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

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.

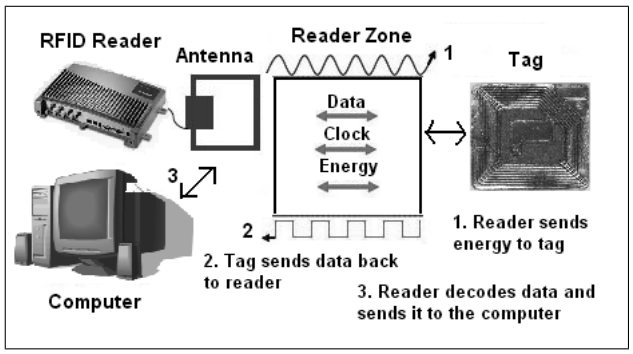


Figure 1: An example of how RFID tag, reader, middleware and application operate.

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 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 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.

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 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), (Carbunar et al. 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 & Widom (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 interrogation 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 identification.

3.2 Deterministic Methods

The *deterministic* 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.

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 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, in 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 an 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 (2006*b*), Myung & Lee (2006*a*) 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, since each query increase 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).

3.3 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 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 which match 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 of identification of tags is mostly affected by the *Collision cycles*, *Idle cycles*, and *similarity of IDs*. Therefore, reducing the number of Collision 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 "Unified Q-ary Tree Protocols" based on *Memoryless QT*. We focus on analysing the impact of EPC Gen-2 encoding scheme and the fact of *bulky items movement* within warehouse, therefore, we only considered static tags where tags have no mobility. We investigated different combination of *Q-ary trees*, to reduce *Collision cycles* and *Idle cycles*, which lead to shorter identification 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 *Memoryless anti-collision protocols* based on QT. This section 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 warehouse scenario which can be classified into four different 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 assume 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.

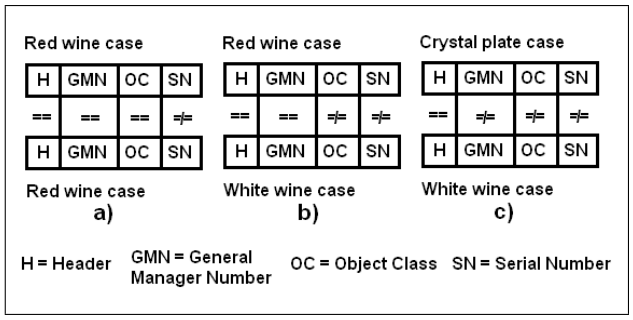


Figure 2: Crystal Warehouse Scenario: a) Two cases of Red-wine have the same Object Class but different Serial Number, b) White-wine case and Red-wine case have different Object Class and Serial Number, and c) White-wine case and Crystal plate case have different General Manager Number, Object Class, and Serial Number.

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.

	Binary	**4-ary**	**8-ary**	**16-ary**
No. of bits	1	2	3	4

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.

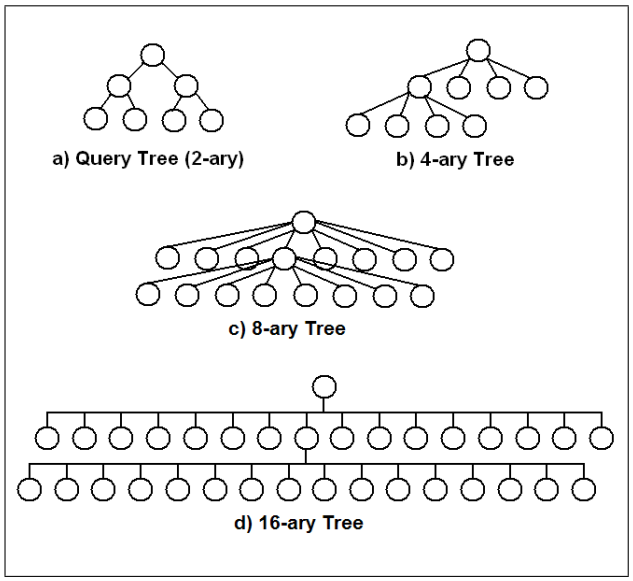


Figure 3: Naive Q-ary Tree: a) Query Tree (2-ary), b) 4-ary Tree, c) 8-ary Tree, and d) 16-ary Tree.

Figure 3 shows the example of the *Naive Q-ary tree*. There are four Naive Q-ary trees shown in the figure: a) Query/Binary/2-ary Tree with 2 child nodes, b) 4-ary Tree with 4 child nodes, c) 8-ary Tree with 8 child nodes, and d) 16-ary Tree with 16 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).

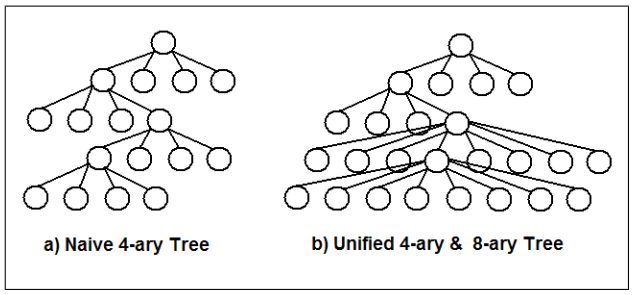


Figure 4: a) Naive 4-ary Tree, and b) Unified 4-ary & 8-ary Tree.

	binary		**4-ary**		**8-ary**		**16-ary**	
	F	**S**	**F**	**S**	**F**	**S**	**F**	**S**
binary	**-**		2	1	3	1	4	1
4-ary	1	2	**-**		3	2	4	2
8-ary	1	3	2	3	**-**		4	3
16-ary	1	4	2	4	3	4	**-**	

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_l is equal to 96 divided by the number of bit/bits needed for each query. For example: for the Naive 4-ary tree, $T_l = 48$ 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_l, y_l, and T_l where x = 36, y = 60:

$$\log_F(2^x) + \log_S(2^y) = \log_4(2^{36}) + \log_4(2^{60})$$

$$\log_4(2^{36}) + \log_4(2^{60}) = \frac{\log_{10}(2^{36})}{\log_{10}(4)} + \frac{\log_{10}(2^{60})}{\log_{10}(4)}$$

$$\log_4(2^{36}) + \log_4(2^{60}) = 4 + 44 = 48$$

OR

$$\log_4(2^{96}) = \frac{\log_{10}(2^{96})}{\log_{10}(4)} = 48$$

Therefore, $x_l = 4$, $y_l = 44$, and $T_l = 48$

For the remaining 12 combinations of *Unified Q-ary tree* in Table 4, we can see that T_l is equal to the sum of x_l and y_l, 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), $T_l = 34$ which is 4 plus 30 (x_l + y_l). 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_l, y_l, and T_l where F = 4, S = 8, x = 8, y = 88:

$$\log_F(2^x) + \log_S(2^y) = \log_4(2^8) + \log_8(2^{88})$$

$$\log_4(2^8) + \log_8(2^{88}) = \frac{\log_{10}(2^8)}{\log_{10}(4)} + \frac{\log_{10}(2^{88})}{\log_{10}(8)}$$

$$\log_4(2^8) + \log_8(2^{88}) = 4 + 30 = 34$$

Therefore, $x_l = 4$, $y_l = 30$, and $T_l = 34$. Note that the outcome in decimal is rounded up to the nearest whole number since level of tree cannot be fractioned.

From Table 4, by combining two *Q-ary trees*, for example binary tree and 4-ary tree, T_l for this combination is different to T_l of the Naive 4-ary tree and T_l of the Naive binary tree. The question is, would the difference between levels of tree have any impact on the total number of *Idle cycles* and number of *Collision cycles* for all tags identification within one interrogation zone? *Identification delay* can be reduced by reducing number of queries, which is a summation of Idle cycles, Collision cycles and Success cycles. *Success cycles* will always be the same using any combination techniques, which leaves Idle cycles and Collision cycles. Furthermore, *Number of bits* per query need to be taken into consideration since different Q-ary tree uses different Number of bits for each query. Thus, the following paragraph shows the difference between performances of the *Naive* approach versus *Unified* approach, and the impact from number of Idle cycles, number of Collision cycles, and total Number of bits.

$\log_F(2^x)$ / $\log_S(2^y)$		x = 8, y = 88			x = 36, y = 60			x = 60, y = 36		
F	**S**	x_l	y_l	T_l	x_l	y_l	T_l	x_l	y_l	T_l
2	**2**	8	88	96	36	60	96	60	36	96
4	**4**	4	44	48	18	30	48	30	18	48
8	**8**	2	30	32	12	20	32	20	12	32
16	**16**	2	22	24	9	15	24	15	9	24
2	**4**	8	44	52	36	30	66	60	18	78
4	**2**	4	88	92	18	60	78	30	36	66
2	**8**	8	30	38	36	20	56	60	12	72
8	**2**	3	88	91	12	60	72	20	36	56
4	**8**	4	30	34	18	20	38	30	12	42
8	**4**	3	44	47	12	30	42	20	18	38
2	**16**	8	22	30	36	15	51	60	9	69
16	**2**	2	88	90	9	60	69	15	36	51
4	**16**	4	22	26	18	15	33	30	9	39
16	**4**	2	44	46	9	30	39	15	18	33
8	**16**	3	22	25	12	15	27	20	9	29
16	**8**	4	30	34	9	20	29	15	12	27

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:

$$NBL = NCN \times NBQ$$

Level	**2**	**2, 4**	**4, 2**	**4**
1	2	2	8	8
2	4	4	16	16
3	6	6	24	24
4	8	8	32	32
5	10	10	18	40
6	12	12	40	48
7	14	14	22	-
8	16	16	24	-
9	18	40	-	-
10	40	48	-	-
11	22	-	-	-
12	24	-	-	-
NBLs	**176**	**160**	**184**	**168**

Table 5: Calculation of Total memory bits required for two Naive Q-ary trees and two Unified Q-ary trees. NBLs shows the Total number of bits required for the specific Naive/Unified Q-ary tree.

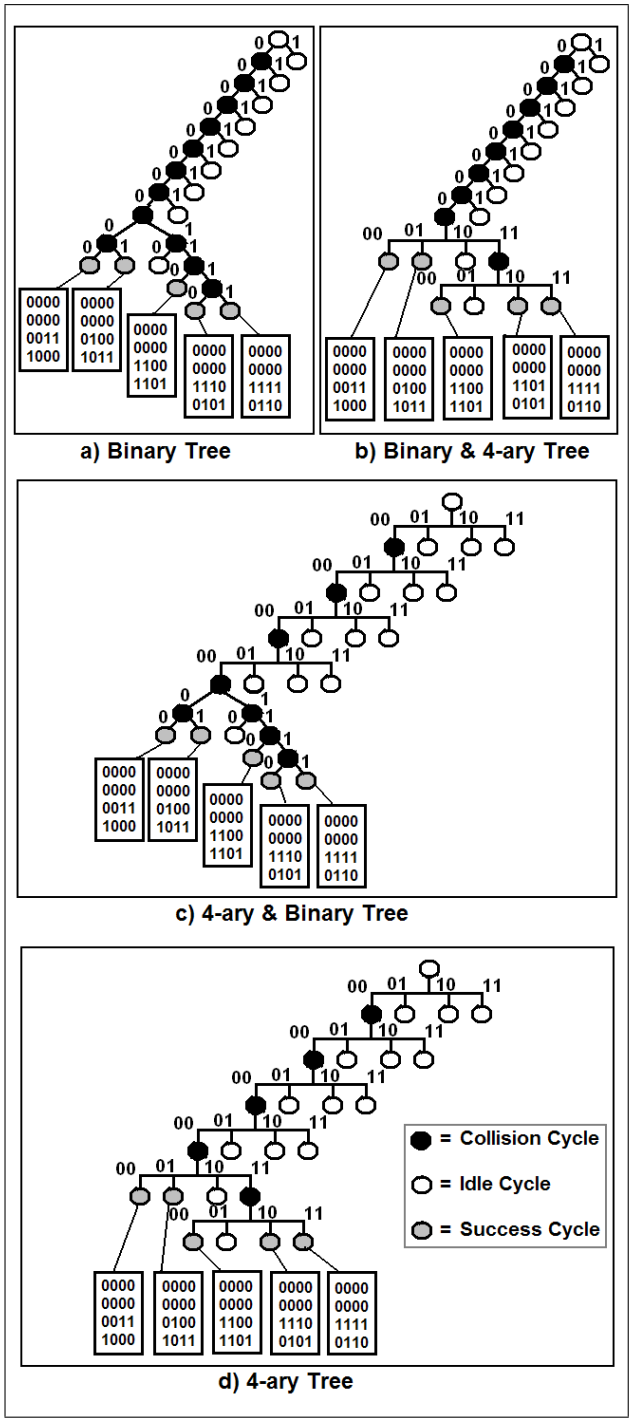


Figure 5: Identification processes of: a) Naive Binary tree, b) Unified Binary & 4-ary tree, c) Unified 4-ary & Binary tree, and d) Naive 4-ary tree.

After calculating the NBL for each level of tree, the *Total number of bits* (NBLs) required can be found by doing the summation of all NBL. For example, in Figure 5 a) it can be seen that the tree has 12 levels where all levels except Level 10 have 2 child nodes each. For each Level, NBQ increased by 1-bit since this is a Naive binary tree. Thus, NBL for each level are (NCN x NBQ): 2 or (2x1), 4 or (2x2), 6 or (2x3), 8 or (2x4), 10 or (2x5), 12 or (2x6), 14 or (2x7), 16 or (2x8), 18 or (2x9), 40 or (4x10), 22 or (2x11), 24 or (2x12) respectively. After adding all NBL together, the NBLs of 176-bits is as shown in Table 5.

Table 6 shows that both Naive binary tree and Unified 4-ary & binary tree, has the same number of *Overall cycles*. However, the *Total number of bits*

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.

Combination	**2**	**2, 4**	**4, 2**	**4**
Collision Cycles (F)	8	8	4	4
Collision Cycles (S)	4	1	4	1
Total Collision Cycles	**12**	**9**	**8**	**5**
Idle Cycles (F)	8	8	12	12
Idle Cycles (S)	1	2	1	2
Total Idle Cycles	**9**	**10**	**13**	**14**
Success Cycles (F)	0	0	0	0
Success Cycles (S)	5	5	5	5
Total Success Cycles	**5**	**5**	**5**	**5**
Overall Cycles	**26**	**24**	**26**	**24**
Number of bits (F)	72	72	80	80
Number of bits (S)	104	88	104	88
Total number of bits	**176**	**160**	**184**	**168**

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.

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.

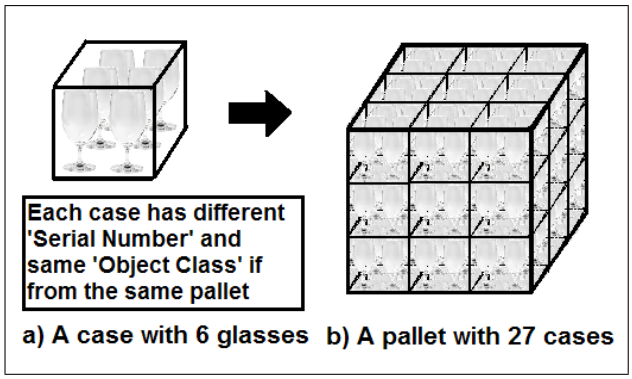


Figure 6: Level-Packaging.

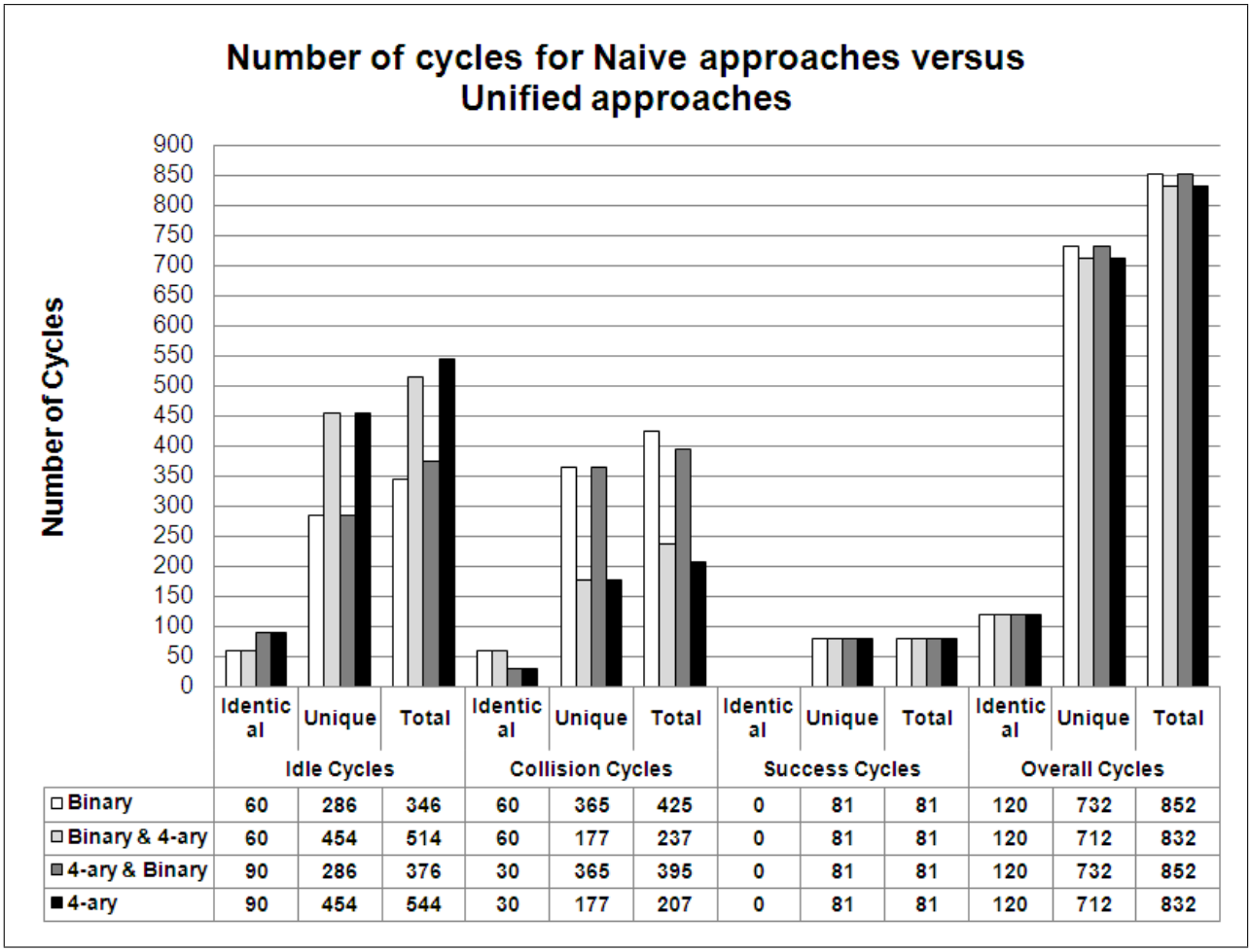
Figure 6 displays a level-packaging, where each case contains 6 glasses and each pallet contains 27 cases. For our experiment, 3 of these pallets will be visible to the reader attached to the dock's door next to the conveyor belt.

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



	Idle Cycles			Collision Cycles			Success Cycles			Overall Cycles		
	Identical	Unique	Total	Identical	Unique	Total	Identical	Unique	Total	Identical	Unique	Total
Binary	60	286	346	60	365	425	0	81	81	120	732	852
Binary & 4-ary	60	454	514	60	177	237	0	81	81	120	712	832
4-ary & Binary	90	286	376	30	365	395	0	81	81	120	732	852
4-ary	90	454	544	30	177	207	0	81	81	120	712	832

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.

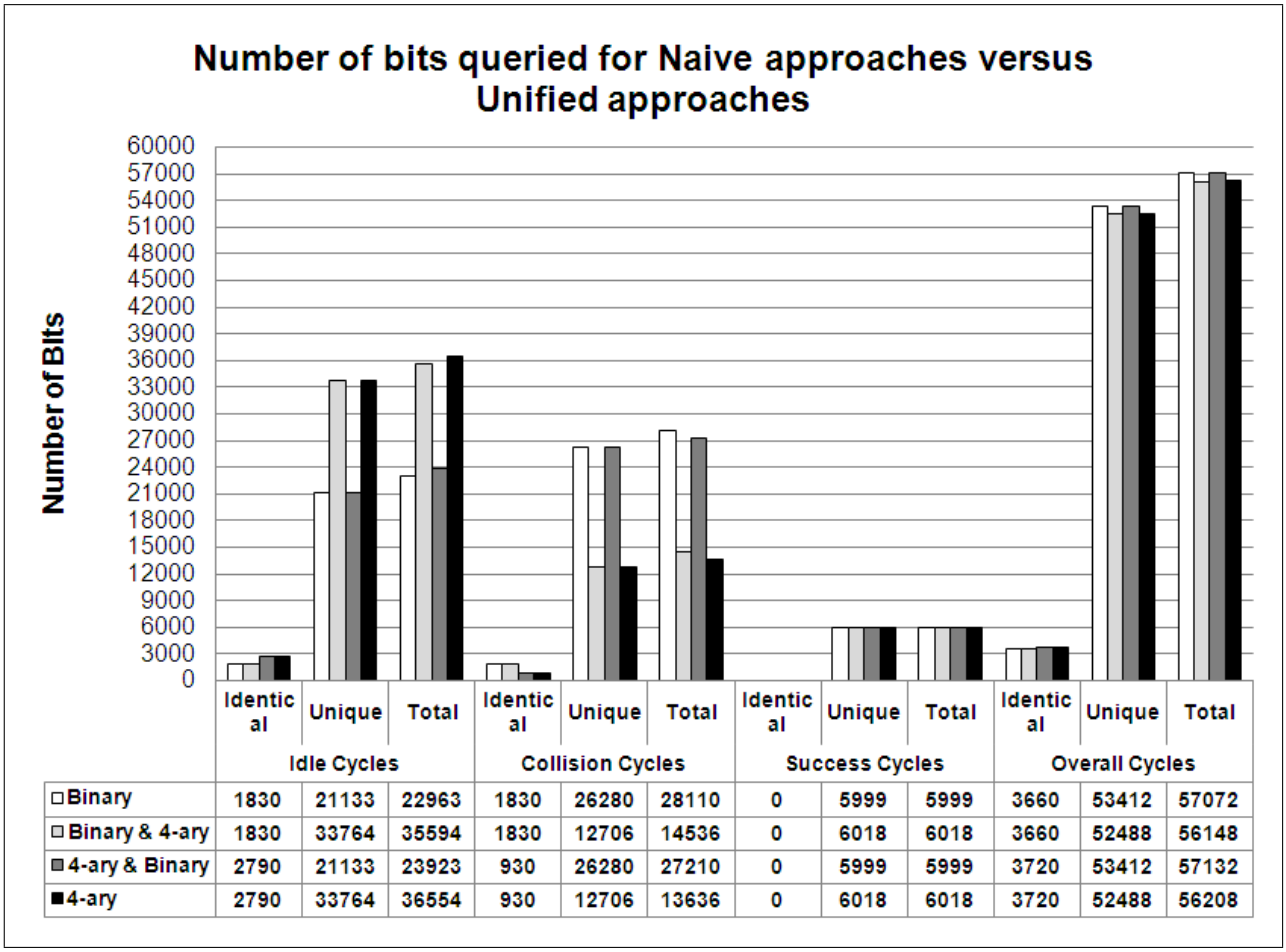
Based on Figure 7 we are now aware of *Success cycles* of all four methods are all equal to 81, which means that all tags in the interrogation zone are 100 percent identified. We can also see that all 81 tags were recognised at the later stages, where all bits (bit no. 61-82) are unique. As for *identical 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



	Idle Cycles			Collision Cycles			Success Cycles			Overall Cycles		
	Identical	Unique	Total	Identical	Unique	Total	Identical	Unique	Total	Identical	Unique	Total
Binary	1830	21133	22963	1830	26280	28110	0	5999	5999	3660	53412	57072
Binary & 4-ary	1830	33764	35594	1830	12706	14536	0	6018	6018	3660	52488	56148
4-ary & Binary	2790	21133	23923	930	26280	27210	0	5999	5999	3720	53412	57132
4-ary	2790	33764	36554	930	12706	13636	0	6018	6018	3720	52488	56208

Figure 8: Results of two Naive approaches (Binary, 4-ary) and two Unified approaches (Binary & 4-ary, 4-ary & Binary) for total number of bit queried for Idle cycles, Collision cycles, Success cycles, and Overall cycles.

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

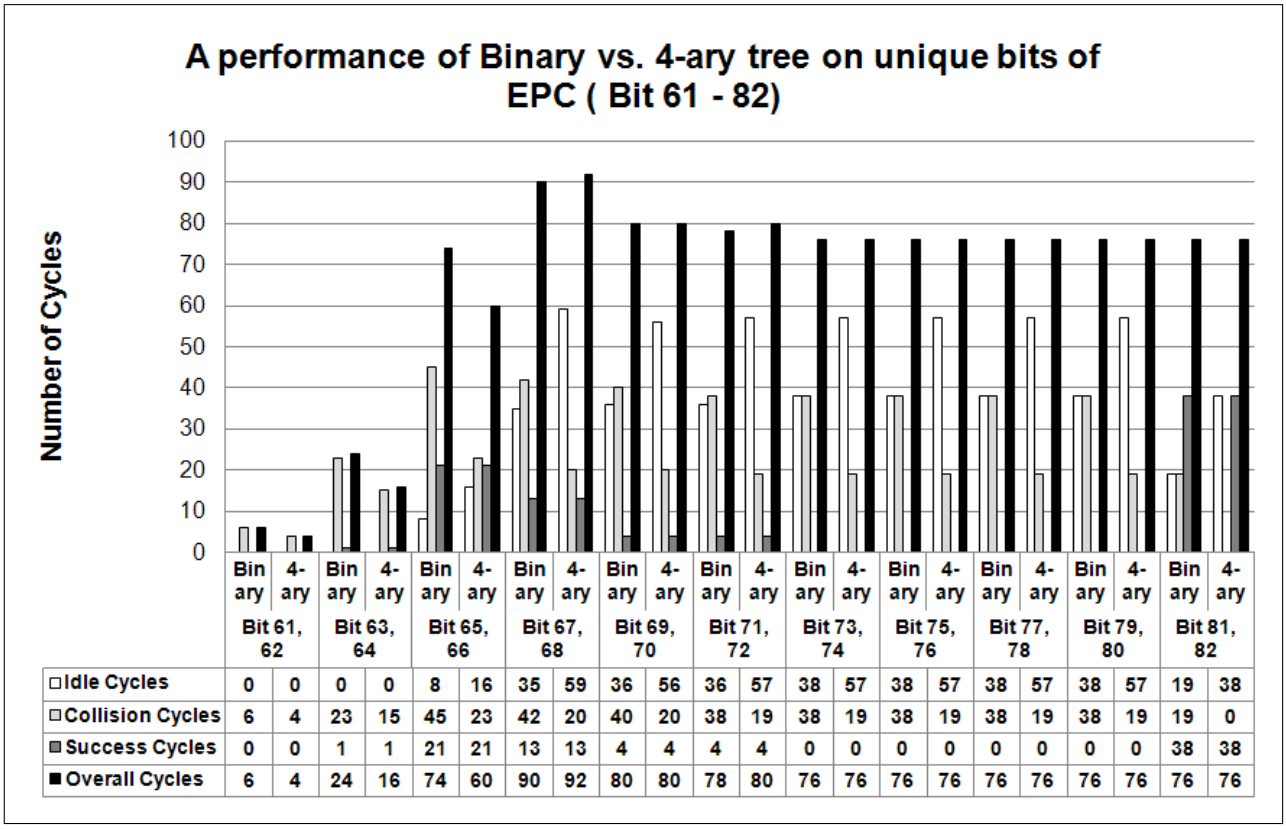
This research is partly sponsored by ARC (Australian Research Council) grant no DP0557303.

References

Bai, Y., Wang, F. & Liu, P. (2006), Efficiently Filtering RFID Data Streams, *in* 'CleanDB Workshop', pp. 50–57.

Brown, M., Patadia, S. & Dua, S. (2007), *Mike Meyers'Certification Passport: CompTIA RFID+ Certification*, McGraw-Hill.

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.



	Binary	4-ary	Binary	4-ary	Binary	4-ary	Binary	4-ary	Binary	4-ary	Binary	4-ary	Binary	4-ary	Binary	4-ary	Binary	4-ary	Binary	4-ary	Binary	4-ary
	Bit 61, 62		Bit 63, 64		Bit 65, 66		Bit 67, 68		Bit 69, 70		Bit 71, 72		Bit 73, 74		Bit 75, 76		Bit 77, 78		Bit 79, 80		Bit 81, 82	
Idle Cycles	0	0	0	0	8	16	35	59	36	56	36	57	38	57	38	57	38	57	38	57	19	38
Collision Cycles	6	4	23	15	45	23	42	20	40	20	38	19	38	19	38	19	38	19	38	19	19	0
Success Cycles	0	0	1	1	21	21	13	13	4	4	4	4	0	0	0	0	0	0	0	0	38	38
Overall Cycles	6	4	24	16	74	60	90	92	80	80	78	80	76	76	76	76	76	76	76	76	76	76

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.

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.

Derakhshan, R., Orlowska, M. E. & Li, X. (2007), RFID Data Management: Challenges and Opportunities, *in* 'IEEE International Conference on RFID 2007', Texas, USA, pp. 175–182.

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.

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, 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.

Landt, J. (2001), Shrouds of Time The history of RFID, An Aim Publication, Pittsburg, PA.

Myung, J. & Lee, W. (2006*a*), 'Adaptive binary splitting: a RFID tag collision arbitration protocol for tag identification', *Mob. Netw. Appl.* **11**(5), 711–722.

Myung, J. & Lee, W. (2006*b*), 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.

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, 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, pp. 693–700.