미리 분배된 난수를 이용하는 빠른 충돌방지 알고리즘

준회원 강 전 일*, 박 주 성*, 정회원 양 대 헌**

Fast Anti-Collision Algorithm Using Pre-distributed Random Address

Jeon il Kang*, Ju sung Park*, Dae hun Nyang* ***Regular Member***

요 약

RFID 시스템의 성능을 결정짓는 가장 중요한 요소 중의 하나가 충돌 방지 알고리즘이다. 충돌 방지 알고리즘의 성능을 높임으로써 단위 시간당 처리할 수 있는RFID 태그의 숫자를 늘릴 수 있다. 현재 사용되고 있는 충돌 방지 알고리즘은 ALOHA 프로토콜에 기반한 방법과 이진트리탐색 방법이 있으며 이들 알고리즘은 아직 개선의 여지가 많이 남아있다. 이 논문에서는 미리 분배된 난수를 사용하여 보다 빠르고 효율적인 충돌 방지 알고리즘인 AAC(Address Allocating & Calling) 방식을 제안한다. 그리고 수학적으로 이 방법을 분석하고 모의실험을 통하여 이 성능을 증명한다.

Key Words : RFID system, Anti-collision algorithm, Binary tree-walking

ABSTRACT

One of the most important factors that decide the overall performance of RFID system is anti-collision algorithm. By enhancing the anti-collision algorithm, we can increase the number of RFID tags that can be processed in unit time. Two anti-collision algorithms are most widely prevailed: one is ALOHA-based protocol and the other is a binary tree walking method, but these are still under research. In this paper, we suggest an anti-collision algorithm named AAC(Address Allocating and Calling) using pre-distributed random address, which is much faster and more efficient than existing ones. Finally, we evaluate our scheme using mathematical analysis and computer simulation.

Ⅰ. 서 론

무선 인식, 즉 RFID 시스템은 물류, 유통 분야 및 금융서비스 등에서 사용되는 바코드 시스템의 대체 이외에 유비쿼터스 컴퓨팅 환경 하에서 컴퓨터의 사물 인식 분야에서 중요한 역할을 하게 될 것으로 예상된다. 이러한 RFID 시스템은 제조사간 호환성 문제와 백엔드 시스템의 부재 등의 문제이외에도 기술적으로 아직 완전하지 않은 시스템이라는 것은 큰 문제를 가진다.

RFID 시스템의 충돌방지 알고리즘은 RFID 태그의 실제 작동 원리를 정의한다고 해도 과언이 아니다. 그러나 현재의 RFID 시스템은 태그가 갖는 물리적인 제약사항과 맞물려 기존의 시스템에서 사용하는 충돌방지 알고리즘을 그대로 사용할 수 없다. 따라서 많은 알고리즘이 만들어졌으며 이 논문에서 논하고 하는 것도 그러한 알고리즘 중에 하나이다.

이 논문은 2장에서 기존의 충돌방지 알고리즘의 문

* 인하대학교 정보통신대학원 정보보호연구실({dreamx, security77}@seclab.inha.ac.kr)
** 인하대학교 정보통신대학원 정보보호연구실(nyang@inha.ac.kr)
논문번호 : KICS2004-08-132, 접수일자 : 2004년 8월 4일

제점을 지적하고 3장에서는 새로운 방법에 대하여 기술한다. 4장에서는 새로운 방법이 갖는 문제점과 이를 보안하여 최종적으로 완성된 모델을 설명할 것이다. 5장에서는 기존의 이진트리탐색 기법과 새로운 방법에 대한 수학적 분석을 한다. 6장에서는 모의실험의 결과와 이를 바탕으로 한 최종적인 성능을 예상해본다. 마지막으로 7장에서 이를 간략히 정리한다.

Ⅱ. 기존의 충돌방지 알고리즘의 문제점

RFID 시스템의 충돌방지 알고리즘은 시간을 축으로 원하는 시간에 데이터를 전송하는 알로하(ALOHA) 방식과 태그의 데이터가 0과 1로 이루어진 이진성을 이용한 이진트리탐색(Binary Tree-walking) 방식을 응용하는 방법들이 있다.[1]

2.1 알로하 기반 기법

알로하 기반 기법의 경우 모든 태그들이 리더에게 데이터를 보내기 위하여 경쟁하는 과정을 거치게 된다. 태그가 미디어의 상태를 체크하기 어려운 이유로 RFID 시스템에서는 시간을 슬롯(또는 프레임)으로 나누고 동기화된 시간에 데이터를 전송하게 된다. 이 방법의 장점은 연속적으로 태그의 데이터를 전송함으로서 확정적 전송 아래에서의 높은 전송속도가 보장된다는 것이다. 반면 단점은 태그의 숫자가 예측 불가능한 상황에서 찾아볼 수 있다. 태그의 숫자가 적을 경우 알로하 기반 알고리즘은 슬롯을 낭비하게 된다. 반대로 태그의 숫자가 지나치게 많을 경우 태그들이 경쟁하면서 발생한 충돌로 인하여 인식에 필요한 슬롯이 폭발적으로 증가하게 되며 심각한 경우 시간이 무한대로 늘어나도 하나의 태그 정보도 알아낼 수 없을 수 있다는 것이다. 또한 태그가 리더에게 데이터를 보내기 위한 시점을 결정하기 위하여 RNG(Random Number Generator)의 사용이 불가피하다.

2.2 이진트리탐색 기반 기법

이진트리탐색 기반 기법을 사용한 RFID 시스템의 경우, 리더는 태그의 정보를 한 비트 씩 읽어 들이게 된다. 리더는 태그에게 'next bit'나 'who is x'와 같은 명령을 전송함으로서 태그를 제어한다. 결과적으로 리더는 트리를 검색하는 것처럼 태그의 정보를 찾아 가게 된다. 이 방법의 장점은 확정적인 데이터 전송에서 찾아볼 수 있다. 리더는 태그가 보내는 모든 데이터를 사용하게 되므로 충돌로 인하여 불필요하게 사라지는 데이터는 없는 것과 같다. 또한 많은 수의 태그가 존재하거나 적은 수의 태그가 존재하여도 이에 따른 전체 성능을 선형적인 수식에서 예측할 수 있다. 반면 단점은 리더가 태그에게 태그가 리더에게 보내는 데이터만큼이나 많은 명령 쿼리를 전송해야 한다는 것이다. 또한 태그는 리더에게 데이터를 보내기 위하여 전력을 충전하고 방전하는 과정이 지나치게 빈번하다는 것도 문제이다.

Ⅲ. 새로운 충돌방지 알고리즘

3.1 목표

지금까지 이 논문에서는 기존의 충돌방지 알고리즘의 문제점을 고찰하였다. 그리고 이제부터는 이를 해결하기 위한 방안으로 새로운 충돌방지 기법을 제안한다. 이 논문에서 AAC(Address Allocating & Calling)라고 부르는 이 기법은 선형적으로 모든 태그들의 데이터를 읽어 들이는 시간을 예측할 수 있게 하며 이 시간은 알로하 기반 기법이나 이진트리탐색 기반 기법에 비하여 빠르다. 또한 태그는 스스로 생성하는 난수를 사용하지 않기 때문에 RNG를 포함하지 않는다.

3.2 가정

이 논문에서는 BT와 IBT이라는 두 가지 전송시간 단위를 사용한다. BT(Bit Time)는 태그에서 리더로, 또는 리더에서 태그로 순수하게 비트만 전송되는 시간을 의미한다. IBT(Inter-Bit Time)는 리더에서 태그로, 또는 태그에서 리더로 데이터의 방향이 바뀔 때 태그나 리더가 필요한 처리지연 시간과 데이터 인식의 안정성을 이유로 필요한BT와 BT 사이의 시간을 의미한다. 그러므로 비트가 연속으로 전송될 경우 IBT가 필요 없게 된다.

알로하기반 기법은 보통 전체 태그를 전송하기

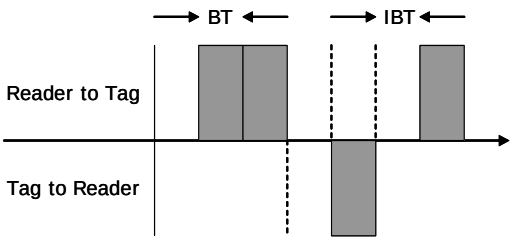

그림 1. 시간 축에서의 BT와 IBT, 회색부분의 시간이 BT, BT이외의 시간이 IBT이다.

위하서는 전체 태그 수의 3~4배의 슬롯이 필요하며 그 효율성은 25~33% 정도이다. 이진트리탐색 기법은 리더에 의해 보내지는 비트와 중복되는 비트를 고려한 수학적 분석 결과에 따르면 100%에 가깝다. 또한, 몇몇 논문과 기술문서[2][3]에 의하면 중복되는 비트의 이득이 없는 완전난수 태그의 경우 알로하 기반 기법이 이진트리탐색기반 기법에 비하여 빠를 수 있음을 알 수 있다. 이러한 사실을 올바르게 설명하기 위해서는 실제 시간을 충분히 고려한 새로운 시간 단위가 필요했으며 이러한 이유로 이 논문에서는 IBT라는 단위를 사용하게 되었다.

또한 이 AAC 기법에서 사용되는 태그의 경우 태그의 정보를 알아내기 위한 일련의 과정이 일어나는 동안에는(리더가 태그에게 전력을 계속 공급해 주는 동안에는) 태그의 메모리상에 어떠한 정보를 저장할 수 있는 능력이 있음을 가정한다. 이 가정은 뒤에 AAC 기법과 직접적인 성능 비교가 될 이진트리탐색 기법에서도 마찬가지로 요구된다.

또 다른 것으로, 리더는 태그가 데이터를 전송하는 과정에서 사이에서 발생하는 데이터 충돌의 위치를 정확히 알 수 있음을 가정한다.

3.3 미리 분배된 난수

AAC에서는 난수를 주소 값으로 사용한다. 이 주소 값은 임시적으로 태그를 구별하는 용도로 사용하게 된다. 태그에 RNG를 포함하여 시작 시간에 난수를 생성할 수 있으나 지속적으로 사용되지 않는다는 점, 그리고 RNG에 의해 발생된 난수를 새로운 주소를 할당하는 데 사용하였을 때 이를 제어할 수 없다는 점을 고려하였을 때 RNG를 사용하여 생성한 난수를 주소 값으로 사용하는 것은 불필요해 보인다. 그러므로 AAC에서는 미리 분배된 난수를 사용하게 된다. 난수의 범위는 모든 태그를 구별할 수 있을 정도로 커야 한다. 하지만 확률적으로 충돌을 회피하기 위해서 예상되는 태그 수의 2배 이상으로 잡을 필요는 없다. 물론, 독립적인 메모리 공간에 꼭 이를 저장할 필요는 없으며, 난수처럼 처리될 수 있다면 태그 데이터의 임의의 부분을 이용하여도 상관없다. 그렇게 하기 위해서는 중복이 잘 일어나지 않는 일련번호(serial number) 부분을 이용하여야 하지만 일련번호가 0으로 채워진 경우 난수를 미리 분배하는 방식을 사용하여야 한다.

이 난수는 태그 제조사에서 임의로 분배할 수 있을 것이다.

3.4 주소를 저장하기 위한 메모리 공간

AAC 기법을 사용한 태그는 리더에 의하여 전원이 공급되면 미리 분배된 난수를 임의의 메모리 공간에 복사하게 된다. 이 메모리 공간은 언제든지 업데이트가 가능하지만 단지 전원이 공급되는 동안만 데이터를 유지하면 된다. 그리고 AAC 태그는 이 메모리 공간에 있는 데이터를 자신의 주소 값으로 인식하게 된다. 리더는 첫 번째 단계에서 이 메모리 공간에 복사된 값을 알아오는 것부터 시작한다. 기본적으로 이진트리탐색 기법을 사용하지만 다른 방식이 있다면 그것을 사용해도 된다.

3.5 주소 값 할당과 호출

이 기법은 기본적으로 태그가 RNG를 포함하는 대신 미리 분배된 난수(즉, 주소)를 가지고 있다는 것이다. 하지만 확률적으로 임의의 한 태그는 다른 태그와 동일한 주소를 가지는 것을 피하지 못한다. 하지만 이 주소는 태그들을 적당히 작은 수의 개체의 그룹으로 분류할 수 있을 것이다. 리더는 그룹의 주소를 호출하여 태그들의 부분 데이터를 얻어온다. 이 때 발생하는 충돌을 기록하고 이 충돌 위치를 바탕으로 리더는 한 그룹의 태그들에게 다른 태그와 구분되는 적당한 주소를 새로이 할당할 수 있다.

리더는 태그의 주소를 호출함으로서 특정한 태그의 행동을 제어할 수 있게 된다.

3.6 단계별 행동

AAC는 4단계를 가지고 있으며 연속적으로 진행되지만, 기본적으로 태그는 플래그 비트가 삽입된 명령 쿼리에 따라서 각 단계에 독립적으로 작동한다. 즉, 태그의 주소를 가지고 있다면 얼마든지 바로 태그의 정보를 알아낼 수 있다. 아래는 이 단계들을 간략히 정리한 것이다.

- 1단계 : 리더는 이진트리탐색 기법(다른 방법이어도 크게 상관은 없다)을 사용하여 태그들이 기본적으로 가지고 있던 난수인 주소들을 알아내어 이를 저장한다.
- 2단계 : 리더는 1단계에서 알아낸 주소를 순차적으로 호출한다. 태그는 자신의 주소가 호출되었을 때만 자신의 마지막 몇 비트를 전송한다. 리더는 주소를 식별자로 하는 레코드에 태그에서 얻어온 비트들을 저장한다. 만약 이 비트들에 충돌이 발생하였을 경우 그 위치를 저장한다.
- 3단계 : 레코드를 순차적으로 검색하여 2단계에서 리더가 저장한 태그들의 마지막 몇 비트에

충돌을 발견했을 경우(즉, 하나의 주소에 여러 개의 태그가 몰려있을 경우) 리더는 이 정보를 바탕으로 태그를 제어하여 태그에게 지금까지 사용되지 않은 새로운 주소를 할당한다.

- 4단계 : 리더는 3단계까지의 과정까지 주소를 식별자로 분류된 태그들을 주소를 사용하여 순차적으로 호출하면서 태그의 나머지 데이터를 얻어온다.

Ⅳ. 보다 향상된 충돌방지 알고리즘

여기서는 위에서 설명했던AAC의 한계를 지적하고 이를 해결하기 위한 보다 향상된 방식을 설명한다. 이를 AAC-P(Parking mode)라고 부르며 이와 구별하기 위해서 AAC를 AAC-O라고 부른다.

4.1 기본 방식의 제한사항

확률적으로 3단계까지 알아낸 비트를 가지고도 4단계에서의 충돌을 완전히 없앨 수는 없다. 이를 해결하기 위해서는 5단계를 두고 마치 3단계에서 했던 것과 유사한 행동을 취할 수 있을 것이다. 그러나 이는 효율성의 측면에서 추천하지 않는다. 또한 태그의 수가 난수의 범위를 초과했을 경우 문제가 발생하게 된다.

그림 2. AAC-P 데이터그램

4.2 변경 사항

AAC-O는 '업데이트 가능한 첫 번째 주소 공간'이 특별히 필요했으나, AAC-P에서는 추가적으로 '업데이트 가능한 두 번째 주소 공간'이 더 필요하다. 이 공간은 기본적으로 '0x0'으로 초기화된다.

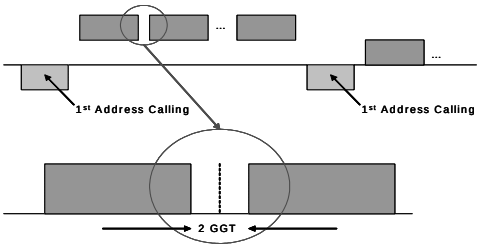

그림 3. 기본적인 AAC-P 블록단위 데이터 전송

그리고 AAC-O가 마지막 4단계에서 전체 태그의 데이터 그램을 한 번에 전송하는 것과 다르게 AAC-P는 블록 단위로 전송하게 되며 블록전송이 완료하면 쿼리의 명령을 기다리고 특별한 명령이 없다면 계속하여 다음 블록을 전송하게 된다.

4.3 네스팅과 파킹(Nesting & Parking)

AAC-P에서는 3단계와 4단계를 혼합한 방식을 사용한다. 이 과정에서 2단계의 필요성이 없어지게 되었으며 AAC-P에서는 1단계(주소 검색 단계)와 2단계(전송 단계)만을 가지고 있다.

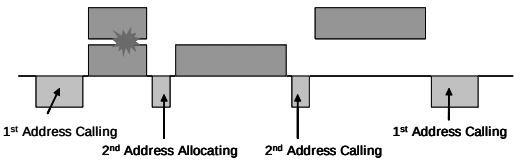

그림 4. 네스팅(Nesting)과 파킹(Parking)

네스팅(Nesting)은 AAC-O의 3단계에서 사용하는 방식을 응용한 것으로 충돌이 발생한 시점부터 새로운 레코드를 할당하고(두 번째 주소 공간을 증가시키고) 모든 데이터가 안전하게 전송되었으면 기존의 레코드로 돌아가(두 번째 주소 공간을 감소시킨다)는 과정을 말한다.

파킹(Parking)은 모든 데이터가 안전하게 전송되었다는 것을 확인하면 자동적으로 첫 번째 주소 공간을 예약된 절대 호출되지 않을 주소(가령 '0x0')로 바꾸고 자신의 주소를 다시 사용할 수 있도록 하는 일련의 과정을 의미한다.

이 방법은 첫 번째 주소를 바탕으로 두 번째 주소를 증가시키거나 감소시키면서 작동하게 된다. 리더가 지시한 위치의 특정 비트에 해당하는 태그를 제외한 첫 번째 주소를 호출당한 태그들은 충돌이 발생할 때마다 두 번째 주소 값을 증가시킨다. 태그는 기본적으로 첫 번째 주소가 호출당하고 두 번째 주소 값이 '0x0'인 경우에만 태그의 데이터 전송을 시도한다. 태그의 모든 데이터가 전송이 완료되면 리더는 두 번째 주소 값을 감소시키라는 명령을 내린다. 첫 번째 주소에서 모든 태그가 전송을 완료하면 리더는 다른 첫 번째 주소를 호출하여 위의 과정을 반복한다.

만약, 네스팅 횟수가 두 번째 주소의 최고값보다 크게 되어 더 이상 네스팅 할 수 없을 경우 빈 첫 번째 주소 값을 할당하고 이렇게 새로운 첫 번째

주소가 할당된 태그들의 두 번째 주소 값을 '0x0'으로 초기화함으로서 네스팅을 이어간다.

만약 빈 첫 번째 주소가 없다면 리더는 다른 첫 번째 주소를 호출함으로서 빈 첫 번째 주소가 발생하길 기대한다. 빈 첫 번째 주소가 발생하는 시점에서 리더는 이전의 네스팅을 이어갈 수 있다.

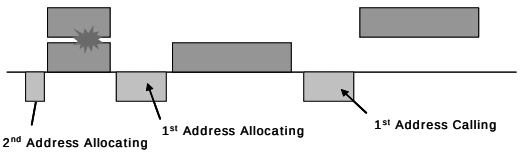

그림 5. 네스팅 과정으로서의 첫 번째 주소 할당

```
read 1st address from TAG

for all 1st address
   send 1st address calling query
   for all block
      read block
      if detect collision in block
         if 2nd address size is max
            if no empty 1st address
               break
            else
               send 1st address allocating query
         else
            send 2nd address allocating query
      else if end of tag
         if max 2nd address size not 0
            send 2nd address calling query

for all break
   if no empty 1st address
      /* do nothing */
   else
      resume break
```

그림 6. AAC-P 의사진행코드(리더)

```
send 1st address to reader
if my 1st address calling
    set 2nd address by 0
    for all block
      if 2nd address is 0
         send block
         if no query
            /* do nothing */
         else if my 2nd address allocating
            increase 2nd address
      else
         if 2nd address calling
            decrease 2nd address
            if 2nd address is 0
               re-send block
   enter parking mode
```

그림 7. AAC-P 의사진행코드(태그)

그림 6과 그림 7은 이러한 과정을 의사진행코드로 표현한 것이다. 이 알고리즘에서 이진트리탐색 기법을 사용하는 부분에서 이진트리탐색 기법에 대한 내용은 그림 8이나 다른 논문을 참고하길 바란다.

```
send next_bit_with_no_collision query

loop ∞
   read bit
   if bit is last one
      save all bit to record
      if collision position list not empty
         pop collision position list
         send wake_up query
      else
         break loop
   else if bit is collision
      push collision position list
      send next_bit_with_collision query
   else
      send next_bit_with_no_collision query
```

그림 8. 이진트리탐색 기법 의사진행코드(리더)

V. 수학적 분석

이 장에서는 성능에 대한 직접적인 비교를 위하여 이진트리탐색 충돌 방지 알고리즘과 제안한 충돌 방지 알고리즘에 대한 수학적인 분석을 수행한다.

5.1 이진트리탐색

확률적으로 예측 가능한 태그를 위하여 태그의 데이터는 완전난수를 가정한다. 그렇게 되면 최소의 트리에서 모든 태그들이 분류가능상태가 되며 그 뒤의 데이터는 리더와 태그가 1:1로 데이터를 주고받는 상태에 놓이게 된다. 트리가 발생하는 부분은 충돌이 발생하는 부분이며 이 부분에 대해서는 리더가 태그에게 트리의 높이 정보를 포함한 'wake up' 쿼리를 전송하게 된다. 이러한 내용은 그림 8에 의사진행코드로 표현되어 있다.

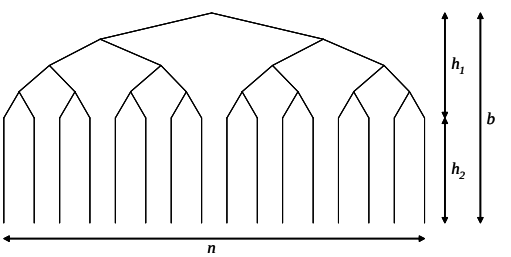

그림 9. 이진트리탐색 충돌방지 알고리즘 동작 모델

그림 9는 이진트리탐색 알고리즘의 동작 모델을 보여준다. 이 모델에서는 태그의 비트가 완전 난수에 가까우며, 태그 하나하나를 구별 가능한 시점의 비트 수를 h_1이라고 하며 전체 비트에서 h_1을 제외한 비트 수를 h_2라고 한다.

표 1. 이진트리탐색 기법에 사용되는 명령 쿼리

비트	이름
0	next bit with no-collision
1	next bit with collision
충돌위치	wake up

b를 '태그의 비트 수', n을 '태그의 수', m을 'b 깊이를 표시할 수 있는 최소 비트 수(즉, $\lceil \log_2 b \rceil$)'라고 하고 그림 8과 표 1에서 제시된 방법을 사용한다고 하였을 때 $h_1 \approx \log_2 n$이므로 $h_2 \approx b - \log_2 n$이 된다. 그러므로 구분가능상태에서부터 전송하는 리더와 태그의 BT의 합은 다음과 같다.

$$2n(b - \log_2 n)$$

그리고 충돌 때문에 발생한 트리에 대한 wake up 쿼리의 BT의 합은 쿼리의 길이와 충돌이 일어난 횟수를 곱하면 되므로 다음과 같다.

$$m \sum_{k=1}^{\log_2 n - 1} 2^k = \lceil \log_2 b \rceil (n-2)$$

또한 구분가능상태가 될 때까지 next bit 쿼리와 태그의 응답 BT의 합은 다음과 같다.

$$2 \sum_{k=1}^{\log_2 n - 2} 2^k = n - 4$$

그러므로 n에 대해서 정리한 이진트리탐색 기법의 BT의 합은

$$(2b - 2\log_2 n + \lfloor \log_2 b \rfloor + 1)n - 2\lceil \log_2 b \rceil - 4 \qquad (1)$$

와 같다. IBT는 BT에서 m이 1과 같은 값을 갖게 되므로

$$2(b - \log_2 n + 1)n - 6 \qquad (2)$$

와 같다.

표 2. AAC-P에 사용되는 명령 쿼리

비트	이름
0	next bit with no-collision
1	next bit with collision
00+충돌위치	wake up
01+첫 번째 주소	1st address calling
10+충돌위치+새로운 첫 번째 주소	1st address allocating
11+0	2nd address calling
11+충돌위치	2nd address allocating

5.2 AAC-P

우선 B를 '블록사이즈', A를 '첫 번째 주소 값에 사용되는 비트 수', D를 'A를 깊이를 표시할 수 있는 최소 비트 수', N을 '두 번째 주소 값에 사용되는 비트 수'로 추가로 변수를 가정하자.

5.2.1 주소 검색 단계에서의 BT

이진트리탐색 기법을 이 분기에 사용하게 되면 위에서 밝힌 바와 같이 이 분기에 사용되는 BT는 $2^A > n$인 경우에 b를 A로, m을 D+2로 치환하는 것과 같다. D는 $\lceil \log_2 A \rceil$와 같고, 특별히 'wake up' 쿼리가 2비트의 플래그 비트를 포함하게 됨을 유의하라.

$$2n(A - \log_2 n) + (\lceil \log_2 A \rceil + 3)(n-2) - 2 \qquad (3\text{-}1)$$

$2^A < n$인 경우에는 모든 트리에서 충돌이 발생한다고 생각할 수 있다. next bit 쿼리의 횟수는 $2^0 + 2^1 + \cdots\cdots + 2^{A-1} = 2^A - 1$와 같고 리더와 태그가 주고 받으므로 'next bit' 쿼리에 사용된 BT는 이의 2배이다. 'wake up' 쿼리의 횟수는 $2^0 + 2^1 + \cdots\cdots + 2^{A-2} = 2^{A-1} - 1$과 같으므로 이 경우 BT는 다음과 같다.

$$2(2^A - 1) + (\lceil \log_2 A \rceil + 2)(2^{A-1} - 1) \qquad (3\text{-}2)$$

5.2.2 전송 단계에서의 BT

$2^A > n$인 경우 네스팅은 발생하지 않는다고 볼 수 있으므로 BT는

$$n(b + A + 2) \qquad (4\text{-}1)$$

와 같다. 반대로 $2^A < n$인 경우 네스팅에 의한 추가 비트가 필요하게 되는데 네스팅이 발생하는 횟

수는 $n-2^A$가 된다. 그러므로 이 경우 BT는

$$n(b+A+2)+(n-2^A)(B+\lceil\log_2 A\rceil+5) \quad (4\text{-}2)$$

이 된다.

5.2.3 주소 검색 단계에서의 IBT

$2^A > n$일 경우 앞의 식들을 참고하면 식(2)와 유사하게

$$2n(A-\log_2 n)+2n-6 \quad (5\text{-}1)$$

가 되고, $2^A < n$인 경우는 식(3-2)와 유사하게

$$2(2^A-1)+2^{A-1}-1=2^{A+1}+2^{A-1}-3 \quad (5\text{-}2)$$

이 된다.

5.2.4 전송 단계에서의 IBT

우선 $2^A > n$인 경우, 이 때 필요한 IBT는 1st Address Calling 쿼리 후에 1IBT와 한 블록을 전송하고 쉬는 2IBT, 그리고 마지막 블록을 전송한 후에는 1IBT만 쓰므로 이를 상쇄하면, 이때 필요한 전체 IBT는

$$n\times(2\times\left(\frac{b}{B}-1\right)+2)=\frac{2nb}{B} \quad (6\text{-}1)$$

이 된다.

네스팅을 고려해야 되는 $2^A < n$인 경우를 생각해 보면, 2nd Address Allocating 쿼리를 전송하려 할 때 충돌이 일어난 블록을 전송한 뒤1IBT를 쉬고 쿼리를 전송하고 다시 1IBT를 쉬므로 이는 네스팅이 없는 상황과 같다. 그리고 2nd Address Calling 쿼리를 전송하려고 할 때 1IBT를 쉬고 쿼리를 전송하고 다시 1IBT를 쉬게 되므로 이는 1IBT를 더 소모하게 되는 경우이다. 그리고 네스팅 횟수만큼 블록의 재전송이 일어나고 2IBT를 쉬게 되므로 이 경우 IBT는

$$\frac{2nb}{B}+2(n-2^A) \quad (6\text{-}2)$$

와 같이 된다.

$2^A > n$인 경우 식(3-1)과 (4-1)을 더하여 n에 대해서 정리한 AAC-P의 BT는

$$(3A-2\log_2 n+\lceil\log_2 A\rceil+b+5)n-2\lceil\log_2 A\rceil-8$$

이고 식(3-2)와 (4-2)를 더한 $2^A < n$인 경우는

$$(b+B+\lceil\log_2 A\rceil+7)n$$
$$-2^{A-1}(2B+2\lceil\log_2 A\rceil+1)+\lceil\log_2 A\rceil+4$$

와 같다.

또한 AAC-P의 IBT는 $2^A > n$인 경우 식(5-1)과 (6-1)을 더하여 n에 대해서 정리하면

$$2(A-\log_2 n+1+\frac{b}{B})n-6$$

이고 $2^A < n$인 경우 식(5-2)와 (6-2)를 더하면

$$2(\frac{b}{B}+1)n+2^{A-1}-3$$

와 같다.

한 가지 주의 할 점은, N은 태그 내부에만 사용되어 성능에는 별다른 영향을 미치지 않는다는 것이다. 단지, N이 크면 극한 상황(즉, 매우 많은 태그가 있어 심하게 네스팅이 발생하는 경우)에서의 작동을 보장해준다는 것이다.

표 3. 비교에 사용할 파라미터 수치

파라미터	수치
tag bit size(b)	128
tag depth(m)	7=⌈log₂128⌉
block size(B)	16
1st address size(A)	10
1st address depth(D)	4=⌈log₂10⌉
2nd address size(N)	5

표 4. 수학적 분석 결과 (log 계산은 자릿수 올림으로 계산)

n	BT		IBT	
	Binary Tree	AAC-P	Binary Tree	AAC-P
5	1332	789	1254	154
10	2702	1574	2494	294
100	27782	15284	24394	2394
500	140982	74484	119994	9994
1000	283982	146984	237994	17994

5.3 이진탐색 기법과 AAC-P 비교

따라서 이러한 파라미터를 사용했을 경우, 계산적으로 AAC-P는 이진트리탐색 기법에 대해서 BT는

53% 정도, IBT에 대해서는 8% 수준임을 알 수 있다. 이는 각각 189%와 1200% 정도 빠르다는 이야기다.

Ⅵ. 모의실험 결과

모의실험에는 리더와 태그의 객체를 선언하였고 리더와 태그는 쿼리를 통하여 통신하며, 리더와 모든 태그들은 독립적으로 작동하게 하였다. 모의실험 결과, 리더 안에는 모든 태그의 데이터가 보관되었으며 각각의 분기에 따른 BT와 IBT의 횟수가 기록되었다.

모의실험은 각각의 경우에 대하여 10회씩 실시하여 평균을 구하는 방식으로 수행되었다.

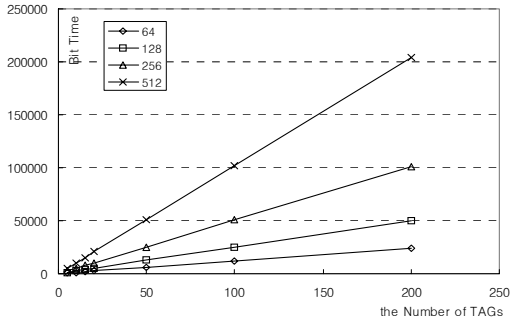

그림 10. 이진트리탐색 기법 모의실험결과(총 BT)

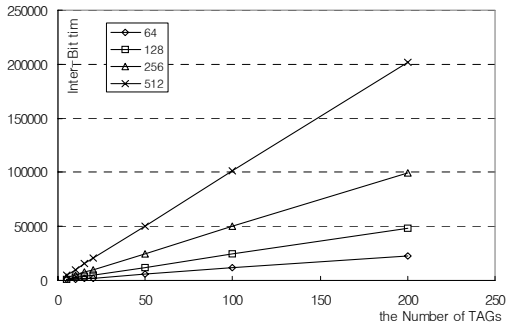

그림 11. 이진트리탐색 기법 모의실험결과(총 IBT)

6.1 이진트리탐색 기법

이진트리탐색 기법은 각각 64, 128, 256, 512 bits에서 수행되었다. 또한 태그의 숫자를 5, 10, 15, 20, 50, 100, 200으로 바꾸어가며 실험하였다. 실험결과, 이진트리탐색 기법은 분석을 통해서 보았던 것과 유사한 결과를 보여주었다.

이진트리탐색 기법은 태그의 수와 태그의 비트수에 비례하여 전체 BT와 IBT가 증가하게 됨을 알 수 있다.

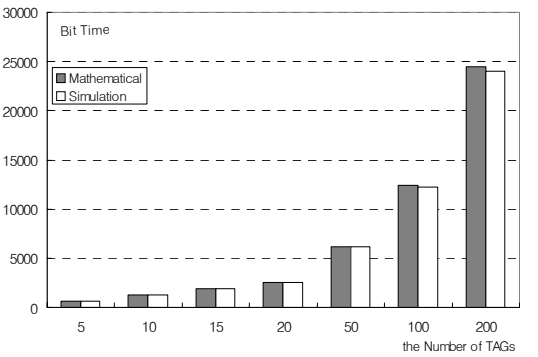

그림 12. 이진트리탐색 기법의 수학적 분석과 모의실험결과 비교(총 BT, 64-bit 태그)

6.2 AAC-O

AAC-O의 모의실험은 주소의 크기를 6-bit에서 12-bit까지 바꾸어가며 실험했으나 여기서는 주소의 크기가 10-bit인 결과만을 제시한다. 10-bit를 선택한 이유는 10-bit가 512-bit나 1024-bit에서 좋은 성능을 보이고, 또한 10-bit 주소 값의 크기는 1000개의 태그의 종류를 보장할 수 있는 안정적인 크기이기 때문이다.

AAC-O의 한 가지 특징은 IBT가 태그의 비트수에 영향을 거의 받지 않는다는 것이다. 그리고 그 수치는 이진트리탐색 기법과 비교할 때 현저하게 낮음을 알 수 있다. 물론, 이 수치는 AAC-P의 IBT 수치보다 더 낮다.

하지만 AAC-O는 확률적으로 모든 태그를 인식하지 않을 수도 있음을 주의하라. 이 실험결과는 AAC-P의 실험결과가 신뢰성이 있음을 보여줄 것이다.

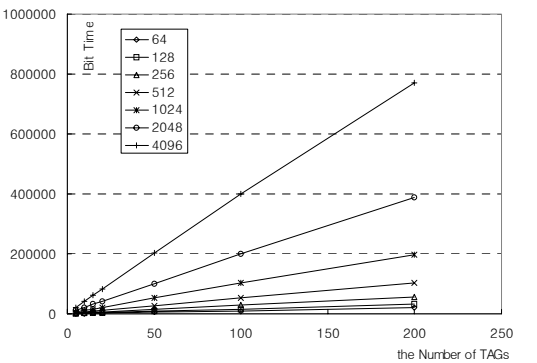

그림 13. AAC-O 모의실험결과(총 BT)

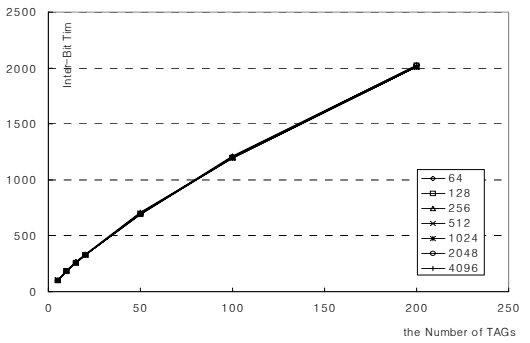

그림 14. AAC-O 모의실험결과(총 IBT)

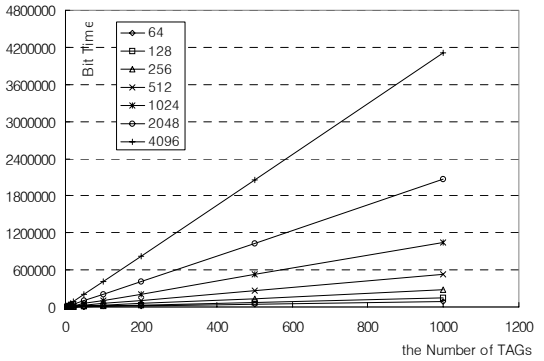

그림 15. AAC-P 모의실험결과(총 BT)

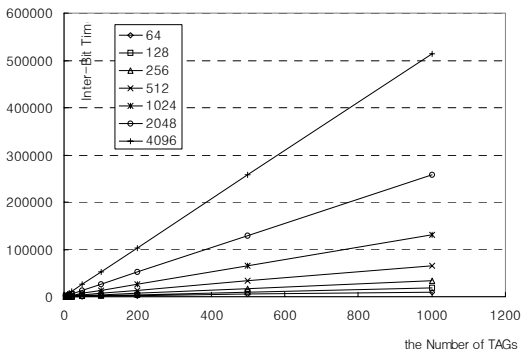

그림 16. AAC-P 모의실험결과(총 IBT)

6.3 AAC-P

AAC-P 모의실험에서는 기존의 실험들과 비교하기 위한 실험 이외에 잠재적 가능성에 대한 한계 실험을 병행하였다. 200개의 태그 이외에 500, 1000개를 추가로 수행하였다.

결과적으로 AAC-P의 BT는 AAC-O의 BT와 비교하여 대략적으로 비슷한 수치를 갖게 된다. 또한 태그의 수와 비트수에 비례하여 BT가 증가하게 된다.

그림 14과 그림 16를 비교하였을 때 AAC-P의 IBT는 태그의 비트수에 영향을 받으며 그 수치 또한 AAC-O에 비하면 경우에 따라서 약 3~20배 정도의 차이가 남을 알 수 있다.

6.4 성능 비교

그림 14과 그림 15에서 보는 것처럼 실험결과, AAC-P는 이진트리탐색기법에 비하여 BT에서는 165%정도, IBT에서는 1134%정도 빠르다는 것을 알 수 있다.

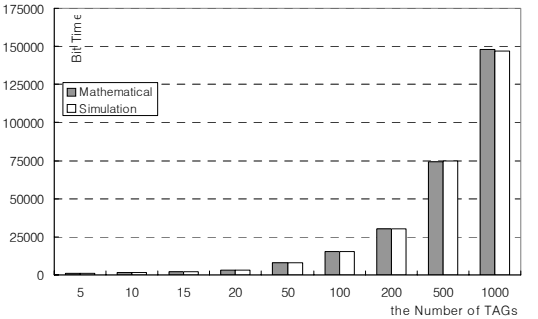

그림 17. AAC-P의 수학적 분석과 모의실험 결과(총 BT, 128-bit 태그)

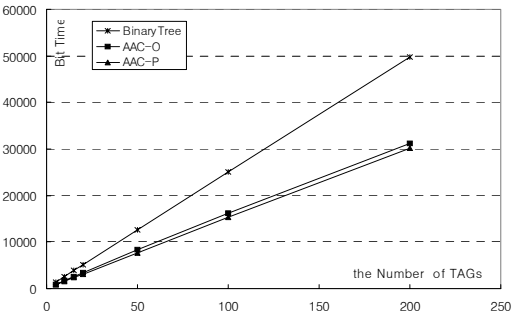

그림 18. 모의실험 총 BT 비교(128-bit)

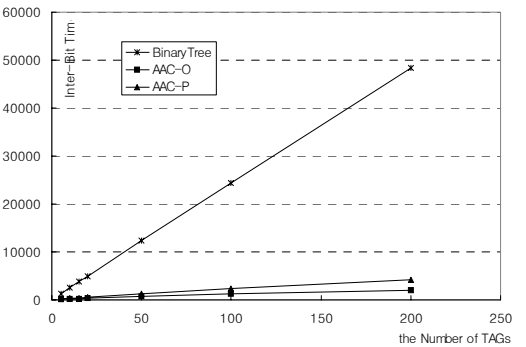

그림 19. 모의실험 총 IBT 비교(128-bit)

표 5. 성능예상표 I (128-bit 태그, BT=10μsec, IBT=10μsec, 단위 초)

TAGs	Binary Tree	AAC-O	AAC-P
5	0.025	0.010	0.010
10	0.051	0.019	0.019
15	0.076	0.028	0.028
20	0.101	0.037	0.037
50	0.249	0.090	0.090
100	0.494	0.173	0.176
200	0.981	0.331	0.345
500			0.840
1000			1.662

표 6. 성능예상표 II (128-bit 태그, BT=10μsec, IBT=50μsec, 단위 초)

TAGs	Binary Tree	AAC-O	AAC-P
5	0.076	0.014	0.016
10	0.151	0.026	0.031
15	0.225	0.038	0.045
20	0.300	0.050	0.059
50	0.741	0.118	0.139
100	1.469	0.221	0.269
200	2.916	0.412	0.516
500			1.224
1000			2.396

6.5 예상 성능

이 장에서는 BT와 IBT에 대하여 특정한 값을 대입하여 실제로 시스템이 구성되었을 때의 성능을 예측하여 보기로 하겠다.

위의 표 5와 표 6에서 보는 바와 같이 잠재적으로 AAC-P는 128-bit 태그를 450~500개를 1초에 분류할 수 있을 것으로 보인다. 또한 1000개를 분류하는 데에 걸리는 시간이 3초 미만이라는 것에는 주목할 만하다. 이것은 이진탐색트리에 비하여 3~4배정도 빠른 것이다.

Ⅶ. 결 론

이 논문에서는 미리 분배된 난수를 주소로 이용하여 태그의 충돌을 회피하는 방법에 대하여 논의하였다. 미리 분배된 난수를 사용함으로써 RNG가 필요 없어졌을 뿐만이 아니라 시간의 낭비와 재전송을 최소화하여 기존의 방법인 이진트리탐색 기법에 비하여 대단히 빠른 성능을 보여줄 수 있게 되었고, 태그의 수가 대폭 늘어나도 예측 가능한 시간에서 안정적인 처리를 보장할 수 있게 되었다.

참 고 문 헌

[1] Klaus Finkenzeller, “*RFID HANDBOOK*”, 2nd Edition in Korea, 이근호 외 3명 공역, 2004, ISBN : 89-314-2769-7

[2] Ching Law, Kayi Lee, Kai-Yeung Siu, “Efficient Memotyless Protocol for Tag Identification”, *MIT*, MIT-AUTOID-TR-003

[3] Tom Ahlkvist Scharfeld, “An Analysis of the Fundamental Constraints on Low Cost Passive Radio-Frequency Identification System Design”, *MIT*, pp. 92-100, 2001

[4] Stephen Auguest Weis, “Security and Privacy in Radio-Frequency Identification Devices”, *MIT*, pp. 24-25, 51-54, May 2003

[5] “125kHz microID Passive RFID Device with Anti-Collision”, *Microchip Technology Inc.*, MCRF250, 2003

[6] Roy Want, Daniel M. Russell, “Ubiqitous Electronic Tagging”, *IEEE* Distributed Systems Online, Volume 1, Number 2

[7] “Radio Frequency Identification - RFID A basic primer”, *AIM Inc.*, White Paper, Sept 1999

[8] “I·CODE1 System Design Guide”, Philips Semiconductors, Revision 1.1 public, Application Note, April 2002

[9] Pete Sorrells, “Passive RFID Basics”, Microchip Technology Inc., AN680, 1998

강 전 일(Jeon il Kang) 준회원

2003년 2월 인하대학교 컴퓨터 공학과 졸업

2004년 3월~현재 인하대학교 정보통신대학원 석사과정

<관심분야> RFID 보안

박 주 성(Ju sung Park) 준회원

2004년 2월 인하대학교 컴퓨터 공학과 졸업
2004년 3월~현재 인하대학교 정보통신대학원 석사과정
<관심분야> RFID 보안

양 대 헌(Dae hun Nyang) 정회원

1994년 2월 한국과학기술원 과학기술 대학 전기 및 전자 공학과 졸업
1996년 2월 연세대학교 컴퓨터 과학과 석사
2000년 8월 연세대학교 컴퓨터 과학과 박사
2000년 9월~2003년 2월 한국전자통신연구원 정보보호연구본부 선임연구원
2003년 2월~현재 인하대학교 정보통신대학원 조교수
<관심분야> 암호이론, 암호프로토콜, 인증프로토콜, 무선 인터넷 보안