DOI: 10.3745/KIPSTA.2008.15-A.5.287

RFID 시스템을 위한 하이브리드 하이퍼 쿼리 트리 알고리즘

김 태 희[†] · 이 성 준[††] · 안 광 선[†††]

요 약

RFID 시스템에서 리더와 태그는 단일 무선 공유 채널을 갖기 때문에 RFID 수동형 태그를 위한 태그 충돌 중재가 태그 인식을 위한 중요한 이슈이다. 본 논문에서는 태그 충돌 방지를 위한 하이브리드 하이퍼 쿼리 트리 알고리즘(H^2QT, Hybrid Hyper Query Tree)을 제안한다. 제안된 알고리즘은 쿼리 트리를 기반으로 태그가 리더에게 ID를 전송하는 시점을 전송ID 상위 3비트 내의 '1'값을 이용하여 결정한다. 또한 전송 받은 Tag의 상위 3비트는 충돌이 발생하더라도 전송 슬롯에 따라 다르므로 제안한 알고리즘에서 예측이 가능하다. 시뮬레이션을 통한 성능 평가에서 다른 트리 기반 프로토콜에 비해 제안한 알고리즘이 쿼리 횟수에서 높은 성능을 갖는다는 것을 보여준다

키워드 : RFID, 충돌방지 알고리즘, 태그 인식, 쿼리 트리, EPC

A Hybrid Hyper Query Tree Algorithm for RFID System

TaeHee Kim[†] · SeongJoon Lee[††] · KwangSeon Ahn[†††]

ABSTRACT

A tag collision arbitration algorithm for RFID passive tags is one of the important issues for fast tag identification, since reader and tag have a shared wireless channel in RFID system. This paper suggests Hyper-Hybrid Query Tree algorithm to prevent the tag-collisions. The suggested algorithms determine the specified point in time for tag to transfer ID to reader by using value 1 of the upper 3 bit based on Query Tree. Also, because the transferred upper 3 bits of tag is different depending on the time of transfer, it is possible to predict in the suggested Algorithm. In the performance evaluation through simulation, it shows the suggested algorithm has higher performance in the number of queries compared to other Tree-based protocols.

Keywords : RFID, Anti-Collision Algorithm, Tag Identification, Query Tree, EPC

1. 서 론

RFID(Radio Frequency Identification)는 유비쿼터스 환경의 기술로서 물품에 부착된 전자 태그의 정보를 리더가 RF 신호를 이용하여 태그를 인식하는 자동 인식 기술이다. RFID는 유통/물류뿐만 아니라 식품, 의료/약품, 도로/교통, 우정, 문화, 생산자동화, 소방/방재, 금융, 환경, 정보유통 등 다양한 분야에 적용이 가능하다[1]. 특히, 유통 분야에서는 이미 바코드를 대체하여 물품에 RFID 칩을 부착하여 관리하고 있다.

RFID 시스템은 태그와 리더 두 가지로 구성된다. 태그는 리더의 요청에 따라 자신의 ID를 RF신호로 리더에게 전송하고, 리더는 태그를 인식하기 위한 작업을 수행한다. RFID 시스템 환경에서 리더와 다수의 태그가 무선 채널을 공유한다. 그러므로 다수의 태그가 응답하는 경우 충돌(collision)이 발생하여 어느 태그도 인식하지 못하는 상황이 발생한다. 그러므로 충돌을 인식하고 새로운 프리픽스를 만들어 태그에게 질의하기 위한 절차가 필요하다[3]. 이러한 절차를 충돌 방지 알고리즘 (anti-collision algorithm)이라고 부른다. 충돌 방지 알고리즘에 있어서 가장 중요한 것은 태그를 인식하는데 걸리는 시간을 줄이는 것이다. 즉, 하나의 태그만이 응답할 수 있는 프리픽스를 빠르게 생성하여 태그에 질의할 수 있는 알고리즘이 필요하다.

태그 충돌 방지 알고리즘은 확률적 방식과 트리 기반 방식으로 분류된다. 확률 기반의 태그 충돌 방지 기법은 확률에 근거하여 태그 아이디를 전송할 임의의 시간을 선택하고, 아이디 전송을 반복하는 방법이다. 이러한 방법은 확률에 의존하여 태그가 응답할 시간을 선택하기 때문에 충돌을 완전히 방지하지 못한다. 그러므로, 두 개 이상의 태그가 동시에 같은 응답 시간 슬롯을 선택한다면 아주 긴 시간 동안 인식되지 못하는 상황이 발생할 가능성을 갖는다[11]. 대표

† 준 회 원 : 경북대학교 대학원 전자전기컴퓨터학부 석사과정
†† 정 회 원 : 경북대학교 컴퓨터공학과 초빙교수
††† 종신회원 : 경북대학교 컴퓨터공학과 교수(교신저자)

논문접수 : 2008년 5월 19일
수 정 일 : 2008년 8월 26일
심사완료 : 2008년 8월 26일

적인 확률기반의 태그 충돌 방지 기법은 ALOHA, Slotted ALOHA, Frame slotted ALOHA가 있다[4,5,6].

트리 기반의 방법은 충돌이 발생하였을 때, 트리를 기반으로 다음 질의 프리픽스를 선택하고, 질의함으로써 영역내의 모든 태그를 인식하는 방법이다. 그러나 트리 기반의 방법은 유사 태그가 많은 경우 충돌이 많아져 인식시간이 지연된다. 대표적인 알고리즘으로 이진 트리[7]와 Query Tree(QT) 프로토콜[8]이 있다.

최근, 두 가지 방식의 단점을 보안하는 하이브리드 알고리즘이 제안되었다. 즉, 이진 또는 그 이상의 고정된 슬롯을 이용하여 강제적인 충돌을 발생시키고, 이를 트리로 구성함으로써 모든 태그를 인식함과 동시에 질의 횟수 감소시키고 있다.[10]

본 논문에서는 이러한 하이브리드 알고리즘을 기반으로 하는 하이브리드 하이퍼 쿼리 트리 알고리즘 (H^2QT, Hybrid Hyper Query Tree algorithm)을 제안한다. 제안된 방법은 4-ary트리를 전가산기와 맨체스터 코드를 이용하여 인식 가능한 비트의 단위를 기존의 2비트에서 3비트로 확장하는 방법이다. 태그는 아이디를 전송할 때 전송을 시작할 비트의 위치로부터 3비트들의 합을 이용하여 전송할 타임 슬롯을 선택한다. 리더는 태그로부터 전송된 데이터 값을 맨체스터 코딩 방법을 이용하여 얻고, 특정 비트에 충돌이 발생하더라도 이를 조사하여 전달된 데이터 값을 파악하는 것이 가능하다. 시뮬레이션에서 제안된 알고리즘은 중복이 많을 수록 좋은 성능을 보였다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 관련 연구로 확률적 방식과 트리 방식의 충돌 방지 알고리즘을 설명한다. 3장에서는 하이퍼 하이브리드 쿼리 트리 알고리즘을 제안하고 동작 방식을 설명한다. 4장에서는 제안 알고리즘의 성능 평가를 하고 마지막으로 5장에서 결론을 기술한다.

2. 관련 연구

2.1 트리 기반 알고리즘

2.1.1 쿼리 트리 알고리즘

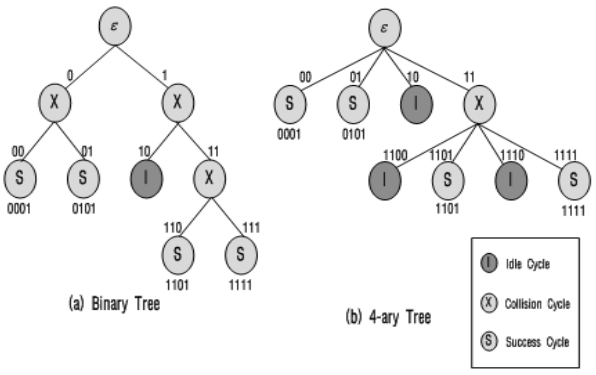
쿼리 트리 알고리즘(QT, Query Tree Algorithm)[12]은 리더의 질의와 태그의 응답을 반복하면서 태그를 식별 한다. QT는 충돌 비트의 위치를 알 수 없는 충돌 감지(collision Detecting)알고리즘을 사용하고 있기 때문에 비트 충돌 정보를 이용할 수 없다.

QT 알고리즘은 구현이 쉽고, 적은 비용으로 생산이 가능하다는 장점을 갖는다. 그러나, 태그 아이디의 분포 상황을 고려하지 않은 상태에서 태그 아이디를 확장하는 것은 유휴 사이클의 원인이 된다. 또한 이러한 환경으로 인해 태그 인식에 많은 질의와 전송 비트 수를 필요로 한다.

2.1.2 하이브리드 쿼리 트리 알고리즘

QT 알고리즘의 단점은 많은 충돌과 그로 인한 질의와 전송에 많은 비트 수를 필요로 한다는 것이다. 하이브리드 쿼리 트리 알고리즘(HQT, hybrid query tree algorithm)[10]은 4-ary 검색 트리 기법을 적용하여 프리픽스가 1비트에서 2비트씩 증가하도록 확장하였다. 이로 인해 태그 들 사이의 충돌 사이클을 줄이고 있으나 유휴 사이클의 증가를 가져왔다. 이러한 문제점을 해결하기 위해 HQT는 슬롯 지연 기법을 적용하였다. 슬롯 지연 기법이란 전달된 프리픽스와 일치하는 태그들이 즉시 응답하는 것이 아니라 특정시간 대기한 후 데이터를 전송하는 기법이다. 이러한 기법을 통해 HQT는 특정 비트 패턴의 존재 유무를 확인한다. 다음 (그림 1)은 쿼리 트리와 HQT를 이용한 태그 인식 과정을 비교한 것이다.

HQT는 슬롯 지연 기법을 이용하여 유휴 사이클의 수를 줄이는 것을 시도하였다. 즉, 시작 위치와 완료 위치를 점검하여 불필요한 프리픽스의 확장을 줄이는 것이다. 그러나, 제안된 방법에서는 비지 상태만을 점검하기 때문에 중간에 보내지지 않은 슬롯의 경우 탐지가 불가능하다. 또한, 슬롯의 개념을 변형하여 한 슬롯 당 시간을 줄이고 있으나 질의 횟수를 늘려야 하는 문제를 발생시킨다. 예를 들어 “01010000”, “01010101”, “01011100”이 존재 한다고 가정하고, 리더가 “0101”의 프리픽스를 전파하면 태그는 각 슬롯에 응답한다. 이때, 만일 각 슬롯이 남은 아이디를 위한 충분한 시간을 확보 하였다면 각 태그는 한번에 인식할 수 있다. 그러나, 제안된 논문에서는 시간과 질의 횟수를 단축하기 위한 방법으로 변형된 슬롯드 알로하 방법을 고려함으로써, 두 개 이상의 유사한 태그가 존재할 때 서로 다른 슬롯에 할당 됨에도 불구하고 인식하지 못한다. 즉 상위 예에서 제안된 방법은 프리픽스 “0101”의 전파시 한 슬롯에 응답하기 때문에 다음 질의에서 모두 인식하는 것이 가능하지만, 슬롯의 대기 시간을 줄여 리더는 한번에 인식하는 것이 불가능하다. 상위 예에서 리더가 “0101”을 전파하고 태그들이 응답한 후, 리더는 충돌을 감지하여 “010100”, “010101”, “010110”, “010111”을 큐에 저장하게 된다. 또한 태그들의 바쁜 시간(busy time)만을 점검하기 때문에 리더는 “010110”의 부재를 알 수 있는 방법이 없다. 그로 인해 새로운 유휴 사이클을 생성한다.



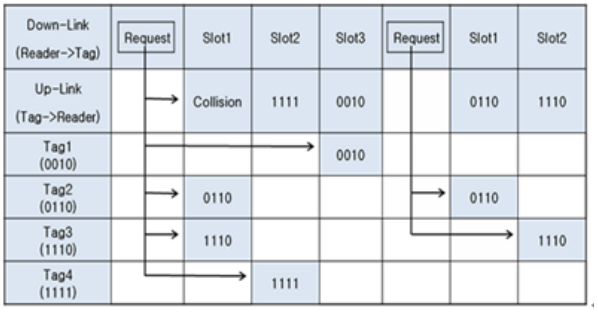
(그림 1) Binary와 4-ary Tree

2.2 확률 기반 알고리즘

2.2.1 슬롯 알로하 알고리즘

알로하 기반의 프로토콜은 보통 슬롯 알로하(Slotted ALOHA) 프로토콜[6]을 말하며 이는 태그의 응답 시간을 고정된 몇 개의 슬롯(Slot)으로 나누고 각 태그들은 자신이 사용할 슬롯을 선택하여 자신의 태그 ID를 전송하는 방식이다. 슬롯이란 시간간격을 의미한다. 이 방식에서 리더는 나누어진 각 타임 슬롯에 태그 ID가 두 개 이상 응답하면 충돌 발생으로 인식한다.

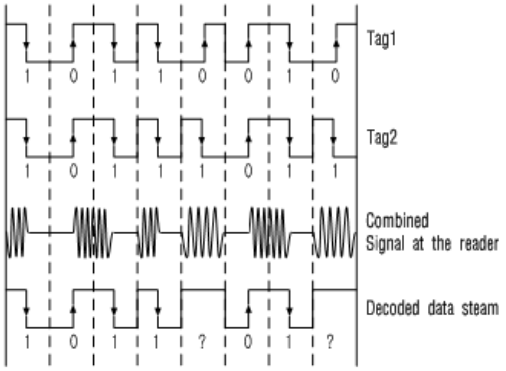
(그림 2)는 슬롯 알로하 프로토콜을 이용하여 리더가 4개의 태그 ID를 식별하는 과정이다. 먼저 리더는 태그들에게 요청 메시지를 전파하면서, 각 태그들에게 슬롯 선택에 대한 정보를 전송한다. 이 때 각 태그들은 전송된 정보를 이용하여 자신의 슬롯을 결정한다[12]. 태그 2와 태그 3은 슬롯 1, 태그 1은 슬롯 3, 태그 4는 슬롯 2에서 응답하였다. 슬롯 2와 슬롯 3은 하나의 태그가 응답하여 태그를 인식하였고, 슬롯 1은 두 개의 태그가 응답하여 충돌이 발생하였다. 충돌이 발생한 태그들은 이러한 과정을 반복하여 모든 태그가 식별 될 때까지 반복한다.



(그림 2) 슬롯 알로하 프로토콜의 동작과정

2.3 맨체스터 부호를 이용한 비트 별 충돌 검출

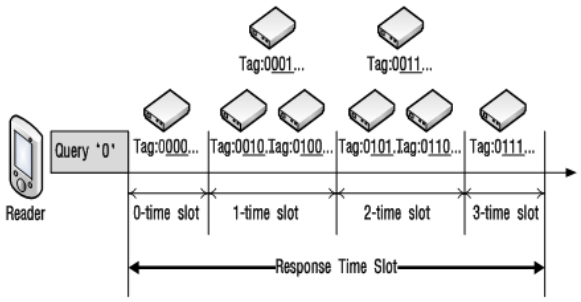
RFID 시스템에서 리더는 태그의 충돌의 상태를 감지할 수 있어야 한다. QT의 알고리즘 태그의 충돌상태만을 감지하고 이진 트리 알고리즘의 경우는 비트 별 충돌위치를 알고 있어야 한다. 태그 전체의 충돌 위치를 알기 위해서는 맨체스터 부호를 쓰면 가능하다. 맨체스터 부호화에서는 (그림 3)과 같이 한 비트구간내의 위상이 변하는 것으로 비트 값을 결정한다. 그러나 위상이 변하지 않는 상태가 존재한다면 충돌이 발생 한 것으로 인식할 수 있다. 그러므로 맨체스터 부호를 사용하면 비트단위로 충돌을 검출할 수 있다. (그림 3)에서 Tag1의 ID는 '10110010'이고, Tag2는 '10111011'일 때 리더의 요청에 따라 태그는 자신의 ID를 전송한다. 리더에서는 두 태그간의 위상이 다른 비트들은 서로가 중첩되어 위상이 변하지 않는 상태가 한 비트 구간 동안 지속이 된다. 이러한 상태를 맨체스터 코딩에서 오류로 판단하고 그 비트 구간을 충돌로 인식한다. 이와 같이 맨체스터 부호화를 이용하면 각 비트 별로 충돌 여부를 검출할 수 있다[9].



(그림 3) 맨체스터 부호를 이용한 비트충돌검출

3. Hybrid Hyper Query Tree protocol

QT 알고리즘은 태그가 많이 존재할수록 태그 충돌이 많이 발생한다. 이로 인해 전체 태그의 인식시간을 지연시킨다. 2비트를 확장하는 4-ary 검색 트리 알고리즘은 태그의 충돌을 줄이는 반면에 유휴 사이클을 증가시켜 QT의 쿼리 수를 줄이지 못했다. 본 논문에서 제안하는 하이브리드 하이퍼 쿼리 트리(H^2QT, Hybrid Hyper Query Tree) 알고리즘은 태그 ID중 잔여 비트 중 앞 3비트가 갖는 1의 개수를 이용하여 응답 슬롯 선택하고, 응답한다. H^2QT 는 기존 알고리즘에 비해 충돌과 쿼리의 횟수를 줄일 수 있다. 또한 제안된 알고리즘은 존재하지 않는 프리픽스를 생성하지 않기 때문에 유휴 사이클을 생성하지 않는다. 다음 (그림 4)는 태그 ID에 따른 슬롯 선택 방법을 보여준다.



(그림 4) H^2QT Tag 응답 슬롯 선택 방법

3.1 태그

다음 (그림 5)는 태그에서 수행되는 작업을 의사 코드로 표현한 것이다. 먼저, 태그는 리더로부터 받은 프리픽스 $prefix_p$를 검사하여 자신이 소유한 ID 비트와 일치하는지 점검한다. 만약 프리픽스가 일치한다면, L_{p+1}에서 L_{p+3}비트의 값 입력으로 하는 전가산기를 이용하여 '1'의 개수를 C에 쉽게 저장한다(그림 5, line 5). 이후 C의 값에 따라 주어진 타임 슬롯의 길이만큼 대기한 후(line 10), 태그의 나머지 ID 값들을 리더에게 전달한다(line 11).

Algorithm 1. H^2QT Tag Operation
1. P = prefix received from a reader
2. L_p = length of prefix P
3. C = 0 //the number of '1'
4. // 리더로부터 받은 프리픽스와 동일한지 검사
5. **If**(p==ID[1..L_p]){
6. // 1의 개수를 계산
7. C = **Sum** (L_{p+1}, L_{p+2}, L_{p+3})
8. }
9. **Else**
10. Return;
11. // ID를 전송할 스롯의 시간까지 대기
12. **Delay** (C * t_d)
13. // 매칭된 프리픽스 이후 전 데이터를 전송
14. **Response** ($TagID_{p+1}$..$TagID_{Length}$)

(그림 5) 제안된 알고리즘에서 태그의 의사 코드

상위 알고리즘에서 보이는 바와 같이 prefix와 일치하는 다음 3비트가 "000"이거나 "111"의 경우 리더는 그 해당 태그가 하나만 존재한다면 바로 인식할 수 있다.

3.2 리더

리더는 모든 태그의 인식을 위해 태그에게 프리픽스를 전달하고, 태그로부터 전달 받은 데이터를 분석하여 태그의 ID를 자동으로 인식하는 부분이다. 다음 (그림 6)은 리더에서 발생하는 작업을 의사 코드로 표현한 것이다.

리더는 질의, 분석, 저장 3단계를 거친다. 질의 단계는 리더가 인식된 프리픽스를 큐로부터 얻어 태그에게 전파하는 과정이다. 자동인식을 시작하기 위해 null을 뜻하는 'ε'문자열을 큐에 저장한다 (그림 6, line2). 리더는 이후 큐가 빌 때까지 반복적으로 작업을 수행한다. 먼저 큐에서 태그로 전달할 프리픽스를 얻고(line 6), 이를 태그에게 전파한다(line 7).

분석 단계는 전달할 프리픽스와 일치한 값을 갖는 태그들이 자신의 ID 값을 전달하면, 이 데이터로부터 원하는 값을 얻는 과정이다. 이 과정에서 충돌의 위치를 파악하기 위해 맨체스터 코드를 사용한다. 또한 분석 단계는 저장 단계와 인접하여 발생한다. 전달된 데이터는 최상위 3비트에 존재하는 '1'의 개수에 따라 다른 슬롯으로 전달된다. 이렇게 전달된 데이터는 충돌이 발생하더라도 태그가 전달한 3비트가 무엇인지 쉽게 인지할 수 있다. 예를 들어 "010"와 "001"로 시작하는 두 태그가 데이터를 1 타임 슬롯에 전달하였다면 버퍼에서는 "0XX"로 충돌이 발생한다. 이를 이용하여, 1타임 슬롯에서는 '1'의 수가 하나만 존재하기 때문에 "010"와 "001"이 전달되어졌음을 알 수 있다. 이와 유사하게 "011"과 "101"의 경우도 충돌의 위치를 인지할 수 있다면 쉽게 파악하는 것이 가능하다. 만일 1타임 슬롯이나 2 타임 슬롯에서 3비트 모두 충돌이 발생하였다면 슬롯내의 모든 가능한 비트 조합이 존재한다는 것이다. 이를 위해 리더는 타임 슬롯의 번호에 따라 버퍼를 다른 값으로 채운다(line 23). 이후 상위 3비트 내에 충돌이 발생한 경우 태그에 전달한 프리픽스와 버퍼의 3비트만을 조합하여 새로운 프리픽스를 생성하고 이를 큐에 저장한다(line 28-34). 상위 3비트는 다른 비트에서 충돌이 발생할 수 있기 때문에 원 비트로 채운다. 상위 3비트에서 충돌이 발생하면 나머지 비트에 대해 고려 대상이 아니기 때문에 다음 슬롯으로 이동한다(line 36 - 37). 마지막으로 상위 3비트가 아닌 다른 비트에서 충돌이 발생하면, 충돌이 발생하기 이전 비트까지를 얻어 이전 태그에 전달한 프리픽스와 접목하여 큐에 저장한다.

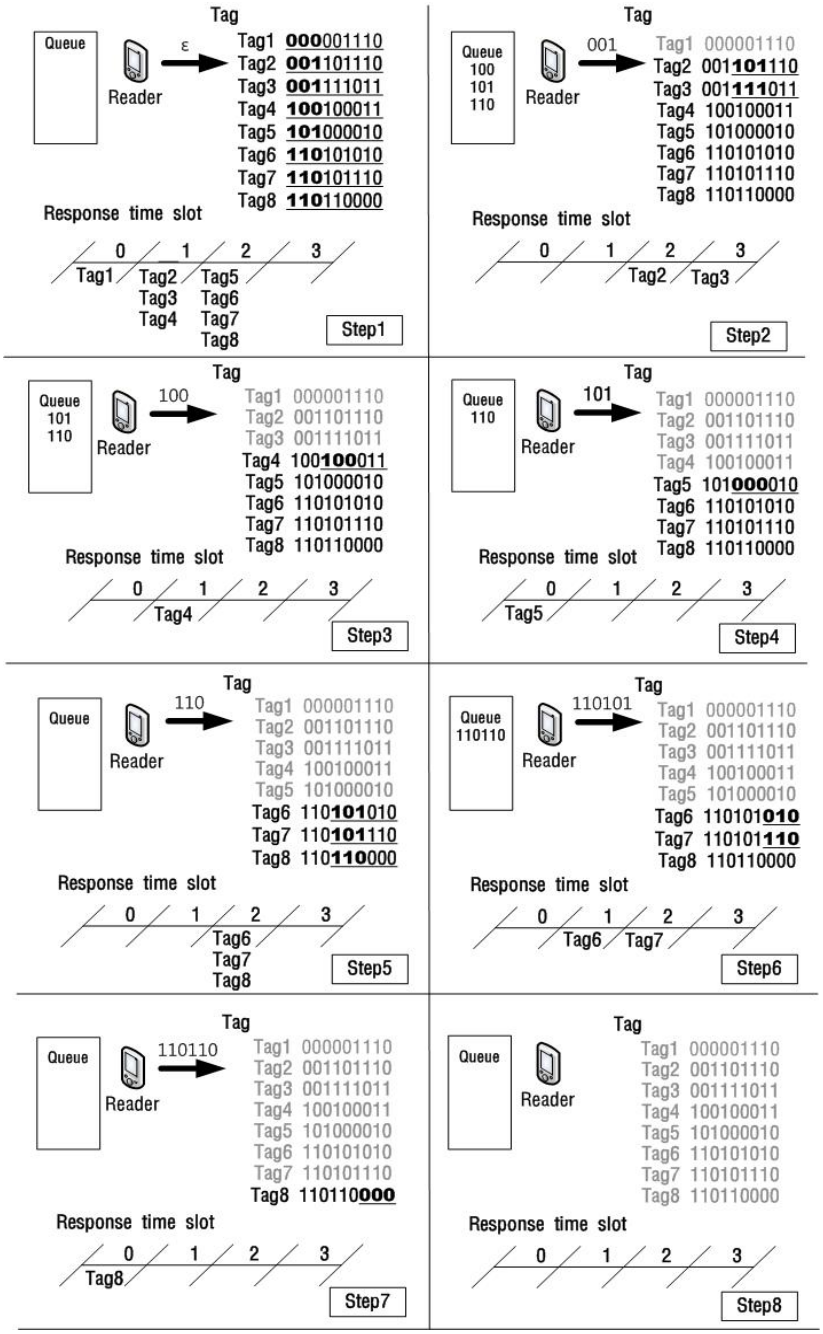
Algorithm 2. H^2QT Reader Operation
```
1.
2.   PushQueue ("ε")
3.
4.   Do While(!isQueueEmpty()){
5.   /* 질의 단계 */
6.   Prefix = get a prefix from Queue
7.   Propagate the prefix to tags
8.
9.   /* 분석 단계 */
10.  For intTimeSlot = 0 to 4   {
11.    R = Receive(TagID)
12.
13.    If (isEmpty(R))
14.      Delay(d)
15.      Continue  // next Time Slot
16.    }
17.
18.    If ( intTimeSlot == 0 or 1)
19.      fillChar = '0'
20.    Else
21.      fillChar = '1'
22.
23.    bytePre = fill(fillChar)
24.    inner = false
25.
26.    For i = 0 to length of R   {
27.      If ( AtChar(R, i) == 'X' )
28.        If ( i < 3 )              {
29.          bytePre[i] = not fillChar
30.
31.          /* 저장 단계 */
32.          PushQueue ( Prefix + subString( bytePre, 3) )
33.          bytePre[i] = fillChar
34.          inner = true
35.        }
36.        Else if ( i == 3 and inner )
37.          break;
38.        Else   {
39.          /* 저장단계  */
40.          PushQueue ( Prefix + subString( bytePre, i ) )
41.          break
42.        }
43.      Else
44.        bytePre[i] = AtChar(R, i)
45.    }
46.  }
47. }
```

(그림 6) 리더의 의사 코드

3.3 동작예제

(그림 7)은 임의적으로 생성된 9-bit 태그 8개를 인식하는 과정을 나타내고 있다. 먼저 리더가 태그에게 큐에서 얻는 프리픽스 'ε' 문자열을 전파하면, 모든 태그는 리더에게 자신의 상위 3비트 값에 따라 응답슬롯을 선택하고 각 슬롯에 자신의 ID를 전송한다.

스텝 1)에서 0 타임에 응답하는 태그는 하나이기 때문에 태그가 인식된다. 1타임 과 2타임에 응답한 태그는 2개 이상이므로 충돌 비트를 확인해 다음 쿼리를 위한 프리픽스를 생성한다. 1타임에 응답하는 태그는 "001101110"와 "001111011", "100100011"이다. 이때 충돌은 최상위 3비트에서 발생하는데 맨체스터 코드에 의해 버퍼에는 "x0x"로 기록되기 때문에 이를 분석하여 큐에는 "001"과 "100" 두 개의 새로운 프리픽스만 저장된다. 이와 유사하게 2타임에서는 버퍼에 "1xx"가 기록되어 새로운 프리픽스 "110"와 "101"이 존재한다는 것을 확인할 수 있다. 이러한 과정을 반복하여 9bit의 태그 8개를 7번의 쿼리를 통해 모두 인식하였다. 예제에서 제시한 태그를 인식하기 위해 필요한 질의 횟수는 QT경우 8개의 총 25회 HQT는 21회가 요구된다. 그러나 제안된 논문의 경우 충돌 이후 아이디의 비트 정보를 파



(그림 7) H^2QT 동작 예제

악할 수 없기 때문에 빈 슬롯의 수가 증가 할 수 있다는 단점을 가지고 있다. 그러나, 제안된 알고리즘은 충돌 사이클을 줄임으로써 트리의 노드 수를 줄이고, 특정 그룹으로 분류되어 있는 경우 좋은 성능을 발휘하는 것으로 성능평가에서 나타났다.

4. 성능 평가

논문에서는 제안하는 H^2QT 알고리즘의 성능 평가를 위해 C#언어로 시뮬레이션 프로그램을 설계하여 기존에 제안되었던 QT, 4-Ary QT 알고리즘 대하여 리더와 태그의 질의/응답 회수, 전송 비트 수를 비교하였다. 그리고 태그 ID의 길이는 EPCglobal에서 제안하는 EPC code를 감안하여 96비트로 하였다.

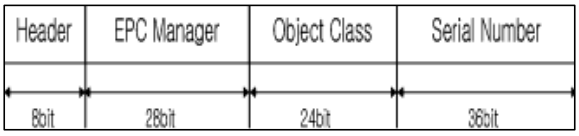
96bit 태그는 (그림 8)과 같이 Header, EPC Manager, Object class, Serial number로 구성되어있다. Header는 데이터 유형 및 길이를 정의하고, EPC Manager는 상품에 대한 분류와 일련번호를 관리하는 기관이나 기업을 표시한다. Object Class는 바코드의 상품 품목코드에 해당하는 것으로 각 품목 또는 단위를 표시하고 Serial Number 는 각 상품들에 대하여 부여되는 고유한 식별번호를 나타낸다

실험에서 실제 환경과 유사성을 고려하기 위해 Header, EPC Manager, Object class 필드를 먼저 무작위로 10개를 생성하고, Serial Number 필드는 두 가지 방법으로 생성한 후 접목 하였다. 첫 번째 방법은 36비트 중 랜덤 값을 선택하고 1씩 증가 시키면서 태그를 생성하는 방법이고, 두 번째 방법은 Serial Number 필드를 다시 임의로 생성하는 것이다.

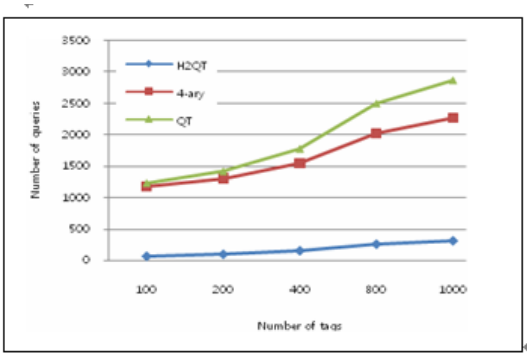
Serial Number필더를 제외한 나머지의 비트가 같다는 것은 동일 제품 코드를 나타내는 것이며 Serial Number가 순차적으로 있는 것은 물류 창고에 동일 제품이 여러 개 쌓여 있는 것을 의미한다.

태그의 개수는 순차적일 경우 각각 10의 군에 대해 10개부터 100개까지 생성하여 전체 태그개수가 100개부터 1000개까지 비교하였다. Serial number 필드가 랜덤한 경우에는 5개 군을 형성해 각 군에 대해 태그 100개부터 500개까지 생성하여 전체 태그의 개수를 1000개부터 2500개까지 실험하였다.

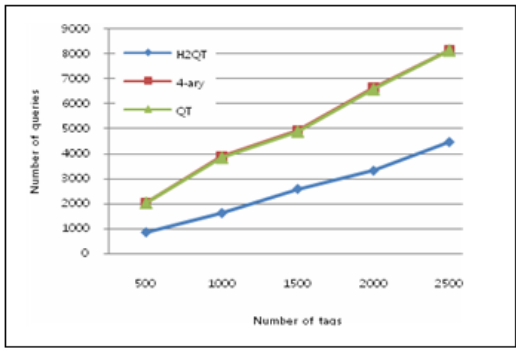
실험에서는 제안한 알고리즘과 4-ary 알고리즘과 QT 알고리즘으로 쿼리 횟수와 1개 태그를 인식하기 위한 평균 쿼리 수를 비교하였다. (그림 9)의 (a)에서의 결과를 보면 H^2QT 에서 Serial Number필더가 순차적인 100개의 태그를



(그림 8) EPC 96 bit Tag Structure

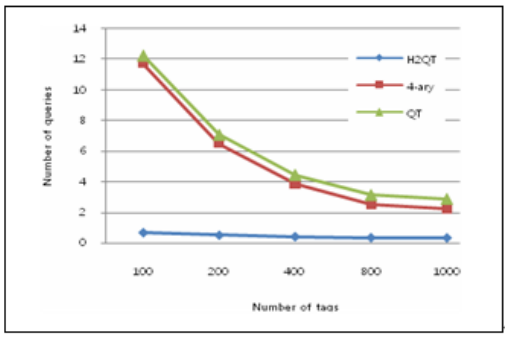


(a) Sequential assignment in serial number field

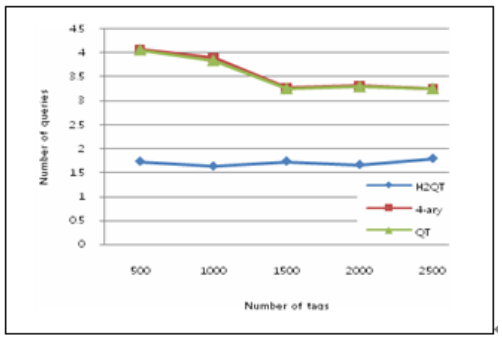


(b) Random assignment in serial number field

(그림 9) 96bit Tag인식을 위한 쿼리 수



(a) Sequential assignment in serial number field



(b) Random assignment in serial number field

(그림 10) 하나의 태그를 인식하기 위한 평균 쿼리 수

인식하기 위해 필요한 쿼리의 수는 66이다. 이는 4-ary 알고리즘의 1173개에 비해 성능이 10배 이상 향상되었다. (그림 9)의 (b)는 Serial Number 필드가 임으로 생성된 태그의 실험 결과이다. 태그가 랜덤한 ID를 가지더라도 인식 성능은 QT와 4-ary트리에 비해 향상되었다. QT와 4-ary 트리의 쿼리 횟수가 거의 같은 성능을 보인다. 이것은 4-ary 트리의 특성상 전체 collision cycle의 수는 줄였으나 idle cycle의 증가로 인해 QT와 비슷한 성능을 보이는 것이다.

(그림 10)은 한 개의 태그를 인식하기 위한 평균 쿼리의 수를 나타내고 있다. Serial number 필드가 순차적일 때 랜덤 할 경우보다 다른 알고리즘에 비해 성능이 더 높게 나타났다.

5. 결 론

RFID 시스템은 다수의 태그를 리더가 빠르게 인식할 수 있는 알고리즘이 필요하다. QT에서는 한 비트씩 확장하면서 태그를 인식하고 4-ary tree에서는 2비트씩 확장하면서 인식과정을 거친다. 하지만 확장과정을 거치면서 idle cycle이 늘어나게 되어 불필요한 질의 응답과정이 생긴다. 본 논문에서는 이러한 문제점들을 개선하여 태그확장을 빠르게 할 수 있는 새로운 H^2QT 를 제안하였다.

그러나 H^2QT 는 태그 비트 확장 속도 면에서는 빠르나 태그가 랜덤하게 분포되어있을 때 응답과정에서 빈 슬롯이 발생하게 된다. 슬롯의 응답타임을 어떻게 설정하느냐에 따라 시스템의 전체 인식시간이 결정될 것이고 만약 응답 슬롯이 타임이 길어진다면 태그 인식 시간 또한 늘어날 것이다. 응답슬롯이 길어짐에 따라 빈 슬롯이 발생한다면 낭비되는 시간 또한 늘어나게 되는 것이다. 향후 낭비되는 빈 슬롯을 줄일 수 있도록 응답 슬롯을 동적으로 할당하는 연구가 필요하다.

참 고 문 헌

[1] F. Mattern, “The Vision and Foundations of Ubiquitous Computing,” Upgrade, Vol.2, No.5, pp2-6, October, 2001.

[2] S. Sarma, D. Brock, and D. Engels, “Radio Frequency Identification and the Electronic Product Code,” IEEE Micro, vol.21, No.6, pp.50-54, November, 2001.

[3] J. Myung, and W. Lee, “Adaptive Binary Splitting: A RFID Tag Collision Arbitration Protocol for Tag Identification,” ACM/Springer Mobile networks and Applications (ACM MoNET), Vol.11, No.5, pp.711-722, October, 2006.

[4] EPC™ Radio-Frequency Identification Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860MHz Version 1.0.8, EPCglobal, Dec., 2004.

[5] Information technology automatic identification and data capture techniques-Radio frequency identification for item management Air interface. Part 6. Parameters for Air interface communications at 860-960 MHz, Final Draft International Standard ISO 18000-6, Nov., 2003.

[6] K. Finkenzeller, RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification. John Wiley & Sons, 2003.

[7] T.Wang, “Enhanced binary search with cut-through operation for anti-collision in RFID systems,” IEEE Commun. Lett., Vol.10, no.4, pp.236-238, April, 2006.

[8] J. H. Chio, D. Lee, H. Lee, “Query tree-based reservation for efficient RFID tag anti-collision,” IEEE Commun. Lett., vol, 11, No.1, pp.85-87, April, 2006.

[9] C. Quan, “Design and Performance Evaluation of High Performance Anti-collision Algorithm in the RFID System,” Ph.D. thesis. Dept. of Computer & Communications Engineering, Daegu Univ., Dec., 2004.

[10] J. Ryu, H. Lee, Y. Seok, T. Kwon and Y.Choi, “A Hybrid Query Tree Protocol for Tag Collision Arbitration in RFID system,”, IEEE ICC, pp 5981-5986, Jun., 2007

[11] J.H. Myung, W. J. Lee, J. Srivastava, T.K. Shih, “Tag-Splitting: Adaptive Collision Arbitration Protocols for RFID Tag Identification”

[12] Ching Law, Kayi Lee, and Kai-Yeung Sju, “Efficient Momoryless Protocol for Tag Identification”, *In Proceedings of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communication*, ACM, pp.75-84. August, 2000.

김 태 희

e-mail : kth3916@hotmail.com

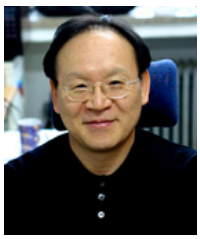
2006년 대구대학교 전자계산학과(학사)

2007년~현 재 경북대학교 대학원 전자전기컴퓨터학부(공학석사)

관심분야 : 유비쿼터스, RFID, Embedded System

이 성 준

e-mail : imggaibi@knu.ac.kr
1997년 경일대학교 컴퓨터공학과(학사)
1999년 경북대학교 컴퓨터공학과 대학원(공학석사)
2007년 경북대학교 컴퓨터공학과 대학원(공학박사)
2008년~현 재 경북대학교 컴퓨터공학과 초빙교수
관심분야 : 유비쿼터스, 스마트 홈, RFID, Embedded System



안 광 선

e-mail : gsahn@knu.ac.kr
1972년 연세대학교 전기공학과(학사)
1975년 연세대학교 전기공학과 (공학석사)
1975년~1976년 스페리유니백 근무
1980년 연세대학교 전기공학과(공학박사)
1976년~현 재 경북대학교 컴퓨터공학과 교수
관심분야 : 유비쿼터스, 센서네트워크, RFID, Embedded System