# RFID 시스템에서 충돌 트리 기반 충돌방지 알고리즘

## (Collision Tree Based Anti-collision Algorithm in RFID System)

서 현 곤 †

(Hyun-Gon Seo)

**요 약** RFID는 RF신호를 이용하여 물체를 식별하는 가장 유망한 미래의 비접촉 기술이다. RFID 리더의 식별영역에 여러 개의 태그가 있는 경우, 리더의 질의에 대하여 모든 태그들이 동시에 응답을 하기 때문에 충돌이 발생되어 태그를 식별할 수 없게 된다. RFID에서 다중 태그 식별문제는 아주 중요한 핵심 기술로 이것을 해결하기 위해 슬롯기반 알로하 알고리즘, 트리 기반 알고리즘 등과 같은 충돌 방지 알고리즘이 제안되었다. 본 논문에서는 RFID 시스템에서 충돌 트리를 이용한 충돌트리 기반 충돌 방지알고리즘을 제안한다. 제안하는 방법은 효과적인 충돌 방지 메커니즘을 제공하며 메모리래스 알고리즘이다. 제안하는 충돌트리는 다중 태그 식별문제를 해결하기 위한 메커니즘으로 리더와 태그사이 질의와 응답과정에서 만들어 진다. 리더가 k 비트로 구성된 프리픽스를 질의하면, 태그는 자신의 식별자와 프리픽스를 비교하여 일치할 경우 식별자의 K+1 비트에서 마지막 비트까지 리더에게 전송한다. 시뮬레이션 결과에 따라 제안하는 충돌 트리 기반 충돌 방지 알고리즘이 기존의 트리 워킹 알고리즘이나 쿼리 트리 알고리즘보다 좋은 성능을 보임을 알 수 있다.

**키워드** : RFID, 충돌방지 알고리즘, 트리기반 알고리즘, 충돌 트리

***Abstract*** RFID (Radio Frequency Identification) is one of the most promising air interface technologies in the future for object identification using radio wave. If there are multiple tags within the range of the RFID tag reader, all tags send their tag identifications to the reader at the same time in response to the reader's query. This causes collisions on the reader and no tag is identified. A multi-tag identification problem is a core issue in the RFID. It can be solved by anti-collision algorithm such as slot based ALHOA algorithms and tree based algorithms. This paper, proposes a collision tree based anti-collision algorithm using collision tree in RFID system. It is a memory-less algorithm and is an efficient RFID anti-collision mechanism. The collision tree is a mechanism that can solve multi-tag identification problem. It is created in the process of querying and responding between the reader and tags. If the reader broadcasts K bits of prefix to multiple tags, all tags with the identifications matching the prefix transmit the reader the identifications consisted of k+1 bit to last. According to the simulation result, a proposed collision tree based anti-collision algorithm shows a better performance compared to tree working algorithm and query tree algorithm.

**Key words** : RFID, Anti-collision algorithm, tree based algorithm, collision tree

## 1. 서 론

RFID(Radio Frequency Identification)은 유비쿼터스 컴퓨팅 환경의 초기 기반 기술로 물체에 전자태그를 부착하여 RF신호로 물체의 태그 정보를 읽어와 물체를 식별하는 비 접촉 인식기술이다. 태그 생산 가격이 저렴해지고 표준화됨에 따라 RFID시스템의 도입 시기는 한층 빨라지고 있다. RFID 기술은 기존의 바코드를 대체하여 상품관리를 네트워크화 및 지능화함으로써 유통 및 물류관리뿐만 아니라 보안, 안전, 소방/방재, 환경관리, 생산자동화 등 다양한 분야에 적용될 수 있다[1].

RFID시스템은 무선태그의 정보를 읽을 수 있는 리더와 사물 또는 물체의 정보를 저장하고 있는 무선 태그로 구성된다. 무선 태그는 전원의 유무에 따라 수동식(passive)태그, 능동식(active)태그로 구분되는데, 능동식 태그는 자체에 전원이 내장되어 있으나 passive 태그는 리더기에서 전송된 반송파를 이용하여 생산된 전원을

† 정 회 원 : 한라대학교 정보통신공학부 교수
hgseo@halla.ac.kr
논문접수 : 2006년 9월 20일
심사완료 : 2007년 7월 16일

이용한다. 또한 RFID는 사용하는 주파수에 따라 저주파(125kHz, 135kHz), 고주파(13.56MHz), 극초단파(433.92 MHz, 860~960MHz) 등으로 구분한다. 하지만 RFID기술의 확산을 위해서는 해결해야 할 문제점들이 있다. 즉, 태그의 저가격, 저전력, 초소형화 문제와 사용자 프라이버시 및 정보 보호를 위한 방법, 리더기 영역 내의 다중 태그들의 충돌문제(anti-collision problem) 등을 해결해야 한다. 특히, 충돌문제는 RFID 시스템에서 가장 중요한 핵심 기술이다.

충돌문제를 해결하기 위해 다양한 충돌방지 알고리즘이 제안되었다. 이들 알고리즘은 크게 트리기반 알고리즘과 알로하 기반 알고리즘으로 구분할 수 있다. 트리 기반 알고리즘의 경우 트리 운행기법으로 리더 영역 내에 있는 모든 태그를 식별할 수 있기 때문에 결정적(deterministic) 충돌 방지 알고리즘이라 할 수 있다. 트리 워킹(tree-walking) 알고리즘[2], Bit-arbitration 알고리즘[3], 메모리래스(memoryless) 알고리즘[4] 등이 대표적인 트리 기반 충돌방지 알고리즘이다. 반면에 알로하 알고리즘은 태그들이 리더에게 응답하는 시간을 달리하여 태그들을 식별하는 것으로 수학적 기초에 의한 확률적(probabilistic) 충돌 방지 알고리즘이라 할 수 이다. Framed Slotted Alhoha[5], I-Code 알고리즘[7] 등이 이에 속하는 대표적인 알고리즘이다.

본 논문에서는 다중 태그들의 충돌 문제를 해결하기 위한 충돌 트리 기반 충돌 방지 알고리즘(CT : Collision Tree based Anti-collision algorithm)을 제안한다. 제안하는 충돌 트리 기반 충돌 방지 알고리즘은 쿼리 트리에 기반 한 메모리래스 알고리즘으로 리더기가 태그들에게 응답을 받으면 비트 단위로 충돌한 비트위치를 찾아내어 새로운 프리픽스를 생성하여 다시 모든 태그들에게 전달함으로 충돌을 해결한다. 충돌 트리 기반 충돌 방지 알고리즘은 기존의 트리워킹 알고리즘이나 쿼리 트리 알고리즘 보다 작은 양의 데이타 전송으로 태그를 식별할 수 있고, 질의/응답 횟수도 다른 알고리즘들 보다 작기 때문에 빠른 시간 내에 모든 태그를 식별할 수 있다. 제안하는 알고리즘의 성능 평가를 위해 C++로 시뮬레이션 프로그램을 작성하였다. 여기서 태그는 128bit이고, 태그 수는 $2^2$에서 $2^{11}$까지 증가시키면서 5번 반복 실험한 결과의 평균값을 가지고 성능을 비교하였다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구를 소개하고 3장에서는 본 논문에서 제안하는 충돌 트리에 대하여 설명한다. 그리고 4장에서는 3장에서 제안된 충돌 트리를 이용하여 본 논문에서 제안하는 충돌 트리 기반 충돌 방지 알고리즘을 설명한다. 5장에서 기존 알고리즘과 본 논문에서 제안하는 방법의 성능을 비교하고 6장에서 결론을 내린다.

## 2. 관련 연구

다중 식별태그 문제는 충돌방지 알고리즘을 통하여 해결할 수 있다. 현재까지 다양한 충돌방지 알고리즘들이 제안되었는데 크게 트리 기반 알고리즘(tree based algorithm)과 슬롯 알로하 기반 알고리즘(slotted aloha)으로 구분된다. 슬롯 알로하 기반 알고리즘은 확률적 알고리즘으로 슬롯간의 시간차를 이용하여 태그 충돌을 해결한다[6]. ISO/IEC 18000-6 Type A는 슬로 알로하 기반 표준 충돌방지 기법으로 슬롯 기반 알로하는 프레임안의 슬롯의 개수에 따라 BFSA(Basic Framed Slotted Aloha), DFSA(Dynamic FSA), AFSA(Advanced FSA)로 구분한다. 슬롯 기반 알로하 알고리즘의 한 라운드(round)의 CAS(Collision Arbitration Sequence)는 다음과 같다. 먼저 리더가 태그에게 사용가능한 슬롯의 범위를 지정하여 tag_id 전송을 요구하며, 태그는 리더로부터 받은 슬롯의 범위정보를 이용하여 자신이 전송할 슬롯을 결정하고 자신의 tag_id를 전송한다. 이때 두 개 이상의 태그가 같은 슬롯 번호에 tag_id를 전송하면 충돌이 발생한다. 그림 1은 슬롯 알로하 기반

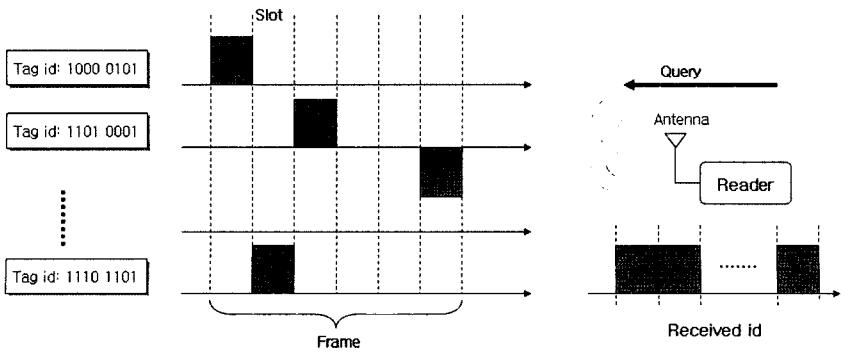


그림 1 슬롯 알로하 기반 알고리즘

알고리즘의 동작을 나타낸 것이다.

BFSA의 경우 프레임 사이즈가 고정되어 있기 때문에 읽어야 할 태그가 많은 경우 잦은 충돌이 발생하여 좋은 성능을 발휘하지 못한다. I-CODE는 BFSA를 기반으로 한 것으로 한 라운드가 끝난 후 충돌비율에 따라 다음 라운드의 프레임 사이즈를 변경하여 다음 라운드를 실행한다. I-CODE의 CAS는 다음과 같다.

*1) 리더 브로드캐스트 : broadcast(I, rnd, N)*
*2) 모든 태그 $sT(data_{T,I})$ ==> 리더*

먼저 리더는 $I$(태그 식별자 범위)와 rnd(랜덤 생성을 위한 시드 값), N∈{1, 4, 8, 16, 32, 64, 128, 256} 프레임 사이즈를 자신의 영역 내의 태그들에게 전달하면, 모든 태그 T는 슬롯 번호 s(0≤s<N)를 생성하여 자신의 tag_id를 리더에게 전달한다. 한 라운드가 끝나면 다음 3가지의 경우 즉, 하나의 슬롯에 하나의 태그만 응답하는 경우, 아무런 응답이 없는 빈 슬롯, 두 개 이상의 태그가 응답하여 충돌이 발생하는 경우로 구분된다. 충돌이 발생하면 리더는 다시 두 번째 라운드를 시작하여 태그를 식별한다. AFSA는 인식범위 내의 태그 수를 추정하여 내부의 프레임 크기를 정하는 방식이다. TEM (Tag Estimation Method)[8-9]에서는 하나의 슬롯에 충돌된 태그 수($C_{tags}$)와 충돌 비율($C_{rate}$: collision rate)의 관계를 $C_{tags} = \frac{1}{C_{rate}} = 2.3922$ 로 정의한 후 한 라운드 후 프레임 내에서 충돌이 발생한 슬롯의 수($M_{cl}$)에 대하여 추정 태그 수 ($n_{est}$) $n_{est} = \lfloor 2.3922 * M_{cl} \rfloor$ 로 정의하여 프레임 크기를 가변 할 수 있도록 제안하였다.

모든 FSA의 성능은 프레임 사이즈 N를 어떻게 결정하는가에 따라 많은 영향을 받게 된다. 따라서 프레임 사이즈 결정이 아주 중요한 성능 평가요소가 된다. 프레임 사이즈 N값을 크게 하면 타임 슬롯의 낭비를 초래하고 너무 작게 하면 과도한 충돌이 발생하기 때문이다. 또한 FSA의 경우 언제 식별을 종료할 것인가를 결정하는 것이 어렵다. 종료하는 시점에 따라 인식률이 많은 영향을 받기 때문이다.

I-CODE에서는 $Q^s q(0)[n] \geq \alpha$ 종료 조건을 정의 하였다[4]. (s는 read cycle, n은 영역 내의 추정 태그 수, Q는 변환 행렬) 무엇보다도 초소형, 저전력, 저가격의 수동형 태그를 생산하기 위해서는 태그에서 수행하는 처리 기능을 감소시켜야 한다. 하지만 슬롯 알로하 기반 충돌 방지 알고리즘의 리더 경우 태그에 전달하는 명령어가 복잡하여 태그에서 처리해야 할 기능이 많은 단점이 있다. 즉, 태그 하드웨어 및 소프트웨어의 복잡도가 증가하여 수동형 태그의 충돌 방지 알고리즘으로 사용하기는 무거운 면이 있다. 따라서 본 논문에서는 트리 기반 충돌 방지 알고리즘에 연구의 초점을 두었다.

트리 기반 알고리즘은 결정적(deterministic) 알고리즘으로 이진 트리를 순회하며 모든 태그를 식별한다. 트리 기반 알고리즘은 메모리형(memory) 알고리즘과 메모리래스형(memoryless) 알고리즘으로 구분하는데, 태그에 대한 질의만으로 태그를 구분할 경우 이를 메모리래스형 알고리즘이라 하고, 리더 질의에 대해 태그마다 상태 정보를 저장하고 관리에 의하여 태그를 식별할 경우 이를 메모리형 알고리즘이라 한다. 메모리형 알고리즘 역시 태그의 저가격, 저전력, 초소형 면에서 태그 하드웨어가 복잡하고 자료처리기능을 요구하기 때문에 수동형 태그의 충돌 방지 알고리즘으로 사용하기엔 무거운 면이 있다. 따라서 본 논문에서는 트리기반 메모리래스형 충돌 알고리즘에 초점을 맞추었다. 본 논문에서 제안하는 충돌 트리 알고리즘은 태그들의 식별을 위해 반복적인 질의, 응답과정을 통하여 이루어지기 때문에 태그에는 인식을 위한 별도의 기억 장치가 필요 없어 태그의 저가격, 저 전력, 초소형화를 할 수 있다.

대표적인 트리기반 메모리래스 충돌 방지 알고리즘으로 트리 워킹 알고리즘, 쿼리 트리 알고리즘, 개선된 쿼리 트리 알고리즘 또는 충돌 추적 알고리즘 등이 있다. 트리기반 메모리래스 충돌 알고리즘의 태그들은 리더에서 전달된 프리픽스와 자신의 tag_id를 비교할 수 있는 매칭(matching) 하드웨어 회로가 필요하다. 프리픽스와 자신의 tag_id가 일치하는 경우에만 태그는 리더에게 자신의 tag_id를 전달함으로 각각의 태그를 식별한다. 다음 절은 기존의 각 알고리즘에 대한 설명이다.

### 2.1 트리 워킹 알고리즘

트리 워킹 알고리즘은 대표적인 메모리래스 알고리즘으로 bit-by-bit 쿼리 방식으로 이진트리의 깊이 우선 탐색방법으로 태그를 식별한다[2]. 리더가 $d$비트 길이의 프리픽스 B(B=$b_1b_2$, ..., $b_d$)를 태그에게 질의하면, 태그들은 자신의 식별자와 프리픽스를 비교하여 그 값이 동일하면 태그는 $d+1$비트를 리더에게 전송한다. 모든 태그가 "0"또는 "1"을 전송했을 경우 충돌이 없기 때문에 리더는 스택에서 새로운 프리픽스 $B$를 pop()하여 다시 모든 태그에게 전송한다. 하지만 "0"과 "1"이 동시에 리더기에 수신된 경우 충돌이 발생하기 때문에 리더는 B ‖ 0 와 B ‖ 1을 프리픽스로 하여 다시 질의를 반복한다.

트리 워킹 알고리즘의 경우 비트단위로 태그를 검출하기 때문에 리더기 영역 내에 태그가 하나만 있는 경우도 마지막 비트까지 질의-응답을 해야 하고, 영역내의 많은 태그가 있을 경우 자주 충돌이 발생하기 때문에 많은 시간과 성능저하의 문제점을 가지고 있다. 트리 워킹 알고리즘의 시간 복잡도는 $O(nk)$이다. 여기서 n은

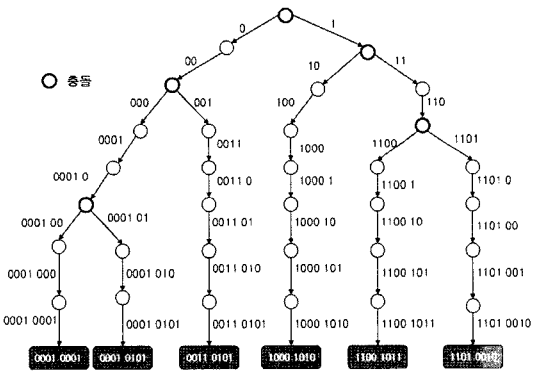


그림 2 트리워킹에서 8비트로 구성된 태그 식별 과정

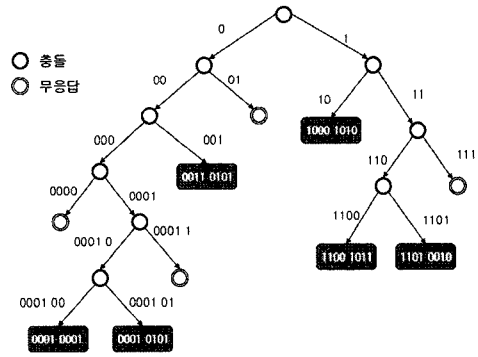


그림 3 쿼리 트리에서 8비트로 구성된 태그 6개를 식별 과정

태그의 개수이고, k는 tag_id의 비트수이다.

그림 2는 트리 워킹 알고리즘에서 8비트로 구성된 태그 6개(0001 0001, 0001 0101, 0011 0101, 1000 1010, 1100 1011, 1101 0010)를 식별하는 과정을 나타낸 것이다. 모든 태그를 식별하는데 총 5번의 충돌이 발생하고, 질의/응답 횟수는 37회가 된다. 트리의 깊이는 태그의 비트와 동일하게 8이며, 에지의 숫자는 프리픽스를 의미한다.

### 2.2 쿼리 트리 알고리즘

쿼리 트리(QT: Query Tree) 알고리즘은 트리 워킹보다 향상된 방법이다. 리더가 $d$비트 길이의 프리픽스 $B(B=b_1b_2, ..., b_d)$를 태그에게 질의하면, 태그들은 자신의 식별자와 프리픽스를 비교하여 그 값이 동일하면 태그는 $d+1$비트에서 마지막 $n$비트까지 리더에게 전송한다[3]. 리더가 한 태그에게서만 응답을 받을 경우 $(b_{d+1}b_{d+2}, ..., b_n)$비트열의 태그가 식별된 것이고, 다수의 태그가 동시에 응답했을 경우 충돌이 발생되므로 리더는 프리픽스 B에 $B \| 0$ 와 $B \| 1$을 만들어 스택에 저장한다. 그리고 새로운 프리픽스를 스택에서 가져와 스택이 빌 때까지 질의를 반복한다. 쿼리 트리의 시간 복잡도는 $O(n*(k+2-\log n))$이다. 하지만 쿼리 트리의 경우 리더의 질의에 대해 무응답인 경우가 발생하고 많은 충돌이 발생하기 때문에 이를 처리하기 위한 부담이 증가한다.

그림 3은 그림 2와 같이 8비트로 구성된 태그 6개를 쿼리 트리 알고리즘을 이용하여 식별하는 과정을 나타낸 것이다. 모든 태그를 식별하는데 총 9번의 충돌이 발생하였고, 모두 18회 질의/응답으로 태그들을 식별한다. 하지만 쿼리 트리 알고리즘의 경우 태그에서 아무런 응답에 없는 경우가 존재한다. 즉, 리더가 프리픽스를 전송하고 난 후 일정시간을 대기 하는 동안 응답이 없는 것으로 해당 프리픽스로 시작하는 태그가 없음을 의미한다. 그림 3에서는 4번의 무응답이 발생한다. 하지만 트리의 깊이는 6으로 트리 워킹보다 작음을 알 수 있다.

### 2.3 개선된 쿼리 트리 알고리즘

최근에 제안된 충돌 추적(collision tracking) 트리 알고리즘[10]과 개선된 쿼리 트리(IQT : Improved Query Tree) 알고리즘[1]은 그 기법이 동일한 것으로, 충돌 위치 비트를 감지하여 태그를 식별하는 기법이다. 리더가 $d$비트 길이의 프리픽스 $B(B=b_1b_2, ..., b_d)$를 모든 태그에게 질의하면, 태그들은 자신의 식별자와 프리픽스의 비교를 통해 매칭이 이루어질 경우 태그 식별자의 $d+1$ 비트에서 마지막 $n$비트까지를 리더기에게 순서적으로 전송한다.

리더는 태그가 전송한 식별자 ID를 수신함과 동시에 수신된 비트의 충돌 발생여부를 검사한다. 이때 "0"과 "1"이 동시에 수신된 경우 충돌이 발생했기 때문에 태그들에게 나머지 비트의 송신을 중지하도록 "전송중지 명령"을 전달한다. 태그가 전송중지 명령을 받으면 즉시 나머지 비트의 전송을 중지하고 새로운 프리픽스 $B'$를 받을 준비를 한다. 리더는 충돌이 발생한 $k$비트 이전까지 비트를 프리픽스 B에 추가하여 새로운 프리픽스 $B'(B'=b_1b_2, ..., b_{d-1}b_d\bar{b}_{d+1}, ..., b_{k-1})$를 생성하여 다시 모든 태그들에게 전송한다. 그림 4는 리더와 태그에서 실행되는 충돌 추적 트리 알고리즘이다.

하지만 개선된 쿼리 트리(IQT) 알고리즘의 경우 해결해야 할 몇 가지 문제점 들이 있다. 충돌이 발생하여 리더가 전송 중지 명령을 태그들에게 전달하면 태그는 이를 식별할 수 있는 별도의 기능이 추가 되어야 한다. 즉, 프리픽스 B를 받고 자신의 tag_id와 매칭 되는 경우 태그는 한 비트씩 차례로 자신의 tag_id를 전달하는 도중에 전송중지 명령을 받기 때문에 이를 감지하고 처리할 수 있는 하드웨어와 소프트웨어가 태그에 추가적으로 포함되어야 한다. 이는 태그의 생산 비용을 증가시킬 뿐만 아니라 초 경량화 및 저전력, 저가격에 기반을 둔 태그의 생산에 많은 어려움을 주게 된다. 따라서 본 논문에서는 기존의 메모리래스 트리 기반 충돌방지

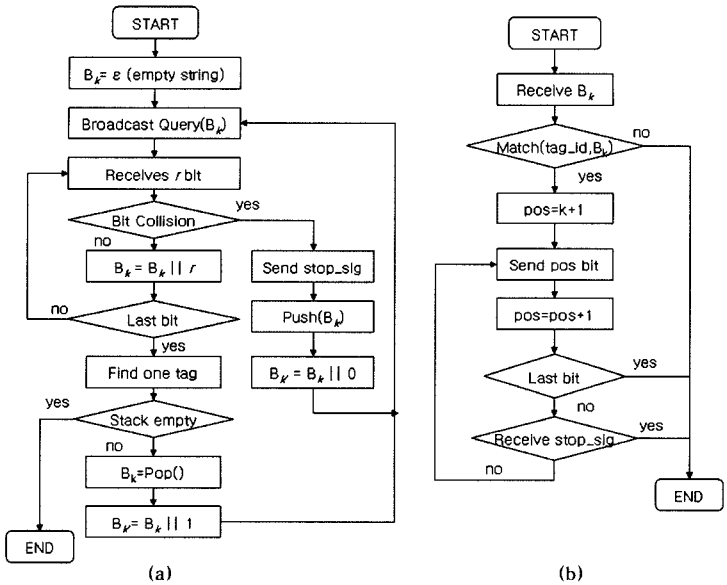


그림 4 충돌 추적 트리 알고리즘, (a) 리더 알고리즘 (b) 태그 알고리즘

알고리즘들의 문제점을 해결하기 위한 충돌 트리 기반 알고리즘을 제안한다.

## 3. 충돌 트리

충돌 트리(collision tree)란 $n$비트로 구성된 두 개 이상의 비트열에 대하여 일치하지 않는 비트를 기준으로 트리를 분리하는 기법으로 비트열을 이진트리에 표현한 트리를 의미한다. 즉, $n$비트로 구성된 두 개 이상의 비트열에서 각 비트의 값이 일치하지 않는 최초 $k$비트를 기준으로 일치하는 비트열($b_0$, $b_1$, $b2$, ..., $b_{k-1}$)을 루트(root)로 하고 충돌 발생한 충돌 비트($b_k$)를 루트노드의 자식(충돌 노드)으로 하며, 충돌 비트 이하의 비트열($b_{k+1}$, $b_{k+2}$, ..., $b_n$)을 충돌노드의 오른쪽 노드, 왼쪽 노드로 취하는 이진트리이다. 충돌 노드의 자식 노드들은 복수개의 비트열을 포함하는 노드와 단 하나의 비트열을 포함하는 노드가 있다. 전자의 경우 복수개의 비트열에 대하여 다시 최초 충돌 비트를 찾아 서브트리의 루트 노드와 충돌 노드 그리고 자식 노드로 분리한다. 후자의 경우 하나의 비트열만 있는 경우로 더 이상의 분리는 일어나지 않는다. 충돌 트리에서 에지의 가중치는 충돌 노드의 자식에만 부여하는데 충돌 노드의 왼쪽 노드로 향하는 에지의 가중치는 0, 오른쪽은 1로 한다.

충돌 트리에서 하나의 노드 안에 두 개 이상의 비트열이 있는 경우 각 노드에 하나의 비트열을 가질 때까지 트리를 분리한다. 즉, 최초 비트열이 2개 이상인 경우 최초 충돌비트의 자식노드들 중에 2개 이상의 비트열을 포함하는 경우가 있다. 이 경우 해당 노드에 속해 있는 비트열에 대하여 다시 각 비트의 값이 일치하지 않는 비트를 찾아 분리하여 각 노드에 하나의 비트열만 가지도록 충돌 트리를 재귀적으로 분리한다. 트리의 분리가 끝나면 단말노드를 제외한 내부 노드들에 대하여 자식노드가 하나인 노드 s를 모두 찾아 노드 s를 삭제한다. 노드 s를 삭제하면서 노드 s의 비트열 값을 s의 자식노드들의 에지 가중치 값에 연결(concatenation)하여 새로운 에지 가중치를 생성한다. 즉, s의 비트열 값이 '01001'이고 s의 자식노드의 에지 가중치 값이 '0'이였다면 새로운 에지 가중치는 '01001'+'0'이 되어 '010010'가 된다.

그림 5는 8비트로 구성된 4개 비트열 n1(0010 0011), n2(0100 1010), n3(0001 0011), n4(001 0111)를 충돌트리로 생성하는 과정을 나타낸 예제이다. 4개의 비트열에서 최초 2번째 비트가 일치하지 않기 때문에 그림 5(a)와 같이 두 번째 비트를 중심으로 상위 첫 번째 비트 '0'을 루트노드로 하고, 두 번째 충돌 비트를 중심으로 왼쪽에는 n1(10 0011), n3(01 0011), n4(01 0111)를 가지는 노드, 오른쪽에는 n2(00 1010)을 가지는 트리로 분리된다. 하지만 그림 5(a)의 왼쪽노드(sub1)는 복수개의 비트열을 가지고 있기 때문에 다시 이들 사이의 최초 충돌 비트를 찾아 분리한다. 그림 5(a)의 왼쪽노드의 n1, n3, n4는 첫 번째 비트에서 충돌이 발생했기 때문에 이를 다시 분리하면 그림 5(b)와 같이 오른쪽 노드에 n1(0 0011), 왼쪽에는 n3(1 0011), n4(1 0111) 두 개의 비트열을 포함하는 sub2가 된다. 역시 그림 5(b)의 sub2는 분리를 해야 하는데 n3, n4의 3번째 비트에서

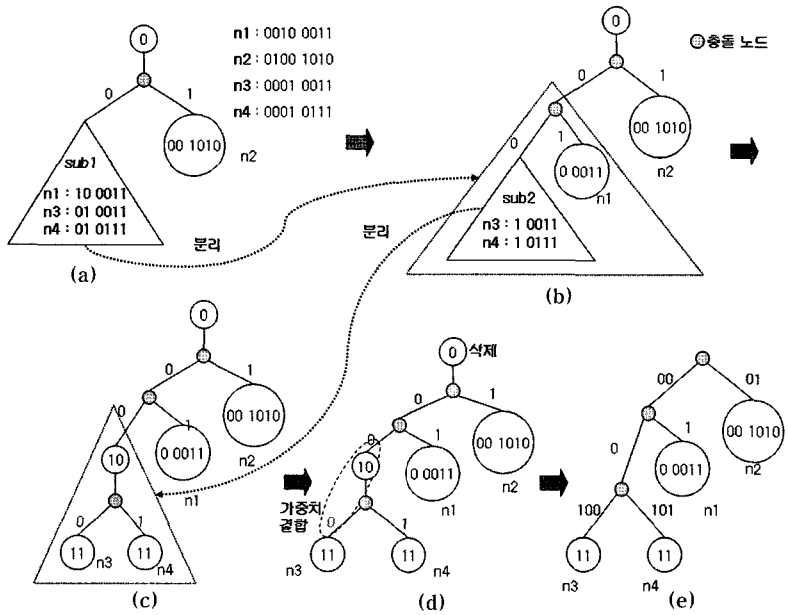


그림 5 충돌트리 생성 예제

충돌이 발생하기 때문에 sub2는 루트노드 '10'과 충돌 노드의 왼쪽에 '11', 오른쪽에 '11' 서브비트열이 저장된 이진트리로 분리된다. 그림 5(c)는 그림 5(b)의 sub2를 분리하였을 때 상태이다. 그림 5(c)에서 모든 단말노드들은 하나의 비트열을 가지고 있으므로 더 이상의 분리는 일어나지 않는다. 그림 5(d)는 단말 노드를 제외한 내부노드들 중에서 자식 노드가 하나인 노드를 모두 찾아 삭제하는데, 삭제 노드의 비트열 값은 자식노드들의 에지 가중치에 연결한다. 그림 5(e)는 8비트로 구성된 4개 비트열 n1, n2, n3, n4의 완성된 충돌 트리이다.

충돌 트리에서 루트 노드의 차수는 항상 1이고 충돌 노드의 차수는 2임으로 충돌 트리는 전이진트리(complete binary tree)이다. 따라서 이진트리의 특징에 따라 충돌 트리는 $n_0$(단말노드의 수), $n_1$(차수가 1인 노드의 수), $n_2$(차수가 2인 노드의 수), n(총 노드의 수), e(에지의 수)에 대하여 다음 수식이 성립함을 쉽게 알 수 있다.

$$n = e + 1 \quad (1)$$

$$e = n_1 + 2n_2 \quad (2)$$

$$n = n_0 + n_1 + n_2 \quad (3)$$

$$n_0 = n_2 + 1 \quad (4)$$

충돌 트리의 단말 노드의 수는 식 (4)에 의하여 충돌 노드의 수에 +1한 것과 같음을 알 수 있다. 즉, 충돌 트리의 단말노드가 $k$개이면 충돌 노드는 $k-1$개이고, 전체 $n$비트 중 $k$개 비트에서 충돌이 발생했음을 알 수 있다. 그림 5에서와 같이 충돌 노드가 3개이면 단말 노드의 수는 4임을 알 수 있다.

$c$비트로 구성된 $n$개 노드를 충돌 트리에 노드를 삽입, 탐색, 제거하는데 최악의 경우 시간 복잡도는 $O(n)$이다. 충돌 트리는 전이진 트리이기 때문에 트리의 평균 깊이는 $O(log_2\ n)$이지만 사향트리가 될 경우 트리의 깊이는 n-1이 될 수 있기 때문에 최악의 경우 $O(n)$이 된다.

충돌 트리에 표현된 비트열에서 원래 비트열 ω를 읽는 방법은 충돌 트리에 대하여 각 노드를 프리오더(preorder)순으로 방문하면서 에지의 가중치 값을 비트열 ω에 추가하면 된다. 단말노드인 경우 노드에 저장된 비트열 값을 비트열 ω에 추가함으로 충돌 트리에 표현된 비트열을 읽을 수 있다. 그림 6은 그림 5(e)의 충돌 트리를 프리오더 순으로 방문했을 때 4개 문자열 n1(00100011), n2(01001010), n3(00010011), n4(0010111)를 나타낸 그림이다.

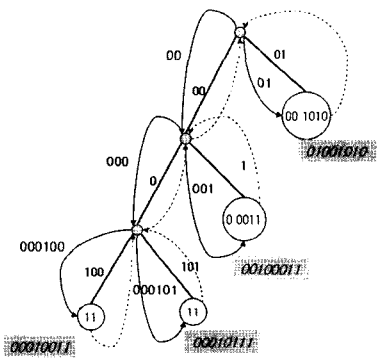


그림 6 문자열 추출을 위한 충돌 트리 순회

## 4. 충돌 트리 기반 알고리즘

본 절에서는 새로운 충돌감지 알고리즘인 충돌 트리 기반 충돌 방지 알고리즘(CT: Collision Tree based anti-collision algorithm)을 제안한다. 충돌 알고리즘은 트리 워킹 알고리즘, 쿼리 트리 알고리즘과 같이 충돌 회피 기법이면서, 개선된 쿼리 트리(improved query tree) 알고리즘과 같이 충돌이 발생한 비트의 위치를 리더가 검출하는 충돌 검출기법이다. 즉, 충돌 알고리즘은 충돌 회피 기법과 충돌 추적 기법을 이용하여 리더 영역내의 태그들을 식별하는 알고리즘이다.

### 4.1 충돌 트리 알고리즘

충돌 트리 알고리즘은 쿼리 트리 알고리즘과 비슷하다. 리더가 $k$비트 길이의 프리픽스 $B_k(B=b_1b_2, ..., b_k)$를 모든 태그에게 질의하면, 태그들은 자신의 tag_id와 프리픽스를 비교하여 그 값이 동일하면 $k+1$비트에서 마지막 $n$비트까지 리더기에게 전송한다. 리더가 한 태그에게서만 응답을 받을 경우 쿼리 트리와 같이 하나의 태그가 식별된다. 그런데 다수의 태그가 동시에 응답했을 경우 충돌이 발생한다. 이 경우 리더는 응답 받은 비트열 $b_{k+1}, b_{k+2}, ..., b_n$사이에서 최초 충돌이 발생한 비트 위치 $c$비트를 찾고(단, $k<c\leq n$), 새로운 프리픽스 $B(=b_1b_2, ..., b_k, b_{k+1}, ..., b_{c-1}) \| 0$ 와 $B(=b_1b_2, ..., b_k, b_{k+1}, ..., b_{c-1}) \| 1$을 생성한다. 이것은 충돌 추적 트리 알고리즘과 개량된 쿼리 트리(improved query tree) 알고리즘에서의 전송중지 명령을 전달하는 방법과는 다르다. 충돌 트리 알고리즘에서는 별도의 추가적인 명령이 필요 없고, 또한 태그에 추가적인 하드웨어나 소프트웨어 없이 리더에서 모든 과정을 수행한다. 그렇기 때문에 충돌 추적 알고리즘과 같이 추가적인 명령과 태그의 하드웨어적인 기능 추가 없이 태그들을 식별할 수 있다.

그림 7에서 리더가 16비트 프리픽스 $B$(1101 0011 1010 1000)를 영역내의 태그들에게 전달하면 tag1을 제외하고 tag2, tag3, tag4가 프리픽스와 일치함으로 tag1를 제외한 나머지 태그들은 프리픽스와 일치하는 비트열 이후(17번 비트)부터 마지막 비트까지 리더에게 전달한다. 리더가 각 태그들에게 응답비트를 받으면 이들 사이에 최초 충돌하는 비트열을 검색한다. 그림 7에서는 7비트에서 충돌이 발생했기 때문에 리더는 응답 비트열의 첫 번째 비트에서 6bit까지의 비트열 0011 01을 기존 프리픽스 $B$에 추가하여 새로운 프리픽스 $B'$(1101 0011 1010 1000 0011 01)를 생성 한다. 다시 리더는 새롭게 생성된 프리픽스 $B'$를 $B' \| 0$ 하여 영역내의 태그들에게 전달하여 위의 과정을 반복하면서 각 태그들을 식별한다.

그림 8은 본 논문에서 제안하는 CT의 리더기와 태그에서 각각 수행되는 알고리즘이다.

그림 9는 그림 2와 같이 8비트로 구성된 태그 6개를 CT알고리즘을 이용하여 식별하는 과정을 나타낸 것이다. 모든 태그를 식별하는데 총 5번의 충돌이 발생하였고, 모두 10회 질의/응답을 한다. 트리의 깊이는 3이고 무응답 경우는 없으며 다른 알고리즘들에 비하여 간단하게 트리가 구성됨을 알 수 있다.

### 4.2 알고리즘 특징 비교

표 1은 기존에 제안된 트리기반 알고리즘과 본 논문에서 제안하는 충돌 트리 알고리즘과의 특징을 비교한 것이다. 4가지 알고리즘 모두 리더에서 전송된 프리픽스 $B$값과 자신의 태그 값을 비교하기 위한 매칭회로가 있어야 한다. 쿼리 트리를 제외한 3가지 기법에서는 무응답 태그가 존재하지 않지만 쿼리트리의 경우 무응답 태그가 존재하기 때문에 리더가 일정시간 대기해야 하는 경우가 발생한다. 이는 태그 식별시간에 영향을 준다.

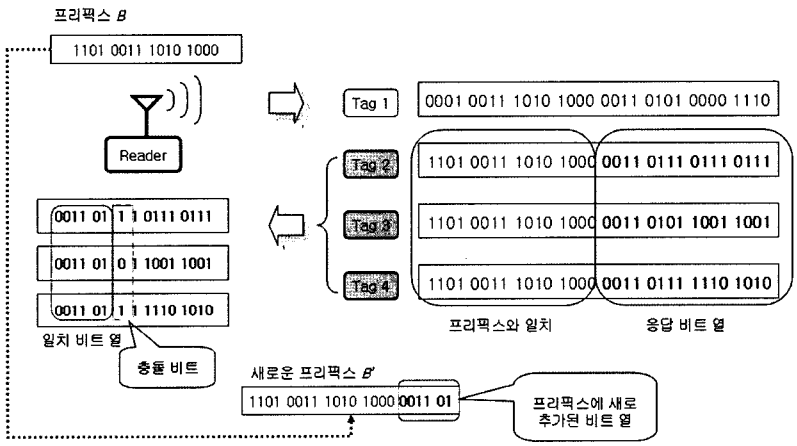


그림 7 CT에 의한 태그 식별 방법

```
Algorithm CT_Reader
{
      k=0;
      Bk=ε;    // init empty string
      do {
            Query(Bk);   // broadcast prefix to Bk all tags
            Reply(tag_idk+1,n );   // receive tag_id
            if (collision detection)
              {
                  Find(collision position c bit);  // 충돌비트 위치 검색
                  Bk = Bk + Bk+1,k+2,.....,c-1;
                     // 충돌이 발생한 c-1 비트까지 비트열을 프리픽스에 추가
                  Push(Bk);
                  Bk= Bk || 0; // 프리픽스에 0추가 }
              else
              {
                     Detection one tag;  // 하나의 태그 확인
                     if(!stack empty)
                     {
                           Bk=Pop();
                           Bk= Bk || 1; // 프리픽스에 1추가 }
              }
      }while(!stack empty)    // stack에 빌 때까지 프리픽스 전달
}

Algorithm CT_Tag
{
      Receive(Bk);
      if (Matching(tag_id,Bk))    // tag_id의 k비트까지 값이 프리픽스와 일치
            Send(tag_idk+1,k+2.....n): // k+1비트에서 마지막 비트까지 전달
}
```

그림 8 CT 알고리즘

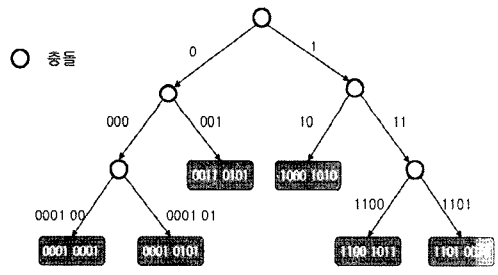


그림 9 CT 알고리즘에서 8비트로 구성된 태그 식별 과정

식별 시간의 경우 트리 워킹 알고리즘에 많은 시간이 소요된다. 왜냐하면 많은 질의/응답과정이 필요하기 때문에 태그 식별시간이 다른 알고리즘에 비해 아주 느리다. 쿼리 트리의 경우 트리워킹 보다 빠르지만 개선된 쿼리 트리와 본 논문에서 제안하는 충돌 트리 알고리즘에 비해서는 속도가 느리다. 개선된 쿼리 트리의 경우 충돌이 발생할 경우 태그에서 이를 처리할 수 있는 추가적인 하드웨어와 소프트웨어가 필요하다. 이는 태그를 생산하기 위한 비용이 증가될 뿐만 아니라 추가적인 명령이 더 필요하기 때문에 태그 식별을 위한 전체적인 오버헤드가 증가 하게 된다. 반면 본 논문에서 제안하는 충돌 트리의 경우 충돌 식별을 리더기에서 실행하기 때문에 태그에 별다른 하드웨어의 추가 없이 쿼리트리보다 더 향상되고 빠른 시간 안에 모든 태그를 식별할 수 있어 더 우수하다고 할 수 있다.

그림 10은 충돌트리와 개선된 쿼리트리의 질의/응답 과정을 나타낸 그림이다. [a]그림은 충돌트리와 개선된 쿼리트리에서 충돌이 발생하지 않을 때의 상태이다. 이 경우 두 알고리즘의 성능은 동일하게 된다. 하지만 충돌이 발생할 경우 충돌 트리는 [a]와 같이 동일한 결과를 나타내지만, 개선된 쿼리트리의 경우 [b]처럼 복잡한 처리 절차를 거쳐야 한다. 여기서 먼저 생각할 것은 충돌 트리의 경우 반이중 방식으로 전송이 가능하지만, 개선된 트리의 경우 전이중 전송이 되어야만 가능하다는 것이다. 왜냐하면 개선된 쿼리 트리의 경우 리더기가 각 태그에게서 보내는 tag_id를 받으면서 충돌을 감지해야 하고, 충돌이 감지될 경우 바로 전송중지 명령을 전달해야 하기 때문에 전이중 방식이 지원되어야 한다. 하지만 충돌트리의 경우 반이중 전송이 가능하기 때문에 이것 역시 본 논문에서 제안하는 충돌트리의 장점이라 할 수 있다.

리더에서 태그로 쿼리를 전달하고 응답을 받을 때까지 시간을 나타내면 다음 수식과 같다.

$$Q_t + M_t + A_t \quad (5)$$

$$Q_t + M_t + A_{ot} \quad (6)$$

여기서

$$A_{ot} = A_k + A_i + S_{pt} \quad (7)$$

$$A_i = C_t + S_t \quad (8)$$

$Q_t$ = *리더에서 태그로 쿼리를 전달하는데 필요한 시간(query time)*

$M_t$ = *tag_id와 프리픽스를 비교하는 시간 (matching time)*

$A_t$ = *태그에서 리더로 나머지 tag_id를 전달하는데 필요한 시간(response time)*

$A_{ot}$ = *충돌이 발생하여 전송중지 명령을 전달할 때까지의 시간(overhead time)*

$A_k$ = *충돌이 발생되기 전까지 전달한 시간(response time before collision)*

$A_i$ = *충돌 발생 후 태그가 전송중지 명령을 받을 때까지 시간(transmit ignore time)*

$S_{pt}$ = *전송중지 명령 처리 시간(stop command*

표 1 트리 기반 충돌회피 알고리즘들의 차이점

| 항 목 / 알고리즘 | Matching 회로 유무 | 무응답 태그 존재 여부 | 태그 생산비용 | 태그 식별시간 | 추가 H/W, S/W | 질의/응답 방식 |
|---|---|---|---|---|---|---|
| Tree Walking | ○ | X | 저가 | 아주느림 | X | bit by bit |
| Query Tree | ○ | ○ | 저가 | 보통 | X | bit by variable bit |
| Improved Query Tree | ○ | X | 고가 | 빠름 | ○ | variable bit by variable bit |
| Collision Tree | ○ | X | 저가 | 아주빠름 | X | variable bit by variable bit |

*processing time)*

$C_t$ = *충돌발생 탐지시간(collision discovery time)*

$S_t$ = *Stop 명령 전달시간*

로 표현 할 수 있다. 식 (5)는 충돌이 일어나지 않은 경우이고 식 (6)은 충돌이 발생된 경우에 시간을 나타낸다. 식 (5)는 쿼리 시간, 비교 시간 및 응답 시간으로 구성되고, 식 (6)은 충돌이 발생했기 때문에 쿼리시간, 비교시간 및 충돌을 해결하기 위한 오버헤드 시간으로 구성된다.

문제는 식 (7)에서 $A_{ot}$시간이다. $A_{ot}$시간에 따라 개선된 쿼리 트리의 성능이 결정되기 때문이다. $A_{ot}$시간에서 $A_t$는 태그가 충돌이 발생한 것을 모르는 상태에서 태그가 전송한 비트이기 때문에 다음 사이클에서 다시 중복해서 보내야하는 부분으로 오버헤드가 된다. 즉, $A_t$의 시간은 $C_t$시간과 $S_t$시간을 더한 것으로 리더기가 충돌을 감시하고 얼마나 빨리 전송 중지명령을 보내느냐에 따라 시간이 결정된다. 하지만 $S_t$시간은 본 논문에서 언급하기 어려운 문제이다. 왜냐하면 $S_t$시간을 처음 명시한 [10]에서 전송중지 명령을 위한 하드웨어와 소프트웨어에 대한 설명이 전혀 없기 때문에 얼마만큼의 시간에 소요되는지 본 논문에서 인위적으로 값을 지정하여 성능을 비교하기란 어려운 문제이기 때문이다. 만약 $S_t + S_{pt} \geq A_t$라면 충돌트리 알고리즘이 더 좋은 성능을 보이고, $S_t + S_{pt} < A_t$라면 개선된 쿼리 트리가 더 좋은 성능을 보일 것으로 예측할 수 있다. 하지만 개선된 쿼리 트리의 경우 추가적인 하드웨어와 소프트웨어를 필요 하고, 태그의 가격이 올라가며, 전이중 전송을 지원해야하기 때문에 충돌트리보다 많은 추가 필용이 발생하는 단점이다. 따라서 본 논문에서는 $S_t$에 대한 정확한 시간을 예측할 수 없기 때문에 이 문제는 차기 연구과제로 남겨둔다.

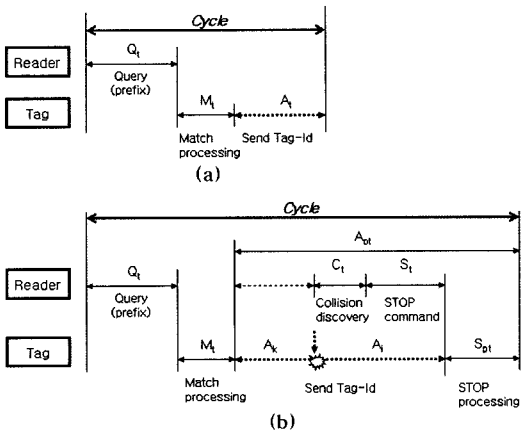


그림 10 충돌트리와 개선된 쿼리 트리의 질의/응답 절차

## 5. 성능평가

본 논문에서 제안하는 충돌 트리 알고리즘의 성능 평가를 위해 C++언어로 시뮬레이션 프로그램을 작성하여 트리 워킹 알고리즘, 쿼리 트리 알고리즘에 대하여 질의/응답 횟수, 충돌 횟수, 전송 비트수를 각각 비교하였다. 개선된 쿼리 트리의 경우 충돌이 발생했을 때 그 위치를 찾아 다음 프리픽스를 생성하는 기본 원리가 본 논문에서 제안하는 충돌 트리 알고리즘과 구조적으로 비슷하고, 충돌 발생 시 전송중지 명령을 위한 추가적인 하드웨어와 소프트웨어가 필요하기 때문에 본 실험에서는 제외하였다. 왜냐하면 충돌 트리 알고리즘의 경우 개선된 쿼리 트리 알고리즘의 태그에서 수행하는 동작을 리더에서 동작함으로 프리픽스를 생성하고 프리픽스를 처리하는 과정이 유사하기 때문이다.

태그 비트의 길이는 EPC Global 96비트로 하였고 태그의 개수는 $2^2 \sim 2^{11}$까지 실험을 하였다. 각 실험 결과는 5회 반복 시험한 결과의 평균값이다. 초당 전송률 및 속도, 태그 특성에 따른 지연 등은 평가 항목에서 제외하였다.

그림 11은 질의/응답 횟수를 비교한 그래프이다. 트리 워킹 알고리즘의 경우 쿼리 트리 알고리즘이아 충돌 트리 알고리즘에 비해 태그 수가 증가 할수록 질의/응답 횟수가 기하급수로 증가함을 알 수 있다. 쿼리 트리의 경우 역시 충돌 트리 알고리즘 보다 질의/응답 횟수가 많음을 알 수 있다. 트리 워킹 알고리즘의 질의/응답 횟수를 100%할 때 쿼리 트리 알고리즘은 트리 워킹에 비해 2.42%, 충돌 트리 알고리즘은 1.69%이다.

그림 12는 그림 11에서 쿼리 트리와 CT의 질의/응답 횟수만 나타낸 그래프이다. 태그의 개수에 상관없이 모든 경우 CT가 기존의 쿼리 트리보다 질의/응답 횟수가 작으며, 약 30%의 성능 향상을 보임을 알 수 있다.

그림 13은 태그 당 평균 질의/응답 횟수를 나타낸 그래프이다. 트리 워킹 알고리즘은 평균 122번, 쿼리 트리는 2.7회, 충돌 트리의 경우 평균 1.9회로 태그 당 평균 질의/응답 횟수에서도 본 논문에서 제안하는 충돌 트리 알고리즘이 우수함을 알 수 있다.

그림 14는 3개의 알고리즘에 대한 충돌 횟수를 나타낸 그래프이다. 트리 워킹 알고리즘과 본 논문에서 제안하는 충돌 횟수가 동일하며 앞장의 수식④에 의하여 항상 전체 태그 수에 -1한 값과 동일하다. 트리 워킹 알고리즘에 충돌이 발생 할 경우 트리가 두 개의 서브트리로 분리되며 서브트리 내에서 다시 충돌이 발생하면 또 다시 분리된다. 하지만 서브 트리 내에서 충돌이 발생되지 않으면 트리의 깊이만 길어질 뿐 더 이상의 분리는 발생하지 않는다. 즉, 서브 트리가 더 이상 분리되

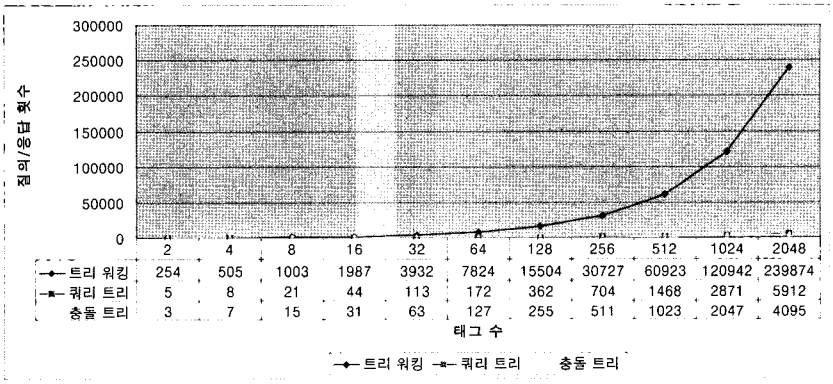


그림 11 질의/응답 횟수 비교

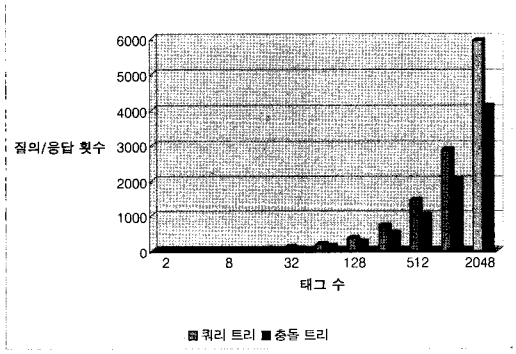


그림 12 쿼리트리와 충돌 트리 알고리즘의 질의/응답 횟수

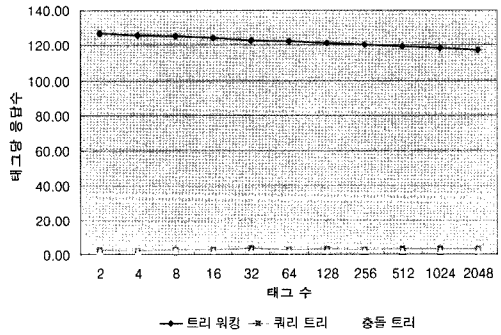


그림 13 태그 당 평균 질의응답횟수

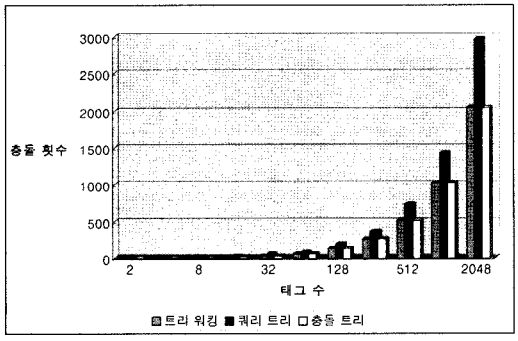


그림 14 충돌 횟수

지 않으면 결국 하나의 단말 노드가 생성되며, 트리 전체에서 단말 노드의 수는 이진트리의 특징에 따라 차수가 2인 노드보다 +1한 값과 동일하다. 트리 워킹에서 차수가 2인 노드는 충돌이 발생한 노드임으로 결국 식 (4)가 성립한다. 충돌 트리 역시 차수가 2인 노드는 충돌 노드임으로 식 (4)가 성립함을 알 수 있다. 따라서 트리 워킹 알고리즘과 충돌 트리 알고리즘은 항상 동일한 충돌 횟수를 가지고 충돌 횟수는 전체 태그 수에 -1 한 값이 된다.

표 2는 태그 개수별 총 질의 비트수, 총 응답 비트수 및 무응답 횟수를 나타낸 표이다. 트리 워킹 알고리즘의 경우 총 질의/응답 비트수에 대하여 질의 비트수는 98.4%이고, 응답 비트수는 1.6%로 대부분 자료 전송은 리더가 프리픽스를 전달하는데 이용된다. 응답 비트수가 1.6%인 원인은 트리 워킹 알고리즘의 매회 질의/응답과정에서 태그가 1비트씩만 응답하기 때문이다. 쿼리 트리 알고리즘의 경우 총 질의 비트는 전체 데이타 전송량의 1.9%이고, 응답 비트수는 98.1%이다. 프리픽스의 비트수가 작은 경우 자신의 tag_id와 일치하는 태그의 수가 많기 때문에 프리픽스의 $k+1$비트에서 마지막 $n$비트까지 전송함으로 쿼리 트리 알고리즘의 경우 응답 비트수가 많게 된다. 충돌 트리 알고리즘의 경우 총 질의 비트수는 1.4%, 응답 비트 수는 98.6%로 쿼리 트리와 큰 차이가 없다. 하지만 쿼리 트리의 경우 무응답인 경우가 발생하기 때문에 리더가 일정시간 대기해야 하는 오버헤드가 존재한다. 이것은 결국 각 태그를 식별하는 시간의 증가를 의미함으로 쿼리 트리 알고리즘의 오버헤드가 된다.

표 2 성능 실험 결과

| Tag 개수 | | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 | 2048 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Tree Walking | 무응답 횟수 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| | 총 질의 비트 수 | 16254 | 32506 | 64992 | 129931 | 259643 | 519174 | 1037453 | 2072963 | 4141559 | 8275356 | 16530708 |
| | 총 응답 비트 수 | 256 | 512 | 1024 | 2048 | 4096 | 8192 | 16384 | 32768 | 65536 | 131072 | 2621440 |
| | 총 질의/응답 비트 수 | 16510 | 33018 | 66016 | 131979 | 263739 | 527366 | 1053837 | 2105731 | 4207095 | 8406428 | 16792852 |
| Query Tree | 무응답 횟수 | 1.00 | 0.67 | 3.00 | 6.67 | 25.00 | 22.33 | 53.33 | 96.33 | 222.67 | 412.00 | 908.67 |
| | 총 질의 비트 수 | 8 | 16 | 72 | 189 | 675 | 1046 | 2704.67 | 5879 | 13997 | 29582 | 67046 |
| | 총 응답 비트 수 | 760 | 1814 | 4903 | 12314 | 31343 | 64351 | 146901 | 326201 | 718321 | 1538797 | 3319758 |
| | 총 질의/응답 비트 수 | 768 | 1830 | 4975 | 12503 | 32019 | 65398 | 149605 | 332080 | 732318 | 1568379 | 3386804 |
| Collision Tree | 무응답 횟수 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| | 총 질의 비트 수 | 4 | 13 | 42 | 121 | 328 | 736 | 1759 | 4082 | 9226 | 20260 | 44539 |
| | 총 응답 비트 수 | 508 | 1604 | 4165 | 10551 | 24800 | 57927 | 132897 | 299176 | 660768 | 1433200 | 3085918 |
| | 총 질의/응답 비트 수 | 512 | 1617 | 4207 | 10672 | 25128 | 58664 | 134656 | 303258 | 669994 | 1453460 | 3130458 |

전체 실험 결과에서 트리 워킹 알고리즘은 쿼리 트리 알고리즘과 충돌 트리 알고리즘보다 성능이 많이 떨어짐을 알 수 있다. 쿼리 트리 알고리즘과 충돌 트리 알고리즘은 성능에 큰 차이는 없지만 대부분의 평가 항목에서 충돌 트리 알고리즘이 좋은 성능을 보임을 알 수 있고 또한 추가적인 하드웨어나 소프트웨어가 필요 없기 때문에 수동형 태그에 적용 할 경우 매우 유리함을 알 수 있다.

## 6. 결 론

본 논문에서는 n비트로 구성된 k개의 비트열에 대하여 이들 사이에 불일치하는 비트를 찾아 트리를 분리하는 기법으로 트리에 표현 할 수 있는 충돌 트리를 제안하였다. 또한 제안된 충돌 트리를 이용하여 RFID시스템에서 리더기와 태그들 사이에 발생하는 충돌을 해결하기 위한 충돌 트리 기반 충돌 방지 알고리즘을 제안하였다. 충돌 트리 기반 충돌 방지 알고리즘은 트기기반 메모래스 알고리즘으로 RFID 리더가 생성한 프리픽스 질의에 대한 태그들의 응답에서 충돌이 발생한 비트의 위치를 찾아 새로운 프리픽스를 생성하는 방법으로 각 태그들을 식별한다. 충돌 트리 기반 충돌 방지 알고리즘의 성능을 비교하기 위하여 기존의 트리 워킹 알고리즘, 쿼리 트리에 대하여 질의/응답 횟수, 충돌 횟수, 질의/응답 비트 수를 비교하였다. 실험 결과 모든 면에서 본 논문에서 제안하는 충돌 트리 기반 충돌 방지 알고리즘이 기존의 알고리즘들 보다 우수함을 알 수 있다.

차기 연구과제는 개선된 쿼리트리에서 전달 중지 명령을 처리하기 위한 오버헤드 비용 산출과 이를 이용하여 본 논문에서 제안하는 충돌트리의 성능을 비교하는 것이다.

## 참 고 문 헌

[1] Feng Zhou, Dawei jin, Chenling Huang and Hao Min, "White Paper: optimize the power Consumption of Passive Electronic Tags for Anti-collision Schemes," Auto-ID center Fudan Univ., October, 2003.

[2] Ari Juels, Ronald L. ivest and Michael Szydlo, "The Block Tag : Selective Blocking of Tags for Consumer Privacy," *Proceedings of the 10th ACM conference on Computer and communication security*, ISBN:1-58113-738-9, pp. 103-111, 2003.

[3] Jacomet M, Ehrsam A and Gehrig U. "Contactless identification device with anti-collision algorithm," *IEEE Computer Society, CSCC'00, Conference on Circuits, systems, computers and communications*, Athens. 4-8 July 1999.

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

[5] Frits C and Scoute, "Control of ALOHA signalling in a Mobile Radio Trunking System," *In International Conference on Radio Spectrum Conservation Techniques*, pp. 38-42. IEEE. 1980.

[6] Jeffrey E. Wieselthier, Anthony Ephremides and Larry A. Michaels, "An Exact Analysis and Performance Evaluation of Framed ALOHA with capture," IEEE Transactions on Communications, COM. Vol.37, No.2, pp. 125-137, 1989.

[7] Harald Vogt, "Efficient Object Identification with Passvive RFID Tags," *In International Conference on Pervasive Computing*, LNCS. Springer-Verlag 2002.

[8] H.S. Choi, J. R Cha and J. H Kim, "Fast Wireless Anti-Collision Algorithm in Ubiquitous ID System," *In Proc. IEEE VTC 2004, L.A.*, USA, September 26-29, 2004.

[9] Jae-Ryong Cha and Jae-Hyun Kim, "Dynamic framed slotted ALOHA algorithms using fast tag estimation method for RFID system," *Consumer*

*Communications and Networking Conference, 2006. CCNC 2006.* 2006 3rd IEEE. Volume 2, Issue, 8-10 Jan. 2006 pp. 768-772.

[10] 권성호, 홍원기, 이용두, 김희철, "RFID 시스템에서 트리 기반 메모리래스 충돌방지 알고리즘에 관한 연구", 정보처리학회논문지 C 제11권 제6호, 12월, 2004.

서 현 곤

1994년 경성대학교 이학사. 1996년 경성대학교 이학석사. 2004년 영남대학교 공학박사. 1994년~1997년 티센크루프동양엘리베이터(주) 기술연구소 주임연구원 2001년~2003년 대구대학교 정보통신공학부 BK21교수. 2004년~2005년 영남대학교 컴퓨터공학과 강의전담교수. 2005년~현재 한라대학교 정보통신공학부 교수. 관심분야는 애드 혹 네트워크, RFID/USN, Embedded System