Improving RFID System To Read Tags Efficiently

Dissertation

submitted in partial fulfillment of the requirements
for the degree of

Master of Technology

by

Naval Bhandari

(Roll Number 04329023)

under the guidance of

Prof. Anirudha Sahoo & **Prof. Sridhar Iyer**

Kanwal Rekhi School of Information Technology
Indian Institute of Technology Bombay

July, 2006

Dissertation Approval Sheet

This is to certify that the dissertation entitled

Improving RFID System To Read Tags Efficiently

by

Naval Bhandari

(Roll no. 04329023)

is approved for the degree of **Master of Technology**.

Prof. Anirudha Sahoo (Guide)	Prof. Sridhar Iyer (Co-Guide)

Prof. Om Damani
(Internal Examiner)

Prof. Varsha Apte
(External Examiner)

Prof. Girish P. Saraph
(Chairperson)

Date: _____

Place: _____

INDIAN INSTITUTE OF TECHNOLOGY BOMBAY
CERTIFICATE OF COURSE WORK

This is to certify that **Mr. Naval Bhandari** was admitted to the candidacy of the M.Tech. Degree in July 2004, and has successfully completed all the courses required for the M.Tech. Programme. The details of the course work done are given below.

Sr.No.	Course No.	Course Name	Credits
		Semester 1 (Jul – Nov 2004)	
1.	IT605	Computer Networks	6
2.	IT619	IT Foundation Laboratory	10
3.	IT623	Foundation Course of IT - Part II	6
4.	IT653	Network Security	6
5.	IT694	Seminar	4
6.	HSS699	Communication and Presentation Skills (P/NP)	4
		Semester 2 (Jan – Apr 2005)	
7.	IT608	Data Warehousing and Data Mining	6
8.	IT610	QOS In Networks	6
9.	IT630	Principles and Practice of Distributed Computing	6
10.	IT680	Systems Laboratory	6
11.	EE701	Introduction to MEMS (Institute Elective)	6
		Semester 3 (Jul – Nov 2005)	
12.	CS601	Algorithms and Complexity (Audit)	6
13.	CS681	Performance Analysis of Computer Systems and Networks	6
		M.Tech. Project	
14.	IT696	M.Tech. Project Stage - I (Jul 2005)	18
15.	IT697	M.Tech. Project Stage - II (Jan 2006)	30
16.	IT698	M.Tech. Project Stage - III (Jul 2006)	42

I.I.T. Bombay — Dy. Registrar(Academic)

Dated:

Abstract

Radio Frequency IDentification (RFID) is slated to become a standard for tagging various products. As more and more products become RFID enabled, fast tag identification mechanisms will become important. Various tag identification (or anti-collision) algorithms have been proposed for RFID systems. This work focuses on methods to improve tag read efficiency in RFID Systems. We propose an Intelligent Query Tree (IQT) Protocol for tag identification that exploits specific prefix patterns in the tags and makes the identification process more efficient. IQT is a memoryless protocol that identifies RFID tags more efficiently in scenarios where tag IDs have some common prefix (e.g., common vendor ID or product ID). IQT is most suitable for readers deployed in exclusive showrooms, and shipment points of big malls, where the products may come from same manufacturers and may have same product type. We provide the worst case complexity analysis of IQT and show the performance improvement of this protocol over traditional Query Tree protocol in different scenarios. For other cases we show the improvement using simulation results.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Radio Frequency IDentification (RFID) is a means of identifying objects using radio frequency transmission[1, 2, 3]. In an RFID system, very small RF chips (called tags or transponders) communicate wirelessly with readers (interrogators) within a certain range. Tags can either be active (powered by battery) or passive (powered by the reader field).

1.1 RFID Components

- RFID Reader (Interrogator) : This unit is connected to the backend and it powers the tag antenna in order to identify it.
- RFID Tag (Transponder) : A programmed memory device, which contains permanent identification number and may also contain other re-writable information.
- RFID Antenna : A coil of wound copper wire, which emits radio frequency signals. It also acts as a receiver.

1.2 Working

In a typical RFID System, tags are attached to objects. Each tag has a certain amount of memory to store information about the object, such as its unique tag ID (serial number), or in some cases, more details, e.g. manufacturing date, expiry date etc. When these tags pass through an electromagnetic field generated by the reader, they transmit this information back to the reader, thereby enabling object identification.

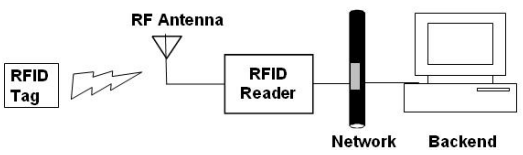

Figure 1.1: RFID System Components

1.3 Types of Tags

There are two types of tags:

- Passive Tags : They operate without an internal battery source, deriving the power to operate from the field generated by the reader.

- Active Tags : They are powered by an internal battery and are typically read/write devices.

Characteristic	Passive	Active
Transponder size	Small	Medium
Range	Short	Long
Data capacity	Small	Large
Application requirements	Small data storage, line of sight read	Large data storage with complex search capability
Typical applications	Fixed asset identification, Bar Code replacement	Identification in wide area , Container data storage and searching

Table 1.1: Features of Active and Passive Tags

1.4 Classes Of Tags

- CLASS 0 (Read Only Tags) These are the simplest type of tags, where the data, which is usually a simple Tag ID is stored only once into the tag during manufacture.

- CLASS 1 (Write Once Read Only (WORM)) These can be factory or user programmed. In this case data can then either be written by the tag manufacturer or by the user only once.

- CLASS 2 (Read Write) Data can be read as well as written into the tag's memory. They contain more memory space than what is needed for just a simple ID number.

- CLASS 3 (Read Write with on board sensors) These are active tags which may contain sensors for recording parameters like temperature, pressure etc. and can record the readings in tag memory.

- CLASS 4 (Read Write with integrated transmitters) These tags can communicate with each other without any help from reader.

1.5 Problems with RFID System

A major problem with RFID systems is that a tag might not be read, in spite of being in the reader's range, due to collisions[4]. A collision is said to have occurred when various devices interfere with each others' operations, or their simultaneous operations lead to loss of data. The reading process is not efficient due to various types of collisions, which are classified as follows :-

- Single Reader-Multiple Tags collision : Multiple tags are present to communicate with the reader. They respond simultaneously and reader is not able to interpret the signal.
- Single Tag-Multiple Readers collision : Single tag is in the range of two or more readers. Tags are mainly passive entities, they do not have enough power to differentiate between frequency range of the readers.
- Reader-Reader interference : Two or more readers within the same frequency range interfere with each others' operations.

These problems need to be resolved to provide efficient solution for tag identification and these are the major research areas, where work needs to be done to practically implement RFID systems.

Chapter 2

MTP Problem Definition

In an RFID system, all tags can not be identified by a reader in a single read cycle due to interference and various environmental factors. To ensure reliability while reading the tags, multiple read cycles are required. We need faster read cycle so as to perform more number of read cycles in given time. There are various ways in which the reader - tag communication can be improved. There are several schemes proposed in literature that attempt to solve this problem. Some papers do this by resolving reader-reader collisions, some concentrate on multiple tag collision avoidance and some describe new MAC protocol for inter-reader communication.

The project aims at improving the tag-reader communication so that tags can be identified efficiently and reliably in less time. The problem can be dealt with in the following ways:

- Use of specific tagID assignment in specific domains.
- Use of previous read cycles' history to reduce the number of collisions.

The first method can be used to devise protocols for specific domains. For example, if we consider the case of godowns, agencies, and exclusive showrooms, where items come in large numbers and there is a high probability that a single lot will contain similar items or items from the same vendor, this information can be utilized for optimization of read process at the shipment point. We have used this information in our Intelligent Query Tree Protocol (IQT) to reduce the number of bits transferred between reader and tag during the tag read process.

The second method deals with using history of previous tag reads to guide the reader so as to reduce the number of single reader - multiple tag collisions.

The proposed IQT Protocol exploits these optimizations in order to improve the tag read efficiency.

2.1 Motivation

RFID mechanism is inherently unreliable. Thus, we need multiple read cycles to improve the reliability of tag identification. The conventional tag identification protocols do not

have any mechanism to use the information contained in tagIDs to improve tag read efficiency. Hence, there is need for a protocol, which can use information about specific prefix patterns in tagIDs so as to speed up the tag read process.

2.2 Problem Formulation

There are practical scenarios where tags have some specific pattern in their tagIDs. For example, items at godowns, shipment points in malls, exclusive showrooms etc., have some common prefix such as EPC version, manufacturer ID or product ID. Our work targets improvement in these deployment scenarios. Tag read efficiency can be improved by reducing the number of bits transmitted during tag identification process and also by using tag read history to reduce the number of collisions in subsequent read cycles.

Chapter 3

Related Work

The proposed IQT protocol belongs to the class of Single Reader-Multiple Tag collision resolution protocols. Other important protocols in this category are Query Tree Protocol[5][6], Binary Tree Protocol[5], Framed Slotted Aloha based I-Code Protocol [7] and Adaptive Memoryless Tag Anti-Collision Protocol[8].

There is another category of protocols to avoid Reader - Reader Collisions (Multiple Reader - Single Tag Collision and Reader - Reader Interference). Important protocols in the category of Multiple Readers collision avoidance protocols are Colorwave Algorithm [9], T-Colorings based Algorithm[10], Q-Learning Algorithm [11]and Pulse Protocol[12]

3.1 Single Reader - Multiple Tags Collision Avoidance Protocols

These protocols are for avoiding the collisions due to presence of multiple tags communicating with a single reader, where simultaneous responses from various tags prevent the reader from interpreting the signal correctly.

Passive tags are constrained in terms of energy since they derive power from the readers' signal only. Such a low energy supply requires the functionality of tag to be very simple so that its power consumption can be reduced. Collision resolution protocols for such kind of RFID tags should be simple enough. Following are various collision resolution protocols that already exist.

3.1.1 Query Tree Scheme

Query Tree (QT) Protocol requires very less tag circuitry. It is a memoryless tag identification protocol, in which the tags do not need to remember their inquiring history. In this protocol, the reader sends a query containing a prefix having length of 1 to n bits. The tags whose prefixes match with the bits sent by the reader, reply back with their tag ID. A queue of such prefixes is maintained and the queries are sent in order from this queue. As and when a query is done, its corresponding entry is removed from the queue. If there is a collision corresponding to any query, i.e., there are more than one tags with the same prefix, the reader removes that query and adds two new queries to the queue,

first by appending a 0, and then a 1 to the current prefix. No collision implies either an ID has been read successfully or there is no tag with matching prefix. An example is shown in Table 3.1 where there are 3 tags with the IDs of 0000, 0010, 1000.

Reader sends	Tags answer	Status	Queue Status(Initially {})
start	*	collision	{0, 1}
0	*	collision	{1, 00, 01}
1	1000	read	{00, 01}
00	*	collision	{01, 000, 001}
01		no response	{000, 001}
000	0000	read	{001}
001	0010	read	{}

Table 3.1: The tag identification process of Query Tree Protocol

In Table 3.1, the first row corresponds to the first query (with null prefix), to read the given set of tags. All tags reply to it. There is a collision as there are more than one tags in the system. Then it sends query with prefix 0, which leads to collision again, as there are two tags starting with 0. Next time it tries 1, there is only one tag starting with 1, which will reply back with its tag ID, and is identified. Similarly process goes on till all the tags have been identified.

IQT Protocol is closely related to QT Protocol. IQT Protocol improves QT Protocol to reduce the communication overhead in the scenarios where tagIDs have some common prefix.

3.1.2 Binary-Tree Scheme

Binary Tree Protocol requires tags to remember the inquiring history in a read cycle. Reader sends one bit at a time and tags reply with the next bit. If there is no collision i.e. only one tag replies with the corresponding bit, the reader repeats the bit sent by the tag. If there is a collision, the reader will send randomly 0 or 1 as the next bit in the inquiry process. Though it is very efficient, it is not good for passive tags, as tags have to remember the previous bits read by the reader, which is not feasible for very low cost passive tags.

An example of inquiring process on 3 tags with tagIDs 001, 011 and 100 is shown in Table 3.2. Asterisk (*) denotes a collision detected by the reader.

3.1.3 Adaptive Memoryless Tag Anti - Collision Protocol

Performance of query tree protocol can be improved by maintaining the history of read cycles in the form of Candidate Queue (CQ). Candidate queue maintains the list of leaf

Reader sends	Tags answer	Identified tag
0	*	
0	1	001
0	1	
1	1	011
0		
1	1	
1	0	110

Table 3.2: The inquiry process of Binary-tree protocol

nodes of the last query process. Leaf nodes can be classified into one of the following categories.

Identified Nodes These correspond to the set of tags which were identified in the last read process.

No Response nodes These correspond to the leaf nodes in query tree which do not lead to response from any tag.

In adaptive memoryless protocol, reader queries for Identified nodes present in candidate queue, as well as it queries for No-Response nodes where the newly coming tagID may lie. Even if some new tagID collides with the previously identified tagID, the total amount of time saved using this protocol is significant. This is because we do not query for Collision nodes.

3.1.4 I-Code Protocol

I-Code Protocol is a probabilistic protocol, based on framed slotted aloha principle, in which tags randomly choose the slots to transmit. Adaptive Memoryless Tag Anti-Collision Protocol is an improvement on Query Tree Protocol, to improve the subsequent read cycles.

3.2 Reader - Reader Collision Avoidance Protocols

These protocols avoid the collisions occurring due to presence of a single tag within the range of two or more readers, or, because of two or more readers interfering with each other. Tags are mainly passive entities. They do not have enough power to differentiate between the frequency ranges of the readers. Following are the protocols to avoid such collisions.

3.2.1 Colorwave

Colorwave is a distributed, online algorithm, based on graph coloring. Q-Learning Algorithm learns from previous collision patterns and effectively assigns frequencies over time to the readers.

3.2.2 Pulse Protocol

In this protocol readers can communicate with each other over control channel to minimize the number of collisions. Reader can send control signal called beacon over separate control channel notifying the neighboring readers that it is going to identify the tags, so that signal from other readers do not interfere with its reading process.

Chapter 4

Intelligent Query Tree (IQT) Protocol

IQT protocol exploits specific prefix patterns in the tagIDs to reduce the communication overhead between reader and tags. The common prefix in tagIDs may be due to the fact that items to which the tags are attached have the same manufacturer or product type. Query Tree Protocol (QT) has been modified for the scenarios where the tags within the range of the reader have some common prefix. It also uses history of read cycles to further improve the tag read efficiency. Although IQT is best suitable for scenarios in which there is some commonality in tagIDs, it performs reasonably well in other cases also, where multiple read cycles are required. Following are its advantages over QT Protocol:

- Reduction in number of bits transmitted by the tag.
- Reduction in number of collisions by maintaining the history of tag read patterns.
- Reduction in number of collisions for subsequent read cycles by using the information of previous read cycles.

4.1 Key Terms

- **EPC Code**[13][14] : Like universal bar code, EPC code is a standard Tag ID assignment code that assigns globally unique IDs to RFID tags. EPC structure is shown above.

EPC version	Manufacturer ID	Product Type	Item ID

Table 4.1: EPC Structure

- **Read Cycle** : Read cycle refers to the set of queries required to read all the tags that are in range of reader at a particular time. In a single read cycle, a reader may not be able to read all the tags due to various factors like collisions, interference etc. So multiple read cycles are required to improve reliability.

- **Read Process** : It refers to the set of multiple read cycles performed by reader to ensure reliability.
- **Query** : It refers to the command containing some prefix sent by the reader, in response to which all the tags with matching prefix reply back with their tag IDs.
- **Invert Query Command** : It is just the opposite of the normal query command. Reader sends some prefix with the command and all the tags having prefixes different from that sent by reader are supposed to reply to this command. This command is issued to ensure that all the tags, that are read by the reader in the current read cycle, have same prefix.
- **Prefix Pool** : It is a set of frequently occurring prefixes sorted according to rank based upon the frequency of their occurrence, and how recently the prefix was used in previous read cycles.

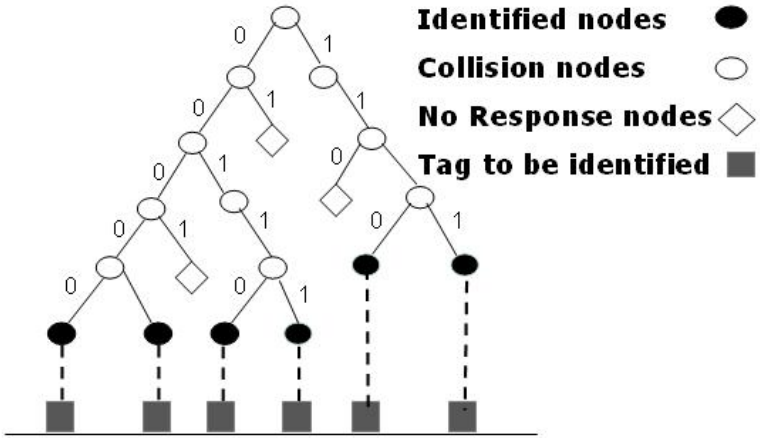

Figure 4.1: Different types of nodes in query tree

- **No-Response Node** : The leaf node in query tree where no tag ID starting with that prefix is found. In Figure 4.1, nodes represented by diamond are No-Response Nodes. For example, if we follow the edges 0 and then 1 starting from the root, it directs to the No-Response node. This node indicates that there is no tag with prefix 01.
- **Identified Node** : The node in the query tree that leads to response from single tag i.e one and only one tag is identified using this prefix. Nodes represented by filled circles in Figure 4.1 are Identified Nodes. If we follow the edges 0, 0, 0, 0, 0, 0 it directs us to the Identified Node.
- **Collision Nodes** : These are the internal nodes in the query tree which leads to response from more than one tag IDs corresponding to a query using a prefix. In Figure 4.1, unfilled circles represent the Collision or Internal Nodes.

4.2 Notations Used

Table 4.2 shows various notations used in this chapter to describe IQT Protocol.

EPC	Number of bits for EPC version.
Manufacturer_id	Number of bits for Manufacturer ID.
Product_id	Number of bits for Product ID.
Item_no	Number of bits for Item ID.
prefix1	Bits corresponding to *EPC* Version + *Manufacturer_id* + *Product_id* in the EPC code.
prefix2	Bits corresponding to *EPC* Version + *Manufacturer_id* in the EPC code.
max_tries	Maximum number times, the reader tries the prefixes from prefix pool, to guess the prefix for current read cycle.
k	Number of bits in tag ID
prefix	Refers to {*prefix1 or prefix2*}
cycle_needed	No of read cycles required in a single read process
rem	$(k - prefix)$ bits

Table 4.2: Notations Used

4.3 Working of Intelligent Query Tree Protocol

Mechanisms used in IQT to identify tags in the first read cycle are different from those for subsequent read cycles. When IQT performs first read cycle, it has no knowledge of tag IDs, but when it performs subsequent read cycles, it knows whether the tags have some common prefix. If they have a common prefix, then it is known to the reader. Hence, IQT does not read those bits again.

4.3.1 First Read Cycle

Intelligent Query Tree Protocol does the following steps to perform first read cycle.

Step 1 [Optional] - Try prefixes from Prefix Pool : For the first read cycle, prefix with highest score from prefix pool is selected and Invert Query Command is executed using first *prefix2* bits to check whether all the tags have the same first *prefix2* bits. If there is a collision, next prefix from the prefix pool is tried and so on.

If it succeeds, then the reader tries other prefixes (which have same *prefix2* bits as that of the one which lead to success) by sending *prefix1* bits. Reader now sends *prefix1* bits to execute invert query command. This guessing phase continues for a maximum of

max_tries number of times. If it succeeds, then it knows the first *prefix* bits that are common in all the tags present.

Step 2 - If Step 1 fails for First Read Cycle : If it is not able to know the prefix in step 1, then it proceeds with normal Query Tree Protocol to know first *prefix1* bits.

Step 3 : After reading first *prefix1* bits either from step 1 or step 2, reader now executes invert query command using *prefix1*, as many number of times as there are cycles in the read process (*cycles_needed* times). If the invert query fails even once (i.e. if any tag replies), we do not execute the invert query command any further.

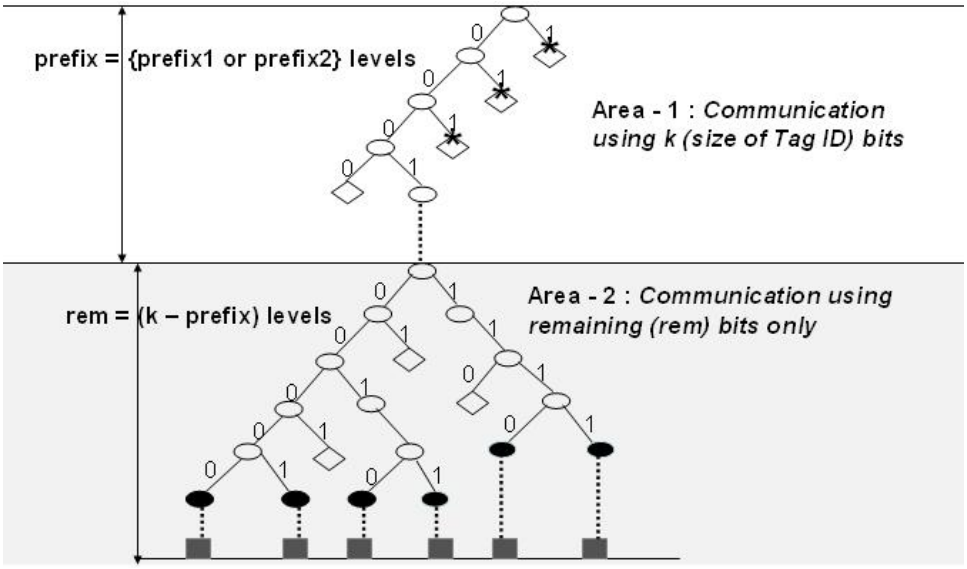

Figure 4.2: Tag Identification using Intelligent Query Tree Protocol

- If no tag replies, it means all tags have the same first *prefix1* bits. Hence, communication can take place using *Item_no* only. This leads to reduction in number of bits transmitted between reader and the tags. Area-2 in Figure 4.2, shows the queries where communication takes place using fewer bits. Reader also ignores all the queries with number of bits less than *prefix1* as it has already read the *prefix1* bits successfully and it knows that all the tags start with this *prefix1*. This leads to saving in number of queries as shown by Area-1 in Figure 4.2. Here the nodes marked with crosses represent the queries saved (not needed) in IQT Protocol. If reader succeeds in guessing the *prefix*, it need not perform any other query in Area-1.

- If some tags reply, then the invert query is executed with *prefix2*, *cycles_needed* number of times. If no tag replies to this command, then it means that all the tags have the same manufacturer ID. Hence, the reader can communicate using only the bits of product type and item number. The reader ignores all the queries with number of bits lesser than *prefix2* for this read cycle.

- If some tags reply to invert query with *prefix2*, it means that there are items from different manufacturers in the lot. In this case normal Query Tree Protocol will be used for tag identification.

4.3.2 Subsequent Read Cycles

IQT improves subsequent read cycles by storing information about previous read cycles. This information is stored in the form of candidate queue as described in [8]. Candidate queue contains Identified Nodes and No Response Nodes only, it does not include Collision Nodes.

For subsequent cycles, query is issued using only the elements of candidate queue i.e. using only the No-Response Nodes and the Identified Nodes of the previous read cycle. If any new tag appears which leads to the collision, then only the subtree rooted at that node is further explored using normal QT Protocol, instead of following the whole query tree from the root.

This helps to avoid the overhead of various read cycles wasted in collisions. In Figure 4.3, the nodes denoted by filled circles and the nodes denoted by diamonds, refer to the elements of candidate queue. Crossed Nodes refer to the saving using this optimization.

Further improvement is possible if all the tags have the same prefix and it matches with the one found in the previous read cycle. It can be checked by running Invert Query Command with the prefix found in the previous read cycle. If all the tags have same prefix which is same as the one found in previous read cycle, then query command with prefixes less than *prefix1* bits need not be executed. That means, all nodes in Area-1, and Collision Nodes in Area-2 of Figure 4.3 represent the saving in terms of number of queries. Hence, query command is only issued for the No-Response Nodes and Identified Nodes of the previous read cycle and that too with lesser (rem) bits.

If a new tag is discovered, then either it will have one of the No-Response Node as its prefix, or it will collide with one of the Identified Nodes. If it has some No-Response Node as prefix, then that node will transform into an Identified Node if only single tag is there with that prefix. Otherwise the node will become a Collision Node if there are multiple tags with same prefix. Similarly, if new tag collides with Identified Node, then that node becomes a new Collision Node and subtree rooted at that node is explored using normal Query Tree Protocol. For example, Collision Node highlighted by arrow in Figure 4.3 was Identified Node in query tree of Figure 4.2, but due to the appearance of some new tag it becomes a Collision Node.

For subsequent read cycles, the protocol works as follows.

- Run invert query with maximum matched prefix (*prefix1 or prefix2*) found in the first read cycle, *cycles_needed* number of times. If the invert query fails even once (i.e. if any tag replies), we do not execute the invert query command any further.

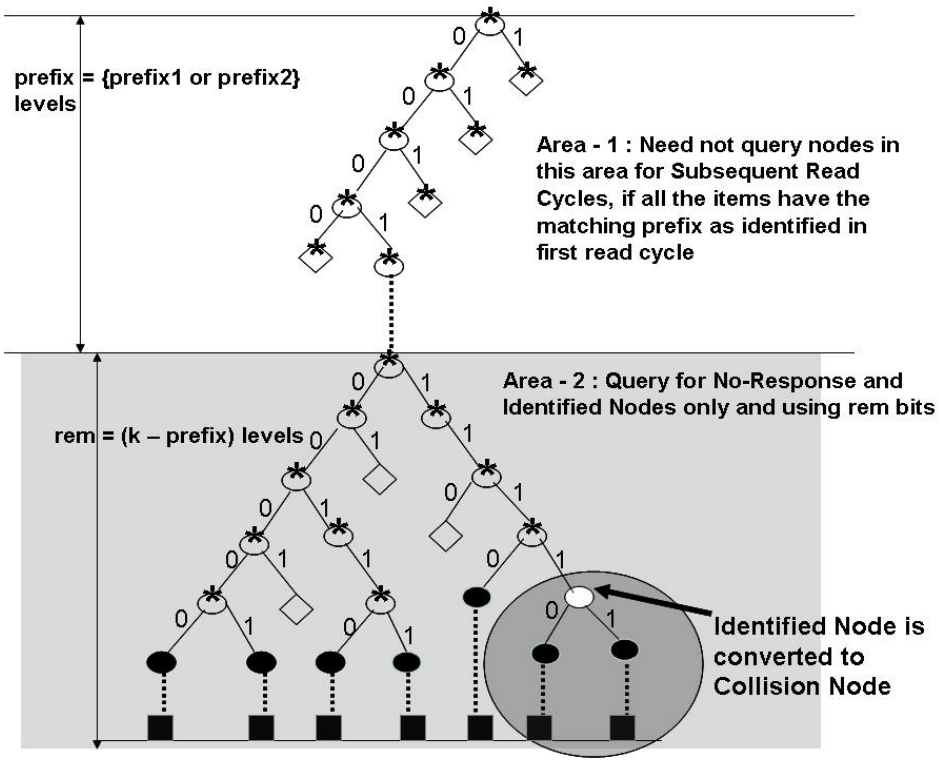

Figure 4.3: Query using candidate queue entries for subsequent read cycles

- If query with *prefix1* succeeds, it means that all the tags have common prefix *prefix1*. Then the reader starts querying using only the No Response Nodes as well as the Identified Nodes, having number of bits more than *prefix1*. It does not query nodes having lesser number of bits, because it already knows the *prefix1* bits and has ensured that no tag has different prefix. Now communication takes place using only *Item_no* (bits below *prefix1*).
- Else If query with *prefix2* succeeds, then reader starts querying with Identified Nodes and No Response Nodes below *prefix2* levels. Communication between reader and tag now takes place using the bits below *prefix2*.
- Else reader has to send queries for all the Identified and No Response Nodes using k bits (all the bits in tag ID).

Chapter 5

Comparative Analysis of IQT Protocol with QT Protocol

IQT Protocol achieves substantially better performance with very minimal change in tag hardware complexity. This is because along with the prefix, the query itself can contain control information to determine whether query will take place using the entire tagID or with the *rem* bits.

Performance of IQT Protocol is compared with normal QT Protocol. Better efficiency can be obtained by decreasing the number of bits transmitted between reader and tag in one query, as well as by reducing the number of queries in one read cycle. Different optimization techniques for the first read cycle and the subsequent read cycles are applied. In the first read cycle, improvement is mainly due to reduction in bits transmitted between reader and tag, while in subsequent read cycles, the improvement is mainly due to reduction in number of queries.

5.1 Where IQT Protocol gains over QT Protocol ?

Figure 4.2 describes the tag reading using Intelligent Query Tree Protocol for the first read cycle. Optimization is due to two factors.

- Crosses marked nodes in Area-1 of Figure 4.2 denote the reduction in number of queries in IQT protocol over normal QT protocol. After reaching $prefix^{th}$ level, the IQT protocol will check whether all the items have prefixes matching with the bit sequence read till that point. If all the items have first *prefix* bits common, then reader need not perform query for some other node, with less than *prefix* bits. Hence on an average, half of the queries corresponding to No-Response Nodes till *prefix* levels will be saved. Savings are even greater if we are able to guess the prefix in certain number of tries, instead of learning it one bit at a time.

- Area-2 in Figure 4.2 denotes the improvement due to reduction in number of bits. Queries in Area-2 need not be performed using the whole tag ID. Hence, if all the items have first *prefix* bits common, then communication can take place using only the remaining (*rem*) bits.

Figure 4.3 describes the saving for subsequent read cycles using IQT protocol over QT protocol. Again, optimization is due to two factors.

- For the nodes in Area-1 of Figure 4.3, we need not perform any query, as we already know the prefix from first read cycle. We need to confirm that no tag with different prefix appears, so we run Invert Query Command using the prefix found in first read cycle. Hence we save all the queries in Area-1, and perform Invert Query Command, *cycles_needed* number of times. This will lead to success most of the times, as we are using the protocol in the scenarios where there is high probability of items from same vendor or product type i.e. tags with some common prefix are highly probable.

- In Area-2 of Figure 4.3, we will save the queries corresponding to Collision Nodes (crossed nodes). Also for the remaining queries we will be using lesser number of bits (*rem*) only.

5.2 Comparisons of IQT with QT in the worst case of tag identification

In this section, we analyze performance of IQT Protocol over QT. We look at the worst case scenario of tag identification, which occurs when the tags present in the system have IDs such that maximum number of queries are required. We look at the worst case, because it gives lower bound in terms of number of tag reads per cycle.

In this study, we look at two performance parameters: 1) the number of queries to identify all the tags and 2) the number of bits transmitted per query. Note that these two parameters decide the performance of a reader in terms of number of tags read per unit time. We show that IQT protocol performs much better than QT. Hence, readers running IQT protocol can read more tags per unit time.

We first calculate the number of queries required in the worst case of tag identification i.e. size of the largest query tree. We present our results using the following lemmas and theorems.

First, we define some important terms used:

Definition 1 TagPair *is a set of two tags, which differ only in the last bit of their tag IDs. This means that tags in the TagPair have the first* $k-1$ *bits common in their tag ID. Hence, to identify a TagPair, the query tree will have collisions upto* $k-1$ *levels and identification of the tags would happen at the* k^{th} *level.* i^{th} *TagPair in a system is denoted as* $TagPair_i$.

Note that, to identify only a TagPair, the query tree will grow to level k, root of the tree being level 0.

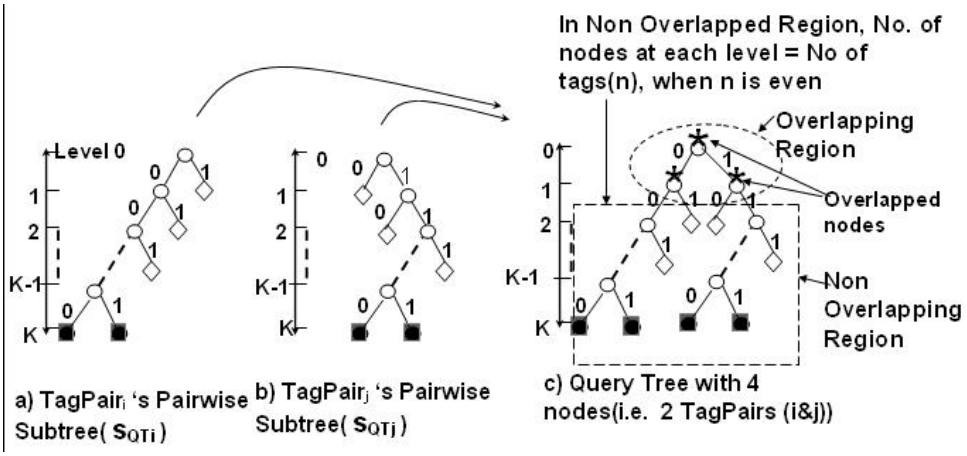

Figure 5.1: Overlapped and Non Overlapped Regions in query tree

Definition 2 Pairwise Subtree ($S_{Q_{T_i}}$) *is defined as the query tree formed while identifying only two tags belonging to* $TagPair_i$. *It is obvious that* $S_{Q_{T_i}}$ *would be a subtree of* Q_T. Q_T *is the query tree formed while identifying all the n tags in the system. Figure 5.1(a) shows the Pairwise Subtree* ($S_{Q_{T_i}}$) *corresponding to the tag pair* $TagPair_i$

Definition 3 Overlapping Region (S_{OL}) *is the subtree of the query tree* Q_T *from level 0 to level L, as shown in Figure 5.1(c), where L is given by*

$$L = \text{maximum level at which } \forall i, \exists j | S_{Q_{T_i}} \bigcap S_{Q_{T_j}} \neq \emptyset. \tag{5.1}$$

In other words, at each level in Overlapping Region, at least one node is common among Pairwise Subtrees.

Definition 4 Non Overlapping Region (S_{NOL}) *is the region of the query tree from level* $L+1$ *to level k where L is given in Equation 5.1. In other words, in this region, there is no overlap among all the Pairwise Subtrees, as shown in Figure 5.1(c).*

Lemma 1 *If we have only two tags with k bit tag IDs, then the query tree will have maximum of k levels with* $2 * k + 1$ *nodes. This happens when the two tags form a TagPair.*

Proof 1 *If there are only two tags, the query tree will have two nodes at each intermediate level before they are identified. At any intermediate level, one node would be a Collision Node and the other one would be a No-Response Node. At the last level, there will be one Identified Node. If first* $i-1$ *bits of the tags are same, then there will be* $i-1$ *intermediate levels (with 2 nodes at each level), and finally the tag will be identified at the* i^{th} *level. Thus, there will be i levels and* $2 * i + 1$ *nodes in the query tree. Maximum possible value of i can be k. Thus, the query tree in this case will have k levels with* $2 * k + 1$ *nodes. And in this case, the tag IDs differ in the last bit only i.e. they form a TagPair.*

Lemma 2 *If we have an even number of tags and all of them form TagPairs, the query tree will have n nodes at every level in the Non Overlapping Region.*

Proof 2 *Each Pairwise Subtree has 2 nodes at each level below the root and there is no node common between two Pairwise Subtrees in the Non Overlapping Region. If we have n tags, all forming TagPairs, then there are $n/2$ Pairwise Subtrees. Hence, the query tree will have $n/2 * 2 = n$ nodes, at each level in Non Overlapping Region.*

Lemma 3 *If the number of tags (n) is odd and the tags form $(n-1)/2$ TagPairs, then the query tree will have $n-1$ nodes at every level in the Non Overlapping Region.*

Proof 3 *If we have odd number tags and the tags form $(n-1)/2$ TagPairs, then we have $(n-1)/2$ Pairwise Subtrees. These Pairwise Subtrees will have $(n-1)/2 * 2 = n-1$ nodes at each level in Non Overlapping Region. The remaining one tag will not lead to any extra node in the Non Overlapping Region. It will just convert one No Response Node to Identified Node.*

Lemma 4 *In the Overlapping Region, query tree having an even number of n tags, which forms $n/2$ TagPairs, will always have less than n nodes at each level. If the number of tags n is odd and they form $(n-1)/2$ TagPairs, then the query tree will have less than $n-1$ nodes at each level in the Overlapping Region.*

Proof 4 *From Lemma 2 and Lemma 3, there are n or $n-1$ nodes at each level in Non Overlapping Region when n is even or odd respectively. As per definition, in the Overlapping Region, there will be some nodes that will be common to two or more Pairwise Subtrees. Hence, at any level in the Overlapping Region, the number of nodes will be less than n or $n-1$ when n is even or odd respectively.*

Lemma 5 *Number of levels in Overlapping Region will be at least $\lfloor log_2(n-2) \rfloor, n > 2$, when there are n tags forming $\lfloor n/2 \rfloor$ TagPairs.*

Proof 5 *From Lemma 4, at any level in the query tree if the number of nodes is less than n (for even n) or less than $n-1$ (for odd n), then that level would be a part of Overlapping Region. So when n is even, the minimum level of Overlapping Region is $\lfloor log_2(n-1) \rfloor, n > 1$ (since it can have upto $n-1$ nodes at that level). Similarly, when n is odd, the minimum level of Overlapping Region is $\lfloor log_2(n-2) \rfloor, n > 2$. Combining the two cases, the lemma follows.*

Theorem 1 *If n is the number of tags, then the maximum number of nodes in the query tree is given by:*

$$\begin{cases} 2^{\lfloor log_2(n-2) \rfloor + 1} - 1 + (k - \lfloor log_2(n-2) \rfloor) * (n) \text{ if } n \text{ is even} \\ 2^{\lfloor log_2(n-2) \rfloor + 1} - 1 + (k - \lfloor log_2(n-2) \rfloor) * (n-1) \text{ if } n \text{ is odd} \end{cases}$$

Proof 1 *From Lemma 1, size of query tree will be maximum if tags occur as TagPairs. Lemma 4 shows that the largest query tree for a given number of n tags (forming $\lfloor n/2 \rfloor$ TagPairs) will be obtained if there are least number of levels in the Overlapping Region. This is because, Overlapping Region has fewer nodes than Non Overlapping Region.*

Nodes in the query tree are divided into two parts:

- *Nodes in Overlapping Region*
- *Nodes in Non Overlapping Region*

Maximum number of nodes in the Overlapping Region is given by the maximum possible nodes in the complete binary tree of height $\lfloor log_2(n-2) \rfloor$, $n > 2$. This is equal to: $2^0 + 2^1 + \ldots + 2^{\lfloor log_2(n-2) \rfloor} = 2^{\lfloor log_2(n-2) \rfloor + 1} - 1$ *for* $n > 2$

Consider the case when n is even. From Lemma 3, each level in Non Overlapping Region will have n nodes. Since there are $(\lfloor log_2(n-2) \rfloor)$ levels in the Overlapping Region i.e. there are $(k - \lfloor log_2(n-2) \rfloor)$ in the Non Overlapping Region, hence maximum number of nodes in Non Overlapping Region is $n * (k - \lfloor log_2(n-2) \rfloor)$. Hence, number of nodes in largest query tree is given by:
$2^{\lfloor log_2(n-2) \rfloor + 1} - 1 + (k - \lfloor log_2(n-2) \rfloor) * (n)$ *for* $n > 2$

Now consider n to be odd. In this case, maximum number of nodes will be equal to nodes in the largest query tree with $n-1$ nodes, since last odd tag will just replace one No-Response Node with Identified Node.
Hence, number of nodes is given by:
$2^{\lfloor log_2(n-2) \rfloor + 1} - 1 + (k - \lfloor log_2(n-2) \rfloor) * (n-1)$ *for* $n > 2$

5.2.1 Performance Improvement in the First Read Cycle

IQT would be deployed where it is highly probable that tag IDs have common manufacturer ID or product ID fields i.e. all the tags have first *prefix* bits same. *prefix* is equal to *prefix1*, if all the items have same EPC version, manufacturer and product type, and it is equal to *prefix2* if all the items have same EPC version and manufacturer, but they are of different product types. In this scenario, we assume that first *prefix* bits of all the tags are same. We will consider only even number of tags to derive expressions for the complexity, since all the calculation remain same, just the term $2^{\lfloor log_2(n-2) \rfloor + 1} - 1 + (rem - \lfloor log_2(n-2) \rfloor) * (n)$ will be replaced by $2^{\lfloor log_2(n-2) \rfloor + 1} - 1 + (rem - \lfloor log_2(n-2) \rfloor) * (n-1)$, for odd values of n, as explained in Theorem 1

Number of queries required in QT Protocol:
Assuming even number of tags and $n > 2$, Query Tree Protocol requires $2*$*prefix*$+1$ queries to identify first *prefix* bits, as shown in Figure 4.2. Theorem 1 shows that with n (even) tags, we require maximum of $2^{\lfloor log_2(n-2) \rfloor + 1} - 1 + (k - \lfloor log_2(n-2) \rfloor) * (n)$ queries. But in our case, we are assuming that tags have common *prefix*, and tag IDs are distributed according

to worst case scenario for the levels below *prefix* i.e. for $(k - prefix)$(i.e. rem) levels. For rem levels the largest query tree will have $2^{\lfloor log_2(n-2) \rfloor + 1} - 1 + (rem - \lfloor log_2(n-2) \rfloor) * (n)$ nodes. Hence, total number of queries required are:

$$\begin{cases} 2 * prefix + 1 + 2^{\lfloor log_2(n-2) \rfloor + 1} - 1 + (rem - \lfloor log_2(n-2) \rfloor) * (n) \text{ for } n > 2 \end{cases} \tag{5.2}$$

Since tags reply with their k bit tag IDs in the QT Protocol, the total number of bits transmitted by tags in the first read cycle is given by

$$\begin{cases} (2 * prefix + 1 + 2^{\lfloor log_2(n-2) \rfloor + 1} - 1 + (rem - \lfloor log_2(n-2) \rfloor) * (n)) * k \text{ bits.} \end{cases} \tag{5.3}$$

Number of queries required in IQT Protocol:

***Case 1: prefix is guessed within* max_tries**:

Suppose reader is able to guess the prefix in *no_of_guesses* tries. Further, *cycles_needed* times, Invert Query Command is required to confirm that all the tags have the same first *prefix* number of bits. Then, total number of queries required by IQT protocol will be:

$$\begin{cases} no\_of\_guesses + cycles\_needed + 2^{\lfloor log_2(n-2) \rfloor + 1} - 1 + (rem - \lfloor log_2(n-2) \rfloor) * n \end{cases} \tag{5.4}$$

Every unsuccessful guess of Invert Query command will be responded by a tag with k bits. But once the Invert Query command succeeds, the remaining queries (queries below *prefix* levels) will only require *rem* bits.
Hence, the total number of bits transmitted by the tags are given by

$$\begin{cases} (no\_of\_guesses + cycles\_needed) * k \\ +(2^{\lfloor log_2(n-2) \rfloor + 1} - 1 + (rem - \lfloor log_2(n-2) \rfloor) * n) * rem \text{ bits.} \end{cases} \tag{5.5}$$

So, performance improvement of IQT over QT, in terms of number of queries, is given by

$$(2 * prefix + 1 - no\_of\_guesses - cycles\_needed) \tag{5.6}$$

and the improvement, in terms of bits transmitted by a tag, is

$$\begin{cases} (2 * prefix + 1 - no\_of\_guesses - cycles\_needed) * k \\ +(2^{\lfloor log_2(n-2) \rfloor + 1} - 1 + (rem - \lfloor log_2(n-2) \rfloor) * n) * (k \text{ - } rem) \text{ bits.} \end{cases} \tag{5.7}$$

***Case 2: prefix is not guessed within* max_tries**: This is the case, when the prefix is not guessed within *max_tries* tries. Hence, first *prefix* bits are learnt using the normal query tree protocol. In the worst case, we need $2*prefix+1$ queries to learn the first *prefix* bits, as needed by normal Query Tree Protocol. Hence performance improvement in terms of the number of bits will only be

$$\begin{cases} (2^{\lfloor log_2(n-2) \rfloor + 1} - 1 + (rem - \lfloor log_2(n-2) \rfloor) * n) * (k - rem) \\ -(cycles\_needed + max\_tries) * k \text{ bits,} \end{cases} \tag{5.8}$$

since we waste max_tries queries to guess the prefix. Here in the equation, *cycles_needed* term is there. This is because, to confirm that first *prefix* bits are same in all the tagIDs, invert query command is executed *cycles_needed* times.

Table 5.1 shows the performance improvement in two scenarios. First, where all the tags have first *prefix1* bits common, and second, where all the tags have first *prefix2* bits common. Second and Third columns in the table show the percentage saving in bits where *prefix* is guessed in 4 tries. Fourth and Fifth columns show the improvement when the reader is not able to guess the prefix and it learns first *prefix* bits using the normal QT Protocol.

5.2.2 Performance Improvement in the Subsequent Read Cycles

If the set of tags is exactly the same as in the first read cycle, we will save queries corresponding to all the nodes in Area-1, and Collision Nodes in Area-2 of Figure 4.3. Also, we will need lesser number of bits for No-Response and Identified Nodes in Area-2 if there is some matching *prefix* among all the tags. In this case, when set of tags is exactly same as that of first read cycle, we will save $2 * prefix + 1$ queries in Area-1, but Invert Query Command needs to be executed *cycles_needed* times to confirm that all the tags have same first *prefix* bits. Hence, number of queries saved in Area-1 will be $2 * prefix + 1 - cycles\_needed$. In Area-2, we will save queries corresponding to all the internal nodes. Since, in worst case, we have $(2^{\lfloor log_2(n-2) \rfloor + 1} - 1) + (rem - \lfloor log_2(n-2) \rfloor) * n$ nodes in the subtree below *prefix* levels, the number of internal nodes will be $[(2^{\lfloor log_2(n-2) \rfloor + 1} - 1) + (rem - \lfloor log_2(n-2) \rfloor) * n + 1]/2 - 1$. Number of leaf nodes (No-Response and Identified Nodes) $= [(2^{\lfloor log_2(n-2) \rfloor + 1} - 1) + (rem - \lfloor log_2(n-2) \rfloor) * n + 1]/2$, which leads to saving of $[(2^{\lfloor log_2(n-2) \rfloor + 1} - 1) + (rem - \lfloor log_2(n-2) \rfloor) * n + 1]/2 * (k - rem)$ in terms of bits, for the leaf nodes at the levels below *prefix*.

Therefore, performance improvement of IQT over QT, in terms of number of queries is

$$\begin{cases} 2 * prefix + 1 - cycles\_needed + [(2^{\lfloor log_2(n-2) \rfloor + 1} - 1) \\ +(rem - \lfloor log_2(n-2) \rfloor) * n + 1]/2 - 1 \end{cases} \tag{5.9}$$

and the improvement, in terms of bits transmitted by a tag, is

$$\begin{cases} (2 * prefix + 1 - cycles\_needed + [(2^{\lfloor log_2(n-2) \rfloor + 1} - 1) \\ +(rem - \lfloor log_2(n-2) \rfloor) * n + 1]/2 - 1) * k \\ +[(2^{\lfloor log_2(n-2) \rfloor + 1} - 1) + (rem - \lfloor log_2(n-2) \rfloor) * n + 1]/2^*(k - rem) \text{ bits.} \end{cases} \tag{5.10}$$

Column number six and seven in Table 5.1, shows the percentage reduction in bits transmitted by the tags, when they have same *prefix1* or *prefix2* respectively. These performance improvements are based on the assumption that the set of tags remains same, as for the previous read cycle and each tag shows up in each of the read cycles. If some tags are different in the subsequent read cycle, performance improvement drops and it depends upon the number of nodes in the new subtree, explored using QT Protocol.

In this chapter, we have analyzed the performance improvement of IQT over QT in the first and subsequent read cycles separately, assuming that same set of tags show up in all the read cycles in a single read process. In the next chapter, we have shown through simulations, the performance improvement for the cases when tags respond with some probability, that is, there can be different set of tags for different read cycles in a read process.

No. Of Tags	First Read Cycle				Subsequent Read Cycles	
.	same prefix1 (guessed in 4 tries)	same prefix2 (guessed in 4 tries)	same prefix1 (unable to guess the prefix)	same prefix2 (unable to guess the prefix)	same prefix1	same prefix2
5	76.25	48.58	33.98	27.50	90.00	75.90
10	71.31	43.74	44.22	31.87	85.83	72.28
15	69.00	41.85	49.01	33.57	85.43	71.74
25	66.81	40.23	53.56	35.04	83.70	70.56
50	64.87	38.92	57.60	36.22	82.54	69.80
75	64.14	38.46	59.10	36.63	81.06	69.01
100	63.76	38.23	59.88	36.84	81.88	69.40
125	63.53	38.09	60.37	36.97	82.38	69.63
150	63.37	38.00	60.70	37.05	80.54	68.73

Table 5.1: Percentage reduction in number of bits

Chapter 6

Simulation Results

We analyze the performance of IQT protocol over QT protocol and simulation results are shown for different scenarios. The improvement in efficiency of IQT protocol over QT protocol is due to reduction in the number of bits transmitted between reader and tag in one query, as well as decrease in the number of queries in one read cycle. Different optimization techniques are applied in the first read cycle and subsequent read cycles. In the first read cycle, improvement is mainly due to reduction in number of bits transmitted between reader and tag, while in subsequent read cycles, the improvement is mainly due to reduction in number of queries.

All the simulations have been performed without considering the intelligent way to guess the tagID prefix. If we use this optimization, there will be a slight improvement in the efficiency in all cases, however this improvement will be constant for different number of tags. So the graphs have been shown for one case only.

According to current EPC standards tag ID is of 96 bits with the following breakup.

- EPC version 8 bits : Manufacturer ID 28 bits: Product Type 24 bits : Item ID 36 bits

Above EPC structure is used for all the simulations. Performance improvement is shown for the following three scenarios:

Scenario 1: Items to which RFID Tags are attached have same manufacturer and product type i.e first *prefix1* bits are common in all the tagsIDs.

Scenario 2: Items have different product type, but are from the same manufacturer i.e. first *prefix2* bits are common in all the tagsIDs.

Scenario 3: Items belong to different manufacturer i.e. tagIDs do not have any common *prefix (prefix1/prefix2)* bits.

For all the graphs, we have plotted average performance improvement of IQT over QT, with the confidence interval of 95%. For scenario 1, 2, and 3, we have assumed tag read probability equal to 0.95. In the graphs, mean performance curves are sandwiched between thin black lines of upper and lower confidence interval.

6.1 Scenario 1 - Items have the same manufacturer as well as product type

This means that the tagIDs have first *prefix1* bits common. In this scenario, tag read speed improves due to two factors - reduction in the number of queries, and reduction in the number of bits transmitted by the tag per query. The reduced number of queries gives the lower bound in tag read speed improvement, independent of the physical communication method. There will be further saving due to reduced number of bits per query, which depends on the communication method used, i.e. whether bits are transmitted one per time cycle, or multiple bits can be transmitted in a single time cycle. That will determine how much time we save by transmitting lesser number of bits in a single query.

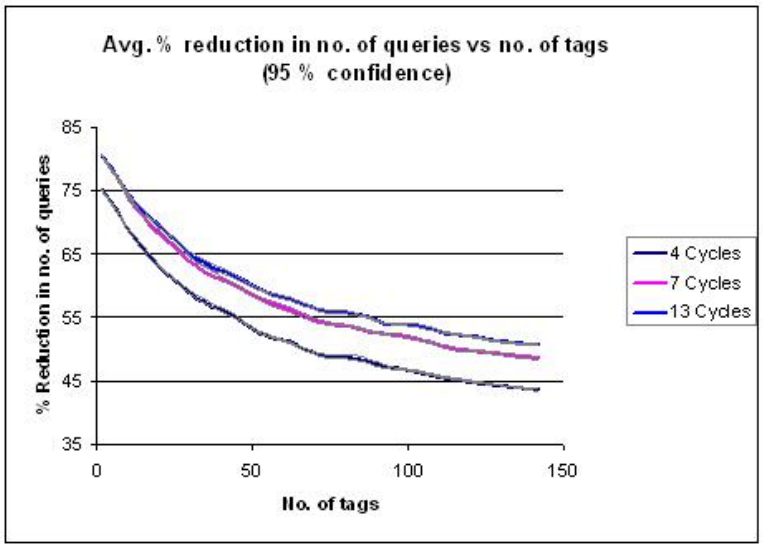

Figure 6.1: % Reduction in number of queries in IQT over QT when tagIDs have first *prefix1* bits common

Figures 6.1 and 6.2 show the improvement in terms of percentage reduction in the number of queries and percentage reduction in the number of bits required respectively. With increase in the number of read cycles in a single read process, percentage saving increases. This is because IQT saves more in subsequent read cycles than in the first read cycle, due to additional information available from the first read cycle. More the number of read cycles in a read process, more will be the saving.

With the increase in number of tags, the performance improvement increases in the case when there are more read cycles per read process, relative to the case when there are lesser read cycles. This is because, with more number of read cycles there is constant extra overhead associated with our protocol, as it has to perform invert query command more number of times (equal to the number of cycles in a read process).

Saving in terms of the number of bits is more than the saving in terms of the number

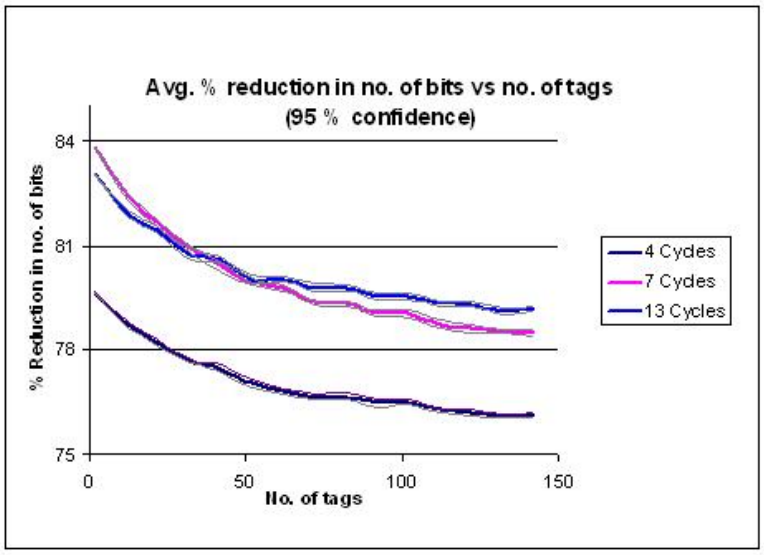

Figure 6.2: % Reduction in number of bits in IQT over QT when tagIDs have first *prefix1* bits common

of queries. This is because reduction in the number of queries also corresponds to the reduction in number of bits; additionally reduction in the number of bits transmitted by the tag per query also leads to further reduction in terms of bits.

In Figure 6.2, the curve with seven read cycles in a single read process intersects with the curve with thirteen read cycles. Initially with lesser number of tags, the case when we have seven read cycles is better. This is because, if we have thirteen read cycles, we need to perform invert query thirteen times for each of the cycles, to confirm that all the tags have same *prefix1* bits, as opposed to seven times in the other case, and with lesser number of tags this constant overhead affects the percentage improvement more.

Figure 6.3 shows the lower bound on the increase in tag read speed by taking into account the effect of reduction in the number of queries only. The actual reduction might be much more because of saving due to transmission of lesser number of bits by the tag. This depends on the underlying physical communication method, and hence is not considered in the simulations.

Due to similar reasons as described for graphs in the figures 6.1 and 6.2, tag read speed will also increase with increase in number of read cycles in single read process as shown in Figure 6.3.

6.2 Scenario 2 - Items have same manufacturer, but different product type

This means that the tagIDs have first *prefix2* bits common. The explanation for the graphs in scenario 2 is similar to those for scenario 1, except that common *prefix* will

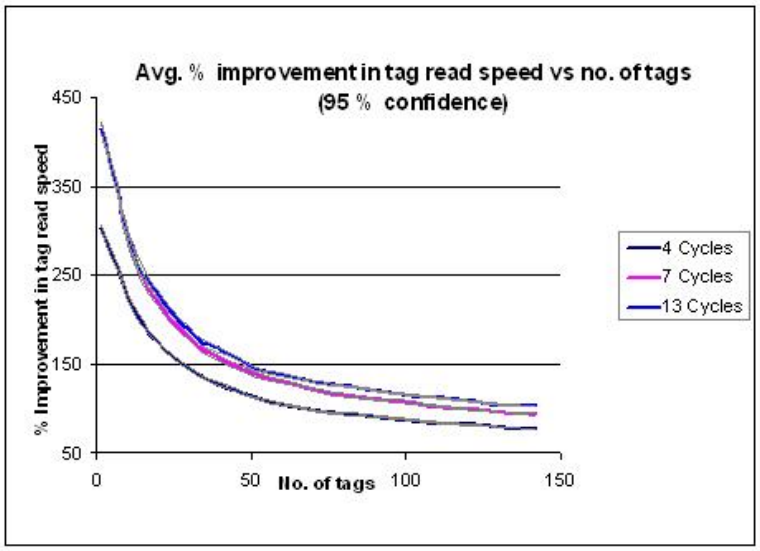

Figure 6.3: % Improvement in tag read speed in IQT over QT when tagIDs have first *prefix1* bits common

be of first *prefix2* bits instead of *prefix1* bits. Trend in graphs will be same, but the performance improvement and tag read speed in this case will be lesser than that in case of scenario 1. Following is the reason for decrease in performance improvement.

In the case of first read cycle, there will be lesser number of crossed nodes (which gives the number of queries saved in IQT) in Area 1 of Figure 4.2 for scenario 2. This is because, in scenario 2, Area 1 has lesser(*prefix2*) levels. Also queries denoted by nodes in Area 2 of the same figure take place using *Product_id* as well as *Item_no* bits in scenario 2, where as in scenario 1 queries take place using only the *Item_no* bits.

In the case of subsequent cycles for scenario 2, all the queries in IQT take place using *Product_id* as well as *Item_no* bits, while in scenario 1, all the queries in IQT take place using *Item_no* bits only.

Figure 6.4 and 6.5 show the percentage reduction in terms of the number of queries, and in terms of the number of bits respectively. Figure 6.6 shows the lower bound on improvement in tag read speed in scenario 2.

6.3 Scenario 3 - Items have different manufacturer ID

Different manufacturer ID means that the tagIDs do not have first *prefix (prefix1/prefix2)* bits common. This is the case when IQT protocol achieves least performance improvement since all the queries take place using the whole tagID. The saving will be in terms of number of queries only and not in terms of number of bits per query.

Since there is no commonality in the tagIDs, there will be no improvement for first

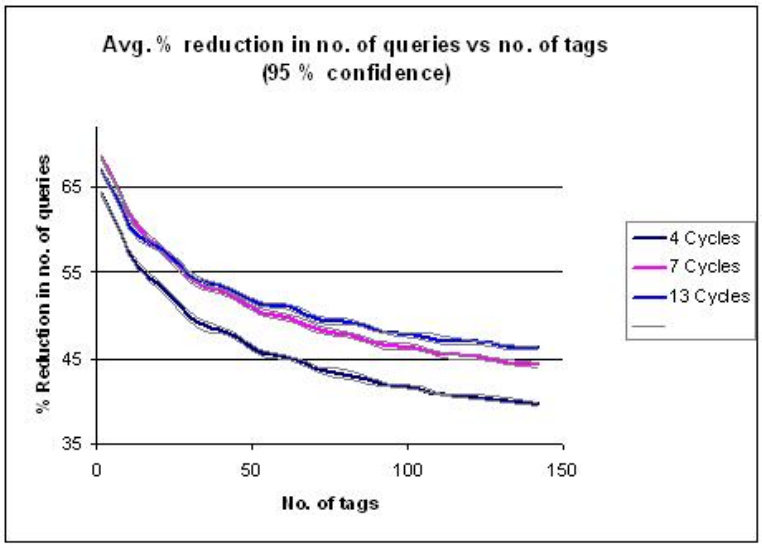

Figure 6.4: % Reduction in number of queries in IQT over QT when tagIDs have first *prefix2* bits common

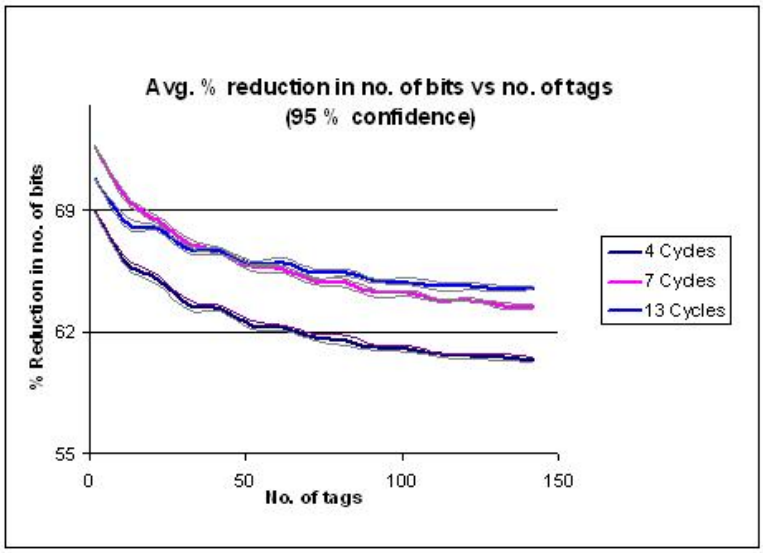

Figure 6.5: % Reduction in number of bits in IQT over QT when tagIDs have first *prefix2* bits common

read cycle, hence IQT protocol gains only in subsequent read cycles. Saving in subsequent cycles depends upon the probability of successful tag reads. Lesser the probability of tag showing up, lesser will be the improvement. This is because IQT protocol needs to explore the subtree for more new tagIDs in each read cycle.

In this scenario, we can directly find the increase in number of tags read per second from the reduction in number of queries. This is shown in Figure 6.7

The saving in number of bits is due to two factors - saving due to number of queries,

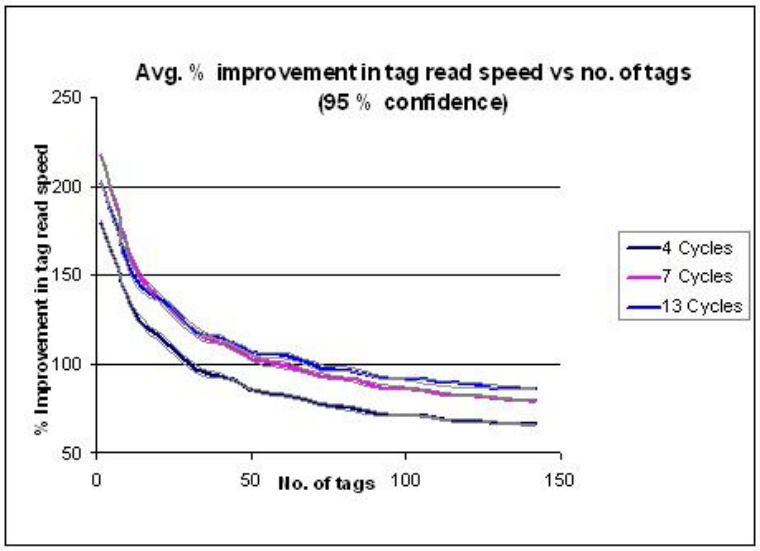

Figure 6.6: % Improvement in tag read speed in IQT over QT when tagIDs have first *prefix1* bits common

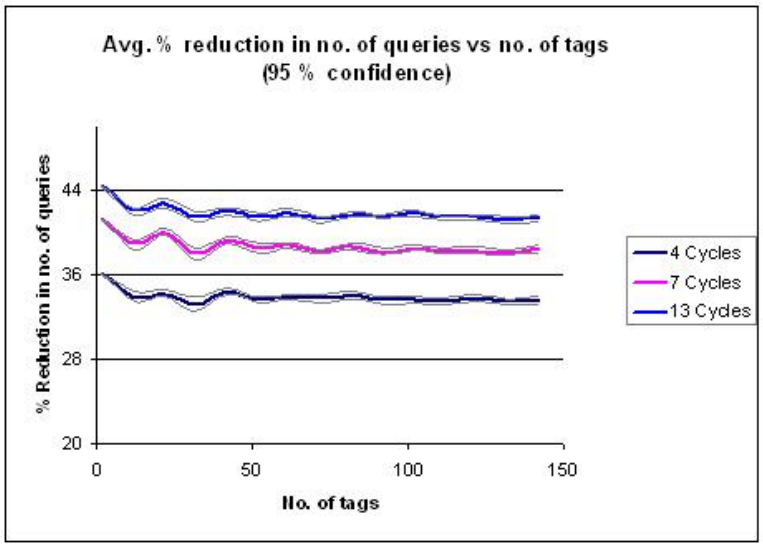

Figure 6.7: % Reduction in number of queries in IQT over QT when tagIDs are random and do not have any common *prefix* bits

and saving in number of bits transmitted per query. In this case, since the entire tagID is used in each query, the reduction is due to the first factor only. Hence the graphs for percentage reduction in number of queries, and that for percentage reduction in number of bits are similar.

The curves are approximately parallel. This is because there is no extra overhead to perform invert query in case of IQT (that was there in other scenarios). In this case invert query will fail in the first time only, and we need not try it again. Secondly we are saving

quite less as compared to the other cases, because we are saving in terms of queries only and also there is no saving in Area 1 of Figure 4.2.

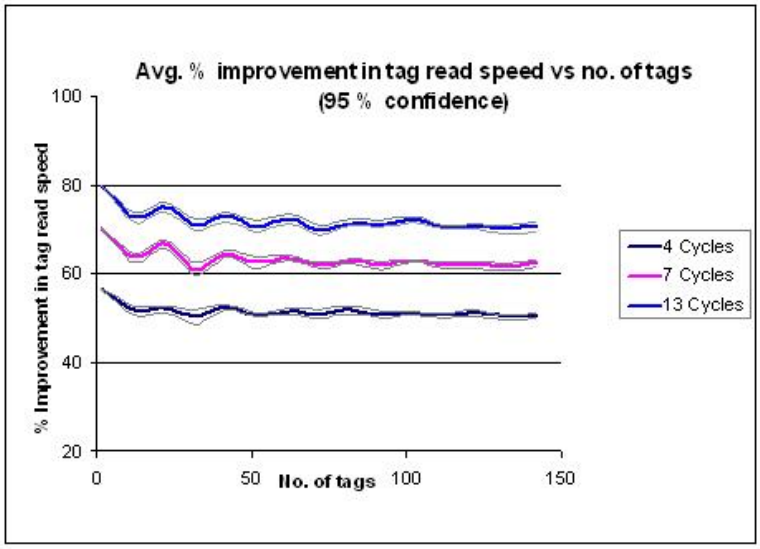

Figure 6.8: % Improvement in tag read speed in IQT over QT when tagIDs are random and do not have any common *prefix* bits

Since there is no extra reduction in the number of bits per query, the improvement in tag read speed can be directly calculated from the reduction in queries. This is shown in Figure 6.8.

6.4 Effect of tag read probability on the performance improvement of IQT over QT

Lesser the tag read probability, lesser will be percentage improvement in the efficiency of IQT with respect to QT. This is because, if the tag read probability is less, more new tags will show up in each of the subsequent cycles. This leads to exploring the nodes in the subtree corresponding to the new tags, hence more queries.

Figure 6.9 shows the effect of tag read probabilities on the tag read speed. In this case tagIDs are totally random and they have first *prefix1* bits common. For simulation, we consider that ten read cycles per read process are required to identify the tags.

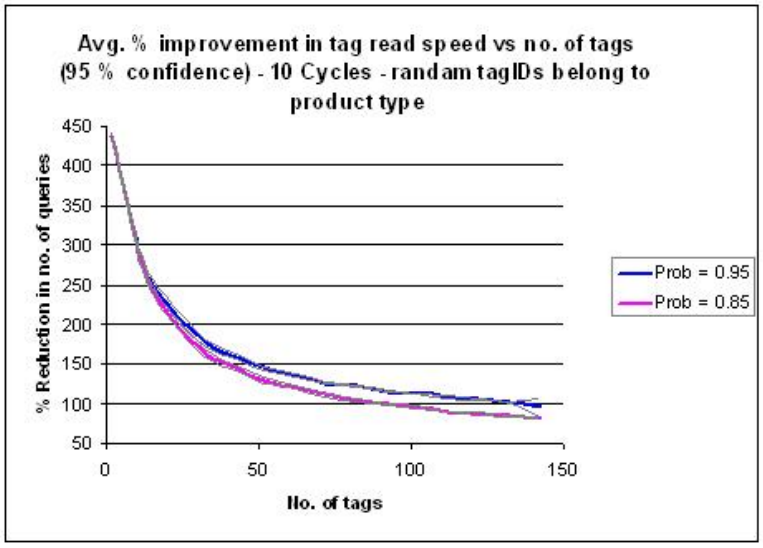

Figure 6.9: % Improvement in tag read speed in IQT over QT when tagIDs are random and do not have any common *prefix* bits

Chapter 7

Conclusions and Future Work

We have presented IQT, an efficient Query Tree based tag identification protocol. IQT is suitable for readers deployed in godowns, exclusive showrooms and shipment points, and large malls etc., where a single lot has similar items (with same product ID and/or vendor ID). The protocol exploits the fact that tags may have common prefixes. IQT also uses the history of read cycles to further optimize the read process. Using history of read cycles leads to fair amount of saving even in the scenario where tagIDs are totally random, that is they do not have same manufacturer or product type.

We have studied the performance of IQT in different tagID distribution scenarios. We have presented the worst case analysis and performed simulations for the other cases.

IQT can even be used in the moving belt scenario with a small feasible constraint. The belt should move discreetly instead of moving continuously and the different chunks should be separated by small RF absorbing partitions. When the reader completes the desired number of read cycles, the belt is moved ahead so that the next chunk is in the range of reader.

As future work, the ranking criteria of prefixes stored in the prefix pool, used to guess the prefix in first read cycle, needs to be tuned to provide better results and make this optimization applicable in more general scenarios. We can further look into how this reader-tag protocol can be adopted along with other reader-reader communication protocols to achieve better read rates.

Acknowledgements

I would like to express my gratitude to my advisors **Prof. Anirudha Sahoo** & **Prof. Sridhar Iyer**. I am thankful to them for their precious guidance, motivation and support.

I would like to thank **Raghuraman Rangarajan** and **Srinath Perur** for discussing and suggesting various ideas that helped in the evolution of this thesis.

Last, but certainly not the least, I would like to thank the people that mean a lot to me, my family. I deeply thank them for their unfailing encouragement and support.

I would also like to thank whole of KReSIT and IIT Bombay family for making my stay here wonderful. Thank you **IIT Bombay**.

Naval Bhandari
KReSIT,
IIT Bombay
July 6, 2006

Bibliography

[1] A basic introduction to RFID technology and its use in supply chain. Technical report, Laran Technologies, January 2004.

[2] K. Finkenzeller. *RFID Handbook : fundamentals and applications in contactless smart cards and identification.* Chichester : John Wiley, Leipzig, dritte edition, 2003.

[3] Radio frequency identification - a basic primer. White Paper, AIM Inc WP-98/002R2, August 2001.

[4] D.W. Engels and S. Sarma. The Reader Collision Problem. *IEEE International Conference on Systems, Man and Cybernetics*, 3:6, 2002.

[5] F. Zhou, C. Chen, D. Jin, C. Huang, and H. Min. Evaluating and optimizing power consumption of anti-collision protocols for applications in RFID systems. In *ISLPED '04: Proceedings of the 2004 international symposium on Low power electronics and design*, pages 357–362, New York, NY, USA, 2004. ACM Press.

[6] C. Law, K. Lee, and K. Siu. Efficient Memoryless Protocol for Tag Identification. *Proceedings of the ACM International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 75–84, 2000.

[7] H. Vogt. Multiple object identification with passive RFID tags. In *IEEE International Conference on Systems, Man and Cybernetics (SMC '02)*, October 2002.

[8] J. Myung and W. Lee. An Adaptive Memoryless Tag Anti-Collision Protocol for RFID Networks. *Dept of Computer Science and Engineering, Korea University, Seoul, Korea*, 2004.

[9] J. Waldrop, D.W. Engels, and S. Sarma. Colorwave : An Anticollision Algorithm for the Reader Collision Problem. *IEEE International Conference on Communications*, 2:1206 – 1210, 2003.

[10] M.B. Cozzens and F.S. Roberts. T-Colorings of Graphs and the Channel Assignment Problem. *Congressus Numeruntium*, pages 191–208, 1982.

[11] K. Ho Junius. Solving the reader collision problem with a hierarchical q-learning algorithm. Master's thesis, Massachusetts Institute of Technology, February 2003.

[12] S. Birari and S. Iyer. PULSE: A MAC protocol for RFID networks. *1st International Workshop on RFID and Ubiquitous Sensor Networks (USN), Nagasaki, Japan*, , Dec 2005.

[13] Electronic product code. `http://www.epcglobalinc.org`.

[14] S. Sarma, D. Brock, and D. Engels. Radio Frequency Identification and the Electronic Product Code. *IEEE Micro, v.21 n..6*, 2001.

[15] R. H. Don and W. Cliff. Analysis of Tree Algorithms for RFID Arbitration. *IEEE International Symposium on Information Theory*, 1998.

Appendix A

Definitions and Key Terms

- **EPC Code** : Like universal bar code, EPC code is standard Tag Id assignment code so that they are globally unique.Following is the structure of EPC Code.
  - EPC version : Manufacturer ID : Product Type : Item ID
- **Read Cycle** : Read cycle refers to the set of queries required to read all the tags that are in range of reader at a particular time. In a single read cycle, a reader may not be able to read all the tags due to various factors like collisions, interference etc. So multiple read cycles are required to improve reliability.
- **Read Process** : It refers to the set of multiple read cycles performed by reader to ensure reliability.
- **Query** : It refers to the command containing some prefix sent by the reader, in response to which all the tags with matching prefix reply back with their tag IDs.
- **Invert Query Command** : It is just the opposite of the normal query command. Reader sends some prefix with the command and all the tags having prefixes different from that sent by reader are supposed to reply to this command. This command is issued to ensure that all the tags, that are read by the reader in the current read cycle, have same prefix.
- **Prefix Pool** : It is a set of frequently occurring prefixes sorted according to rank based upon the frequency of their occurrence, and how recently the prefix was used in previous read cycles.
- **prefix1** = First (EPC + Manufaturer_id + Product_id) bits
- **prefix2** = First (EP + Manufaturer_id) bits