This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the ICC 2007 proceedings.

A Hybrid Query Tree Protocol for Tag Collision Arbitration in RFID systems

Jiho Ryu, Hojin Lee, Yongho Seok, Taekyoung Kwon and Yanghee Choi
School of Computer Science and Engineering
Seoul National University, Seoul, Korea
Email: {jhryu, lumiere, yhseok, tk, yhchoi}@mmlab.snu.ac.kr

***Abstract*—In this paper, we propose a hybrid query tree protocol that combines a tree based query protocol with a slotted backoff mechanism. The proposed protocol decreases the average identification delay by reducing collisions and idle time. To reduce collisions, we use a 4-ary query tree instead of a binary query tree. To reduce idle time, we introduce a slotted backoff mechanism to reduce the number of unnecessary query commands. For static scenarios of tags, we extended the proposed protocol by adopting two phases. First, in *leaf query phase* for existing tags, the interrogator queries *leaf*-nodes directly to reuse query strings in the previous session. Second, in *root query phase* for new arriving tags, the interrogator starts the query process from the *root*-node. Simulation reveals that the proposed protocol achieves lower identification delay than existing tag collision arbitration protocols regardless of whether tags are mobile or not.**

I. Introduction

The Radio Frequency IDentification (RFID) system is a new emerging technology to identify a label which is attached to a product. To make RFID systems practical, the cost of RFID tag is very important. And, an interrogator should identify a number of tags in the reading range in a short time. The RFID tags can be classified as *Passive*, *Semi-passive*, and *Active* tags. Among them, a passive tag is charged by the interrogator's electromagnetic waves, instead of using an internal power supply. Thus, the passive tag is the cheapest solution for RFID systems and we focus on passive tags in this paper. In the following, we just use a *tag* instead of a passive tag.

The most important role of an interrogator is identifying tags (or reading IDs of tags) in its reading range. After that, the interrogator can instruct tags to carry out such as a *read* or *write* command. And a tag is able to perform reading or writing when the interrogator sends the command. Since the tag has no carrier sense ability, tags' responses can be collided. Depending on the number of tags respond to the interrogator, there are three cases at communication between tag and interrogator.

- Collision cycle: Number of tags that respond to the interrogator is more than one. The interrogator cannot identify the ID of tags.
- Idle cycle: No response. It is a waste that should be reduced.
- Success cycle : Exactly one tag responds to the interrogator. The interrogator can identify the ID of the tag.

The performance of RFID system depends on the speed of identifying a number of tags. For this purpose, many *Tag Anti-Collision* protocols [1], [2], [3], [4], [5], [6], [7] are proposed to reduce the number of tag collisions. For reducing the implementation cost of tags, a carrier sensing mechanism such as CSMA/CA protocol cannot be used in tags. At large, there are two types of tag anti-collision protocols, *ALOHA*- [8] and *Tree*-based protocols. ALOHA-based protocols can reduce the collision probability, but they have the tag starvation problem that a particular tag may not be identified for a long time. Tree-based protocols do not cause the tag starvation problem, but they have relatively long identification delay. Therefore, we consider tree-based protocols and seek to reduce the identification delay. Collision resolution algorithms of tree-based protocols are very similar to the binary search algorithm. After each collision, they split the set of tags into two subsets and attempt to recognize the subsets one by one. Binary tree splitting (BTS) protocol [1] and query tree (QT) protocol [2] are the most popular algorithms in the tree-based approach. In the BTS protocol, each tag needs a random generation function to get a random binary number and maintains a counter to arbitrate the schedule of transmission of tags. Thus, the BTS protocol minimizes the number of messages sent by an interrogator compared to the QT protocol but its implementation has more complexity. Hence, we propose to extend the QT protocol by reducing idle and collision cycles.

The QT protocol is the most early and representative tag collision arbitration protocol. In this protocol, each tag corresponds to the leaf-node of a full binary-tree and each node of the tree matches a *query* command of an interrogator. The interrogator starts a query process with an initial string 0 or 1. Assume that the query always investigates 0 first. When the interrogator sends a query with a string, p, if responses of tags (whose IDs from the most significant bit match p) are collided, the interrogator sends query again with a 1-bit longer string, p0, in next cycles. After all the tags with p0 prefixes are recognized, the interrogator will send query with string p1. Thus, internal-nodes of the full binary-tree will be collision cycles and leaf-nodes will be idle or success cycles. This QT protocol, to our best knowledge, is the very first tag collision arbitration scheme based on tree structure. It does not need any memory at tags, and its operation is simple.

This paper proposes a hybrid tag anti-collision protocol that is extended from the QT protocol. The proposed hybrid query tree (HQT) protocol has two features: a 4-ary search tree mechanism and a slotted backoff mechanism. First, the interrogator carries out the 4-ary search tree mechanism. In the 4-ary search tree mechanism, a collided query string next time will be extended by 2-bits unlike of 1-bit in the QT protocol. In this way, we can reduce a number of collisions substantially [9]; that is, more efficient query is possible. Likewise, if we extend the collided query string 3- or 4-bits (8- or 16-ary tree, respectively) instead of 2-bits (4-ary tree), we can reduce even more collisions, but at the same time idle cycles will be also increased. In the literature [10], the authors showed that a 3-ary tree is optimal and in a q-ary tree, if q is greater than 3, the performance (tag reading rate) is decreased as q is increased. Thus, we select a 4-ary tree since a 3-ary tree is impossible in the query tree protocol.

Second, a tag carries out the slotted backoff mechanism before replying to an interrogator. When the 4-ary search tree mechanism is used, the number of collisions will be reduced but the idle cycles will be increased. Thus, we propose to reduce idle cycles by incorporating the slotted backoff mechanism. When a tag responds to an interrogator, it sets its backoff timer using a part of its ID. If there is a collision (multiple tags respond), the interrogator can partially deduce how the IDs of the tags are distributed and potentially reduce unnecessary queries.

Lastly, we also consider the static scenarios of tags. The static scenario means there are two types of tags: new arriving tags in an interrogator's reading range and existing tags which have stayed in the reading range. Adaptive Query Splitting (AQS) [3] is the most recent study on the tag collision arbitration mechanism based on the QT protocol. If the interrogator already knows the IDs of tags (or query strings of tags) which are obtained by the previous queries, it will be stored in a candidate queue and this information can be used to efficiently schedule the next queries. So the more efficient query process is possible with the static scenarios. But, to guarantee the recognition of all tags (including new arriving tags), it needs to query for many idle cycles even though there are no new arriving tags. Thus, we extend the AQS protocol and propose a *Leaf-Root Query* mechanism. In the leaf-root query mechanism, we divide the query process into two phases: *leaf query phase* and *root query phase*. In the leaf query phase, we directly query tags using the history of query strings. Here, we does not query for idle cycles but only for success cycles. In the root query phase, we query new arriving tags separately from existing tags which has not moved out the reading range of the interrogator.

This paper is organized as follows. First, we explain the proposed hybrid query tree protocol in Section II. For mobile RFID systems, we extend the hybrid query tree protocol in Section III. Finally, in Section IV, we evaluate our proposed protocol by using NS-2 simulator, and then concluding remarks will be given in Section V.

II. Hybrid Query Tree Protocol

In this section, we introduce our query tree based protocol with the slotted backoff based response technique. The HQT protocol can decrease the number of collisions using a 4-ary search tree mechanism. However, the drawback of the 4-ary search tree mechanism is the increasing number of idle cycles. We need to reduce the number of idle cycles and for this purpose, we introduce the slotted backoff mechanism.

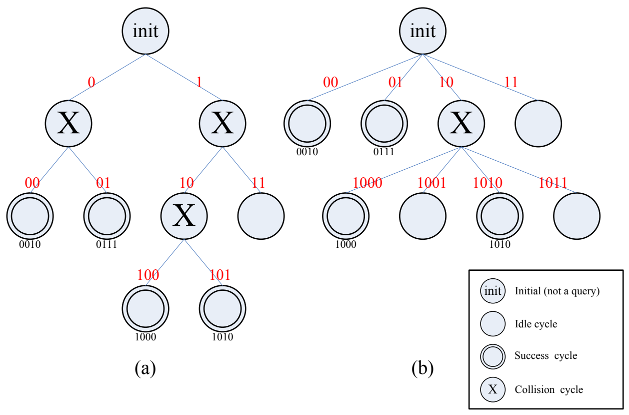


Fig. 1. (a) Query tree protocol (binary search tree), (b) 4-ary search tree algorithm. Using the 4-ary search tree algorithm, collision cycles will be decreased but idle cycles will be increased.

A. 4-ary Search Tree Mechanism

The QT protocol splits the tags into two groups when a collision happens. It expands a collided query string by appending a single bit (0 or 1) at the end of the string, and sends a *query* command again with the new string. Figure 1 (a) illustrates how the QT protocol operates.

In HQT, we suggest a 4-ary search tree mechanism instead of binary search. In this mechanism, if a collision happens in query process, the collided query string will be expanded 2-bits instead of 1-bit. Figure 1 (b) shows the 4-ary search tree mechanism. Using the 4-ary search tree mechanism, the number of collision cycles will be decreased but the number of idle cycles will be increased as a side-effect. Note that the internal-nodes in the query-tree correspond to collision cycles. Let us assume there are n tags to recognize, then a full m-ary query tree (m is 2 or 4) will be created for n tags. According to [9], the number of internal-nodes (the number of collision cycles) is calculated by:

- binary search tree algorithm : $n + k_2 - 1$
- 4-ary search tree algorithm : $\frac{n+k_4-1}{3}$

Here, k_m is the number of idle cycles in a full m-ary tree, and is decided by the distribution of tag IDs. The k_m is increased as m is increased. At best, the number of collisions for the 4-ary search tree mechanism can be decreased to almost one third of that of the binary search tree algorithm. But, it completely depends on the distribution of tag IDs.

B. Slotted Backoff Tag Response Mechanism

In this section, we explain the proposed tag response mechanism based on the slotted backoff mechanism. In the QT protocol, if a query string from the interrogator matches the prefix of the tag ID, then the tag immediately responds to an interrogator. However, in the HQT protocol, the tag should

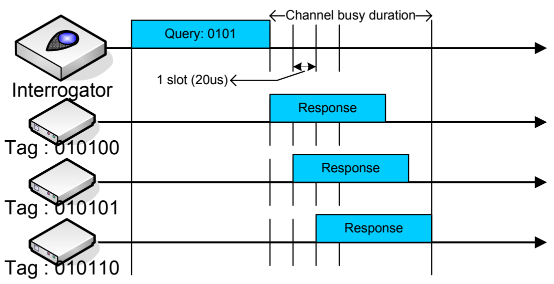


Fig. 2. Operation of the proposed slotted backoff mechanism

defer its response by a backoff time. The backoff time of each tag is determined from the 2-bits which follow the prefix of tag ID identical to the query string. For example, tags do not defer their response if it is '00'. If it is '01', tags will defer 1 backoff time slot until they respond to the interrogator. In other words, when the tags respond to the interrogator, they should set the backoff timer using the very next 2-bits of the prefix of the tag ID which are match the query string. For example, Figure 2, the interrogator sends a query with string '0101', and suppose that there are 3 tags which have following tag ID: '010100', '010101', and '010110'. In this case, each tag sets its backoff time to 0, 1, or 2 and responds to the interrogator when their backoff time expires. For the implementation, a tag needs only one timer. The reason why we apply the slotted backoff scheme in the tag response mechanism is to reduce the unnecessary idle cycles which in turn reduces the number of query messages.

C. *Query Reducing Mechanism*

How can we reduce the number of the idle cycles? When a collision occurs with the slotted backoff tag response mechanism, an interrogator can partially detect to which sub-trees tags belong. We assume that interrogators figure out the transmission time of a response from the knowledge of the bit rate, length of tag's response (length of tag ID), and parameters of the RFID system.

After an interrogator sends the query command, it senses a channel in order to know whether the collision happens or not. If the collision happens, the interrogator can know a busy duration which will be used to figure out the range of bits that tags used for their backoff process. Let us look again the above example in Figure 2. In this case, each tag sets their backoff time to 0, 1, and 2, respectively and responds to the interrogator. The interrogator needs to obtain two values to infer the range of backoff time, *a busy duration* and *a busy starting time*. With these two values, the interrogator can estimate the minimum backoff time and the maximum backoff time of the collided tags. Therefore, the interrogator does not need to send corresponding to the bits of non-existing tags. Again, in the above example, the interrogator can detect there are one or more tags which have '00' bits because it detects that the channel becomes immediately busy. After that, it also senses the channel to be busy during *2 slot times + (length of tag's response/bit rate)*. Here, we can ignore the propagation time because it is very short time. Thus, the interrogator figures out the last response tags use '10' bits for their backoff time. In this way, the interrogator chooses its new query strings as '010100', '010101', and '010110' after '0101'. The interrogator does not send the *'010111' query* string (*unnecessary idle cycle is removed*) and hence the number of queries is decreased than the QT protocol. Note that this scheme cannot detect all idle cycles. For example, in Figure 2, if there are no tags of ID *'010101'*, the interrogator still sends a query with string '010101'.

Table I and II compare operations of the QT protocol and proposed HQT protocol for the example in Figure 1. As you can see, our proposed protocol reduces the number of query commands as well as the number of collisions. More extensive evaluations will be discussed in Section IV.

TABLE I
PROCESS OF QUERY TREE PROTOCOL

Step	Query string	Query result	Query queue
1	0	collision	1,00,01
2	1	collision	00,01,10,11
3	00	success	01,10,11
4	01	success	10,11
5	10	collision	11,100,101
6	11	idle	100,101
7	100	success	101
8	101	success	empty

TABLE II
PROCESS OF PROPOSED HYBRID QUERY TREE PROTOCOL

Step	Query string	Query result	Query queue
1	empty string	collision	00,01,10
2	00	success	01,10
3	01	success	10
4	10	collision	1000,1001,1010
5	1000	success	1001,1010
6	1001	idle	1010
7	1010	success	empty

III. EXTENDED HYBRID QUERY TREE PROTOCOL

In this section, we present our extended hybrid query tree protocol that considers static scenarios of tags. When the tags have a very low mobility, i.e, their location is rarely changed, an interrogator may identify almost the same set of tags after the first query process. Thus, if the interrogator keeps the list of the query strings corresponding to success cycles, it can be used to fasten the next query process. This idea is already introduced in the AQS protocol, but in AQS, every leaf-node of the tree (idle cycles are also included) will be queried. It will waste resources when there are no new arriving tags.

In our proposed protocol, the query process is divided into two phases, for already existing tags and for new arriving ones. First, in *leaf query phase* for existing tags, the interrogator queries leaf-nodes directly to use keeping query strings except idle cycles. Second, in *root query phase* for new arriving tags, the interrogator starts query process from the *root*-node. Thus we call our extended scheme as *Leaf-Root Query* (LRQ).

A. *Leaf-Root Query Mechanism*

Figure 3 shows the AQS protocol and the HQT + LRQ protocol. The basic idea is similar to the AQS protocol, but our scheme does not store query strings corresponding to idle

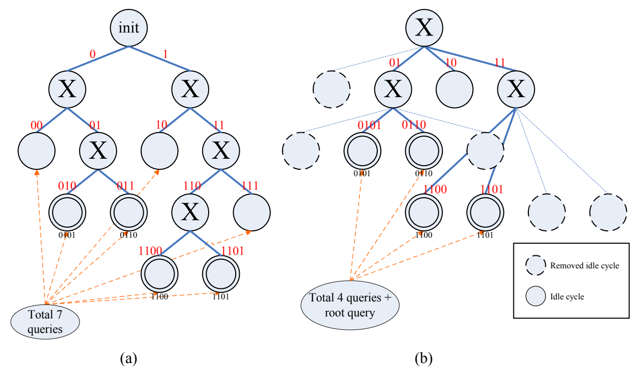


Fig. 3. (a) AQS and (b) HQT + Leaf-Root Query. The unnecessary idle cycles are reduced by introducing two query phases: leaf query phase and root query phase. In our protocol, new arriving tags are recognized in root query phase by using LPM mechanism.

cycles. Only the *leaf*-nodes corresponding to success cycles will be directly queried (leaf query phase). As you see in Figure 3, HQT + LRQ sends fewer query messages than AQS. If some existing tags move out of the reading range, some idle cycles will happen. For new arriving tags, the query command is initiated and sent again from the *root*-node (root query phase). In this way, we make only the new arriving tags contend for responding to interrogator.

Note that even if we apply the current LRQ mechanism to the HQT protocol, the collision may occur between new arriving tags and already existing ones, since new tags' IDs may be located as a sub-tree of a leaf-node (success cycle). To avoid this problem, we also introduce a *Longest Prefix Matching* (LPM) algorithm in the LRQ mechanism.

B. Longest Prefix Matching Algorithm

The key idea of LRQ is that the tags which are already queried in leaf query phase do not respond at root query phase to decrease the total identification delay by reducing collisions. For this purpose, each tag maintains the longest length of matching string (*ls*), i.e, the longest length of the query string that matched the prefix of ID so far. The initial value of *ls* is 0. Whenever the tag receives a query command, it compares *ls* to the length of the current query string. If *ls* is longer than the length of query string then the tag ignores this query command. If *ls* is 2-bits shorter (Since the query string is always increased by 2-bits whenever collision occurs.) than or equal to the length of the current query string, then tag checks if its prefix of the ID matches the query string. Thus, tags which responded at leaf query phase will not respond to the interrogator at root query phase because, in general, *ls* of tags in leaf query phase may be much longer than the length of query strings in root query phase. And new arriving tags will not respond to the interrogator at leaf query phase, because they will initialize their *ls* value when an interrogator's ID in the query command is different from the old one.

Table III and IV compare operations of the AQS protocol and extended HQT protocol for the example in Figure 3. As you see, the proposed protocol reduces the number of query commands because idle cycles are decreased.

TABLE III
PROCESS OF THE AQS PROTOCOL

Step	Query string	Query result	Query queue
1	00	idle	010,011,10,1100,1101,111
2	010	success	011,10,1100,1101,111
3	011	success	10,1100,1101,111
4	10	idle	1100,1101,111
5	1100	success	1101,111
6	1101	success	111
7	111	idle	00,010,011,10,1100,1101,111

TABLE IV
PROCESS OF EXTENDED HQT PROTOCOL

Step	Query string	Query result	Query queue
1	0101	success	0110, 1100, 1101
2	0110	success	1100, 1101
3	1100	success	1101
4	1101	success	empty
5	empty (root query)	idle	0101, 0110, 1100, 1101

IV. Performance Evaluation

We use the NS-2 simulator to evaluate our proposed protocol. We also evaluate the QT protocol and the AQS protocol for comparison purposes. The RFID system network in this simulation consists of one interrogator and n tags to recognize; n varies from 25 to 200. Each tag has a unique ID of 128 bits long, and we assume all tags are in the reading range of the interrogator when tags without mobility scenario (static scenario). Mobility scenario will be described later. The size of a *query* command and responses of tags are 128 bits each. Both the query and tag's responses are transmitted by the same transmission rate, 128Kbps [11]. One backoff time slot is set to 20us and the idle cycle is set to 80us (20us + maximum time slot(3)) + *query transmission time* [11] in our protocol and 20us + *query transmission time* in others. We do not consider frame errors due to the link condition. We compare three performance metrics in 2 different scenarios: average identification delay, number of collision cycles, and number of idle cycles.

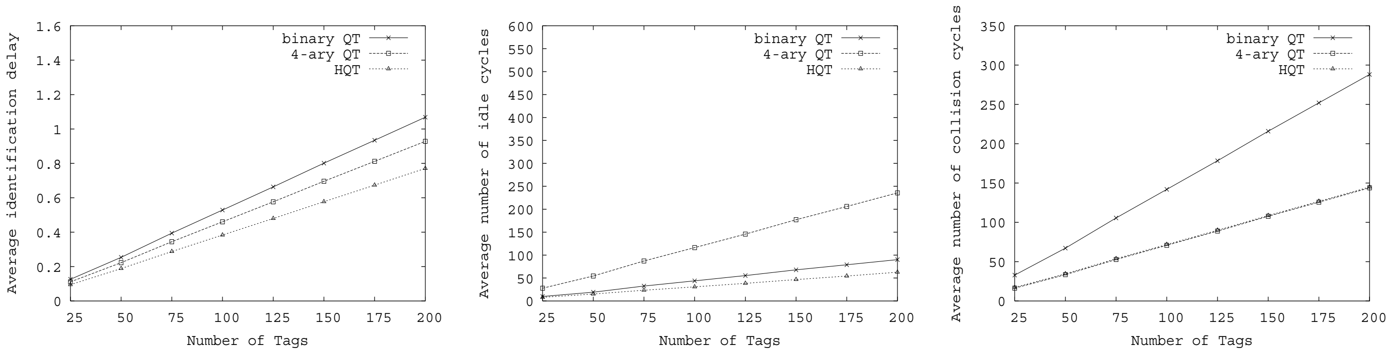
A. Scenario 1: First query process - without mobility

In this scenario, there are no information of query strings in the interrogator and we assume there are no new arriving tags within this query process. The interrogator should identify all tags starting with the initial query string (0 or 1 in QT and empty in HQT) and expanding the query string step by step.

TABLE V
RESULTS OF EACH PROTOCOL WITH 200 TAGS IN SCENARIO 1

Protocol	Identification delay	Number of collision cycles	Number of idle cycles	Efficiency
binary QT	1.06 sec	288.1	90.1	-
4-ary QT	0.92 sec	143.9	235.1	13.1%
HQT	0.77 sec	144.9	62.8	27.6%

We compare binary QT protocol, 4-ary QT protocol, and HQT protocol. Each simulation is repeated 20 times for each number of tags: 25, 50, 75, 100, 125, 150, 175, and 200. Figure 4(c) shows the number of collision cycles of each protocol. Both the 4-ary QT protocol and HQT reduce the number of collision cycles since both of them use the 4-ary tree algorithm instead of binary tree algorithm. Average identification delays,



(a) Identification delay (b) Number of idle cycles (c) Number of collision cycles

Fig. 4. Scenario 1: Static tags only. There are 3 metrics, identification delay, idle cycles, and collision cycles with 3 different protocols. As you see, all graphs form a linear line.

in Figure 4(a), of both the 4-ary QT and the HQT protocols are lower than that of binary QT. But, the number of idle cycles, in Figure 4(b), of the 4-ary QT protocol is much larger than that of binary QT protocol even though the HQT protocol reduces the number of idle cycles less than that of binary QT protocol. Hence, the identification delay of the HQT protocol is the lowest in this scenario.

Table V shows average value of output of 20 simulations only with 200 tags due to space limit. Here, *efficiency* means how faster identification is possible compared to the binary QT protocol. You can expect the other cases because the results show linear performance according to Figure 4. Since HQT starts a query process with empty string, there are just one more collision cycle compared to 4-ary QT [1]. In the case of the 4-ary QT protocol, it decreases the number of collision cycles to 50% of binary QT protocol. But, the number of idle cycles is increased by 150% than that of the binary QT protocol. But in the HQT protocol case, the number of idle cycles is also decreased to 68% of binary QT protocol, thus the HQT protocol can reduce the overall identification delay by 27.6%.

B. Scenario 2: Mobility scenario - Static tags with new tags

In this scenario, there are some moving tags, thus idle cycles are increased and collision cycles will occur more because some tags are moved into the interrogator's reading range. We setup the scenario that 10% and 50% of tags moved out and moved in this interrogator's reading range. The number of tags to recognize is not changed since we make the the number of tags moved in equal to the number of tags moved out. We compare binary QT protocol, AQS protocol, and extended HQT protocol. Each simulation is also repeated 20 times for each number of tags.

Figures 5 and 6 show 3 measurement results of each protocol with 10% and 50% mobility respectively. In case of binary QT protocol, mobility of tags does not influence performances since the QT protocol does not use query string obtained by prior query process, thus mobility of tags is insignificant to the QT protocol. Hence, its results are similar to those of scenario 1 in both 10% and 50% mobility. But in the case of the AQS protocol, mobility of tags has a substantial influence. Similar to Figures 5(b) and 6(b), the number of idle cycles are increased substantially as mobility of tags increased. Thus, as Figure 6(a) shows, there is little difference between the AQS protocol and the QT protocol in terms of average identification delay with 50% mobility.

[1]The reason why HQT starts the query process with empty string is that it can be useful when there are no new tags (save 3 idle cycles). Even when there are new tags; the first query with empty string may make collision, HQT may be able to reduce unnecessary queries due to the slotted backoff mechanism.

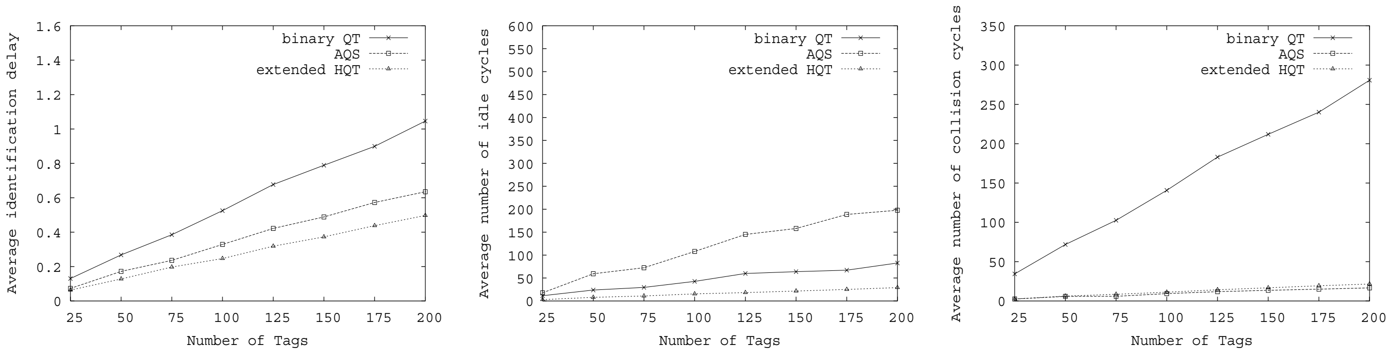
TABLE VI
RESULTS OF EACH PROTOCOL WITH 200 TAGS IN SCENARIO 2(50%)

Protocol	Identification delay	Number of collision cycles	Number of idle cycles	Efficiency
AQS	1.05 *sec*	54.9	532.8	-
HQT	0.74 *sec*	83.5	120.4	29.5%

As Figures 5(a) and 6(a) show, the extended HQT protocol achieves less identification delay than the QT protocol and the AQS protocol. In Figures 5(c) and 6(c), the number of collision cycles of extended HQT is increased as the mobility of tags increased even faster than that of the AQS protocol. But the extended HQT protocol reduces many idle cycles significantly, compared to the AQS protocol in Figures 5(b) and 6(b), the overall identification delay is lower than that of both the QT and the AQS protocols. Results of 50% mobility with 200 tags are summarized in Table VI. The extended HQT protocol makes more collisions than the AQS protocol. Because if there are new arriving tags, it starts the query process from the *root*-node thus; more collisions are possible since the AQS protocol starts query process from the *internal*-node. However, the extended HQT protocol obtains almost 30% performance improvement over the AQS protocol because there are too many idle cycles at the AQS protocol.

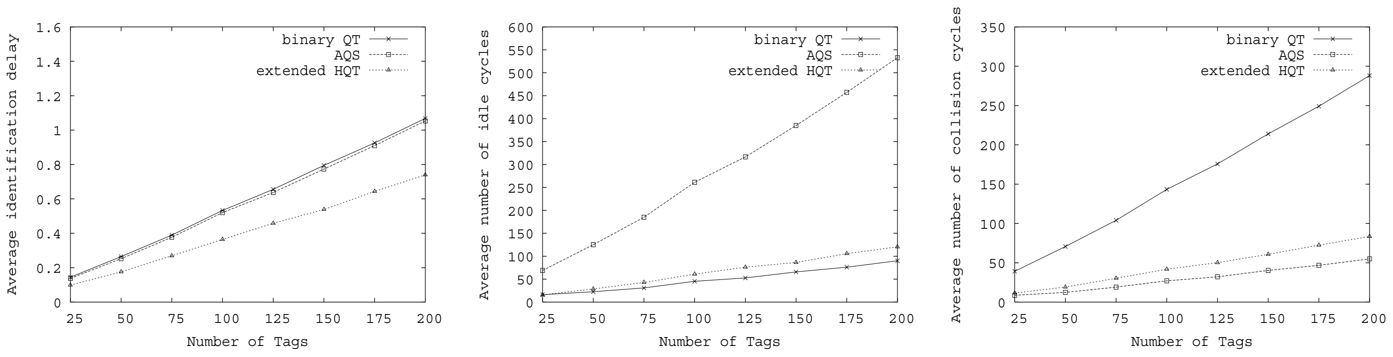
V. CONCLUSION

In this work, we propose a new tag collision arbitration protocol, *hybrid query tree* (HQT), that combines the query tree based protocol with a slotted backoff mechanism. We use a 4-ary query tree instead of a binary query tree for



(a) Identification delay (b) Number of idle cycles (c) Number of collision cycles

Fig. 5. Scenario 2: 10% mobility. As you see, performance of the QT protocol is similar to that of scenario 1.



(a) Identification delay (b) Number of idle cycles (c) Number of collision cycles

Fig. 6. Scenario 2: 50% mobility. The HQT protocol surpasses performance of the both QT and AQS protocols. The AQS protocol does not show good performance in this scenario compared to the QT protocol.

reducing the number of collisions between tags. To decrease an additional idle cycles, the slotted backoff mechanism is also introduced. For static scenarios of tags, we extended the HQT protocol by the *leaf-root query* and the *longest prefix matching* algorithm. Separating a query process of new arriving tags from a query process of existing tags, we allow an interrogator to store much less query strings than the AQS protocol and can save the overall identification delay. In our extended HQT protocol, each tag should maintain the backoff timer, the longest length of the query string that matches the prefix of ID, and the interrogator's ID and defer its response until its backoff timer expires. This complexity overhead would be comparable to that of the BTS protocol. Simulation results show that the proposed protocol achieves less identification delay than both the QT protocol and the AQS protocol. Our future work will include an analysis of the HQT protocol and consider other application scenarios of RFID systems.

VI. Acknowledgment

This work was supported in part by the Brain Korea 21 project of the Ministry of Education, 2006, Korea. The ICT at Seoul National University provides research facilities for this study.

References

[1] J. I. Capetanakis, "Tree algorithms for packet broadcast channels," in IEEE Trans. Inform. Theory, vol. 25, no. 4, pp. 505-515, September 1979.

[2] C. Law, K. Lee, and K.-Y. Siu, "Efficient Memoryless Protocol for Tag Identification." in International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, August 2000.

[3] J. Myung and W. LEE, "Adaptive Splitting Protocols for RFID Tag Collision Arbitration." in ACM Mobihoc, May 2006.

[4] H. Vogt, "Efficient object identification with passive RFID tags," in International Conference on Pervasive Computing, April 2002.

[5] S. Lee, S.D. Joo, and C.W. Lee, "An enhanced dynamic framed slotted aloha algorithm for RFID tag identification," in ACM Mobiquitos, July 2005.

[6] Maurizio A. Bonuccelli, Francesca Lonetti, and Francesca Martelli, "Tree Slotted Aloha: a New Protocol for Tag Identification in RFID Networks," in IEEE WoWMoM, June 2006.

[7] 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 International Symposium on Low Power Electronics and Design, August 2004.

[8] N. Abramson, "The Aloha system - another alternative for computer communications," in AFIPS Conference Proceedings, vol. 36, pp. 295-298, 1970.

[9] K. Rosen, "Discrete Mathematics and Its Applications," 5th edition, McGraw-Hill.

[10] P.Mathys and P.Flajolet, "Q-ary collision resolution algorithms in random-access systems with free or blocked channel access," in IEEE Trans. Inform. Theory, vol. 31, no. 4, pp. 217-243, March 1985.

[11] "EPC^{TM} Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860MHz-960MHz Version 1.0.9," EPCglobal, January 2005.