocols since they can achieve 100 percent identification. We also concentrate on the impact of similarity of EPC data, especially in warehouse environment where most items have bulky movement. These items are usually manufactured from the same company which evidently used the same Encoding Schemes and have the same Company Prefixes. In simulated experimental study, we show that our method reduces Collision cycles and Idle cycles, which resulted in less Identification delay and improve a quality of captured data.

The remainder of this paper is organised as follows: In section 2, some general background on RFID and information related to Missed reads including their causes is provided. In section 3, we discuss the related works to our proposed method and their limitations, which include probabilistic and deterministic anti-collision methods, and Query tree protocols. In section 4, we are presenting a new technique, the “Unified Q-ary Tree Protocols” including methodology and scenarios. In section 5, we present experimental results, analysis and discussions, and finally in section 6 we provide our conclusion and future work.

2 RFID Background

RFID may only consist of a tag and a reader but a complete RFID system involves many other components, such as computer, network, Internet, and software such as middleware and user applications. A typical RFID system is divided into two layers: the physical layer and Information Technology (IT) layer (Brown et al. 2007).

The physical layer consists of: one or more reader antennas, one or more readers (Interrogator), one or more tags (Transponder), and deployment environment.
The IT layer consists of one or more host computers connected to readers (directly or through a network), and appropriate software such as device drivers, filters, middleware, databases, and user applications.

![RFID Reader Diagram](Image)

**Figure 1**: An example of how RFID tag, reader, middleware and application operate.

Figure 1 shows that RFID reader retrieves information from tag and sends that information back to host computer via middleware. Middleware first needs to convert raw data retrieved by the reader to a meaningful data before sending them to an application layer assigned on a host computer.

### 2.1 Electronic Product Code (EPC)

EPC Class 1 Generation 2 is widely used in UHF range for communications at 860-960MHz. This passive tag is also referred to as EPC Gen-2 tag, where the standards have been created by EPC-Global (EPCGlobal 2006). The most common encoding scheme currently widely used includes: General Identifier (GID-96), Serialised Global Trade Item Number (SGTN-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 due to page limitation. The implementation and experiment will be determined by the impact of EPC Encoding Scheme and bulky movement of items, therefore at present, only one type of encoding is necessary.

<table>
<thead>
<tr>
<th>Encoding Scheme</th>
<th>Bit</th>
<th>Max. Decimal/Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>GID-96</td>
<td>96</td>
<td>0011 0101</td>
</tr>
<tr>
<td>Header</td>
<td>8</td>
<td>268,435,455</td>
</tr>
<tr>
<td>GMN*</td>
<td>28</td>
<td>16,777,215</td>
</tr>
<tr>
<td>Object Class</td>
<td>24</td>
<td>68,719,476,735</td>
</tr>
<tr>
<td>Serial Number</td>
<td>36</td>
<td></td>
</tr>
</tbody>
</table>

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 are completely determined by the Header value. Table 1 shows an example of GID-96 EPC generation 2 encoding scheme. Only Header is shown in binary, the rest are shown in decimal number.

### 2.2 Warehouse Distribution Justification

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, 10 pallets with 60 cases of crystal glasses each may be ready to leave the warehouse and deploy to different retailer. At this point, 10 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, and 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.

### 2.3 Errors in RFID Data Streams

Due to the characteristics of RFID streaming data, which does not contain much information and can be captured very fast, some of these data need to be filtered before being stored into the database. Such filtering is called ‘Edge cleaning/filtering’. 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, Garofalakis & Franklin 2006), (Curth and others 2005), (Fishkin et al. 2004); however, these techniques only filter specific kind of errors generated. A research on Noise and Duplication data filtering has been done very well previously; however, the Unreliable reads can be prevented only at some point. This depends on the deployment of readers, tags, and an environment.

### 2.4 Problems with “Missed reads” and their solutions

**Missed Reads** are very common in RFID applications and often happened in a situation of low-cost and low-power hardware, which leads to a frequently **Dropped Reading** referred to in other work (Derakhshan et al. 2007). Another cause of Missed reads is usually when multiple tags are to be detected by a reader but RF collisions occur causing RF signals to interfere with each other preventing the reader from identifying any tags. Dropped reading can be easily filtered using “Smoothing” technique proposed by Jeffery, Alonso, Franklin, Hong & Wixlom (2006), where missing data from specific time can be filled. However, preventing data resulting from RF collisions can be harder and in order to solve this problem, anti-collision can be performed at the edge to prevent two or more tags from responding to a reader at the same time.

### 3 Related Works

The various types of anti-collision methods for multi-access/tag collision can be reduced to two basic types: **probabilistic method** and **deterministic method**.

#### 3.1 Probabilistic Methods

In a probabilistic method, tags respond at randomly generated times. If a collision occurs, colliding tags will have to identify themselves again after waiting at a random period of time. This technique is faster than deterministic but suffers from **Tag starvation problem** where not all tags can be identified due to the random nature of chosen time.

The probabilistic methods are based on “Slotted-ALOHA” protocol (Quan et al. 2006), which introduces discrete time-slots for tags to be identified by reader at the specific time. To improve the performance, a “Frame Slotted ALOHA” (Shin et al. 2007) based anti-collision algorithm has been suggested, where each frame is formed of specific number
of slots that is used for the communication between
the readers and the tags. Each tag in the interroga-
tion zone arbitrarily selects a slot for transmitting the
tag’s information. However, the probabilistic method
can only be improved to a very high throughput rate
but they still cannot achieve 100 percent tag identifi-
cation.

3.2 Deterministic Methods
The deterministic method starts by asking for the
first number of the tag (Query Tree algorithm) un-
til 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.

Such deterministic 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 Arbi-
tration Algorithm”, the reader’s inquiries and the re-
sponses of the tags are stored and managed in the tag
memory, resulting in an equipment cost increase espe-
cially for RFID tags. In contrast, in the Memoryless
based algorithm, the responses of the tags are not de-
determined by the reader’s previous inquiries. The tags’
responses and the reader’s present inquiries are deter-
mined 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 an effective anti-collision technique for pas-
sive UHF tags. However, there are other improved
anti-collision methods based on Query Tree such as
an “Adaptive Query Splitting” (AQoS) proposed by
Myung & Lee (2006b), Myung & Lee (2006a) and a
“Hybrid Query Tree” (HQT) proposed by Ryu et al.
(2007). AQoS keeps information which is acquired dur-
ing 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 cy-
cles, while extra memory needed also increases as an
identification process gets longer, since each query in-
crease the prefixes by 2-bits instead of 1-bit. Accord-
ingly, 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).

3.3 Query Tree Based Protocols
The Query tree is a data structure for representing
prefixes which is sent by the reader in the 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 recog-
nises the tag. When more than one tag tries to re-

spond 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 which match the query. For identify-

ing tags that lead to the collision, the reader tries to
query with 1-bit longer prefixes in next loops. By ex-
tending 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 communi-
cation 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 of identification of tags is mostly af-
fected by the Collision cycles, Idle cycles, and simi-
larity of IDs. Therefore, reducing the number of Col-
lision cycles and eliminating Idle cycles, can improve
the identification ability of the reader.

In order to overcome shortcomings of existing
methods for collision resolution, we propose a “Uni-
fied Q-ary Tree Protocols” based on Memoryless QT.
We focus on analysing the impact of EPC Gen-2 en-
coding scheme and the fact of bulky items movement
within warehouse, therefore, we only considered static
tags where tags have no mobility. We investigated dif-
ferent combination of Q-ary trees, to reduce Collision
cycles and Idle cycles, which lead to shorter identifi-
cation time.

4 Unified Q-ary Tree Methodology
In order to reduce Collision cycles and Idle cycles,
and minimise total Identification delay, a “Unified Q-
ary Tree Protocols” or a combination of two Q-ary
trees are employed. The Unified Q-ary tree is a Mem-
oryless anti-collision protocols based on QT. This sec-
ction will describe the Q-ary tree, the scenarios where
EPC data are similar, and what can be improved by
combining two Q-ary trees together.

4.1 Q-ary Tree
Instead of using a Query Tree, which 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
more collision but at the same time, Idle cycles will
also increase. In the literature (Ryu et al. 2007), the
author used a 4-ary tree (HQT) to optimise the anti-
collision performance, which increases a lot of Idle
cycles despite reducing number of Collision cycles,
and requires extra memory and time to avoid them.
Therefore, the best way to solve the problem is to
produce both Collision cycles and Idle cycle as low as
possible in order to improve identification time.

4.2 Warehouse Distribution Scenarios
In this paper, we are examining a specific scenario
based on the assumption that items tend to move and
stay together through different locations especially in
a large warehouse. We are focusing on Crystal ware-
house scenario which can be classified into four dif-
ferent scenarios as follows:

**Scenario One**: Two collided tags are captured and
they have the same Encoding Scheme (Header), same
General Manager Number (Company Prefixes), same
Object Class, and different Serial Number. We can as-
sume that all items are from the same warehouse that
uses the same Encoding Scheme throughout the warehouse, and the warehouse also keeps different kind of product from different companies.

For example, the company warehouse produces different kind of crystal wine glasses, and all glasses that have the same sculpture will be packed in the same case and pallet. Therefore, crystal Red-wine glasses and crystal White-wine glasses should be packed in different case and pallet since they are different type of wine glasses. Within this scenario, each case of wine glasses will have a unique Serial Number attached to it with different Object Class for each pallet of White-wine or Red-wine.

![Figure 2: Crystal Warehouse Scenario](Image)

As for scenario one, by using the crystal warehouse example from Figure 2 a), when two collided tags are captured and they have the same Encoding Scheme, same General Manager Number, same Object Class, and unique Serial Number; we believe that both tags are each attached to two different cases of Red-wine.

**Scenario Two:** Two collided tags are captured and they have the same Encoding Scheme, same General Manager Number, different Object Class, and different Serial Number.

As for scenario two, by using the crystal warehouse example from Figure 2 b), when two collided tags are captured and they have the same Encoding Scheme, same General Manager Number, unique Object Class, and unique Serial Number; we believe that one tag is attached to Red-wine case, while the other tag is attached to White-wine case.

**Scenario Three:** Two collided tags are captured and they have the same Encoding Scheme, different General Manager Number, different Object Class, and different Serial Number.

As for scenario three, by using the crystal warehouse example from Figure 2 c), when two collided tags are captured and they have the same Encoding Scheme, unique General Manager Number, unique Object Class, and unique Serial Number; we believe that one tag is attached to Crystal plate case, while the other tag is attached to White-wine case. We can also make the assumption that there are two different companies producing separate crystal ware; and the wine glasses and plates are from different company but share the same warehouse since they are both crystal.

**Scenario Four:** Two collided tags are captured and they have the different Encoding Scheme, different General Manager Number, different Object Class, and different Serial Number. We can assume that all items are from different company that uses different encoding schemes. For example, two wine glasses with different sculpture, one made from crystal and one made from plastic, are allocated in the same warehouse. This scenario will not be discussed any further in this paper since we are only looking at a large warehouse distribution where most items move together as a group. Therefore, most items from the same type of manufacturing will stick together until deployed to smaller retailer.

### 4.3 Unified Q-ary Tree

Instead of using a plain Q-ary tree, which uses every 2-, 3-, or 4-bits of tag ID to split a tag set, we propose a “Unified Q-ary tree” or a combination of two Q-ary trees (12 combinations), which can reduce more collision and at the same time, *Idle cycles* can be minimised. For example, we can combine 4-ary tree with 8-ary tree and apply this *anti-collision* to 96-bits EPC; however, we need to configure the right partition so that 4-ary tree can be applied to the first half bits of EPC and 8-ary tree can be applied to the remaining bits. The remaining of this section will focus on two approaches: 1) a Naive approach, where Q-ary tree is non-unified and only a single Q-ary tree is used as an anti-collision; and 2) a Unified approach, where two Q-ary trees are combined as an anti-collision with 12 possible combinations.

**Naive Approach - Non-Unified Q-ary tree:** The Naive approach is a *non-unified Q-ary tree* that does not have a combination between two different Q-ary trees. There are four non-unified Q-ary trees investigated in this paper: binary QT, 4-ary tree, 8-ary tree, and 16-ary tree. Table 2 represents a *Number of bits* needed for each query using different Q-ary tree.

<table>
<thead>
<tr>
<th>No. of bits</th>
<th>Binary</th>
<th>4-ary</th>
<th>8-ary</th>
<th>16-ary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: The non-unified Q-ary Tree is where no combination between two Q-ary trees is necessary. The Table represents 4 non-unified Q-ary tree: binary, 4-ary, 8-ary, and 16-ary tree with 1, 2, 3, and 4 bits needed for each query respectively.

![Figure 3: Naive Q-ary Tree](Image)
Figure 3 shows the example of the Naive Q-ary tree. The figure shows four Naive Q-ary trees with 2 child nodes, 4-ary Tree with 8 child nodes, 8-ary Tree with 16 child nodes, and 16-ary Tree with 32 child nodes. The Highest Level Tree for each Naive Q-ary tree is calculated as shown in Table 4. The first four rows represent the Highest Level Tree for a binary QT, 4-ary tree, 8-ary tree, and 16-ary tree. Calculation for this Highest Level Tree will be explained in detail under heading ‘Highest Level Tree for each combination’.

Unified Approach - Unified Q-ary tree: The Unified approach is a unified Q-ary tree with 12 possible combinations. This approach will be applied on each collided tags EPC which will be split using every 1, 2, 3, or 4-bits of tag ID for the first few queries; and then at one point every 1, 2, 3, or 4-bits will be queried. With the fact that most items from warehouse have bulky movement, first few bits of EPC will be identical. For example, first 8-bits of EPC are ‘Header’, which will be the same for all items using the same encoding and they usually came from the same company and in the same pallet. These 8-bits of EPC can be bypassed faster using 4-ary tree instead of binary tree but by doing so, too many Idle cycles will be produced. By using 4-ary tree instead of binary tree, the Number of bits needed for each query also accumulates faster. Thus, we need to optimise the performance of “Unified Q-ary tree” by configuring the right separating point between the two Q-ary trees. The objective of Unified Q-ary tree is to minimise the Number of Bits used for querying all tags within an interrogation zone. Figure 4 shows the example of the Naive 4-ary Tree (4a) and the Unified 4-ary & 8-ary Tree (4b).

Figure 4: a) Naive 4-ary Tree, and b) Unified 4-ary & 8-ary Tree.

<table>
<thead>
<tr>
<th></th>
<th>binary</th>
<th>4-ary</th>
<th>8-ary</th>
<th>16-ary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F</td>
<td>S</td>
<td>F</td>
<td>S</td>
</tr>
<tr>
<td>binary</td>
<td>-</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4-ary</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>8-ary</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>16-ary</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 3: The Unified Q-ary Tree can be merged into 12 different combinations. Each EPC can be divided into two parts: First half (F) of EPC where all bits are identical, and Second half (S) of EPC where most bits are unique for all EPC in the reader zone. 1, 2, 3, and 4 represent the Number of bits queries each time for splitting tags when collision occurred.

For Highest Level Tree, Idle cycles, Collision cycles, and Number of bits estimation calculation purposes; let ‘F’ be the first half of EPC where bits are identical, and let ‘S’ be the second half of EPC where bits are unique. Table 3 shows possible combinations between four of the Q-ary trees; binary, 4-ary, 8-ary, and 16-ary.

Highest Level Tree for each combination: We now present a Highest Level Tree for all combinations of both Naive Q-ary tree and Unified Q-ary tree. Table 4 shows the Highest Level Tree where \( x_l \) is the maximum level tree of ‘x’ (first partition variable); \( y_l \) is a maximum level tree of ‘y’ (second partition variable); and \( T_l \) is a maximum level tree of \( x_l + y_l \). From the Table, for the first four Naive Q-ary tree, we can see that \( T_2 \) is equal to 96 divided by the number of bit/bits needed for each query. For example; for the Naive 4-ary tree, \( T_2 \) is 48 which is 96 divided by 2. In the case of a Naive Q-ary tree, \( F \), \( S \), and variable ‘x’ and ‘y’ does not play any major role. Sample calculation of 4-ary tree for \( x_1, y_1 \), and \( T_1 \) where \( x = 36, y = 60 \):

\[
\log_2(2^x) + \log_2(2^y) = \log_2(2^{36}) + \log_4(2^{60})
\]

\[
\log_4(2^{36}) + \log_4(2^{60}) = \log_{10}(2^{36}) + \log_{10}(2^{60})
\]

\[
\log_{10}(2^{36}) = 4 + 44 = 48
\]

OR

\[
\log_{10}(2^{60}) = 48
\]

Therefore, \( x_1 = 4, y_1 = 44, \) and \( T_1 = 48 \)

For the remaining 12 combinations of Unified Q-ary tree in Table 4, we can see that \( T_1 \) is equal to the sum of \( x_1 \) and \( y_1 \), where the sum of \( x \) and \( y \) equal to 96. For example: for 4-ary tree combining with 8-ary tree (\( F = 4, S = 8 \)), \( T_1 = 34 \) which is 4 plus 30 (\( x_1 + y_1 \)). In the case of a Unified Q-ary tree, ‘F’; ‘S’, and variable ‘x’ and ‘y’, play any major role.

Sample calculation of a Unified 4-ary & 8-ary Tree, for \( x_1, y_1 \), and \( T_1 \) where \( F = 4, S = 8, x = 38, y = 88 \):

\[
\log_2(2^x) + \log_2(2^y) = \log_4(2^8) + \log_8(2^{88})
\]

\[
\log_4(2^8) + \log_8(2^{88}) = \log_{10}(8) + \log_{10}(8)
\]

\[
\log_{10}(8) + \log_{10}(8) = 4 + 30 = 34
\]

Therefore, \( x_1 = 4, y_1 = 30, \) and \( T_1 = 34 \). Note that the outcome in decimal is rounded up to the nearest whole number since level of tree cannot be fractioned.
Table 4: Each combination 1 to 16 are applied with three different variables of ‘x’ and ‘y’, where $x_l =$ number of highest tree level for ‘F’; and $y_l =$ number of highest tree level for ‘S’. $T_l$, which is the summation of $x_l$ and $y_l,$ also represents the number of queries needed for the worst case of identification where two collided tags have all identical bits except for the last bit (bit 96).

Sample comparison between a performance of Naive approach versus Unified approach: We are now initiating a comparison between the performance of Naive binary tree, Naive 4-ary tree, Unified binary & 4-ary tree, and Unified 4-ary & binary.

Figure 5 shows a comparison between Unified approach (binary & 4-ary, 4-ary & binary) and Naive approach (4-ary, binary) on the five EPC data. We can see that the Naive 4-ary tree have the shortest level of tree, however, by examining Table 6, 4-ary tree does not have the lowest Total number of bits. This proves that levels of tree have an impact on the Total number of bits and Overall cycles, but does not necessarily result in the best performance of tree.

In order to calculate a Total number of bits required for the whole identification process, information on Number of Child Nodes (NCN) for each level of tree and Number of Bits per Query (NBQ) for that specific level, is needed. Number of bits per Level (NBL) can be calculate as follows:

$$NBL = NCN \times NBQ$$

<table>
<thead>
<tr>
<th>Level</th>
<th>2</th>
<th>2, 4</th>
<th>4, 2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>6</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>8</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>10</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>12</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>14</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>16</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>9</td>
<td>18</td>
<td>18</td>
<td>72</td>
<td>72</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>20</td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>11</td>
<td>22</td>
<td>22</td>
<td>88</td>
<td>88</td>
</tr>
<tr>
<td>12</td>
<td>24</td>
<td>24</td>
<td>96</td>
<td>96</td>
</tr>
</tbody>
</table>

| NBLs  | 176 | 160 | 184 | 108 |

Table 5: Calculation of Total memory bits required for two Naive Q-ary trees and two Unified Q-ary trees. NBL shows the Total number of bits required for the specific Naive/Unified Q-ary tree.
for the two approaches is different. The same goes with Naive 4-ary tree and Unified binary & 4-ary tree where Overall cycles are the same but have a different Total number of bits. As for the impact of EPC data, we can see that when EPC IDs are identical (bit 1-8), a binary tree works better since it uses less Number of bits than 4-ary tree. This difference cannot be seen without calculating a proper Total number of bits since for the ‘F’, both binary and 4-ary tree have the same number of Collision cycles and Idle cycles. However, for each of these cycles, different Number of bits are used for querying, thus 4-ary tree uses more bits than binary tree. For ‘S’, 4-ary tree uses less Number of bits than binary tree since the number of Collision cycles happened more in binary tree. Although a 4-ary tree produces more Idle cycles than binary tree in the second half, it still produces less total number of Collision cycles and Idle cycles. We can now conclude that for identical bits of EPC, lower level tree can perform better than higher level one and for unique bits of EPC, a higher level tree is more suitable.

Table 6: Results of Collision cycles, Idle cycles, Success cycles, Overall cycles, Number of bits, and Total number of bits for 5 tags identification using Naive approach and Unified approach.

<table>
<thead>
<tr>
<th>Combination</th>
<th>2</th>
<th>2, 4</th>
<th>4, 2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collision Cycles (F)</td>
<td>8</td>
<td>8</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Collision Cycles (S)</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Total Collision Cycles</td>
<td>12</td>
<td>9</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>Idle Cycles (F)</td>
<td>8</td>
<td>8</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Idle Cycles (S)</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Total Idle Cycles</td>
<td>9</td>
<td>10</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>Success Cycles (F)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Success Cycles (S)</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Total Success Cycles</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Overall Cycles</td>
<td>26</td>
<td>24</td>
<td>26</td>
<td>24</td>
</tr>
<tr>
<td>Number of bits (F)</td>
<td>72</td>
<td>72</td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>Number of bits (S)</td>
<td>104</td>
<td>88</td>
<td>104</td>
<td>88</td>
</tr>
<tr>
<td>Total number of bits</td>
<td>176</td>
<td>160</td>
<td>184</td>
<td>168</td>
</tr>
</tbody>
</table>

Number of tags in the interrogation zone also has an impact on Collision cycles and Idle cycles. When number of tags increases, number of collision also increases, thus higher level trees are more suitable since they provide more unique queries at each level of tree. In this paper, we will only test the combination of binary and 4-ary tree and analyse a performance to see how much improvement can be formed. In the future, further study will be done with higher level trees and a larger number of tags will be used for an experiment.

5 Experiment and Results

In this section, we present experiment conducted to evaluate the performance of “Unified Q-ary tree”. As a result, analysis discussions are evaluate between the performance of Naive approach versus Unified approach.

5.1 Environment

To study the proposed “Unified Q-ary Tree” and compare with the performance of Naive approach, experiments are performed according to a Crystal warehouse scenario. The experiment is set 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, 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 Pentium 4 CPU with 2.80GHz processor and 2GB RAM is used for testing. A Microsoft Window XP professional with Service Pack 3 is installed on the computer. Algorithms are implemented using Java JCreator.

Figure 6: Level-Packaging.

Figure 6 displays a level-packaging, where each case contains 6 glasses and each pallet contains 27 cases. For our experiment, 3 of these pallets will be visible to the reader attached to the dock’s door next to the conveyor belt.

5.2 Data Set

Results presented are related to the first scenario mentioned earlier in Section 4.2. We performed ten runs on the data set and present the average results. For the data set, there are 81 tags/EPC used in the experiment. Each tag contains 60 identical bits for ‘F’ and 36 unique bits for ‘S’. Each pallet contains 27 tags (See Figure 6) and 3 pallets are assumed to be visible to the reader each time. We applied the Naive approach, binary and 4-ary tree, to the data set with no partition. On the other hand, we applied Unified approach to the data set using ‘x’ = 60 and ‘y’ = 36 based on the nature of scenario one where the first 60-bits are identical.

5.3 Result, Analysis and Discussion

Based on the experiment simulation, Figure 7 shows the average results, from ten runs, on all four combinations: Naive binary, Unified binary & 4-ary, Unified 4-ary & binary, and Naive 4-ary tree. From Figure 7, we can see that the Naive 4-ary tree produced the most Idle cycles while the Naive binary tree produced the least. In contrast, the Naive binary tree produced the most Collision cycles while the Naive 4-ary tree produced the least. Both Naive binary and Unified 4-ary & binary have the same total number of cycles, which corroborate our methodology. In addition, the total number of cycles for Naive 4-ary tree and Unified binary & 4-ary tree are also equal. The total number of cycles can, at one point, clarify the performance of all four methods. We notice that both Naive 4-ary tree and Unified binary & 4-ary tree have less total cycles than binary and 4-ary & binary. This means that these first two methods will use less Number of bits in querying for all 81 tags than the other two. However, without looking into the actual results of
Figure 7: Results of two Naive approaches (Binary, 4-ary) and two Unified approaches (Binary & 4-ary, 4-ary & Binary) for number of Idle cycles, Collision cycles, Success cycles, and Overall cycles.

Number of bits, we still cannot conclude which of the two methods will achieve less identification time for querying.

Based on Figure 7 we are now aware of Success cycles of all four methods are all equal to 81, which means that all tags in the interrogation zone are 100 percent identified. We can also see that all 81 tags were recognised at the later stages, where all bits (bit no. 61-82) are unique. As for identical bits of Idle cycles and Collision cycles, the sum of Idle cycles and Collision cycles have an outcome of 120 cycles, which means that both methods of binary or 4-ary tree have no impact in the sense of cycles count but as mentioned earlier, we need to calculate the actual Number of bits in order to clarify the difference of the performance of both methods. The next Figure (Figure 8) shows the Number of Bits for Idle cycles, Collision cycles, Success cycles, and Overall cycles, of each method.

Figure 8 shows all the actual bits for all queries that occur during tags identification. We now notice that the Unified binary & 4-ary tree have the lowest Number of bits queried for entire identification process. This verify our theory that by using a lower level tree for identical bits of EPC and higher level tree for unique bits of EPC, Number of bits queried can be minimised and identification process can be accelerated. There is not much difference in results but we can assume that as the number of tags in an interrogation zone increases, and other combinations of Q-ary tree are used, we will be able to see more differences in the outcome.

For identical bits of EPC, there is a slight difference between the Number of bits queried by the four methods. While Figure 7 shows that there is no difference between total number of cycles for identical bits for all four methods, we can see clearly that Total number of bits is different for each case in Figure 8. This is because each query inquired each time issues different Number of bits. For example, 4-ary tree issues 2 extra bits from the last query (from the parent node), while binary tree only append 1 extra bit to the last query. The Unified binary & 4-ary tree performed the best overall and required 60 bits less than the Naive 4-ary tree, and 924 bits less than the Naive binary tree. In contrast, the Unified 4-ary & binary tree performed the worst out of all four methods. This is because a higher level tree was used at the earlier stages of identification where all bits are identical. This means that more than 75 percent of the queries were Idle cycles which are waste of resources (See Figure 8 - 4-ary & Binary: Idle cycles/Collision cycles = Ratio of 3:1 or 75%/25%). By using binary tree instead of 4-ary tree for identical bits, 60 bits of queries were reduced (3720 minus 3660).

For unique bits of EPC, Number of bits query rises rapidly compared to identical bits. Figure 8 shows that, by using 4-ary tree for unique bits of EPC, number of queries and bits were slightly reduced (see Total bits queried for unique bits). The performance of each method on unique bits of EPC will be specified in detail in Figure 9.

Figure 9 shows the number of Idle cycles, Collision cycles, Success cycles and Overall cycles produced in each query loop. We can see that at bit 63-64 to bit 65-66, the difference between Overall cycles of binary
and 4-ary tree grows. After bit 67-68, there is not much difference between the two. From bit 73-74 to bit 79-80, there are no Success cycles for both methods; therefore, there are no differences for their Overall cycles. We can now assume that at bit 61-62 to bit 71-72, the EPC are similar but not identical, which results in the unstable change in number of Overall cycles. On the other hand, at bit 73-74 to bit 79-80, we can assume that all bits become identical again resulting in no change in Overall cycles. The number of collided tags at bit 73-74 to bit 79-80 are exactly two since the ratio of Idle cycles to Collision cycles is 1:1 for binary tree and 3:1 for 4-ary tree respectively. At last, all tags were identified at bit 81-82 resulting in the same number of Overall cycles for both binary tree and 4-ary tree.

We can now summarise that by using a lower level tree for identical bits of EPC, and by using a higher level tree for unique bits of EPC, the Total number of bits for querying can be decreased. By reducing the Total number of bits, identification time for each round can be minimised.

6 Conclusion

In this study, we identified the significance of RFID tags anti-collisions and developed efficient method to minimise the use of memory bits; and at the same time to ensure that all RFID tags are 100 percent identified, which is essential to provide correct RFID data before they can be further processed, transformed, and integrated for RFID-enabled applications. We proposed a “Unified Q-ary tree”, which combines two Q-ary trees together in order to reduce Collision cycles and Idle cycles; and to minimise identification time. In the experimental evaluation, we showed that our method performs better, ensures 100 percent tags identification and reduces Overall cycles, and Total number of bits queried, which leads to faster identification time.

As of future work, we intend to test other combinations of Q-ary trees. Different pallet sizes will also be inspected to determine the impact of packaging and density on quality of captures data. Different number of tags will be tested for the impact of number of tags within one interrogator zone. Also, different Encoding Scheme other than GID-96 will be observed.

Acknowledgements

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

References


Figure 9: Performance Analysis of Binary tree vs. 4-ary tree on unique bits of EPC, Bit 61 - 68, until all tags are identified. Results of Idle cycles, Collision cycles, Success cycles, and Overall cycles are displayed.


