Intelligent Query Tree (IQT) Protocol to Improve RFID Tag Read Efficiency

Naval Bhandari, Anirudha Sahoo and Sridhar Iyer
Kanwal Rekhi School of Information Technology
Indian Institute of Technology, Bombay, Powai, Mumbai, India, 400076
email: {naval, sahoo, sri}@it.iitb.ac.in

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. In this paper, we propose an Intelligent Query Tree (IQT) Protocol for tag identification that exploits specific prefix patterns in the tags and make 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 suitable for readers deployed in exclusive showrooms, shipment points of big malls, where the products may come from same manufacturer and may have same product IDs. 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.

1. Introduction

Radio Frequency Identification (RFID) is a means of identifying objects using radio frequency transmission[3, 7, 2]. 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).

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[6]. We are mainly concerned with Single Reader-Multiple Tags collision. Here multiple tags communicate with the reader. They respond simultaneously and reader is not able to interpret the signal. 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.

1.1. Motivation

RFID mechanism is inherently unreliable. Thus, we need multiple read cycles to improve reliability of tag identification. The conventional tag identification protocols do not have any mechanism to use the information contained in tag IDs to improve tag read efficiency. Hence, there is a need for different tag identification protocol, which can use information about specific prefix patterns in tag IDs so as to speed up the tag read process.

1.2. Problem Formulation

There are practical scenarios where tags have some specific pattern in their tag IDs. 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.

The rest of the paper is organized as follows: Section 2 discusses related work. Section 3 presents our proposed Intelligent Query Tree (IQT) Protocol. Section 4 analyzes the worst case complexity of IQT Protocol and compares it with the complexity of Query Tree (QT) Protocol. Finally, concluding remarks are given in Section 5.

2. Related Work

The IQT Protocol proposed in this paper belongs to the class of Single Reader-Multiple Tag collision resolution protocols. Other important protocols in this category are Binary Tree Protocol[14], Query Tree Protocol[14][9], Framed Slotted Aloha based I-Code Protocol [12] and Adaptive Memoryless Tag Anti-Collision Protocol[10].

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.

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 1 where there are 3 tags with the IDs of 0000, 0010, 1000. In Table 1, the first row corresponds to the first query (with null prefix), to

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 1. The tag identification process of Query Tree Protocol

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.

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.

When multiple readers try to read tags simultaneously, and their reading operation interfere with each other, multiple reader collision avoidance algorithm is used. Important protocols in this category are Colorwave Algorithm [13], T-Colorings based Algorithm[5], Q-Learning Algorithm [8] and Pulse Protocol[4]. Optimizations based upon these protocols may also be used in future, to improve tag read efficiency.

3. Intelligent Query Tree Protocol

Intelligent Query Tree Protocol exploits specific prefix patterns in the tag IDs to reduce the communication overhead between reader and tags. The common prefix in tag IDs 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. Following are its advantages over Query Tree Protocol:

- Reduction in number of bits transmitted by 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.

3.1. Key Terms

- **EPC Code**[1] : Like universal bar code, EPC code is a standard Tag ID assignment code that assigns globally unique IDs to RFID tags. EPC code essentially has four fields: *EPC Version, Manufacturer ID, Product Type ID, 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.
- **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.

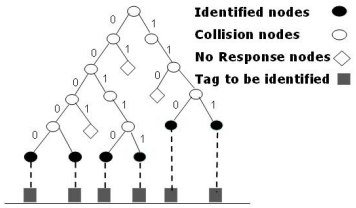

Figure 1. Different types of nodes in query tree

- **No-Response Node** : The leaf node in query tree where no tag ID starting with that prefix is found. In Figure 1, nodes represented by diamond are No-Response Nodes.
- **Identified Node** : The node in the query tree that lead to response from single tag i.e one and only one tag is identified using this prefix. Nodes represented by filled circles in Figure 1 are Identified Nodes.
- **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 1, unfilled circles represent the Collision or Internal Nodes.

3.2. Notations Used: Table 2 shows various notations used in this paper.

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*}
rem	(k − *prefix*) bits

Table 2. Notations Used in this paper

3.3. Working of Intelligent Query Tree Protocol

Mechanism used in IQT to identify tags in the first read cycle is different from 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.

3.3.1. First Read Cycle

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

Step 1 - Try prefixes from Prefix Pool : For the first read cycle, prefix with highest score from prefix pool is selected and Invert Query Command (see Section 3.1) 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 *prefix1* 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*.

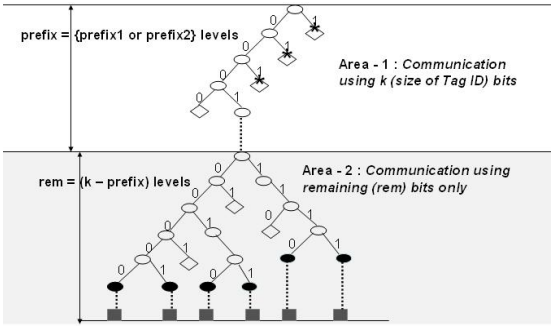

Figure 2. Tag Identification using Intelligent Query Tree Protocol

- 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 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 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*. 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 less 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.

3.3.2. 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 [10]. 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 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 as shown in 3.

Further improvement is possible if all the tags have the same prefix and it matches with the one found in the previous read cycle. In that case 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 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. For subsequent read cycles, the protocol works as follows.

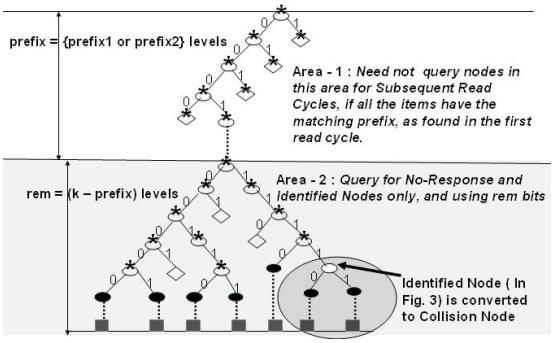

Figure 3. Query using candidate queue entries for subsequent read cycles

Run invert query with maximum matched prefix (*prefix1 or prefix2*) found in the first read cycle.

- 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).

4. 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 Intelligent Query Tree Protocol is compared with normal Query Tree 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.

4.1. Where IQT Protocol gains over QT Protocol ?

Figure 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 2 denote the reduction in number of queries in IQT Protocol over normal Query Tree Protocol. After reaching *prefix*th level, the Intelligent Query Tree 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 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 3 describes the saving for subsequent read cycles using Intelligent Query Tree Protocol over Query Tree Protocol. Again, optimization is due to two factors.

- For the nodes in Area-1 of Figure 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 just one Invert Query Command. 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 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.

4.2. Comparisons of IQT Protocol with QT Protocol 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 indirectly 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.

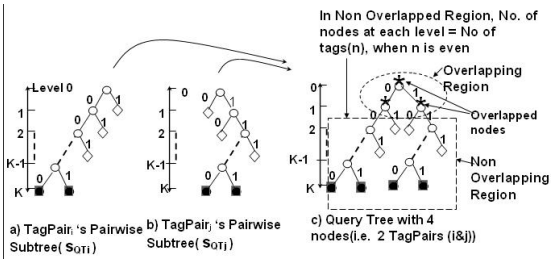

Figure 4. Overlapped and Non Overlapped Regions in query tree

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 4(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 4(c), where L is given by*

$$L = \text{maximum level at which } \forall i, \exists j | S_{Q_{T_i}} \bigcap S_{Q_{T_j}} \neq \emptyset. \quad (1)$$

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 1. In other words, in this region, there is no overlap among all the Pairwise Subtrees, as shown in Figure 4(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.*

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.*

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.*

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.*

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.*

Proofs of above lemmas can be found in [11].

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

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

Proof: 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 and 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:

$$\begin{cases} 2^0 + 2^1 + \ldots + 2^{\lfloor log_2(n-2) \rfloor} \text{ for } n>2 \\ = 2^{\lfloor log_2(n-2) \rfloor+1} - 1 \text{ for } n>2 \end{cases} \quad (3)$$

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:

$$2^{\lfloor log_2(n-2) \rfloor+1} - 1 + (k - \lfloor log_2(n-2) \rfloor) * (n) \text{ for } n>2 \quad (4)$$

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:

$$2^{\lfloor log_2(n-2) \rfloor+1} - 1 + (k - \lfloor log_2(n-2) \rfloor) * (n-1) \text{ for } n>2 \quad (5)$$

■

4.2.1. 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 calculations 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 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:

$$\begin{cases} 2 * \textit{prefix} + 1 + 2^{\lfloor log_2(n-2) \rfloor+1} - 1 \\ + (rem - \lfloor log_2(n-2) \rfloor) * (n) \text{ for } n>2 \end{cases} \quad (6)$$

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

$$\begin{cases} (2 * \textit{prefix} + 1 + 2^{\lfloor log_2(n-2) \rfloor+1} - 1 \\ + (rem - \lfloor log_2(n-2) \rfloor) * (n)) * k \text{ bits.} \end{cases} \quad (7)$$

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, one 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: *no_of_guesses*$+1+2^{\lfloor log_2(n-2) \rfloor+1} - 1 + (rem - \lfloor log_2(n-2) \rfloor) * n$ 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 *no_of_guesses* $+1) * k + (2^{\lfloor log_2(n-2) \rfloor+1} - 1 + (rem - \lfloor log_2(n-2) \rfloor) * n) * rem$ bits. So, performance improvement of IQT over QT, in terms of number of queries, is given by

$$(2 * \textit{prefix} - \textit{no\_of\_guesses}) \quad (8)$$

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

$$\begin{cases} (2 * \textit{prefix} - \textit{no\_of\_guesses}) * k \\ + (2^{\lfloor log_2(n-2) \rfloor+1} - 1 + (rem - \lfloor log_2(n-2) \rfloor) * n) * \\ (k \text{ - } \textit{rem}) \text{ bits.} \end{cases} \quad (9)$$

***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

$$\begin{cases} (2^{\lfloor log_2(n-2) \rfloor+1} - 1 + (rem - \lfloor log_2(n-2) \rfloor) * n) * \\ (k - rem) - max\_tries * k \text{ bits,} \end{cases} \quad (10)$$

since we waste max_tries queries to guess the prefix.

Table 3 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.

4.2.2. 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 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 $2 * \textit{prefix} + 1$ queries in Area-1, but one Invert Query Command needs to be executed to confirm that all the tags have same prefix. Hence, number of queries saved in Area-1 will be $2 * prefix$. 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) are given by $[(2^{\lfloor log_2(n-2) \rfloor + 1} - 1) + (rem - \lfloor log_2(n-2) \rfloor) * n + 1]/2$, 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

$$\begin{cases} 2 * prefix \\ +[(2^{\lfloor log_2(n-2) \rfloor + 1} - 1) + (rem - \lfloor log_2(n-2) \rfloor) * n + 1]/2 - 1 \end{cases} \quad (11)$$

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

$$\begin{cases} (2 * prefix + [(2^{\lfloor log_2(n-2) \rfloor + 1} - 1) \\ +(rem - \lfloor log_2(n-2) \rfloor) * n + 1]/2 - 1) * k \\ +[(2^{\lfloor log_2(n-2) \rfloor + 1} - 1) + (rem - \lfloor log_2(n-2) \rfloor) * n + 1]/2 \\ *(k - rem) \text{ bits.} \end{cases} \quad (12)$$

Column number six and seven in Table 3, show 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. 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.

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
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
150	63.37	38.00	60.70	37.05	80.54	68.73

Table 3. Percentage reduction in number of bits

5. 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 to 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 prefixes in common. IQT also uses the history of previous read cycles to further optimize the read process. The ranking criteria of prefixes stored in the prefix pool needs to be tuned to provide better results and makes this optimization applicable in general scenario. We presented a comparative complexity analysis of IQT and Query Tree protocol and have shown performance improvement.

We intend to study the performance of IQT in different tag ID distribution scenarios. We would like to look at how this reader-tag protocol can be adopted along side other reader-reader communication protocols to achieve better read rates.

References

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

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

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

[4] 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.

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

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

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

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

[9] 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.

[10] 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.

[11] A. Sahoo, S. Iyer, and N. Bhandari. Improving rfid system to read tags efficiently. `http://www.it.iitb.ac.in/research/techreport/reports/18.pdf`, 2006.

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

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

[14] 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.