EPC 네트워크를 위한 다중 RFID 태그 식별 알고리즘의 분석☆

권 성 호* 김 희 철*

◈ 목 차 ◈

1. 서 론

모든 공간의 사물이 지능화되고 언제 어디서나 제한 없는 접속이 이루어지는 컴퓨팅 환경의 도래로 인하여 인류역사에는 농업혁명, 산업혁명, 정보혁명에 이어 새롭게 유비쿼터스 혁명이 시작되고 있다[1,2,3]. 현재 세계 각 국은 국가적 차원에서 유비쿼터스 컴퓨팅 관련 연구 및 개발에 박차를 가하고 있다. 유비쿼터스 환경 실현의 필수기술인 RFID(Radio Frequency IDentification) 관련 산업은 IT 분야의 신성장동력 산업 중의 하나로 급부상하고 있다.

RFID는 유비쿼터스 시대의 가장 핵심적인 기술로서 무선기술과 소형 칩을 이용하여 사물(Object)을 비접촉 방식으로 자동 식별함으로서 국방, 의료, 유통, 교통, 보안, 제조, 서비스, 행정 등의 다양한 응용분야에 적용될 수 있다[4,5]. RFID 시스템은 사물에 부착된 작은 태그(Tag)와 태그의 정보를 읽을 수 있는 장치인 태그 리더(Tag Reader, 이하 *리더*)로 구성된다. 태그들은 에너지원의 보유 여부를 기준으로 에너지원을 보유하고 있는 능동식 태그(Active Tag)와 리더의 신호로부터 에너지를 공급받아 동작하는 수동식 태그(Passive Tag)로 구분 할 수 있다. 본 논문에서는 소형화가 가능한 수동식 태그에 초점을 두고 있다.

* (중)연변대학교 컴퓨터과학과
** 대구대학교 공과대학 정보통신공학부 교수
☆ 본 연구는 2001년도 대구대학교 학술 연구비 지원에 의한 논문임.

RFID 태그는 기존의 상품이나 물류와 관련하여 널리 사용되고 있는 향후 바코드(Bar Code)를 대체할 수 있는 새로운 기술로서 각광을 받고 있다. 현재, 태그 당 가격이 5센트 이하로 떨어지고 있다. 이 가격은 향후 3년에서 5년 사이에 1센트 이하로 낮아지게 될 것으로 예상되므로 모든 사물에 RFID 태그가 부착된 유비쿼터스 환경 실현이 먼 미래가 아님을 알 수 있다. 또한 최근 표준화가 진행되고 있는 EPC(Electronic Product Code), RFID 태그, 리더(Reader), 미들웨어, ONS(Object Name Service)를 포함하는 EPC 네트워크 기술을 기반으로 가상공간(Bits)과 물리공간(Atoms)을 포함하는 네트워크가 구축되면 모든 사물들이 인터넷에 연동되는 진정한 유비쿼터스 환경이 구현될 것으로 예상된다[6,7].

사물에 대한 식별은 사물에 부착된 태그에 대한 리더의 질의로 시작된다. 리더의 인식 영역(Read Range) 내에 여러 개의 태그가 존재할 경우, 이 태그들이 동시에 리더의 질의에 응답 하게 되므로 충돌이 발생한다. 이러한 충돌을 방지하며 여러 개의 태그들을 동시에 식별해야 하는 문제를 다중 태그 식별 문제(Multiple Tag Identification Problem)라 한다. 이 다중 태그 식별 문제는 기존의 바코드, 교통 카드, 무선 결제 시스템과 같은 태그와 리더간의 일대일 통신 환경에서는 발생하지 않는다. 따라서 본 논문에서는 RFID 시스템의 다중 태그 식별 문제를 해결하기 위한 대표적인 알고리즘을 소개하고 구현 복잡도, 성능, 실용성에 대한 분석결과를 제시한다.

논문의 나머지 부분은 다음과 같이 구성된다. 2장

에서는 RFID 시스템 소개한 후 다중 태그 식별 문제의 정의 및 다중 태그 식별 알고리즘의 분류에 관하여 기술한다. 3장에서는 대표적인 다중 태그 식별 알고리즘들에 대한 상세한 설명을 제공한다. 4장에서는 다중 태그 식별 알고리즘의 분석결과를 기술하며, 마지막으로 5장에서는 결론을 맺는다.

2. 다중 태그 식별 문제

RFID 시스템은 크게 리더와 태그로 구성되며 리더는 태그와의 통신을 위한 송/수신부와 정보전달을 위한 외부 인터페이스로 구성된다. 송/수신부는 안테나를 통하여 태그로 신호를 전달하고 태그로부터 필요한 정보를 수신하게 된다. 태그는 트랜스폰더(Transponder)라고도 하며 태그의 식별자 정보와 리더와의 통신에 필요한 기본적인 회로만을 가지고 있다. 리더의 인식 영역 내에 있는 태그들은 리더의 신호로부터 에너지를 공급받아 자신의 정보를 리더에게 송신하게 된다.

태그 식별은 리더가 사물에 부착된 태그에 대한 질의와 태그의 리더에 대한 응답으로 구성되는 리더와 태그사이의 통신을 통하여 이 이루어진다. 이러한 RFID 시스템에서의 리더와 태그사이의 통신은 아래와 같은 이유로 기존의 무선 통신과 구별된다.

- 태그는 리더의 신호로부터 에너지를 공급받아 동작함으로 전력소비에 민감하다.
- 태그들은 내부 메모리나 연산 능력에 많은 제한이 있다.
- 태그들은 주위의 태그들과 서로 통신을 할 수 없다.
- 리더는 인식 영역 내에 있는 태그들의 개수를 알 수 없다.

사물에 부착된 태그들은 리더의 명령에 따라 태그에 포함된 정보를 리더로 전송한다. 만약 여러 개의 태그가 리더의 인식 영역 내에 존재할 경우, 여러 개의 태그들이 동시에 리더의 질의에 응답할 수 있다. 이 때, 리더에서는 충돌이 발생하게 된다. 이러한 충돌이 발생하면 리더는 수신된 정보를 정확히 인식할 수 없게 되면서 재전송을 요구한다. 이러한 재전송으로 인하여 대역폭 낭비가 발생하고 태그가 부착된 사물의 식별에 소요되는 시간이 늘어난다. 이러한 충돌 문제는 리더의 인식 영역 내에 있는 태그의 개수가 많을수록 더욱 심각해진다. 궁극적으로 다중 태그 식별의 알고리즘의 구현은 충돌 방지 기법으로 귀결된다.

다중 태그 식별 알고리즘은 크게 결정형(Deterministic) 충돌 방지 방식과 확률적(Stochastic) 충돌 방지 방식으로 분류된다. 먼저 결정형 충돌 방지 방식의 알고리즘의 동작은 아래와 같이 요약된다.

① 리더가 인식 영역 내의 태그들에게 식별자 요청의 질의를 보낸다.
② 리더의 질의에 대하여 충돌이 발생한 태그들 중에서 일부 태그들을 그 다음 사이클의 질의에는 응답하지 않도록 한다.
③ 오직 한 개의 태그만 응답하여 해당 태그가 식별되어질 때까지 ①~②과정을 반복한다.
④ 식별된 태그들은 다른 모든 태그들이 식별되어질 때까지 응답을 하지 않도록 한다.
⑤ 리더 영역 내의 모든 태그들이 식별되어질 때까지 ①~④과정을 반복한다.

이러한 결정형 방식의 다중 태그 식별 알고리즘으로서 Tree-walking 알고리즘[8], Tree 알고리즘[9], Bit-arbitration 알고리즘[10], Memoryless 알고리즘[11] 등이 있다.

확률적 충돌 방지 방식의 알고리즘은 일반적으로 Framed-Slotted Aloha 프로토콜[12]을 기반으로 하고 있다. 리더의 질의 영역 내의 태그들은 주어진 N개의 슬롯에서 태그의 정보를 전송할 슬롯을 임으로 선정하여 해당 식별자를 전송하게 된다. 일반적으로 인식 영역 내의 태그의 개수를 알 수 없는 상황으로 인하여 태그의 개수에 적정한 슬롯의 개수와 종료 시점의 계산은 확률적 접근방식을 사용한다. 대표적인 방식으로 I-Code 알고리즘[13]이 있다.

효율적인 태그 식별을 위해서는 알고리즘 설계 시에 아래와 같은 사항들이 고려되어야 한다.

- 전체 태그 식별 시간: 리더 인식 영역 내에 있는 사물에 부착된 모든 태그 식별에 소요되는 시간을 말한다. 가능한 최소화할 수 있어야 한다.

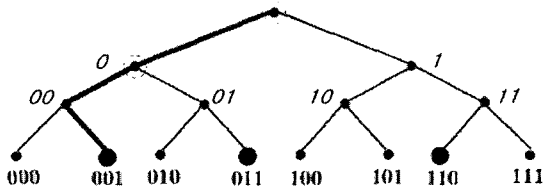
- 태그 복잡도: 알고리즘 구현 시 필요한 내부 메모리, 매칭에 필요한 계산 회로, 동기화에 필요한 클락 회로 등과 관련이 있다. 태그의 소형화를 위하여서는 가능한 간단하게 설계되어져야 하고 복잡한 회로는 그만큼 소요하는 전력도 많다.
- 통신 메시지 양: 태그 식별을 위한 과정에서 리더와 태그들 사이에서 전송되어지는 메시지의 양을 의미한다. 많은 양의 메시지 전송을 위하여서는 그만큼 많은 양의 전력을 요구한다.
- 태그 이동성: 리더의 인식 영역 내의 태그들은 식별 과정 중에 인식 영역의 범위를 벗어나거나 새로운 태그들이 인식 영역의 범위에 진입할 수 있다. 따라서 태그 식별 알고리즘들은 이러한 태그집합의 특성을 고려하여야 한다.
- 식별 완전성(Completeness): 리더는 인식 영역 내에 있는 모든 태그들을 인식할 수 있어야 한다.
- 태그집합 확장성(Scalability): 리더 인식 영역 내의 태그의 개수의 증가에 따라 유연하게 확장될 수 있어야 한다.
- 신뢰성(Reliability): 태그 식별의 과정은 정확하게(Correctly) 이루어져야 한다. 전달된 신호를 정확히 인식할 수 없는 경우의 재전송 메커니즘, CRC를 통한 신호의 검증, 변/복조 등 알고리즘 및 통신과 관련된 내용들이 포함될 수 있다.
- 강건성(Robustness): 태그 식별의 과정은 주위의 환경 조건과 관계없이 이루어져야 한다. 알고리즘보다는 불필요한 신호들의 간섭 등 통신과 관련된 내용이 포함될 수 있다.

3. 태그 식별 알고리즘

3.1. Tree-walking 알고리즘

Tree-walking 알고리즘[8]은 비트 단위의 질의와 응답 과정을 반복하며 깊이 우선(Depth-first) 이진트리 탐색과 유사한 방식으로 수행된다.

각 태그의 식별자는 k 개의 비트로 구성되며 $ID=b_1b_2...b_k$로 표시한다고 가정하자. 여기서 b_1, b_2, b_k는 각각 ID의 첫 번째, 두 번째, k 번째 비트를 의미하며 각 비트는 0 또는 1의 값을 가진다. ID의 시작 d ($d \leq k$)개의 비트 스트링은 프리픽스(Prefix) B라고 지칭하며 $B=b_1b_2...b_d$로 표시한다.



(그림 1) Tree-walking 사례[8]

Tree-walking을 위한 깊이가 k인 이진 탐색 트리의 구성은 다음과 같다. 각 노드는 노드의 깊이(Depth)에 해당하는 비트 크기를 갖는 프리픽스를 노드 값으로 갖는다. 예를 들면, 루트노드는 노드의 깊이 값이 '0'이므로 프리픽스 값이 정의되지 않으며 그 자식 노드는 깊이 값이 '1'이므로 $B=b_1$으로 표현되는 프리픽스 값을 갖는다. 즉, 루트노드의 왼쪽 자식노드는 b_1이 '0'이 되고, 오른쪽 자식 노드는 b_1이 '1'이 된다. 이와 같이 깊이가 d인 노드들은 $B=b_1b_2...b_d$로 표시되는 프리픽스를 갖는다. 깊이가 k인 각 단말 노드(Leaf Node)는 하나의 태그를 나타내며 그 프리픽스는 해당 태그의 식별자에 해당한다.

Tree-walking 알고리즘의 태그 식별 동작 과정은 아래와 같다.

① 이진 탐색 트리에서 리더가 위치한 노드가 $B=b_1b_2...b_d$라고 할 때 리더는 영역 내의 태그들에게 B 값으로 질의를 한다. B를 프리픽스로 가진 태그들만 질의에 응답을 한다. 만약 B가 루트 노드일 경우에는 영역 내의 모든 태그들이 응답한다.
② 응답을 수행할 태그들은 태그 식별자의 b_{d+1} 비트를 리더에게 전송한다.
③ 태그로부터의 응답에 대하여 리더는 아래와 같은 두 가지 상황을 각각 처리한다. (a) '0'과 '1'이 동시에 수신되어 충돌이 발생했을 경우: 리더는 $B=b_1b_2...b_d0$ 혹은 $B=b_1b_2...b_d1$로 설정하고(설정 순서는 관계없음) 충돌이 발생했음을 기억한다. (b) 충돌이 발생하지 않을 경우, 즉 모두 '0'이 수신되었거나 아니면 '1'이 수신된 경우: $B=b_1b_2...b_db_{d+1}$로 설정한다. 여기서 b_{d+1}은 태그로부터 수신된 비트의 값이다.

④ 새로 설정된 B 값으로 단말 노드에 도달할 때까지, 즉 깊이 d가 k와 동일한 값이 될 때까지 과정 ①~③을 반복한다.

⑤ 한 개의 태그 식별이 끝나면 가장 최근에 충돌이 발생한 노드로 되돌아가 영역 내의 모든 태그가 식별될 때까지 과정 ①~④를 반복한다.

그림 1은 Tree-walking의 사례를 보여준다. 그림에서 큰 점으로 표시된 단말 노드 '001', '011', '110'은 리더 영역 내에 존재하는 세 개의 태그의 식별자(k=3)이며 각 중간 노드의 프리픽스는 이탤릭체로 표시되어 있다. '001'의 식별자를 갖는 태그의 식별 과정은 굵은 선으로 표시되어 있고 동그라미가 표시되어 있는 노드는 식별 과정에서 충돌이 발생하는 경우를 나타내고 있다.

Tree-walking 알고리즘의 수행 시간의 복잡도는 영역 내의 태그의 개수(n)와 태그의 식별자의 비트 길이(k)의 곱, 즉 $O(n \cdot k)$가 된다.

3.2 Tree 알고리즘

Tree 알고리즘[9]은 분할(Splitting) 혹은 Tree Search 알고리즘이라고도 하며, 충돌이 발생한 태그들을 B개의 서로소 집합(Disjoint Set)으로 분할하는 과정을 충돌이 발생하지 않을 때까지 반복적으로 수행하며 리더 영역 내의 모든 태그들을 식별한다.

영역 내의 모든 태그들의 집합을 S로 표시한다고 하자. 첫 번째 분할에 의하여 B개의 서로소 집합들을 $S_0, S_1, S_2, \ldots, S_{B-1}$가 생성된다. 또한 이러한 서로소 집합 중에서 S_2의 경우, 새로운 분할을 통하여 서로소 집합 $S_{20}, S_{21}, S_{22}, \ldots, S_{2B-1}$이 생성되게 된다. 마찬가지로 S_{22}도 다시 B개의 서로소 집합으로 분할될 수 있다. 이러한 과정은 B진 트리로 표시될 수 있으며 그 루트 노드는 S로 표시하고 각 자식 노드는 분할된 서로소 집합중의 하나를 나타낸다.

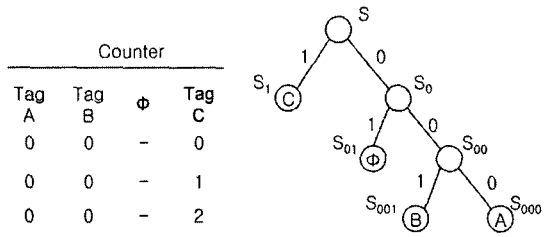
태그 식별을 위한 트리 알고리즘의 동작 과정은 아래와 같다.

① 리더는 영역 내의 모든 태그들에게 질의를 전송한다. 리더의 질의에 대한 응답으로 태그들은 식별자를 리더에게 전송한다. 이 태그들의 응답에 대하여 리더는 피드백 정보로서 충돌 발생 여부를 다시 태그들에게 전송한다.

② 리더의 피드백 신호를 참조하여 충돌이 발생한 각 태그는 Random number를 생성시켜 그 값에 준하여 B개의 서로소 집합 중 해당 집합에 할당되도록 한다.

③ 지정된 순서대로 한 개의 서로소 집합이 다음 타임 슬롯에 선택되고 이 집합에 포함된 태그들은 식별자 정보를 리더에게로 전송한다.

④ 영역 내의 모든 태그들을 식별할 때까지 과정①~③을 재귀적으로 수행한다.



Counter

Tag A	Tag B	Φ	Tag C
0	0	-	0
0	0	-	1
0	0	-	2
0	1	-	3
√	0	-	2
-	√	-	1
-	-	√	0
-	-	-	√

Random Number

Node	Tag A	Tag B	Tag C
S	0	0	1
S_0	0	0	-
S_{00}	0	1	-

(그림 2) 트리 알고리즘 사례

각 태그들은 해당하는 서로소 집합의 선택 시점을 인식하기 위하여 카운터(Counter)를 사용한다. 충돌이 발생했을 경우, 각 태그의 카운터 값은 과정 ②에서 할당된 서로소 집합의 인덱스 번호의 값으로 설정된다. 이미 식별된 태그와 충돌이 발생한 태그를 제외한 나머지 태그들의 카운터 값은 충돌이 발생할 때마다 '1' 증가하고 충돌이 발생하지 않는 경우에는 '1' 감소한다. 카운터 값이 '0'이면 태그는 자신의 식별자 정보를 리더에게 전송한다. 트리 순회과정의 구현은 일반적으로 스택이 사용된다.

그림 2는 트리 알고리즘의 사례를 보여준다. 이 사례에서 리더의 영역 내에 3개의 태그가 존재하고 B값은 2, 즉 태그의 집합들은 두 개의 서로소 집합으로

분할된다. 그림에서 각 태그가 보유하고 있는 카운터의 값들과 충돌이 발생했을 경우 각 태그들이 생성한 Random number 값들을 볼 수 있다. 그림에서 보는 바와 같이 알고리즘의 수행 결과로서 이 리더 영역 내의 태그의 식별은 A, B, C 순으로 이루어진다.

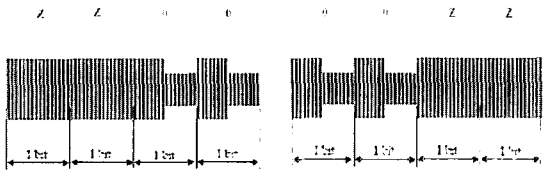
트리 알고리즘에서 각 태그들은 상태 정보를 유지·관리하여야 한다. Random number 발생을 위한 회로가 필요하나 태그 식별자로 대체하여 사용할 수도 있다. 태그 식별을 위한 시간은 영역 내의 태그의 개수(n)에 비례하며 태그와 리더 간의 메시지 전송량은 $O(nlogn)$의 복잡도를 갖는다[9].

3.3 Bit-arbitration 알고리즘

Bit-arbitration 알고리즘[10]은 비트 방식으로 중재를 활용한 충돌 방지 알고리즘으로서 태그의 식별자에 대하여 한 비트 단위로 식별을 진행한다.

태그 식별을 위한 알고리즘의 동작 과정은 아래와 같다.

① 리더는 영역 내의 모든 태그들에게 질의를 발송한다.
② 리더의 질의에 대하여 활성화(Active) 상태에 있는 태그들은 식별자의 주어진 위치의 비트(BitVal)를 리더에게 응답으로 전송한다.
③ 리더는 응답된 비트의 값(BitVal)에 따라 '0' 또는 '1'을 선택하여 태그로 전송한다. 이 값을 ContBit이라고 한다.
④ ContBit을 전송 받은 태그들은 자신이 보낸 BitVal와 ContBit를 비교하여 같은 경우, 액티브한 상태로 유지하며 식별자의 다음 위치의 비트를 리더로 전송하는 비트-중재(Bit-arbitration) 과정(②~③)을 반복한다. 만약, BitVal와 ContBit가 다를 경우, 대기(Wait)상태로 진입하면서 활성화 상태가 될 때까지 기다린다.
⑤ 태그 식별자의 처음 위치의 비트부터 시작하여 마지막 비트까지 중재(Arbtration) 처리가 종료되면 한 개의 태그 식별이 완료된다.
⑥ 한 개의 태그가 식별된 후에는 대기상태의 새로운 태그가 활성화(Active) 상태로 변환되어 새로운 비트-중재 과정이 시작된다.



(그림 3) 논리 '0'과 논리 '1'[10]

⑦ 위의 과정 ②~⑥이 영역 내의 모든 태그가 식별될 때까지 반복된다.

리더는 응답된 비트의 값(BitVal)에 따라 ContBit를 결정할 때 충돌이 발생하면 '0' 이나 '1' 중에서 하나를 선택하고 충돌이 발생하지 않으면 BitVal로 ContBit를 설정한다. 즉 이 경우는 응답된 비트의 값이 모두 '0'이나 모두 '1'임을 의미한다. 리더에서의 충돌 감지는 논리 '0'와 '1'에 대한 변조기법에 기반한다. 그림 3과 같이 논리 '0'는 'ZZ00'의 형태로 변조되고 논리 '1'은 '00ZZ'의 형태로 변조된다. 여기서 'Z'는 높은 임피던스 상태를 가리키며 따라서 충돌이 발생하면 '0000'형태의 신호가 감지된다.

Bit-arbitration 알고리즘에서 각 태그들은 상태 정보를 유지·관리하여야 하고 동기화를 위한 클락 회로가 필요하다. 태그 식별을 위한 시간 복잡도는 $O(2^k)$이고 여기서 k는 식별자의 비트 길이이다. 태그와 리더사이의 메시지 전송량은 $O(n*(k+1))$의 복잡도를 갖는다[10].

3.4 Memoryless 알고리즘

Memoryless 알고리즘[11]은 k 개의 비트 크기를 갖는 스트링의 모든 값을 가능한 최적의 방법으로 탐색함으로서 궁극적으로 영역 내의 모든 태그들을 식별한다. 'Memoryless'는 리더의 질의에 대한 태그의 응답은 오직 현재의 질의에만 의존한다는 의미를 나타낸다. 따라서 각 태그는 프리픽스 매칭(Prefix matching)을 위한 회로만이 필요하고 리더와의 통신에 사용되는 메시지는 매우 간단하다는 장점으로 갖는다. QT 알고리즘라고 불리는 태그 식별 과정은 계층적으로 이루어지며 쿼리 트리(QT, Query Tree) 형태로 표현된다 (그림 4).

QT 알고리즘의 대략적인 동작 과정은 아래와 같다.

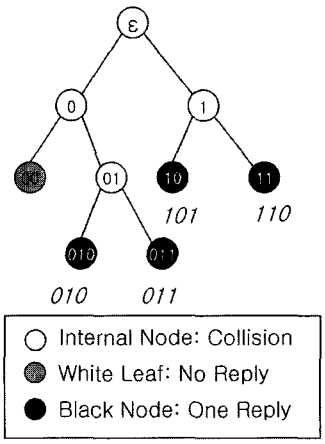
① 리더는 주어진 프리픽스 p를 태그로 전송한다.
② p를 프리픽스로 갖고 있는 태그는 자신의 식별자를 리더로 전송한다. 여기서 과정 ①~②를 메시징 사이클(Messaging cycle)이라고 한다.
③ 메세징 사이클의 결과에 따라 리더에서는 아래와 같이 세 가지 상황이 발생할 수 있다. a) 충돌 발생: p를 프리픽스로 갖고 있는 태그의 개수가 두 개 이상일 경우에 해당한다. 이 때 프리픽스 p에 '0' 또는 '1'을 뒤에 추가하여 새로운 프리픽스를 생성시킨 후 그 값을 큐에 저장한다. b) 태그 식별: p를 프리픽스로 갖고 있는 태그가 한개만 존재하는 경우로서 리더에게 한 개의 식별자가 전송되므로 리더는 유일하게 태그를 식별할 수 있다. c) 무응답: 프리픽스 p를 갖고 있는 태그가 존재하지 않아서 리더에게 식별자가 한개도 전송되지 않는 경우이다.
④ 큐로부터 새로운 프리픽스를 가져온다.
⑤ 과정 ①~④를 영역 내의 모든 태그들이 식별될 때까지 반복한다.

표 1은 알고리즘의 동작 과정을 단계별로 보여준다. 이 사례에서 리더의 영역 내에 '010', '011', '101', '110'의 식별자를 갖고 있는 태그들이 사물에 부착되어 있다. 첫 번째 단계는 초기에 값을 갖지 않는 프리픽스를 사용한 태그에 대한 리더의 질의와 태그로부터의 응답 결과를 보여준다. 표에서 'ε'는 빈 스트링(Empty String)을 나타낸다. 이 단계에서는 영역 내의 모든 태그들이 응답하며 따라서 리더에서 충돌이 발생하는 것을 볼 수 있다. 네 번째 단계를 살펴보면, 리더가 프리픽스 '00'을 갖는 질의를 수행한다. 이 때 영역 내의 태그에서 '00'를 프리픽스로 갖는 태그들이 존재하지 않으므로 응답이 발생하지 않는다. 단계 6에서 9까지는 해당 프리픽스를 갖고 있는 태그가 한 개만 존재하기 때문에 각 단계별로 한 개씩의 태그가 차례로 식별된다. 따라서 단계 9를 종료하면 영역 내의 모든 태그에 대한 식별이 완료된다.

(표 1) 리더와 태그사이의 통신 사례[11]

Step	Query Prefix	Response
1	ε	Collision
2	0	Collision
3	1	Collision
4	00	No Response
5	01	Collision
6	10	101
7	11	110
8	010	010
9	011	011

본 Memoryless 알고리즘에 보다 구체적인 이해를 위하여 공식적인 정의를 살펴본다. $A=U^{k}_{i=0}\{0,1\}^{i}$는 비트 길이가 k인 스트링의 집합으로 정의한다. Q는 A에 포함되어 있는 스트링들의 순서집합(Sequence set)으로 큐(Queue)로 구성된다. q는 A에 포함되어 있는 스트링으로서 리더의 질의에 사용되는 프리픽스를 나타내며, w는 $\{0,1\}^{k}$에 포함된 스트링으로서 태그로부터의 응답을 나타낸다. 태그의 식별자는 k 비트로 구성되며 $ID=b_1b_2...b_k$로 표시하며 'ε'는 빈 스트링을 나타낸다. 완전 이진 트리 형태인 QT(Query Tree)는 Q에 대하여 재귀적으로 정의되는 형태를 갖는다. QT의 루트 노드는 'ε' 프리픽스에 해당되고 그 하위 트리의 각 노드는 특정한 프리픽스 값에 대한 질의를 나타낸다. 즉, q를 갖는 노드는 프리픽스 q값에 대한 질의를 나타내며 $q0$, $q1$($q0$, $q1\in Q$)은 각각 노드 q의 왼쪽과 오른쪽 자식 노드가 된다. 표 1의 예제에 대한 QT는 그림 4



(그림 4) Query Tree 사례[11]

와 같다.

좀더 명확한 알고리즘[11]은 아래와 같다.

In the Reader:

```
1:  Begin
2:   Q ← <q1, q2, q3, ...,qi>;
3:   Do while (Q is not empty)
4:    begin
5:     Get q from Q;
6:     Broadcast query q to the tags;
7:     Receive response from the tags;
8:     If (collision is detected) Put q0, q1 to Q;
9:     Else if (only one tag replied)
10:    Identify one tag;
11:    Else // In case of no reply, do nothing;
12:   end;
13: End.
```

In the Tag:

```
1: Begin
2:  Receive query;
3:  If (q = 'ε' or q = b1b2...b|q|)
4:   Sends w to the reader; // w=ID
5: End.
```

Memoryless 알고리즘에서 태그 식별을 위한 시간 복잡도는 $O(n*(k+2-\log n))$이고 여기서 n과 k는 각각 태그의 개수와 식별자의 비트 길이를 나타낸다. 태그와 리더사이의 메시지 전송량은 $O(2.21k*\log n+4.19k)$의 복잡도를 갖는다[11].

위에서 설명한 기본적인 Memoryless 알고리즘에 대한 성능향상 기법으로서 *Short Cutting, Aggressive Advancement, Categorization, Dynamic Version of QT, QT-sl, QT-im* 등이 있다[11].

Short Cutting 기법은 QT의 노드 q에서 충돌이 발생하고 그 다음 질의인 노드 $q0$에 대해서는 응답이 발생하지 않는 경우에는 노드 $q1$에 대한 질의를 수행하지 않고 직접 노드 $q00$에 대한 질의를 수행하여 리더와 태그사이의 메시지 전송량을 감소시키는 기법이다. 이 기법에 준하여 그림 4를 살펴보면 '00'에 대한 질의를 수행한 후 '01'에 대한 질의를 하지 않고 직접 '010'에 대한 질의를 수행하면 된다.

Aggressive Advancement 기법은 사전 지식을 바탕으로 하거나 예측을 통하여 얻은 영역 내의 태그 개수 정보를 활용하여 메시징 사이클에서 기존 프리픽스에 추가하는 비트의 개수를 한개 이상으로 하는 기법이다. 예를 들면, 노드 '0'에 대한 질의를 수행한 후 그 다음 메시징 사이클에서는 '01'이 아니라 '001'이나 '0001' 등과 같은 값을 프리픽스로 질의를 수행할 수 있다. 이 기법도 Short cutting 기법과 마찬가지로 리더와 태그사이의 메시지 전송량 감소를 통하여 성능 향상을 얻는다.

Categorization 기법은 영역 내의 태그들에 대한 사전 지식을 이용하여 태그들을 여러 개의 그룹으로 나누어서 식별하는 기법이다.

그 외에 리더와 태그 간의 통신의 신뢰성 보장을 위한 결함 내성(Fault Tolerance)에 대한 연구도 진행되고 있으며 사전 정의된 차수만큼의 질의를 반복하는 동적 버전(Dynamic Version)의 QT 알고리즘도 제안되었다.

QT-sl 기법은 태그로부터 1-bit 길이의 응답을 요청할 경우에는 단형 질의(Short query)를 사용하고 태그로부터 식별자를 요청할 경우에는 장형 질의(Long query)를 사용하는 방식으로 동작한다. 예를 들면, 주어진 프리픽스에 한 개의 태그만이 매칭될 것이라는 사실을 알고 있을 경우에는 장형 질의를 사용한다. 이 기법은 리더와 태그사이에서 전송되어지는 메시지의 비트 개수를 감소시켜 성능 향상을 얻는다.

QT-im 기법은 리더가 점진적 매칭(Incremental Matching) 방식을 사용하여 1-bit의 추가 프리픽스 정보만을 태그로 전송하는 원리로 동작한다. 이를 위하여 각 태그는 프리픽스와 관련된 정보를 저장하고 있어야 한다. 따라서 엄밀한 의미에서는 이 기법은 Memoryless 알고리즘으로서의 성격을 상실했다고 할 수 있다. QT-sl과 같이 이 기법 또한 리더와 태그간의 메시지의 비트 개수의 감소를 목표로 한다.

3.5 I-Code 알고리즘

I-Code 알고리즘[13]은 Framed-Slotted Aloha 프로토콜[12]을 기반으로 하는 확률적 충돌 방지 기법이다.

알고리즘에서 하나의 리더의 사이클은 여러 개의 슬롯(Slot)으로 구성되는 하나의 프레임으로 정의된다. 태그 식별을 위해서 리더는 먼저 태그에게 <*I, rnd,*

N> 정보를 질의로 전송한다. 여기서 *I*: 태그 식별자 범위, *rnd*: Random number 생성에 필요한 seed 값, *N*: 한 프레임 내의 슬롯의 개수를 나타낸다. 이 질의에 대하여 영역 내의 각 태그는 프레임에서 하나의 슬롯을 임으로 선택하여 그 슬롯에 식별자 정보를 적재하며 리더는 프레임의 각 슬롯에 적재된 식별자 값을 사용하여 태그를 식별하게 된다. 이러한 과정은 영역 내의 모든 태그들이 식별되어졌다고 판단될 때까지 반복된다. 리더와 태그사이의 한 번의 질의와 응답과정을 한 사이클이라고 한다.

위의 식별 과정에는 *N 값의 크기 결정* 및 *실별 종료 시점 결정*의 두 가지 문제가 제기된다. 첫 번째 문제와 관련하여 살펴보면 지나치게 큰 *N* 값은 타임 슬롯의 낭비를 초래하고 반면에 지나치게 작은 *N* 값은 충돌 발생률을 증가시킨다. 따라서 적합한 *N* 값의 결정은 최적의 성능을 위해서 필수불가결하며 I-Code 알고리즘에서는 다음과 같은 방식을 사용한다. 먼저 이 알고리즘에서 각 태그는 임으로 하나의 슬롯을 선택함으로 인하여 태그로부터 응답 받은 프레임 내의 각 슬롯은 1) 비어는 경우 2) 한 개의 태그의 식별자 만이 적재되어 있는 경우 3) 여러 개의 태그들이 동시에 할당하여 해당 식별자들을 적재한 경우의 세 가지로 구분된다. 그러므로 이 결과로서 리더가 수신한 프레임은 위의 각 경우에 준하여 $<c_0, c_1, c_k>$로 표현되는 슬롯의 개수 분포를 갖는다. 여기서 c_0: 비어 있는 슬롯의 개수, c_1: 한 개의 태그 정보만 전송되어 진 슬롯의 개수, 그리고 c_k: 여러 개의 태그 식별자 정보들이 전송되어진 슬롯의 개수를 나타낸다. 위 분포 값을 이용하여 리더는 먼저 영역 내의 태그의 개수 최소한계치(Minimum bound) n 값을 $n=c_1+2c_k$ 관계를 이용하여 산출한다. 산출된 n 값을 표 2의 n_{Low} 와 n_{High} 값과 비교하여 해당 슬롯 개수 N 값을 결정한다. 예를 들면, 계산된 n 값이 15라고 가정하면 그 값은 위의 테이블에서 N 값이 32인 경우의 n_{Low} 와 n_{High} 값의 범위에 속하므로, 즉 (n_{Low}:10) < (n:15) < (n_{High}:27), N 값은 32로 결정한다. 참고로 표 2의 값들은 실험을 통하여 얻은 결과이다[13].

두 번째 문제, 즉 식별 종료 시점 결정에 관해서는 I-Code 알고리즘은 태그 식별과정을 Homogeneous Markov Process를 이용한 모델을 도입하여 그 해결책을 제시

(표 2) 프레임 사이즈 테이블

N	1	4	8	16	32	64	128	256
nLow	-	-	-	1	10	17	51	112
nHigh	-	-	-	9	27	56	129	Inf

하고 있다. s 사이클 후에 영역 내의 모든 태그들이 식별될 확률은 $Q^s q(0)[n]$으로 정의된다[13]. 여기서 Q는 변환행렬(Transformation matrix)이고 Q^s는 s 사이클 후의 변환행렬이다. $q(0)$는 초기 상태이고 n은 영역 내의 태그 개수의 예측 값이다. 예를 들면, $N=64$, $n=40$ 일 경우에 9번의 사이클을 종료한 후 40 개의 모든 태그를 식별할 수 있는 확률은 96.45%인 것으로 나타났다[13]. 태그 식별은 사이클의 반복을 통하여 수행되며 각 사이클이 종료된 후에 $Q^s q(0)[n] > a$ 조건이 만족되면 식별과정이 종료된다. 여기서 a는 확률값으로 일반적으로 0.99 이상의 값으로 지정된다. 정적(Static) 태그집합에 대한 I-Code 알고리즘[13]은 아래와 같다.

Algorithm *IdentifyStatic* ()

```
1:  Begin
2:   N ← 16; //초기 슬롯의 개수.
3:   n_est ← 0; //태그 개수의 추정치.
4:   stepN ← 0; //실행 사이클 카운터 변수.
5:   Do
6:    begin
7:     stepN++;
8:     c ← performRead(N); // read 사이클
9:     t ← estimateTags(N, c); // N값을 결정
10:    If (t > n_est)
11:     begin
12:      n_est ← t;
13:      N0 ← adaptFrameSize(N, n_est);
14:      // N값을 테이블과 비교.
15:      If (N0 > N)
16:       // restart with new frame size
17:       begin
18:        stepN ← 0;
19:        N ← N0;
20:       end;
21:     end;
22:   end;
23:  while (stepN < maxStep(N, n_est));
24: End.
```

(표 3) 충돌 방지 알고리즘 비교

알고리즘		태그 식별 시간	태그의 복잡도	통신 메시지 양	태그의 이동성	완전성	확장성
Deterministic	Tree-Walking	$n*k$	기본회로	$n*k$	-	100%	Good
	Tree	n에 비례	카운터, 랜덤 값 발생 회로	$n\log n$	-	100%	Good
	Bit-Arbitration	2^k	상태정보, 클락 회로	$n(k+1)$	-	100%	Bad
	Memoryless	$n*(k+2-\log n)$	기본회로	$2.21k*\log n+4.19k$	-	100%	Good
Stochastic	I-Code	$s*t$	랜덤 값 발생회로, 클락 회로	$s*n$	지원	Close to 100%	Good

n: 태그의 개수, k: 식별자의 비트 길이, t: 한 사이클 평균 소요시간, s: 수행 사이클 수

동적(Dynamic) 태그집합에 대한 I-Code의 알고리즘은 종료되지 않고 계속 수행된다는 점 이외에는 앞에서 설명한 정적 알고리즘과 유사하다[13].

I-Code 알고리즘에서 태그 식별을 위한 시간 복잡도는 $O(t*s)$이다. 여기서 t는 한 사이클에 소요되는 평균 시간을 나타낸다. 한편 태그와 리더사이의 메시지 전송량은 $O(n*s)$의 복잡도를 갖는다[13]. I-Code의 장점은 모든 해의 공간을 탐색하는 트리 기반 기법보다 일반적으로 더 짧은 시간에 태그식별을 수행할 수 있으며 태그의 이동성, 즉 동적 태그집합을 지원한다는 점이다. 하지만 I-Code 알고리즘은 주어진 입력 태그 집합에 대하여 모든 태그의 식별이 가능은 하지만 100%의 식별은 보장하지 못하는 단점을 갖는다.

4. 다중 태그 식별 알고리즘 분석결과

앞에서 다섯 가지의 대표적인 태그 식별 알고리즘에 대하여 살펴보았다. 이러한 기법들은 상호 장단점을 지니고 있으며 본 절에서는 이러한 장단점들을 태그 구현성 측면, 성능 측면, 실용화 측면으로 구분하여 비교·분석한 결과를 기술한다. 표 3에 그 내용을 간략하게 요약하여 나타내었다.

먼저 태그의 구현성 측면에서 보면, Memoryless 알고리즘과 Tree-walking 알고리즘을 위한 태그의 구현에는 단순한 기본적 회로만 요구되는 반면 그 외의 다른 알고리즘들에 대해서는 카운터와 같은 상태정보를 위한 회로, Random number 발생 회로, 클락 회로 등을 필요 한다. 일반적으로 수동식 태그의 특성상 전원 공급에 따른 상태 정보의 유실, 인접 리더 영역으로의 이동시 상태 정보의 설정 등 상태정보 유지·관리에 다양한 복잡성 등으로 인하여 상태정보를 기반으로 하는 알고리즘들의 실용화는 매우 어렵다고 볼 수 있다. 한편 리더와 태그사이의 통신 메시지 양 측면에서는 QT-sl 기법과 같은 Memoryless 알고리즘이 매우 우수한 성능을 갖는다. 이러한 우수성은 메시지 통신부의 저전력화 구현이 가능하므로 이러한 저전력화 구현 측면에 있어서는 Memoryless 알고리즘이 가장 우수하다고 할 수 있다.

알고리즘의 성능을 결정하는 대표적인 인자는 주어진 태그집합에 대한 전체 식별 시간과 성된다. 이전에 설명한 바와 같이 I-Code 알고리즘은 일반적으로 트리 기반의 탐색 기법보다 우수한다. 이러한 알고리즘들은 현재 초당 200개를 식별할 수 있으며 향후, 300개, 500개의 식별을 목표로 하고 있다. 트리 기반 알고리즘의 경우, 전제 태그 식별 시간은 지수형식을 갖는 Bit-arbitration 알고리즘을 제외한 다른 트리 기반 알고리즘들은 태그의 개수에 비례한다. 향후 성능 향상을 위한 방향으로 멀티-채널, 멀티-안테나 등을 이용한 병렬 전송 기법, 한 개의 심벌에 여러 비트의 정보를 송/수신하는 변/복조 기법, 기존의 질의/응답 구조가 아닌 연속질의-연속응답 등의 구조를 선택할 수도 있다. 따라서 향후, 이런 기법과 관련된 연구가 진행되어야 할 것이다.

실용화 측면에서는 태그의 식별 완전성(Completeness), 태그 이동성 지원 여부, 태그집합 확장성이 우선적으로 고려되어야 한다. 먼저 주어진 입력태그집합에 대하여 100% 식별 지원 능력을 나타내는 식별 완전성 측면에서는 I-Code 알고리즘을 제외한 나머지 네

개의 알고리즘은 식별 완전성을 지원하므로 이러한 알고리즘들이 I-Code 알고리즘보다 우수하다. 태그 이동성은 입력 태그 집합의 구성이 동적으로 변화하는 환경에서 태그 식별이 가능한지 여부를 나타낸다. I-Code 알고리즘을 제외한 나머지 네 개의 알고리즘은 태그 이동성을 지원하지 못한다. 태그집합 확장성은 태그 개수 또는 태그 비트 크기의 증가에 따른 성능 식별시간 측면의 확장성을 의미한다. $O(2^k)$로 표현되는 지수적인 시간 복잡도를 갖는 Bit-arbitration 알고리즘의 경우 일반적으로 k 값이 64 이상인 점을 고려하면 태그 집합 확장성 측면에서 실용성이 떨어진다고 볼 수 있다.

실용화 측면에서 위의 알고리즘 분석내용을 종합하면 기존의 대표적인 알고리즘들은 각각 실용화 측면에서 문제점을 지니고 있다. 즉, I-Code 알고리즘의 경우 식별 완전성을 지원하지 못하며, 트리 기반 알고리즘의 경우는 태그 이동성을 지원하지 못하고, Bit-arbitration 알고리즘은 태그 집합 확장성을 지원하지 못한다. 따라서 이러한 알고리즘들의 EPC 네트워크[6] 적용은 많은 문제 발생의 소지가 있다. 이 알고리즘들은 실용화 측면에서 살펴본 위의 세 가지 요소뿐만 아니라 그 외에 개인 프라이버시와 관련된 문제점도 현재 정확하게 해결방안을 제시하고 있지 않다.

5. 결 론

본 논문에서는 RFID 시스템의 멀티 태그 식별 문제의 해결을 위한 기존의 대표적인 알고리즘인 Bit-arbitration, Tree-walking, Tree 알고리즘, Bit-arbitration 알고리즘, Memoryless, I-Code 알고리즘에 대하여 구현성, 성능, 실용성에 대한 분석을 통하여 상호 장단점을 비교결과를 제시하였다. 먼저 태그 구현성 측면 있어서는 Memoryless 알고리즘이 태그 식별 과정에서 태그 내에 상태정보 유지에 필요한 회로를 요구하지 않으며 통신회로의 저전력화 구현이 가능하므로 가장 우수한 기법으로 나타나고 있다. 하지만 전반적으로 모든 알고리즘들이 EPC 네트워크와 같은 실용화 환경에서의 제반 요구사항을 만족시키지 못하고 있는 수준이다. 즉, I-Code 알고리즘이 전체 태그 식별 시간 측면의 성능에서는 나머지 알고리즘보다 우수하지만 태그 식별 완전성을 지원하지 못하는 단점을 지니고 있으며 트리 기반 알고리즘은 태그 식별 완전성은 보장하지만 태그 이동성은 보장하지 못하고 있다. 또한 Bit-arbitration은 태그 집합 확장성 측면에서 문제를 지니고 있다. 본 연구결과는 현재 전 세계적으로 추진되고 있는 EPC 네트워크의 실용화를 위해서는 태그 구현성, 식별 성능, 실용성 측면에서 기존 알고리즘의 단점을 극복할 수 있는 새로운 태그 식별 알고리즘의 개발이 필수불가결하다는 점을 시사한다.

참고문헌

[1] M. Weiser. The Computer of the 21st Century. *Scientific American.* pages 94-100. September 1991.

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

[3] 하원규, 김동환, 최남희, 유비쿼터스 IT 혁명과 제3공간. *전자신문사.* 2002.

[4] R. Want, K. Fishikin, A. Gujar, and B. Harrison. Bridging Physical and Virtual Worlds with Electronic Tags. *In ACM Conference on Human Factors in Computing Systems(CHI99).* Pittsburgh, PA. May 1999.

[5] K. Romer, T. Schoch. Infrastructure Concepts for Tag-Based Ubiquitous Computing Applications. Workshop on Concepts and Models for Ubiquitous Computing at Ubicomp 2002. Goteborg, Sweden. September 2002.

[6] EPC global. http://www.epcglobalinc.org

[7] EPC service. http://www.verisign.com/nds/directory/epc/

[8] A. Juels, R. Rivest, and M. Szydlo. The Blocker Tag: Selective Blocking of RFID Tags for Consumer Privacy. *Proceedings of the 10th ACM conference on Computer and communication security*, ISBN:1-58113-738-9, pages 103-111. 2003.

[9] Hush, Don R. and Wood, Cliff. Analysis of Tree Algorithms for RFID Arbitration. *In IEEE International Symposium on Information Theory*, pages 107-. IEEE, 1998.

[10] Jacomet M, Ehrsam A, Gehrig U. Contactless identi-

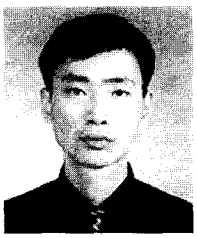
fication device with anti-collision algorithm. *IEEE Computer Society, CSCC'99, Conference on Circuits, Systems, Computers and Communications*, Athens. 4-8 July 1999.

[11] Law, Ching, Lee, Kayi and Siu, Kai-Yeung. Efficient Memoryless protocol for Tag Identification. *In Proceedings of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 75-84. ACM. August 2000.

[12] Frits C. Schoute. Control of ALOHA signalling in a Mobile Radio Trunking System. *In International Conference on Radio Spectrum Conservation Techniques*, pages 38-42. IEEE. 1980.

[13] Vogt, H. Efficient Object Identification with Passive RFID Tags. *In International Conference on Pervasive Computing*, LNCS. Springer-Verlag 2002.

◐ 저 자 소 개 ◑



권 성 호

1992년 (중)연변대학교 전자공학과 졸업
2001년 대구대학교 대학원 정보통신공학과 졸업(공학석사)
2003년 대구대학교 대학원 정보통신공학과 박사수료
1992년~1999년 (중)연변대학교 컴퓨터과학과 재직
관심분야 : RFID, EPC 네트워크, Grid, 지능 컴퓨팅

김 희 철

1983년 연세대학교 전자공학과 졸업(공학사)
1991년 University of Southern California (Computer Eng. M.-S.)
1996년 University of Southern California (Computer Eng. Ph.D.)
1983년~1988년 (주)삼성전자 주임연구원
1996년~1997년 (주)삼성SDS 수석연구원
1997년~현재 : 대구대학교 공과대학 정보통신공학부 교수
관심분야 : RFID, EPC 네트워크, Grid, 지능 컴퓨팅