

Silence Coding for RFID Tag Anti-collision

Ming-Kuei Yeh

National Taipei University of Business
Taipei, Taiwan
jamesyeh@ntub.edu.tw

Yung-Liang Lai

Taoyuan Innovation Institute of Technology
Jhongli, Taiwan
945402011@cc.ncu.edu.tw

Jehn-Ruey Jiang

National Central University
Jhongli, Taiwan
jrjiang@csie.ncu.edu.tw

Abstract—This paper proposes a tree-based RFID tag anti-collision protocol, called the silence coding query tree (SCQT) protocol, based on a novel coding scheme, namely the silence coding, to speed up tag identification. The proposed protocol can identify multiple tags at a time even at the appearance of tag signal collisions. Simulation results show that the SCQT protocol significantly outperforms other related tree-based protocols in terms of the number of cycles to identify tags.

Index Terms—Internet of Things, RFID, Anti-Collision, Query Tree Protocol, Collision Tree Protocol.

I. INTRODUCTION

The *Radio Frequency Identification (RFID)* technology plays an important role in the development of *Internet of Things (IoT)* [1]. It is integrated into many kinds of devices to endow them with the capability to access and manipulate objects in the physical world. There are many RFID-based applications, such as healthcare, logistic control, supply chain management, asset tracking, and industrial manufacturing.

The front-end of an RFID system consists of readers and tags. A tag with a unique ID is usually attached to an object, and a reader can recognize the object by initiating the *identification procedure* (or *interrogation procedure*) to identify the tag ID through wireless communications between the reader and tags. During the identification procedure, the reader sends interrogation commands to tags and the tags respond with their IDs to the reader. If only one tag responds, the reader can identify the responding tag successfully. However, if two or more tags respond simultaneously, responding signals collide and no tag can be identified successfully, which is called the *tag collision problem* [2].

Several tag anti-collision protocols are proposed to solve the tag collision problem. They can be categorized into two classes: *ALOHA-based* protocols and *tree-based* protocols [3]. In ALOHA-based protocols [4-5], tags respond to the reader by transmitting their IDs in a probabilistic manner. For example, in the slotted ALOHA protocol [4], on receiving a reader request, each tag randomly selects a time slot and transmits its ID to the reader at the beginning of the slot. ALOHA-based protocols are simple; however, as shown in [5], they have obviously worse system efficiency than tree-based protocols, where system efficiency is defined to be the ratio of the number of tags to the number of slots required to identify all tags. This paper thus focuses on tree-based protocols.

In tree-based protocols [5-14], the group of tags to be identified is split into subgroups cycle by cycle at the

appearance of tag signal collisions. In a cycle, a reader sends a command to tags, while tags respond or do not respond according to their IDs [7-8] or counters [5-6]. All tags in a subgroup respond to the reader request at the same time. If only one tag responds, the reader can identify the sole responding tag successfully. If two or more tags respond, the reader sends a command to further split responding tags into smaller subgroups until there is only one tag in a subgroup. The tag group splitting is tracked by the reader to identify all tags and can be represented by a tree structure. This is the reason why such a class of protocols is called tree-based.

The query tree (QT) protocol [8] is probably the most promising tree-based tag anti-collision protocol. It is simple and memoryless, since it does not require tags to have tag memory for tracking the tag identification status. Many studies adopt various techniques, such as bit-tracking [9-11], ID-revising [12-13], and re-identifying [14], to improve the QT protocol. This motivates us to study if the QT protocol can be further improved.

This paper proposes a novel coding scheme, namely the *silence coding*, to design a QT protocol variant, called the *silence coding query tree (SCQT)* protocol, for accelerating tag identification. The proposed SCQT protocol can identify tags even when tag signals collide. The SCQT protocol is simulated for performance evaluation. The simulation results are compared with those of the original QT protocol [8] and the CT (collision tree) protocol [9], which is believed to be the QT protocol variant that has very fast tag identification.

The rest of this paper is organized as follows. Section II introduces some related work, and the proposed protocol is elaborated in Section III. Performance evaluation is shown in Section IV, and conclusions are drawn in Section V.

II. RELATED WORK

In this section, we describe the QT protocol [8] and the CT protocol [9]. In the QT protocol, a reader first enqueues two query strings “0” and “1” into an empty queue, then dequeues a query string S from the queue and broadcasts it to tags to start a *cycle* (or *iteration*) of the tag identification procedure. If a tag has an ID prefix matching with S , it responds to the reader with its ID remainder. According to tag responses, there are three cases of cycles for the reader to take different actions. (1) *Idle cycle*: No tag responds; the reader does nothing. (2) *Readable cycle*: Only a tag responds; the reader identifies the tag successfully. (3) *Collision cycle*: Two or more tags respond; tag signals collide and no tag can be identified. For such a case, the reader enqueues two longer query strings $S||0$ and $S||1$ into the

queue. After taking one of the actions, the reader then dequeues a query string to start the next cycle. The identification procedure continues until the queue is empty. In that way, all tags can be identified properly. Note that the original QT protocol adopts the queue data structure, while many studies assume that the QT protocol adopts the stack data structure. However, using the stack or the queue only affects the identification order of tags but does not affect the correctness or performance of the protocol.

The CT protocol improves the QT protocol by reducing the number of idle cycles and collision cycles. Its basic concept is to encode tag IDs with the *Manchester coding*, which encodes bit “0” as a positive (low-to-high) level transition and encodes bit “1” as a negative (high-to-low) level transition in a bit slot. As shown in Fig. 1, when a tag responds a positive transition and another tag responds a negative transition in the same bit slot, the reader recognizes a collision. However, if all tags respond a positive (resp., negative) transition in the same bit slot, the reader can still recognize “0” (resp., “1”). Thus, the reader can detect the first bit encountering collision.

In the CT protocol, if the bits responded by all tags are the same for the first k bits (i.e., they all respond $b_1 \dots b_k$), the reader can detect the first collided bit to be the $(k+1)^{th}$ bit and enqueues $S||b_1 \dots b_k 0$ and $S||b_1 \dots b_k 1$ (instead of $S||0$ and $S||1$ in the QT protocol) into the queue. The number of cycles needed to identify all tags are thus reduced significantly.

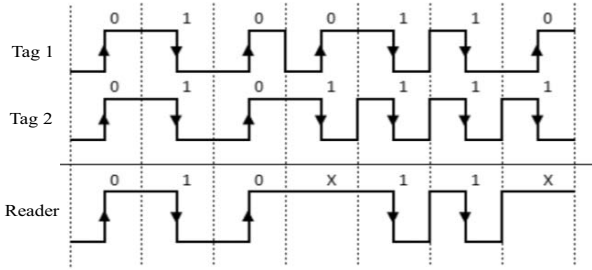


Fig. 1. Using Manchester coding to detect collisions (denoted by X).

III. PROPOSED PROTOCOL

Like the CT protocol, the proposed SCQT protocol encodes response ID bits based on the Manchester coding. Moreover, it incorporates the *silence coding* for the following two purposes: (1) to allow a tag to send the “0” and “1” signals or to keep silent (denoted by the “ $\emptyset$ ” signal) and (2) to allow a reader to precisely detect the “0”, “1”, silence (i.e., “ $\emptyset$ ”), or collided signal (denoted by “X”). The silence coding is described below. **[Silence Encoding]** As shown in Table I, a tag encodes every pair of response bits by combining one silence signal with either “0” or “1” signal. For example, if the pair of response bits is “10”, the encoded signal is “1 $\emptyset$ ”, which means the tag transmits “1” in the first bit slot and keeps silent (i.e., transmits no signal) in the second bit slot. Note that we assume the tag ID length is even. For the case of the odd ID length, we only need to add a padding bit “0” and remove the bit after the tag is identified.

[Silence Decoding] Since there are 4 possible signals (i.e., “0”, “1”, “ $\emptyset$ ”, and “X”) in one bit slot, there are 16 combinations of signals in a pair of time slots. For each of the combinations, the reader can infer (decode) the response bit pairs sent by all tags. Table II shows the mapping of signal combinations to the inferred (decoded) response bit pairs. By this table, the reader can exactly infer what response bit pairs are sent by tags, and it can also infer the minimum number of responding tags. For example, when the reader receives signals “ $\emptyset 0$ ”, it can infer that there is only one tag responding bit pair “00”. For another example, when the reader receives signals “00”, it can infer that there are two or more tags responding bit pairs “00” or “01”.

TABLE I. SILENCE ENCODING

Response bit pair	Encoded signal pair
00	$\emptyset 0$
01	0 $\emptyset$
10	1 $\emptyset$
11	$\emptyset 1$

TABLE II. SILENCE DECODING

Received signal pair	Set R of bit pairs decoded	Min. number m of responding tags
$\emptyset \emptyset$	{ }	0
$\emptyset 0$	{00}	1
$0 \emptyset$	{01}	1
$1 \emptyset$	{10}	1
$\emptyset 1$	{11}	1
00	{00, 01}	2
10	{00, 10}	2
$\emptyset X$	{00, 11}	2
$X \emptyset$	{01, 10}	2
01	{01, 11}	2
11	{10, 11}	2
$X 0$	{00, 01, 10}	3
$0 X$	{00, 01, 11}	3
$1 X$	{00, 10, 11}	3
$X 1$	{01, 10, 11}	3
XX	{00, 01, 10, 11}	4

The SCQT protocol is elaborated as follows. Initially, a reader empties a queue and enqueues a *NULL* query string into the queue to initiate a new *round* of tag interrogation. The reader tries to identify all tags cycle by cycle. In every cycle, the reader dequeues a query string Q from the queue and broadcasts it. On receiving a query string from the reader, a tag whose ID prefix matches with Q will respond with its ID remainder on the basis of the silence encoding, as shown in Table I. Note that the query string Q is *NULL* in the first cycle, which makes all tags respond their ID remainders.

In every cycle, the reader receives and processes (decodes) signals pair by pair according to the silence decoding, as shown in Table II. It tries to find out the set R of possible response bit pairs sent by all responding tags and the minimum number m of responding tags for every pair of received signals. Let $k=|Q|/2$, $l=|ID|/2$, where $|Q|$ and $|ID|$ is the length of the query string Q and the length of the tag ID, respectively. The reader needs to figure out $R_{k+1}, \dots, R_l$ and $m_{k+1}, \dots, m_l$, since tags only respond ID remainder but do not respond ID prefix that matches with Q . If

every m_i , $k+1 \leq i \leq l$, is 1, then the reader can exactly infer the response bits (i.e., ID remainder) to be the bit pair concatenation $p_{k+1}||\dots||p_l$, where p_i is the only bit pair in R_i , $k+1 \leq i \leq l$. Furthermore, if every m_i , $k+1 \leq i \leq l$, $i \neq j$, is 1, and only m_j is not 1 (more than 1), then the reader can also exactly infer the response bits to be the following m_j bit pair concatenations: (1) $p_{k+1}||\dots||p_{j-1}||p_{j,1}||\dots||p_l$, (2) $p_{k+1}||\dots||p_{j-1}||p_{j,2}||\dots||p_l$, ..., (m_j) $p_{k+1}||\dots||p_{j-1}||p_{j,m_j}||\dots||p_l$, where p_i is the only bit pair in R_i , $k+1 \leq i \leq l$, $i \neq j$, and $p_{j,1}, p_{j,2}, \dots, p_{j,m_j}$ are the m_j bit pairs in R_j . On the contrary, if there exist 2 or more indexes, say $j, j', \dots$, such that $m_j, m_{j'}, \dots$ are larger than 1, where $j < j' < \dots$, then the reader cannot exactly infer the response bits and have to enqueue some prefixes into the queue as query strings in future cycles. The prefixes to enqueue are the following m_j bit pair concatenations: (1) $p_{k+1}||\dots||p_{j-1}||p_{j,1}||\dots||p_{j'-1}$, (2) $p_{k+1}||\dots||p_{j-1}||p_{j,2}||\dots||p_{j'-1}$, ..., (m_j) $p_{k+1}||\dots||p_{j-1}||p_{j,m_j}||\dots||p_{j'-1}$, where p_i is the only bit pair in R_i , $k+1 \leq i \leq j'-1$, $i \neq j$, and $p_{j,1}, p_{j,2}, \dots, p_{j,m_j}$ are the m_j bit pairs in R_j .

The algorithm in Fig. 2 shows the details about how the reader exactly infers the response bits or the future query strings. In the algorithm, the functions $\text{NoOfRespondingTags}(s_i)$ and $\text{SetOfBitPairsDecoded}(s_i)$ return “the minimum number of responding tags” and “the set of bit pairs decoded” corresponding to s_i according to the silence decoding shown in Table II. The function $\text{CAT}(R_{k+1}, \dots, R_z)$ returns the set D of the decoded string $p_{k+1}||\dots||p_z$ if every m_i , $k+1 \leq i \leq z$, is 1, where p_i is the only bit pair in R_i , $k+1 \leq i \leq z$. It returns the set D of m_j strings of (1) $p_{k+1}||\dots||p_{j-1}||p_{j,1}||\dots||p_z$, (2) $p_{k+1}||\dots||p_{j-1}||p_{j,2}||\dots||p_z$, ..., (m_j) $p_{k+1}||\dots||p_{j-1}||p_{j,m_j}||\dots||p_z$, if $m_i=1$, for $k+1 \leq i \leq z$, $i \neq j$, and $m_j > 1$, where p_i is the only bit pair in R_i , $k+1 \leq i \leq z$, and $p_{j,1}, p_{j,2}, \dots, p_{j,m_j}$ are the m_j bit pairs in R_j . If $z=l$, then the strings in D are ID remainders. Otherwise, the strings in D are enqueued into the queue as query strings used in future cycles.

Algorithm DecodeSignalPairs:

Input: received signal pairs $s_{k+1}s_{k+2}\dots s_l$

Output: a set of bit strings decoded

$i \leftarrow k$ // i is the loop index

$c \leftarrow 0$ // c is the counter

Repeat

$i \leftarrow i+1$ // $i=k+1, k+2, \dots$

$m_i \leftarrow \text{NoOfRespondingTags}(s_i)$

$R_i \leftarrow \text{SetOfBitPairsDecoded}(s_i)$

If ($m_i > 1$)

$c \leftarrow c+1$

$\text{position}[c] \leftarrow i$

Until ($c \geq 2$ or $i \geq l$)

If ($c \geq 2$)

$z \leftarrow \text{position}[2]-1$

Else

$z \leftarrow l$

Return $\text{CAT}(R_{k+1}, \dots, R_z)$

Fig. 2. The algorithm for decoding received signal pairs

A round of tag identification proceeds cycle by cycle until the queue is empty. The queue being empty implies that all tags have been identified. Then the reader initiates a new round of tag identification. By the silence coding, the reader can infer all

possible bit patterns that cause collisions and enqueues these patterns into the queue or identifies multiple tag IDs based on these patterns at a time. This is the reason why the SCQT protocol can significantly accelerate the tag identification.

Table III shows an example of the identification procedure of the SCQT protocol for a scenario of 8 tags, whose IDs are “000001”, “000010”, “001010”, “001100”, “010001”, “010101”, “110110”, and “111010”. In cycle 1, the reader sends the query string $Q=NULL$ to tags. In this cycle all tags respond, since the query string is $NULL$. Every tag responds signals based on its ID according to the silence encoding shown in Table I. For example, the tag whose ID is “00 00 01” responds with “00 00 00”. In cycle 1, the reader receives signals “0X XX X0”. According to the silence decoding shown in Table II, the first and the second signal pairs correspond to at least 3 and 4 responding tags, respectively. Thus, according to silence decoding, the reader decodes the first signal pair “0X” as “00”, “01”, or “11”, and enqueues $Q||00=“00”$, $Q||01=“01”$, and $Q||11=“11”$ into the queue. In cycle 2, the reader dequeues string $Q=“00”$ from the queue as the query string. Since the four tags with IDs “00 00 01”, “00 00 10”, “00 10 10”, and “00 11 00” have the prefix “00”, they encode their ID remainders and send encoded signals. In this cycle, the reader receives signals “1X X0”. Both the first and the second signal pairs correspond to at least 3 responding tags, so the reader decodes the first signal pair “1X” as “00”, “10”, or “11”, and enqueues $Q||00=“0000”$, $Q||10=“0010”$, and $Q||11=“0011”$ into the queue. In cycle 3, the reader dequeues string $Q=“01”$ as the query string and two tags with IDs “01 00 01” and “01 01 01” respond. In this cycle, the reader receives signals “00 00”. The first and the second signal pairs correspond to at least 2 and at least 1 responding tags, respectively. Thus, the reader decodes the first signal pair “00” as bit pairs “00” or “01”, and decodes the second signal pair “00” as bit pair “01”. Since signals are processed up to the last pair, the reader can identify the tags with IDs $Q||00||01=“010001”$ and $Q||01||01=“010101”$. Cycles 4 to 7 proceed similarly. In cycle 7, the queue is empty and all tags are identified.

TABLE III. AN EXAMPLE OF THE SCQT PROTOCOL

Cycle No.	Queue	Query String	Tag IDs	ID Remainders	Signals Responded	Signals Received	IDs Identified
1	Empty	NULL	00 00 01 00 00 10 00 10 10 00 11 00 01 00 01 01 01 01 11 01 10 11 10 10	00 00 01 00 00 10 00 10 10 00 11 00 01 00 01 01 01 01 11 01 10 11 10 10	00 00 00 00 00 10 00 10 10 00 01 00 00 00 00 00 00 00 01 00 10 01 10 10	0X XX X0	
2	01 11	00	00 00 01 00 00 10 00 10 10 00 11 00	00 01 00 10 10 10 11 00	00 00 00 10 10 10 01 00	1X X0	
3	11 0000 0010 0011	01	01 00 01 01 01 01	00 01 01 01	00 00 00 00	00 00	010001 010101
4	0000 0010 0011	11	11 01 10 11 10 10	01 10 10 10	00 10 10 10	X0 10	110110 111010
5	0010 0011	0000	00 00 01 00 00 10	01 10	00 10	X0	000001 000010
6	0011	0010	00 10 10	10	10	10	00 10 10
7	Empty	0011	00 11 00	00	00	00	00 11 00

IV. SIMULATION AND COMPARISONS

The proposed SCQT protocol is simulated for performance evaluation for scenarios of 100, 200, ..., 2000 tags. It is assumed that tag IDs are uniformly distributed and the length of the tag ID is 64. The simulated results are compared with those of the QT protocol and the CT protocol in terms of the average number of cycles needed to identify a tag. As shown in Fig. 3, the average number of cycles to identify a tag is independent of the number of tags for all three protocols. Specifically, the SCQT protocol needs the smallest average number (about 2) of cycles, the CT protocol needs about 2.5 cycles, and the QT protocol needs the largest average number (about 2.8) of cycles to identify a tag.

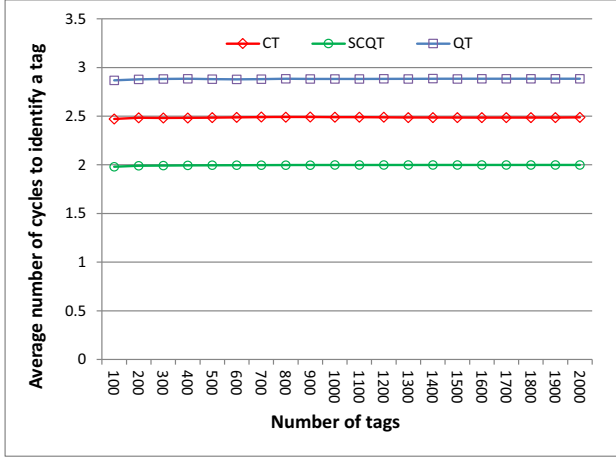


Fig. 3. Simulation result comparisons for the QT, CT, and SCQT protocols.

V. CONCLUSIONS

This paper proposes the SCQT protocol to speed up the tag identification procedure. The novel idea of the SCQT protocol is to use the silence coding scheme, in which transmitting the signal of “0”, transmitting the signal of “1”, and no transmission are all used to carry information in a bit slot. Therefore, at the appearance of tag signal collisions, the reader can infer the bit patterns causing the collisions and can identify multiple tags in a single cycle. Simulation results show the SCQT protocol outperforms other related protocols in terms of the number of cycles needed to identify tags. In the future, we plan to apply the same coding scheme to other existing tag anti-collision protocols to design even more efficient protocols.

ACKNOWLEDGMENT

This work was supported in part by the Ministry of Science and Technology (MOST), Taiwan, under grant number 104-2221-E-008-017- and grant number 105-2218-E-008-008-.

REFERENCES

- [1] S. Li, L. D. Xu, and S. Zhao, “The Internet of Things: a Survey,” *Information Systems Frontiers*, vol. 17, issue 2, pp. 243-259, 2015.
- [2] J.-R. Jiang and M.-K. Yeh, “Anti-collision protocols for the RFID system,” *RFID and Sensor Networks: Architectures, Protocols, Security and Integrations*, Ed. Yan Zhang et al., Auerbach Publications, Taylor&Francis Group, USA, 2009.
- [3] D. K. Klair, K.-W. Chin, and R. Raad, “A Survey and Tutorial of RFID Anti-Collision Protocols,” *IEEE Communications Surveys and Tutorials*, vol. 12, no. 3, pp. 400-421, 2010.
- [4] L. Liu, and S. Lai, “ALOHA-Based Anti-Collision Algorithms Used in RFID System,” in *Proc. of the International Conf. on Wireless Communications, Networking and Mobile Computing (WiCOM 2006)*, pp. 1-4, Sep. 2006.
- [5] M.-K. Yeh, J.-R. Jiang, and S.-T. Huang, “Adaptive Splitting and Pre-Signaling for RFID Tag Anti-Collision,” *Computer Communications Journal*, Vol. 32, Issue 17, pp. 1862-1870, Nov. 2009.
- [6] M.-K. Yeh, and J.-R. Jiang, “Parallel Splitting for RFID Tag Anti-Collision,” in *Proc. of International Journal of Ad-Hoc Ubiquitous Computing*, vol. 8, pp. 249-260, 2011.
- [7] H. Choi, J. R. Cha and J. H. Kim, “Fast Wireless Anti-Collision Algorithm in Ubiquitous ID System”, in *Proc. of IEEE VTC '04*, 2004.
- [8] C. Law, K. Lee, and K.-Y. Siu, “Efficient Memoryless Protocol for Tag Identification (Extended Abstract),” in *Proc. of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, 2000.
- [9] F. Zhou, D. Jin, C.L. Huang, M. Hao, “Optimize the Power Consumption of Passive Electronic Tag for Anti-Collision Schemes,” in *Proc. of the Fifth International ASIC Conference*, pp. 1213-1217, 2003.
- [10] H. Landaluce, A. Perallos, and I. Angulo, “Managing the Number of Tag Bits Transmitted in a Bit-Tracking RFID Collision Resolution Protocol,” *Sensors*, vol. 14, pp. 1010-1027, 2014.
- [11] X. Jia, F. Quanyuan, and L. Yu, “Stability Analysis of an Efficient Anti-Collision Protocol for RFID Tag Identification,” *IEEE Transactions on Communications*, vol. 60, pp. 2285-2294, 2012.
- [12] C. Jung-Sik, S. Jea-Dong, and K. Sung Kwon, “RFID Tag Anti-Collision Protocol: Query Tree with Reversed IDs,” in *Proc. of 10th International Conference on the Advanced Communication Technology (ICACT)*, pp. 225-230, 2008.
- [13] Y. Tian, G. Chen, and J. Li, “An Effective Temporary ID Based Query Tree Algorithm for RFID Tag Anti-Collision,” *Internet of Things*, Y. Wang and X. Zhang, Eds., ed: Springer Berlin Heidelberg, vol. 312, pp. 242-247, 2012.
- [14] J. Myung, and W. Lee, “Adaptive Splitting Protocols for RFID Tag Collision Arbitration,” in *Proc. of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pp. 202-213, 2006.