An Optimized Query Tree Algorithm in RFID Inventory Tracking –A case Study Evidence

Tanvi Agrawal[1], P.K.Biswas[2] and A.D.Raoot[3]

**[1] ITM, NITIE, Vihar Lake, Powai
Mumbai, Maharashtra 400087, India**

**[2] ITM, NITIE, Vihar Lake, Powai
Mumbai, Maharashtra 400087 , India**

**[3] ITM, NITIE, Vihar Lake, Powai
Mumbai, Maharashtra 400087 , India**

Abstract

One of the challenged technical issue faced in inventory tracking applications with Radio frequency identification(RFID) implementation environment, is the collisions due to responses from multiple tags on the shared channel between reader and the tags, which consumes further the energy as well as the delay in tag identification process. To avoid collisions by some anti-collision algorithm means, may create large amount of data, adding complexity at the circuit level and also consumes energy for on-going transmissions. Therefore an optimized anti-collision algorithm can be designed to avoid the collisions with minimal amount of data transmissions for lowering the communication overhead as it is critical for wireless battery operated devices. In this paper author has proposed an optimized query tree anti-collision algorithm. It uses minimal number of bits to resolve the collision issue by making groups and solving collisions group by group by utilizing the same query sent by reader in the very first round.

Keywords: *Query Tree Algorithm, Inventory Tracking, Radio Frequency Identification (RFID).*

1. Introduction

Radio Frequency Identification (RFID) system is a form of wireless technique used to identify physical objects [4]. RFID technology improves the potential benefits of supply chain management through reduction of inventory losses, increase of the efficiency, speed of processes and improvement of information accuracy [1] as compared to old BARCODE technology. Now-a-days RFID is widely being used in supply chain for inventory tracking applications, which employs large and densely packed tag populations.

The key issue faced in large scale RFID systems' applications such as inventory tracking, is the Collisions taking place between readers and the tags creating huge amount of data which exaggerates the problem in interpretation of useful data for full tag identification process of tags [3].These much number of transmissions further affect the two parameters critically in i.e. time and energy consumed in the process of identification [5]. So an efficient anti-collision algorithm is the need of hour that avoids unnecessary transmissions without losing useful information.

RFID systems can be classified into two types as active and passive systems and both are having its own requirement for algorithm to be energy efficient and optimized. In case of passive tag systems as tag extracts the energy from the reader to communicate back, so battery of the readers may deplete at much faster rate, as being continuously used for providing energy while in case of active tag systems tags itself are having the sensors and battery so tuple of data is consumed in providing the tag related information to the reader leading to energy consumption and also complexity at algorithm level, so a tag anti-collision algorithm could be developed to be optimized leading to energy efficient solution. Further enormous consumption of energy may add up to the maintenance cost, because of batteries of wireless devices viz. reader in case of passive system and tags in case of active systems, in multiple readers and tags implementation environment present in any retail store or warehouse, in inventory tracking application. One of the ways to achieve energy efficiency at algorithm level is, by minimizing the number of bits transacted between readers and the tags, product identification process fulfillment, which may lead to energy efficient and optimized solution.

Here in this paper author has proposed a new Query Tree Anti-Collision algorithm which utilizes fewer numbers of bits than the previously existing query tree algorithm making it more efficient and optimized. A case study of a retail store has been considered here, showing the practical working of the algorithm, which comprises six sections, having some number of tagged items (items 1 to 10) needed to be identified, to get the idea of its presence.

2. Background Work

In Retail store applications where a number of tagged items are present at the same time to be tracked, collisions may occur due to simultaneous answers of tags in this process. If multiple tags and readers are present then responses by them at the same time on the shared channel creates collision problem.

Collisions can further be classified into two (a) Tag collisions (b) Reader collisions. Between the two, tag collisions can’t be avoided completely, due to limited computational capability, while the later one may utilize the carrier sensing for collision avoidance. Tag anti-collision algorithms can be classified as follows:

(1) ALOHA based algorithms(Probabilistic Algorithms)
(2) Tree based algorithms(Deterministic Algorithms)

In ALOHA based algorithm, tag transmits its ID in a slot, also known as a time interval, assigned in the Query command by the reader in the Query through a parameter Q[6] ,and then tags choses the slot by generating any number randomly between 0-2^Q-1, finally the tags having 0 as slot count, is going to answer first its ID to reader. The slot is chosen randomly in every cycle. ALOHA based algorithms are simple and faster but works efficiently when the tags population size is small [6] and also faces a problem known as “tag starvation”, due to which a particular tag may not be identified, within a time bound making this algorithm probabilistic in nature. Further division of ALOHA on the basis of choice of Frame (group of slots) is given below.

(i) Frame slotted ALOHA (Tags transmits in the chosen slot selected randomly in every cycle or Frame)
(ii) Dynamic Frame slotted ALOHA (Frame length or number of slots are dynamically changed in every cycle based on the tags responses (collisions, no answer and single tag answer))

Tree based algorithms are deterministic in nature which guarantees the identification of each and every tag within time bound. Tree based algorithms can be classified in following two broad categories.

(i) Binary Tree Algorithm
(ii) Query Tree Algorithm

In binary tree algorithm, firstly all the tags set their counter value as zero in the starting then the reader sends the query command and tags randomly decrements the counter value finally the tags having counter value as zero answers its ID to reader. Now if collision occurs due to multi tags answer then reader split the tags into two sets, and whenever the collision occurs, it resolves the tag collisions completely in single set one by one, ensuring that all tags get identified at the end of the identification cycle. Query tree algorithm is memoryless in nature irrespective of binary tree algorithm, means that it is not required for the tags to remember the counter value to answer for as the case with binary tree, in this case reader sends the prefix first (some user defined number of bits) and only those tags, which have their portion of ID matched with answers first to reader as 8 bit response. Whenever the collision occurs in a cycle, one bit longer prefix is sent by reader as the query in corresponding cycle and tags answers the next 8 bits in response to get identified. During the process completion of identification in tree based algorithm, a big deal of energy and time is consumed in terms of transmitted bits as compared to ALOHA based Algorithms.

In some applications like Inventory tracking one requirement is that multiple products present in large numbers, is to be tracked within certain predefined time period, as any single unidentified item over a period of time, may lead to loss. This goal can be achieved efficiently through deterministic tree based algorithms with an additive limitation of massive time and energy consumptions. Hence in this paper an optimized query tree algorithm has been proposed, availing all the benefits of tree based algorithms, and attempts to minimize the limitations associated with previous existing query tree algorithm. Existing query tree can further be improved, if the same query sent by the reader can be reutilized, by acquiring the information from tags responses, which can save the time as well as reduce the number of bit transactions to give an optimized solution.

3. Case Study Evidence

A retail store case study having 1-10 items of different categories kept in 6 different sections has been chosen for analysis. As each item in different section is having a RFID tag affixed to it. Table 1 shows the retail store outlet.

Table 1 Retail store overview

Retail Store[Retail bazaar]	
Section-I(Food Items) Item1- .. Item10-	Section-II(Cosmetics) Item1- .. Item10-
Section-III(Apparels) Item1- .. Item10-	Section-IV(Furniture) Item1- .. Item10-
Section-V(Electronics Items) Item1- .. Item10-	Section-VI(Glossary) Item1- .. Item10-

In Table-2 a sample set of 10 tag Ids for EPC class-1 standard of 96 bits long length is shown for simulation purpose of algorithm simulation.

Table 2 Example set of 10 Tag Id's for 1-10 items in a section

Item no.	***Tag ID number***
Item 1	'010111000100001100110010010001000001110000101111010111010110000000001001000100111000101010000000'
Item 2	'000010100100100100110010011000110100110100101100000111100100010101001001001110110110010000000000'
Item 3	'001001110101000101011101001100010101100100010100010100010010001000111000010101010000001100000000'
Item 4	'001010010010111110010001100010101010000100010000100110010010110010001000000110101000101100000000'
Item 5	'001000010100111000101100010000110100110001010010010011100101001000100101010001110101000000000000'
Item 6	'010000100101110000110100010000110001110100110000001000000010110000111110010101010000001000000000'
Item 7	'111000100101000100010101010101110000000000011010001001000001110100111010001111010001101000000000'
Item 8	'110111000101001000001000000100100100100100010010010111110101000001011100001010010011101000000000'
Item 9	'100011000100110100000001001000011010110110110000100101110000111110001101100110110010111110000000'
Item 10	'100100000110010001001101001010110100001001001001010011010100000000010000000011000011110000000000'

.

4. Simulation Environment

MATLAB 7.0 is used for algorithm evaluation. Assuming tags (Active tags) having 96 bits long id (as per EPC standards), in conjunction with single reader for inventory tracking is used in retail store for simplification of the problem.

5. Proposed Algorithm

Our proposed Algorithm is similar to Query Tree Algorithm in that it sends the prefix from reader but requires the memory unlike Query tree, to remember the particular bit position which suffers collision so that further query can be sent to avoid collisions occurred at that particular bit position using the same query by just flipping the bits at the position where collision occurred in next round. Proposed Algorithm works in following manner.

1. Firstly Reader broadcasts the Make group command and k as user defined number so that the tag responds with only k bits in subsequent rounds.

2. Then all the tags make two groups as Group 1 and Group 0 ,according to reader command, according to the initials of the ID as '1' and '0' respectively.

3. Then reader sends Query command firstly to the group 1 by XORing the k bits from the available database of tag sets population being inventoried as prefix.

4. Tags send k bit responses to the reader and then two subgroups are made according to the responses sent by the tags one subgroup of the tags which are having same collided bit positions (bits at the same positions are collided) and another one are the tags which are not having the same collided bit positions.

5. Then again the query command is within two subgroups one by one subgroup 1 having different collided bit positions is sent the same command as in previous round, by just flipping the number of bits at corresponding colliding bit position as prefix, and the subgroup 2 an alternate query is sent by Xoring the next k bits.
6. If at the step number 5 all tags are identified then tags send the rest of the bits (user defined) of ID number for full tag identification and go for step 7 otherwise another query command as prefix by Xoring the next k bits from the database of tag sets is sent to the tags and steps 4 to 6 are repeated again for group1.
7. Reader sends the Acknowledgement to those tags which are identified fully.
8. Steps 1 to 7 are repeated for group 2 if all tags from group 1 are identified.
Flow chart of the algorithm has been given in Figure 1

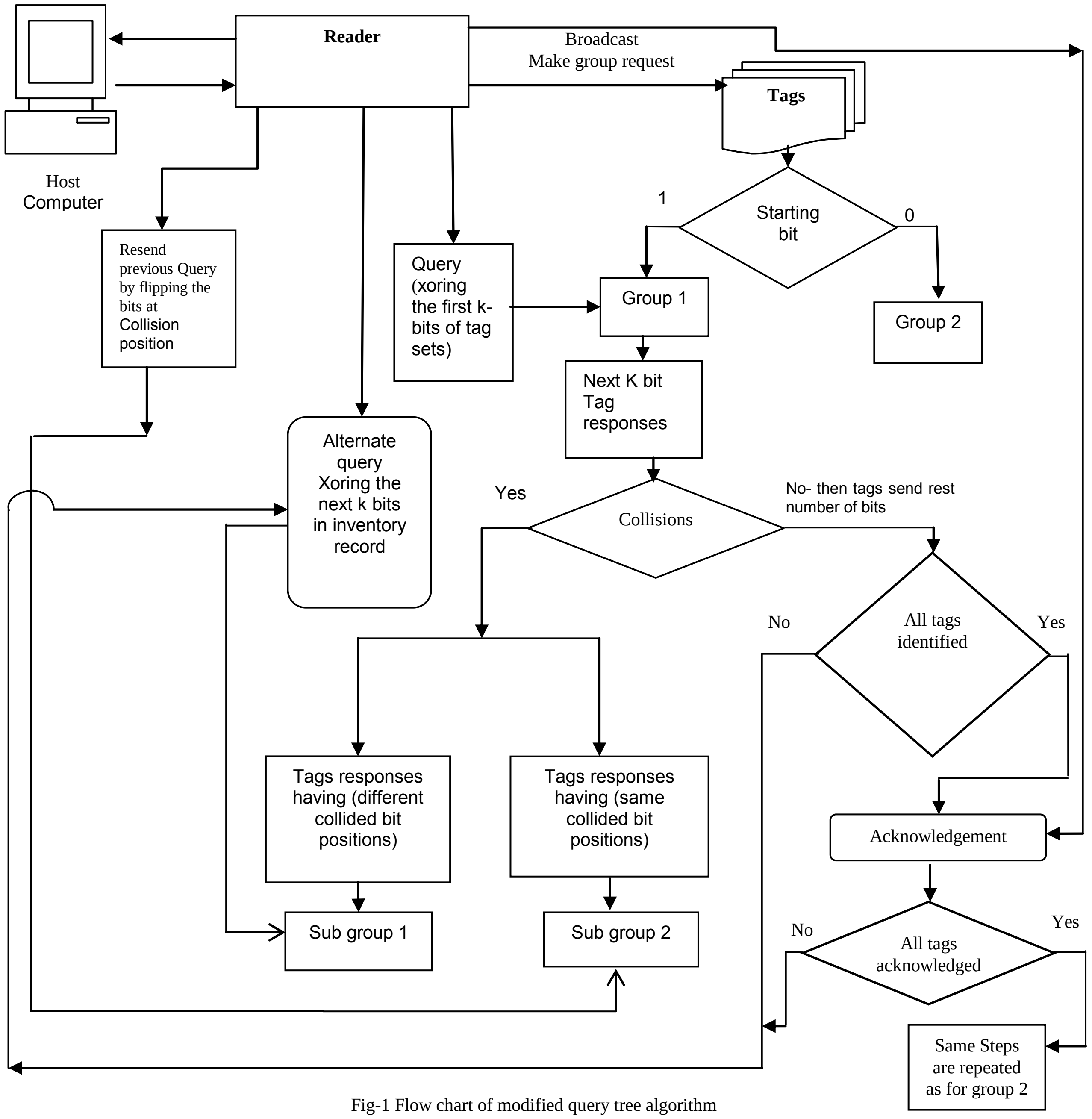
Reader
Broadcast
Make group request
Tags
Host Computer
Resend previous Query by flipping the bits at Collision position
Query (xoring the first k-bits of tag sets)
1
Starting bit
0
Group 1
Group 2
Next K bit Tag responses
Alternate query Xoring the next k bits in inventory record
Yes
Collisions
No- then tags send rest number of bits
No
All tags identified
Yes
Tags responses having (different collided bit positions)
Tags responses having (same collided bit positions)
Acknowledgement
Sub group 1
Sub group 2
No
All tags acknowledged
Yes
Same Steps are repeated as for group 2

Fig-1 Flow chart of modified query tree algorithm

Performance analysis of proposed algorithm vs. existing algorithm

According to the above proposed algorithm, firstly the reader broadcasts the make group command and user defined k bits to inform the tags, to answer only the k bits in response and make groups of two, as group 1 and group 2. Tags make the group into two according to the starting bit of their id numbers as '0' or '1'. After making two groups, command is sent by reader, one by one, within each group, to resolve the issue within one group first. Query by the reader is sent by Xoring the k ID bits of tags in group 1 as information is available with company database. Then the tag responses of k bits are judged by the reader for every bit position whether it suffered from collision or not, if no collision case then one tag may get identified and reader issues acknowledgement command to tag in next round otherwise in collision case two conditions may arise (i) collision occurrence at the same bit positions for more than one tag (ii) collision occurrence at different bit positions. So accordingly two subgroups (subgroup 1 and 2 respectively) may be formed and in subgroup2 case the same command is used in subsequent rounds utilizing the information gathered from the tag responses only by complementing the bits which suffered collisions until all tags from the same subgroup get identified while for subgroup1 an alternate query by Xoring the next k bits of the tag sets from this subgroup is sent and abovementioned steps are repeated. Same process steps are followed by group 2. Here for this algorithm simulation k (user defined number) is taken as 4. Performance analysis between existing and proposed algorithm shows better efficiency in identification with minimum number of transmissions between reader and tags (table-2 and table 3). Graph (fig-2 and fig -3) shows number of collisions in different rounds in modified and existing algorithms here x denotes collisions occurred at that position. Table-2 is the round wise detail of transmitted bits in optimized query tree algorithm and in Table-3 same information about existing query tree algorithm has been shown. Further table-4 gives the comparative performance evaluation between the two above mentioned algorithm.

Table-2: Optimized query tree algorithm

Round	Status	Reader Command / Query	Tags Response
1	Nil	Reader broadcasts –'Make Group command and , k = 4'	Make two groups (Group1-Tags whose ID bit is starting from '0' (Tag 1,Tag2,Tag 3,Tag 4,Tag 5,Tag 6) Group 2- whose ID bit is starting from '1' (Tag 7,Tag 8,Tag 9,Tag 10)
2	NIL	'0111'(Next by Xoring the 2nd bit to 2nd+kth bit of tag ID bits of all tags from Group 1)	Tag1-'xx11' (k=4 bits response starting from 2nd bit to 2nd+kth bit) Tag2-'0xx1' (k=4 bits response starting from 2nd bit to 2nd+kth bit) Tag3-'01xx' (k=4 bits response starting from 2nd bit to 2nd+kth bit) Tag4-'01x1' (k=4 bits response starting from 2nd bit to 2nd+kth bit) Tag5-'01xx' (k=4 bits response starting from 2nd bit to 2nd+kth bit) Tag6-'xxxx' (k=4 bits response starting from 2nd bit to 2nd+kth bit)
3	Tag 1 identified	'1011'('0111'Query is utilized by just complementing the corresponding tag bits at 'x' position)	Tag1-'1000'(matched prefix case, hence next 4 bits response)
4	Tag 2 identified	'0001'('0111'Query is utilized by just complementing the corresponding tag bits at 'x' position)	Tag2-'0100'(matched prefix case, hence next 4 bits response)
5	Tag 4 identified	'0101'('0111'Query is utilized by just complementing the corresponding tag bits at 'x' position)	Tag 4-'0010'(matched prefix case, hence next 4 bits response)

Round	Status	Reader Command / Query	Tags Response
6	Tag 3 identified	'0100'('0111'Query is utilized by just complementing the corresponding tag bits at 'x' position)	Tag3-1110(matched prefix case, hence next 4 bits response)
7	Tag 5 identified	'1101'('0111'Query is utilized by just complementing the corresponding tag bits at 'x' position)	Tag5- '0010'(matched prefix case, hence next 4 bits response)
8	Tag 6 identified	'1000'('0111'Query is utilized by just complementing the corresponding tag bits at 'x' position)	Tag6-'0100'(matched prefix case, hence next 4 bits response)
9	NIL	'0100'(Next Query by Xoring the 2nd bit to 2nd+kth bit of tag ID bits of all tags from Group 2)	Tag 7-'x100'(k=4 bits response starting from 2nd bit to 2nd+kth bit) Tag 8-'xxxx'(k=4 bits response starting from 2nd bit to 2nd+kth bit) Tag 9-'0x0x'(k=4 bits response starting from 2nd bit to 2nd+kth bit) Tag 10-'0xx0'(k=4 bits response starting from 2nd bit to 2nd+kth bit)
10	Tag 7 identified	'1100'	Tag 7-'0100'
11	Tag 8 identified	'1011'	Tag8-'1000'
12	Tag 9 identified	'0001'	Tag9- '1000'
13	Tag10identified	'0010'	Tag 10-'0000'

No of rounds performed : 13
No. of transmitted bits from reader : 48
No. of transmitted bits from tags : 80

Table-3: Existing query tree algorithm

Number of Round	*Identified Tags*	*Reader Query*	*Tag Response*	*Status*

1	NIL	'0'	Tag1-10111000(next 8 bits)
			Tag2-00010100(next 8 bits)
			Tag3-01001110(next 8 bits)
			Tag4-01010010 (next 8 bits)
			Tag5-01000010(next 8 bits)
			Tag6-10000100 (next 8bits)
				Collision

2	NIL	'00'	Tag2- 00101001(next 8 bits)
			Tag3-10011101(next 8 bits)
			Tag4-10100100(next 8 bits)
			Tag5-10000101(next 8 bits)
				Collision

3	Tag1	'01'	Tag 1-01110001(next 8 bits)	Tag 1 identified (No collision)
4	NIL	'10'	Tag 9- 00110001(next 8 bits) Tag 10- 01000001(next 8 bits)	Collision
5	Tag 2	'000'	Tag2-01010010(next 8 bits)	Tag 2 identified (No collision)

6	NIL	'001'	Tag3 -00111010(next 8 bits)
			Tag4-01001001(next 8 bits)
			Tag5-00001010(next 8 bits)
				Collision

7	Tag 6	'010'	Tag 6-'00010010'(next 8 bits)	Tag 6 identified (No collision)
8	NIL	'011'	No response	_

Number of Round	*Identified Tags*	*Reader Query*	*Tag Response*	*Status*
9	NIL	'1'	Tag 7-11000100(next 8 bits) Tag 8- 10111000(next 8 bits) Tag 9- 00011000(next 8 bits) Tag 10-00100000(next 8 bits)	Collision
10	NIL	'10'	Tag 9-'00110001'(next 8 bits) Tag 10-'01000001'(next 8 bits)	Collision
11	NIL	'11'	Tag 7-10001001(next 8 bits) Tag 8-01110001(next 8 bits)	Collision
12	NIL	'100'	Tag 9-'01100010'(next 8 bits) Tag 10-'10000011'(next 8 bits)	Collision
13	NIL	'101'	No response	_
14	Tag 8	'110'	Tag 8-11100010(next 8 bits)	Tag 8 identified (No collision)
15	Tag 7	'111'	Tag-7-00010010 (next 8 bits)	Tag 7 identified (No collision)
16	Tag 9	'1000'	Tag 9- 11000100(next 8 bits)	Tag 9 identified (No collision)
17	Tag 10	'1001'	Tag 10-00000110(next 8 bits)	Tag 10 identified (No collision)
18	NIL	'001'	Tag 3-'00001010'(next 8 bits) Tag 4-'01001001'(next 8 bits) Tag 5-'00001010'(next 8 bits)	Collision
19	NIL	'0010'	Tag3- 01110101(next 8 bits) Tag4- 10010010(next 8 bits) Tag5- 00010100(next 8 bits)	Collision
20	NIL	'00100'	Tag3- 11101010(next 8 bits) Tag5- 00101001(next 8 bits)	Collision
21	Tag 3	'001001'	Tag 3- 11010100(next 8 bits)	Tag 3 identified (No collision)
22	Tag 5	'001000'	Tag 5- 01010011(next 8 bits)	Tag 5 identified (No collision)
23	Tag 4	'00101'	Tag 4 -00100101(next 8 bits)	Tag 4 identified (No collision)

No of rounds performed: 23
No. of transmitted bits from reader : 73
No. of transmitted bits from tags : 344

Table 4: Performance comparison table

	Query Tree	**Modified Query Tree**
No. of rounds performed	**23**	**13**
% reduction in transmitted bits from reader(transmitted reader bits from existing algorithm- transmitted reader bits from proposed algorithm)/(transmitted reader bits from existing algorithm)	--	34.24%
% reduction in transmitted bit from tags(transmitted tag bits from existing algorithm- transmitted tag bits from proposed algorithm)/(transmitted tag bits from existing algorithm)	--	76.74%
Throughput= (No. of rounds with successful identification /Total no. of rounds)x100	43.47%	76.92%
Efficiency= Number of transmitted sent in successful identification round(reader+tags)/Total number of transmitted bits(Tags + Reader)	28.53%	62.5%

In following figures graphical representation of comparative performance evaluation between the above mentioned two algorithms has been shown. Figure 4 and 5 are the existing Query Tree algorithm representation wile figure 6 and 7 are the optimized Query Tree Algorithm.

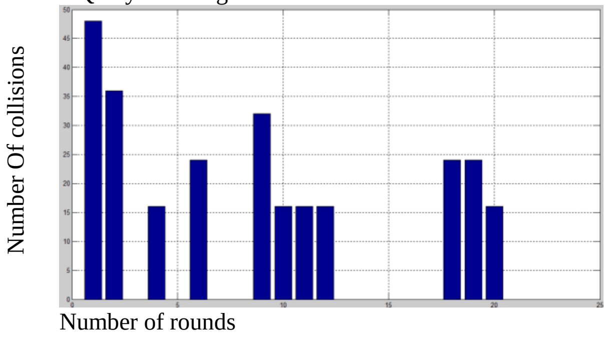

Fig. 4 Number of collisions vs. round (existing Query Tree Algorithm)

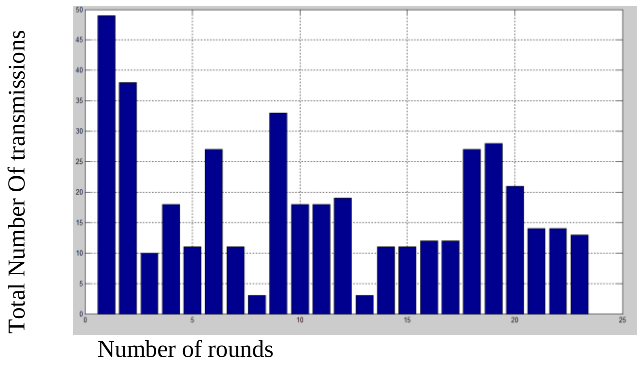

Fig. 5 Number of transmissions vs. round (existing Query Tree Algorithm)

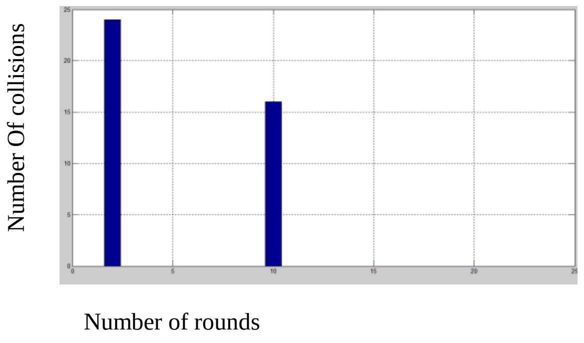

Fig. 6 Number of collisions vs. round (Optimized Query Tree Algorithm)

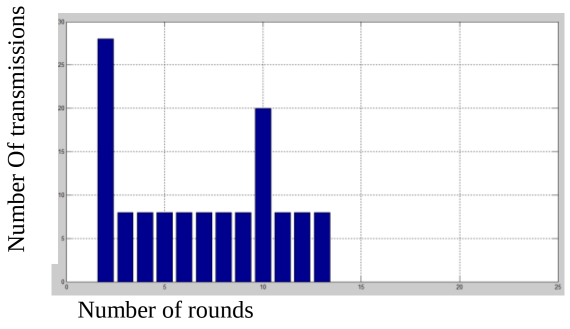

Fig. 7 Number of transmissions vs. round (Optimized Query Tree Algorithm)

6. Conclusions and Future Scope

Performance analysis shows that achieved throughput and efficiency is better as compared to existing query tree algorithm. Number of rounds performed is lesser achieving lesser time required than the case of existing query tree algorithm with minimum number of transmitted bits**.** To identify multiple numbers of tags by reducing the transmitted data with minimum number of collisions, without losing information has become a great issue in RFID Inventory tracking applications. Hence our proposed Algorithm has made an attempt to resolve this issue by optimizing the number of transmission bits, giving energy efficient and faster solution as compared to traditional query tree algorithm. In future this algorithm can be made more practical by utilizing greater number of tags.

References

[1] Aysegul Sarac, Absi Nabil, Dauzere-Peres Stephane, "A literature review on the impact of RFID technologies on supply chain management", Elsevier International Journal of production Economics , pp.78-79, 2010.

[2] Dheeraj K. Klair, Kwan-Wu Chin, Raad and Raad, "An investigation into the energy efficiency of pure and slotted Aloha Based anti-Collision Protocols", IEEE international symposium on world of wireless, mobile and multimedia networks, 2007, pp.1-4.

[3] Derakhshan, R., Orlowska, M.E and Xue Li, "RFID Data Management: Challenges and Opportunities", IEEE International Conference on RFID, 2007, pp.175-182.

[4] Guangsong Yang, Mingbo Xiao and Chaoyang Chen "A simple energy balancing method in RFID sensor Network" IEEE workshop on Anti-counterfeiting, security, identification, 2007, pp.306 – 310.

[5] Lei Pan and Hongyi Wu "Smart Trend-Traversal: A Low Delay and Energy Tag Arbitration Protocol for Large RFID Systems", IEEE Transactions on wireless communications, 2011, vol.-10, pp. 2571- 2573,

[6] Yan Xin-qing, Yin Zhou-ping and Xiong You-lun 'A comparative Study on the Splitting Tree based Protocols for RFID Tag Identification', IEEE journal of ICCS, pp. 332-336, 2008.

[7] Lei Zhu, Tak-Shing and Peter Yum, "A critical survey and analysis of RFID anti-collision mechanism", IEEE communications magazine, 2011, vol.-49, pp. 214-221.