고속 RFID 필터링 엔진의 설계와 캐쉬 기반 성능 향상

정회원 박 현 성*, 김 종 덕*

Design of a High-Speed RFID Filtering Engine and Cache Based Improvement

Hyun-Sung Park*, Jong-Deok Kim* ***Regular Members***

요 약

본 논문은 다수의 RFID 태그가 사용되고 있는 환경에서 고속 필터링을 수행하기 위한 필터링 엔진을 설계한다. 이를 위하여 우리는 고속 라우터나 방화벽에 적용되었던 고속 패킷 필터링 기법이 RFID 데이터 필터링과 매우 유사함을 보이고 그 중 대표적인 기법인 Bit Parallelism 기반의 Aggregated Bit Vector(ABV)를 고속 RFID 필터링 엔진에 적용한다. 또한, RFID 데이터 필터링의 성향을 관찰한 결과 태그 인식 및 필터 부합의 시간적 중복성을 발견하고 두 가지 캐쉬(태그 캐쉬, 필터 캐쉬)를 적용하여 추가적인 필터링 성능 향상을 꾀하였다. 설계한 RFID 고속 필터링 엔진의 성능 평가를 위해 프로토타입 애플리케이션을 제작하여 시뮬레이션을 수행하였다. 결과로써 기존의 순차적인 RFID 데이터 필터링에 비해 고속의 필터링 성능을 보이며 특히 필터의 수가 증가할수록 필터링의 효율이 높아짐을 보인다.

Key Words : RFID filtering, Readfilter, Bit Parallelism, ABV, Cache

ABSTRACT

In this paper, we present a high-speed RFID data filtering engine designed to carry out filtering under the conditions of massive data and massive filters. We discovered that the high-speed RFID data filtering technique is very similar to the high-speed packet classification technique which is used in high-speed routers and firewall systems. Actually, our filtering engine is designed based on existing packet classification algorithms, Bit Parallelism and Aggregated Bit Vector(ABV). In addition, we also discovered that there are strong temporal relations and redundancy in the RFID data filtering operations. We incorporated two kinds of caches, tag and filter caches, to make use of this characteristic to improve the efficiency of the filtering engine. The performance of the proposed engine has been examined by implementing a prototype system and testing it. Compared to the basic sequential filter comparison approach, our engine shows much better performance, and it gets better as the number of filters increases.

Ⅰ. 서 론

월마트 등의 국제적 대형 유통 및 물류회사들의 주도로 RFID 시스템의 적용 범위가 빠르게 늘고 있으며 실시간으로 처리해야 할 RFID 태그 정보의 양도 급격하게 늘어나고 있다. 그런데 엄청난 양의 RFID 태그 정보 중 실제 서비스 측면에서 의미 있는 정보는 제한적이다. 가까운 미래에 구축될 것으

※ 이 논문은 교육인적자원부 지방연구중심대학육성사업(차세대물류IT기술연구사업단)의 지원에 의하여 연구되었음.
* 부산대학교 컴퓨터공학과 이동통신 연구실 ({jesse, kimjd}@pusan.ac.kr)
논문번호 : KICS2006-01-015, 접수일자 : 2006년 1월 15일, 최종논문접수일자 : 2006년 4월 24일

로 보이는 지구적 규모의 RFID 망의 핵심 기능은 불필요한 태그 정보를 중도에서 삭제하고 필요한 정보는 그것을 필요로 하는 시스템이나 응용으로 라우팅하는 것인데 이를 RFID 필터링이라 할 수 있다.

RFID 필터링의 중요성이 부각되고 있는 것에 비해 관련 연구 성과는 제한적이다. 물론 EPCglobal 등 RFID 표준화 기구를 중심으로 기본적인 필터링 명세를 정의하였으며 RFID 미들웨어 공급사는 필터링 기능을 제공하고 있다. 그러나 대용량의 태그 정보 처리를 위한 고속 필터링의 구체적 기법을 다루고 있지는 못하며 이와 관련한 연구 성과도 찾기 어려운 실정이다.

1.1 EPCglobal 리더 프로토콜

RFID 산업화 및 기술을 주도하고 있는 EPC global은 RFID 리더 장비와 호스트와의 통신을 위한 표준 프로토콜로 리더 프로토콜(Reader Protocol)을 정의하였다. 이 프로토콜은 RFID 리더에서 수행할 수 있는 리드 필터라는 표준 RFID 태그 필터링 기능을 정의하고 호스트에 있는 응용이나 미들웨어가 이를 이용할 수 있게 한다. 리드 필터는 (Value, Mask) 라는 조합으로 표시되며 이에 부합된 태그의 통과/삭제 여부에 따라 Inclusive/Exclusive로 나뉜다. Inclusive 필터는 적어도 하나의 필터에는 부합하여야 한다는 의미이며 Exclusive 필터는 하나의 필터라도 부합하면 제외한다는 의미이다.[1]

1.2 SUN JAVA RFID 미들웨어에서 필터링

대표적인 RFID 미들웨어 업체 중 하나인 SUN 사의 RFID 미들웨어는 Smoothing, BandPass, Delta, EPC 필터의 4가지 필터링 기능을 제공하고 있다. Smoothing 필터는 EPCglobal 리더 프로토콜에서 명시된 바와 동일하며[1], BandPass 필터는 리더기 자체의 EPC에 대해 필터링 하는 것을 의미한다. Delta 필터는 해당 RF 주파수만을 걸러내는 필터를 의미하며 마지막으로 EPC 필터는 태그의 EPC 정보를 필터링 하는 것을 의미한다.[2] 따라서 EPC 필터를 리드 필터라 할 수 있으며 본 논문은 다수의 리드 필터에 대한 구체적인 필터링 기법을 소개하려 한다.

본 논문은 다수의 RFID 필터에 대해 고속의 필터링 작업을 수행하기 위한 RFID 필터링 모듈 구현 방안을 제시한다. RFID 필터링 문제가 기존 Packet Classification 문제와 유사함을 인식하고 기존 Packet Classification의 대표적인 기법 중 하나인 Bit Parallelism을 기반으로 RFID 필터링 모듈을 설계한다. 그런데 RFID 시스템의 특성을 활용하면 Bit Parallelism 알고리즘의 성능을 상당 부분 개선할 수 있다. RFID 시스템의 경우 동일한 태그가 연속으로 읽힐 가능성이 높다. 또한 동일한 태그는 아닐지라도 유사한 속성으로 인해 이전 태그와 동일한 필터에 부합하는 태그가 인식될 가능성도 높다. 전자를 태그 인식의 시간적 중복성, 후자를 필터 부합의 시간적 중복성이라고 하자. 논문은 2가지 시간적 중복성을 활용하여 Bit Parallelism 기반으로 한 RFID 필터링 모듈의 성능을 개선할 수 있는 태그 캐쉬 및 필터 캐쉬의 구조와 그 동작 방법을 제시한다.

Ⅱ. RFID 리더 토콜과 Bit Parallelism

본 장에서는 EPCglobal 리더 프로토콜에서 정의한 RFID 태그 구조와 표준 리드 필터에 대하여 살펴본 후 기존 Packet Classification 기법과 RFID 리드 필터링과의 유사성을 이끌어낸다. 그래서 대표적인 Packet Classification 기법 중에서 Bit Parallelism 알고리즘 기반의 ABV(Aggregated Bit Vector)를 RFID 리드 필터링에서 구현하는 방법을 제시한다.

2.1 RFID 태그 구조

EPCglobal은 [표 1]에 정리한 바와 같이 RFID 태그의 기본 골격을 정의하였다.

태그 데이터 구조는 기존의 EAN-UCC (European Article Numbering-Uniform Code Council)에서 정의한 다양한 형식을 수용하고 있으나 기본적으로 종류에 관계없이 1) Version 정보, 2) 제조사 정보를 나타내는 Domain Manager, 3) 제품 정보를 나타내는 Object Class, 그리고 마지막으로 4) Serial Number의 4가지 필드로 구성된다. [표 1]은 96 비트로 구성된 EPCglobal TDS (Tag Data Standard) 1.3에 명시된 GID-96 태그 (General ID) 구조의 예이다.[3]

표 1. TDS 1.3 태그 구조 (GID-96 예)

	Version Header	General Manager Number	Object Class	Serial Number
GID-96	8	28	24	36

2.2 RFID 리드 필터링

RFID 리드 필터는 일반적으로 16진수 필터 값(V)과 필터 마스크(M)로 표현된다. 즉, 아래와 같은 등호 관계가 성립될 때 태그 값이 필터링에 부합한다고 말한다.[3]

```
if ((V bitand M) == (A bitand M))
  then TagIDMatchesTheFilter()
```

간단한 비트열에 대한 예를 들면,

Filter mask M = 1C (00011100)

Filter value V = 10 (00010000)

Actual TagID A = 55 (01010101)

V bitand M = 10 bitand 1C

= 00010000 & 00011100 = 00010000 = 10

A bitand M = 55 bitand 1C

= 01010101 & 00011100 = 00010100 = 14

0 ≠ 14 : Deny

두 비트단위 AND 연산 결과가 상이하므로 해당 태그는 필터에 부합하지 않는다.

2.3 RFID 리드 필터링과 Packet Classification의 유사성 도출

[표 2]는 RFID 태그 정보를 크게 4가지 부분(Dimension)으로 나누어 나타내고 있다.

표 2. RFID 리드 필터 예제

Filter	Header	General Manager Number	Object Class	Serial Number
F1	35 FF	4AB8012 FF80000	856001 FFFFC0	0012ABC90 FFFE00000
F2	34 FF	4AB8012 FF80000	856001 FFFFFF	0012ABC91 FFFE00000

이는 [표 3]의 Packet Classification[4]에서 표현하는 바와 매우 유사함을 알 수 있는데 필터링 자체의 관점에서는 서로 동일하다. [표 2]에서 RFID 태그의 비트열은 편의상 16진수로 표현하였고 Action 필드가 없는 대신 Inclusive / Exclusive 필터로 그 기능을 대신하며 필터링 마스크는 [표 3]의 서브넷(Subnet) 마스크와 같은 역할을 담당한다. 따라서 TCP/IP에서 선행 연구되었고 검증된 대표적인 Packet Classification 알고리즘 중 Bit Parallelism과 이에 기반을 둔 ABV를 RFID 필터링 엔진에 적용하여 고속 필터링을 구현하였다.

표 3. Packet Classification 예제

Rule	Destination	Source	Port	Pro-tocol	Action
R1	164.12.34.15 255.255.0.0	174.12.34.16 255.255.0.0	www	*	Deny
R2	164.12.70.13 255.255.0.0	174.12.34.10 255.255.0.0	*	udp	Accept

2.4 Bit Parallelism기반의 비트 벡터 생성

Bit Parallelism을 설명하기 위하여 [표 4]는 간단한 2개의 부분(D1, D2)으로 구성된 필터들의 데이터이다.

표 4. Dimension 2개, 필터 8개 예제

Filter	D1	D2
F1	00*	11*
F2	10*	11*
F3	11*	10*
F4	0*	10*
F5	0*	01*
F6	0*	0*
F7	00*	00*
F8	1*	0*

[표 4]를 이용하여 [그림 1]과 같은 비트 벡터 Trie를 만들었다. Trie는 부분(Dimension)마다 하나씩 생성되며 비트 벡터의 자릿수는 필터의 개수 8개와 같고 각각의 비트는 필터 번호에 해당한다. Trie는 필터 값의 자릿수에 해당하는 0 또는 1을 보고 0이면 좌측, 1이면 우측으로 가지를 치며 내려오게 된다. 와일드카드(*)를 제외한 최대 자릿수는 2이므로 Trie의 깊이(Depth)는 2가 되며 마지막 노드에 다다랐을 때 자신의 자릿수에 해당하는 비트를 1로 세팅한다. 필터링 해야 하는 데이터가 입력되었을 때 이와 같이 생성된 비트 벡터 Trie로 부합하는 필터를 찾는 과정을 Bit Parallelism이라고 한다.[5]

예를 들어 간단한 8비트 태그 ID (00100111)가 리더로부터 읽혀졌고 D1(4Bit), D2(4Bit)로 이루어졌다고 할 때 [그림1]의 Trie를 타고 내려오면서 회색으로 표시된 노드의 비트 벡터를 각각 구할 수 있다. D1, D2의 비트 벡터를 비트단위 AND 연산하면,

10011110 & 00001101 = 00001100 = {F5, F6}

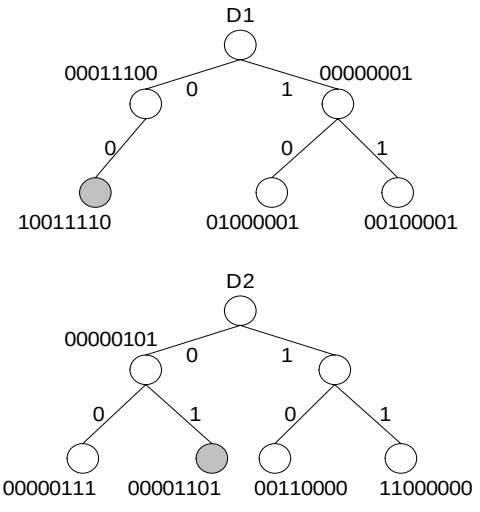

그림 1. Bit Parallelism을 이용한 필터링 예

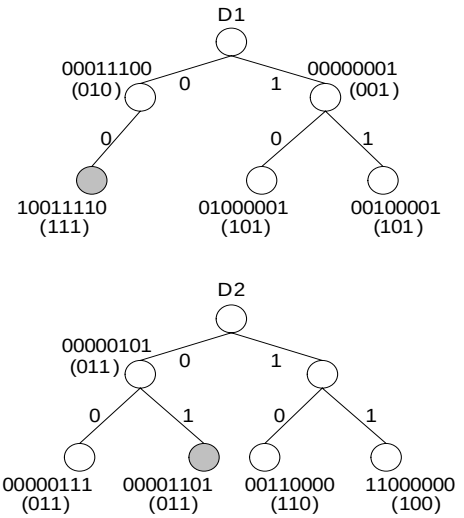

그림 2. ABV Trie를 이용한 필터링 예

태그 ID에 부합하는 필터는 F5와 F6임을 알 수 있으며 Inclusive 필터링라고 가정한다면 해당 태그 ID는 필터에 부합한다. 이때 Packet Classification의 경우에서와는 달리 우선순위(priority)를 고려할 필요는 없다.

2.5 RFID 리드 필터링에 ABV의 적용

ABV(Aggregated Bit Vector) 알고리즘의 핵심은 비트 벡터의 일부분을 비트단위 OR 연산하여 0로 나타나는 부분의 연산을 배제하고 1로 나타나는 부분만을 고려함으로써 실제 AND 연산의 횟수를 줄여 연산의 효율성을 꾀하는데 있다. 이 때 비트 벡터의 일부분을 OR 연산으로 더해서 비트수를 줄인다고 하여 Aggregated 되었다고 한다.[6] 이는 비트 벡터의 구성이 0의 개수가 1의 개수보다 많고, 필터의 개수가 많아서 벡터의 길이가 길어질수록 연산 이득이 증가한다. 이 때 최적의 Aggregate 크기를 결정하는 것은 알고리즘의 성능에 중대한 영향을 미치는 요소이다.

본 논문에서 다루려고 하는 바가 다수의 태그가 존재하고 이를 필터링하기 위한 다수의 필터들이 존재하는 시스템에서 속도의 효율성을 향상시키려 하므로 ABV 알고리즘에 적합한 모델이라 할 수 있다. 그러나 만들어진 비트 벡터들의 구성이 0의 개수가 1의 개수보다 상대적으로 많지 않다면 ABV를 적용하는 것이 오히려 오버헤드로 동작하게 되는 점에 유의하여야 한다.

앞서 예제에서 사용한 8비트 벡터에 ABV를 적용해 보자. Aggregate 크기는 3으로 정하였다. 따라서 3비트, 3비트, 2비트씩 비트단위 OR연산으로 합한다. [그림 2]의 각 노드에서 ()에 해당하는 비트 벡터가 ABV로써 추가하였다.

앞서 예에서와 동일한 실제 태그 ID (00100111)이 리더로부터 읽혔다고 하면,

ABV (111) & ABV (011) = 011

비트단위 AND 연산의 결과인 011은 비트 값 0을 가진 F1에서 F3에는 부합하는 필터가 없고 F4에서 F8사이에 부합하는 필터가 존재할 가능성이 있다는 것을 의미이다. 따라서 11110 & 01101 에 한해서만 비트단위 AND 연산이 필요하다. 만약 이와 같은 연산에서 만족하는 필터가 하나도 나오지 않을 가능성도 있는데 이를 두고 매칭 오류(False Match)가 발생하였다고 한다.[6]

보기로 들은 간단한 예제에서는 8비트의 태그 비트열이 최종적으로 부합하는 필터를 찾는 과정에서 3비트의 비트단위 연산 이득이 발생함을 보였다. 물론 연산과정에서 부가적인 메모리를 더 사용하였으며 ABV를 계산하는 부가연산도 발생하였다. 그러나 예제를 통해 비트 벡터들의 0의 개수가 1의 개수보다 많은 경우 필터의 개수가 증가할수록 부가적인 연산의 오버헤드를 상쇄하고도 남을 연산 이득이 발생할 수 있음을 알 수 있다.

Ⅲ. 캐쉬를 활용한 성능 개선 방안

본 장에서는 RFID 데이터 필터링의 성향을 관찰

한 결과 태그 인식 및 필터 부합의 시간적 중복성을 발견하고 두 가지 캐쉬(태그 캐쉬, 필터 캐쉬)를 적용하여 추가적으로 필터링의 성능을 향상시킨다.

3.1 제안 배경

II장에서 우리는 Packet Classification 알고리즘의 하나인 Bit Parallelism과 이를 개선시킨 ABV 알고리즘을 RFID 필터링 고속화 알고리즘으로 사용하였다. 그러나 여기서 다시 캐쉬를 사용하여 성능 향상을 꾀하려고 하는 이유는 RFID 시스템을 관찰한 결과 태그 인식 및 필터 부합의 시간적 중복성을 발견하였기 때문이다. 우리는 이를 다음과 같이 2가지 대전제로써 정의하였다.

- 단 기간 동안 동일한 태그가 중복되어 리더기로 읽혀질 가능성이 높다.
- 일정 기간 동안 한번 적용된 필터가 다시 적용될 확률이 매우 높다.

첫 번째 대전제에 대한 예를 들어 보자. RFID 시스템이 도입된 대형 할인 매장에 A라는 제품이 있다. 구매자가 A제품을 들고 다수의 리더기가 설치된 게이트를 통과할 당시에 한하여 중복된 RFID 태그 정보가 올라올 가능성이 매우 높으나 리더기가 설치되어 있지 않은 진열대에 계속 남아 있을 경우에는 리더기로 태그 정보가 전혀 전달되지 않는다. 즉, 리더기의 RF 감지 영역 안에 태그가 진입했을 때 리더기가 태그를 감지하는 주기에 따라 해당 태그 정보만 계속해서 읽혀진다는 것이다.

두 번째 대전제를 위하여 RFID가 도입된 대형 할인 매장에서 가나다 전자의 냉장고가 종류별로 입고되고 있는 상황을 가정하였다. 이 때 창고나 매장의 진열대로 진입하는 리더기에 읽혀질 태그는 같은 회사의 같은 종류의 제품이 연속적으로 이어질 가능성이 높으며 이는 곧 같은 종류의 필터가 연속적으로 적용될 확률이 매우 높다는 것을 의미한다. 따라서 우리는 [그림 3]과 같은 다이어그램을 그려서 두 가지 대전제에서 기인한 태그 캐쉬와 필터 캐쉬를 적용하였다.

먼저 태그 ID가 입력되면 태그 캐쉬를 통과한다. 태그 캐쉬에서는 첫 번째 대전제에서 가정한 바와 같이 이미 읽혀진 태그 ID인지 키(key)를 이용하여 조회한다. 이미 읽혀진 태그라면 값(value)을 조회하여 부합여부를 바로 알아낼 수 있다. 만약 새로운 태그 ID라면 필터 캐쉬를 거치게 된다. 필터 캐쉬

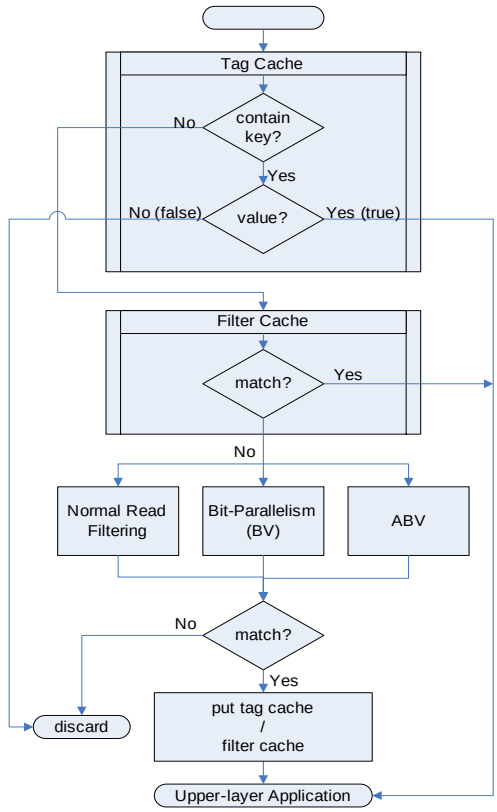

그림 3. 캐쉬를 적용한 필터링의 전체 흐름도

는 두 번째 대전제를 위한 캐쉬이며 필터링 알고리즘을 거치기 전에 확률적으로 부합할 가능성이 높은 필터를 먼저 연산에 사용한다.

3.2 태그 캐쉬의 적용

태그 캐쉬에 사용될 알고리즘은 해쉬를 이용한 해쉬 테이블(Table)과 해쉬 맵(Map)을 사용하기로 하였다. Bit Parallelism이나 ABV를 통하여 이미 최적 필터를 찾아낸 [표 5]의 예제 데이터를 가지고 해쉬 처리를 해보자. 평균적으로 고르게 분포되어야 할 해쉬 함수로 패러티(Parity)를 먼저 적용하였다. ABV 크기와 같이 패러티를 적용할 크기를 3비트로 하였으나 비트단위 OR 연산하지 않고 1의 개수가 기수이면 1, 우수이면 0으로 하여 [표 5]의 패러티 컬럼을 완성하였다.

[표 5]의 패러티 컬럼으로 해쉬 버킷(Bucket)을 분류하면 [표 6]과 같은 테이블을 만들 수 있다. 각각의 테이블 항목은 링크드 리스트로 구성하였으며 키 값(태그 ID)과 부합여부 (True / False)를 같이 저장하도록 하였다. 그리고 각각의 버킷 용량이 오버플로우(Overflow)되는 것을 방지하기 위하여 큐

표 5. 태그 캐쉬 적용을 설명하기 위한 예

No.	TagIDs	Matched?	Parity
1	00100011	T	100
2	11110011	T	110
3	10101100	T	000
4	10111011	F	000
5	00011111	T	010
6	10001101	T	101
7	11101011	T	110
8	01010111	T	100
9	10011001	F	101
10	10100101	T	011

표 6. 태그 캐쉬로 사용된 해쉬 테이블

Hash Bucket	Table Entries: key(value)
000	10101100(T) → 10111011(F)
001	-
010	00011111(T)
011	10100101(T)
100	00100011(T) → 01010111(T)
101	10001101(T) → 10011001(F)
110	11110011(T) → 11101011(T)
111	-

(Queue) 형식으로 오래된 필터 정보를 제거하도록 하였다. 큐의 크기를 결정하는 것은 적용하는 시스템에 따라 최적값으로 정해주어야 한다. 단 여기서 강조하고자 하는 바는 RFID 시스템의 특성상 일정 기간이 지난 태그 정보는 다시 읽혀질 경우가 희박하다는데 있다. 즉, 앞서 시간적 중복성에 기인한 2가지 대전제에 의하여 LRU(Least Recently Used) 캐쉬가 필요하므로 LRU 캐쉬의 구현을 통해 가장 오래된 태그 정보를 큐에서 제거해 주는 것이 효율적이다. 실제적인 구현은 자바(Java)를 기반으로 하였으며 자바 API의 LinkedHashMap 클래스를 사용하여 Doubly Linked List로 구성하였다.[7]

3.3 필터 캐쉬의 적용

필터 캐쉬에 사용될 알고리즘은 하나의 LRU 큐를 사용하였다. 두 번째 대전제에서 가정한 바와 같이 같은 종류의 필터가 사용될 확률이 다른 필터에 비해 매우 높으므로 LRU 캐쉬의 큐에 이미 찾아낸 최적 필터를 기억해 두고 필터 캐쉬에서 필터링을 우선적으로 수행하게 하였다.

3.4 캐쉬를 조회(Lookup) 의사코드

태그 캐쉬와 필터 캐쉬를 적용하여 전체적인 흐름을 의사코드로서 정리해 보자. 먼저 필터링 알고리즘을 수행하기 전에 캐쉬 조회를 통하여 태그 ID의 부합여부를 미리 결정할 수 있는 캐쉬의 조회(lookup) 부분을 살펴보자. 아래는 의사코드가 캐쉬 조회 부분에 해당한다.

```
Pseudo codes for cache lookup:
 1  Get TagID(D1, . . . ,Dk);
 2  HashValue ← HashFunction(TagID);
 3  if tagCache[HashVal].containsKey(TagID) then
 4    if tagCache[HashVal].getValue(TagID) then
 5      postToUpperApp(TagID);
 6      return;
 7    else
 8      return;
 9  else if filterCacheFunc(TagID) then
10    postToUpperApp(TagID);
11    return;
```

3.5 캐쉬를 적용한 ABV 의사코드의 작성

다음은 캐쉬를 적용한 필터링 알고리즘을 소개한다. ABV 알고리즘이 Aggregated 되기 전의 Bit Parallelism을 포함하고 있으므로 여기서는 ABV 알고리즘만 의사코드로 예를 들었다.

```
Pseudo codes for ABV filtering:
(Continues with the above)
...
12  for i ← 1 to k do
13    Ni ← LongestPrefixMatchNode(Trie_i, Di);
14  Aggregate ← 11 . . . 1;
15  for i ← 1 to k do
16    Aggregate ← Aggregate ∩ Ni.aggregate;
17  for i ← 0 to sizeof(Aggregate) - 1 do
18    if Aggregate[i] == 1 then
19      for j ← 0 to A - 1 do
20        if ∩_{l=1}^{k} Nl.bitVect[i x A + j]==1 then
21          tagCache[HashVal].put(TagID,true);
22          filterCache.put(i x A + j);
23          postToUpperApp(TagID);
24          return;
25  tagCache[HashVal].put(TagID,false);
26  return;
```

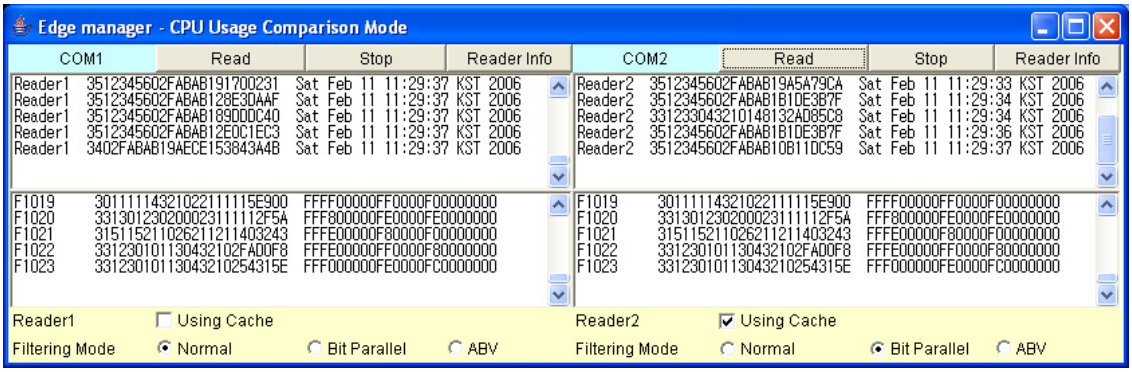

그림 4. 고속 필터링 알고리즘을 시뮬레이션 하는 프로토타입의 테스트 응용(EdgeManager)

Ⅳ. 구현 및 평가

본 장에서는 논문에서 제안한 바를 구현하고 모의 실험을 통하여 평가를 한다. 먼저 모의 실험의 데이터 소스가 되는 의사(pseudo) 태그를 생성하고 가상 리더를 통해 프로토타입의 테스트 응용에 전송하여 일반적인 필터링과 Bit Parallelism 알고리즘에 기반한 필터링과의 성능 비교를 수행하고 평가한다.

4.1 구현을 위한 준비

논문에서 제안한 방안을 실제 테스트하기 위해서는 다수의 리더기와 태그들이 구비되어 있는 대규모 RFID 시스템 환경에서 가능하다. 그러나 현실적으로 연구실에서 테스트 환경을 갖추기가 쉽지 않으므로 가상적으로 의사 태그들을 생성하여 RFID 미들웨어로 보낼 수 있는 가상 리더기 애플리케이션을 개발하였다. 가상 리더기를 통해 태그 ID를 읽고 제안한 고속 필터링을 시뮬레이션 할 수 있는 프로토타입 테스트 응용프로그램도 같이 제작하였다. [그림 4]는 현재 자체 개발 중인 RFID 미들웨어(EdgeManager) 중의 일부에 고속 필터링 알고리즘을 탑재하여 모의 실험을 하는 프로토타입의 테스트 응용 화면을 캡쳐 하였다.

평가를 위해 사용한 시스템 사양은 다음과 같다.

- CPU : AMD 1GHz
- 메모리 : 1GByte
- OS : EdgeManager(Windows XP), 가상리더(Redhat Linux)
- 개발 언어 : EdgeManager(Java), 가상리더(C)

4.2 의사(pseudo) 태그 데이터의 생성

평가의 소스 데이터가 되는 의사 태그의 생성은 가상 리더기의 다음 3가지 태그 생성 옵션을 사용하였다.

- Random 생성
- Common Prefix 패턴 생성
- Sequential 패턴 생성

Random 생성 방법은 완전한 난수(uniform random)로 발생시키는 것이며, Common Prefix 패턴 생성은 미리 정해둔 패턴을 포함시켜서 태그를 생성시키는 방법이다. 그리고 Sequential 패턴 생성은 먼저 생성된 하나의 태그 ID에 Serial Number만 1씩 증가시켜서 다음의 태그 ID를 생성하는 방법이다.

실제 시뮬레이션 시에는 3가지 옵션을 모두 사용하여 2가지 대전제를 반영하는 96 비트 태그 데이터를 생성하였다. 먼저 16진수로 [표 1]의 Version Header, General Manager Number와 Object Class의 일부를 파일에 미리 생성해 두고 Common Prefix 패턴 옵션에 의하여 임의의 패턴이 하나씩 읽혀지도록 하였다. 그리고 Random 생성 옵션에 의하여 96 비트 중 모자라는 나머지 태그 길이를 난수로서 완성한다. 완성된 의사 태그는 Sequential 패턴 옵션에 의하여 프로토타입의 테스트 응용에 전송될 때 Serial Number만 1씩 증가하여 배치(batch) 패킷의 형태로 전송된다. 이 때 2가지 대전제에서 제시한 시간적 중복성을 만족시키기 위해 N개 이하에서 임의의 수로 중복될 수 있게 하였다. 아래는 N=8 이하의 임의의 중복을 허용하고 세 가지 옵션을 모두 사용하여 프로토타입의 테스트 응용에 전달되는 96비트 의사 태그 데이터의 일부분

을 태그 ID만 발췌하여 나열하였다.

```
…
346BEE14D0FFC2CC3FCAD1B8
346BEE14D0FFC2CC3FCAD1B8
346BEE14D0FFC2CC3FCAD1B9
346BEE14D0FFC2CC3FCAD1BA
346BEE14D0FFC2CC3FCAD1BA
346BEE14D0FFC2CC3FCAD1BA
346BEE14D0FFC2CC3FCAD1BA
346BEE14D0FFC2CC3FCAD1BA
3201CE3043210266EEE8B552
3201CE3043210266EEE8B552
3201CE3043210266EEE8B553
…
```

4.3 필터링 속도 비교를 통한 성능 평가

평가는 크게 2가지 방법을 이용하였다.

첫 번째 방법은 다수의 의사 태그 데이터를 가상 리더기를 통해 파일에 미리 만들어 두고 알고리즘을 사용하지 않은 일반적인 경우와 알고리즘을 사용한 경우, 그리고 캐쉬를 적용한 경우에 대하여 속도측정을 하였다. 여기서 순수한 Bit Parallelism과 ABV와 세분화 하지 않은 이유는 성능에서 큰 차이를 보이지 않았기 때문이다. 이는 비트 1이 비트 0보다 매우 적어야 한다는 ABV의 특성과 성능 향상을 위한 필터 재정렬 (rearrangement)을 구현하지 않았기 때문이다.[6]

[그림 5]는 3가지 옵션을 통해 파일로 미리 작성해둔 총 1만개의 태그 ID들에 대해 필터의 개수를 128개 간격으로 1024개 까지 증가시키며 알고리즘을 사용하지 않은 일반적인 경우와 Bit Parallelism을 적용시킨 경우, 그리고 캐쉬를 적용한 경우의 성능 비교 결과이다. Bit Parallelism과 캐쉬 기반의 필터링이 기존의 순차적인 RFID 데이터 필터링에 비해 고속의 필터링을 수행하며 특히 필터의 수가 증가할수록 필터링의 효율성이 높아짐을 알 수 있다.

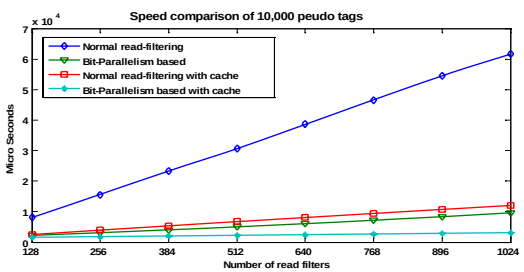

그림 5. 필터링 속도 비교

4.4 CPU 사용량 비교를 통한 성능 평가

두 번째 평가 방법은 다수의 태그 데이터의 필터링으로 인하여 RFID 미들웨어에서 발생하는 부하의 정도를 간접적으로 비교 측정해 보았다. 이를 위하여 마이크로소프트 윈도우즈 운영체제의 CPU 사용량을 측정하는 애플리케이션을 이용하였는데 이는 윈도우즈 운영체제에서 기본적으로 제공하는 신뢰성 있는 애플리케이션이며 손쉽게 사용할 수 있는 장점이 있다.

평가를 위하여 테스트 응용에 2개의 가상 리더기를 연결시키고 3가지 태그 생성 옵션을 모두 사용하고 초 단위로 의사(pseudo) 태그를 생성하여 실시간으로 전송하였다. 먼저 알고리즘을 사용하지 않은 일반적인 경우 초당 평균 500개의 태그를 전송하여 미들웨어가 1024개의 필터로 필터링하였을 때 [그림 6]과 같이 CPU 사용량이 거의 100%에 다다르는 한계를 보였다.

[그림 6]의 한계 상황과 동일한 조건으로 [그림 7]에서 Bit Parallelism 알고리즘을 적용하였을 경우와 [그림 8]에서 캐쉬를 추가로 적용한 경우에 대해서 각각 측정하였다.

Bit Parallelism 알고리즘을 사용한 경우와 캐쉬를 추가로 적용하였을 때의 한계치는 가상 리더기에서 전송하는 의사 태그의 개수를 증가시키며 실험을 계속한 결과 [표 7]에서와 같은 수치를 기록하였다.

[그림 6, 7, 8]에서 우리는 동일한 하드웨어에 추가적인 하드웨어의 확장 없이 필터링 알고리즘과

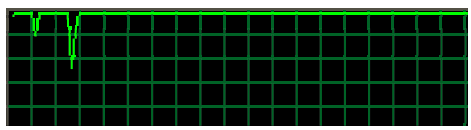

그림 6. 2개의 가상 리더기로 초당 평균 500개의 의사태그를 전송하여 미들웨어가 1024개의 필터로 필터링하였을 때의 CPU 사용량

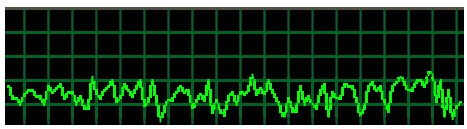

그림 7. Bit Parallelism 기반의 알고리즘을 적용한 경우의 CPU 사용량

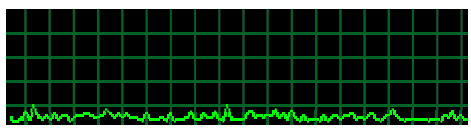

그림 8. Bit Parallelism 기반의 알고리즘에 캐쉬를 적용한 경우의 CPU 사용량

표 7. CPU 사용량을 통한 한계치의 비교

필터 수 1024개	일반적인 리드 필터링	Bit Parallelism 기반 필터링	캐쉬 적용한 Bit Parallelism 기반 필터링
	500개 /s	3500개 /s	5000개 /s

캐쉬의 적용만으로도 RFID 미들웨어의 성능을 소프트웨어적으로 개선시킬 수 있음을 보였으며 [표 7]에서는 Bit Parallelism과 캐쉬를 적용하였을 때 일반적인 리드 필터링에 비하여 약 10배 가량 CPU 부하를 감소시킴을 알 수 있었다.

V. 결 론

본 논문에서는 RFID 필터링의 중요성에 대해서 인식하고 관련 자료를 분석하여 효율적인 필터링 방안에 관하여 모색해 보았다.

먼저 Packet Classification 분야에서 선행 연구가 진행된 여러 알고리즘 중 Bit Parallelism과 ABV (Aggregated Bit Vector)의 원리에 착안하여 RFID 데이터 필터링 연산의 효율을 꾀하고자 하였다. 그리고 Packet Classification과는 구별되는 RFID 시스템의 특징 중 시간적 중복성을 발견하고 캐쉬를 도입하여 필터링을 더욱 효과적으로 할 수 있는 방안에 대해서도 제안하였다.

RFID 태그 정보의 필터링은 적용하려는 시스템에 따라 커스터마이징을 많이 필요로 한다. 그리고 처리해야 하는 태그 데이터의 전체적인 특성을 파악하여 최적의 필터링 알고리즘을 적용하여야 한다. 특히 캐쉬를 가지고 필터링의 효율성을 높이고자 한다면 본 논문에서 제안한 방안을 개량하거나 향후 추가적인 연구와 실험을 통하여 여러 가지 새로운 방안을 도출 할 수 있을 것이다.

참 고 문 헌

〔1〕 *EPCglobal*, “Reader Protocol 1.0” Last Call Working Draft Version of 17 March 2005.

〔2〕 Administration Guide Sun Java™ System RFID Software 2.0, *Sun Microsystems, Inc.*, www.sun.com Part No. 819-1697-10, April 2005.

〔3〕 *EPCglobal*, “Tag Data Standards Version 1.3” Standard Specification, 9 Sep, 2005.

〔4〕 Pankaj Gupta and Nick McKeown. “Algorithms for Packet Classification” Computer Systems Laboratory, Stanford University Stanford.

〔5〕 T.V. Lakshman and D. Stiliadis. “High-Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching”, *Proceedings of ACM Sigcomm*, pages 191-202, September 1998.

〔6〕 Florin Baboescu and George Varghese, “Scalable Packet Classification,” *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 199-210, 2001, University of California, San Diego.

〔7〕 Sun Microsystems, “Java API Specification., Java™ 2 Platform, Standard Edition, v 1.4.2 API Specification,” *Sun Microsystems, Inc.*, 2005.

〔8〕 Alien Technology, “Reader Interface Guide ALR-9780 ALR-8780 ALR-9640 Doc. Control #8101938-000 Rev D,” *Alien Technology Corporation*, Nov 2004.

〔9〕 *EPCglobal*, “EPCglobal Draft Protocol Specification for a 900 MHz Class 0 Radio Frequency Identification Tag,” February 2003.

〔10〕 Free Software Foundation, “The GNU C Library, Edition 0.10, Last Updated 2001-07-06, of The GNU C Library Reference Manual, for Version 2.3.x.,” *Free Software Foundation, Inc.*

박 현 성 (Hyun-Sung Park) 정회원

1999년 2월 경성대학교 산업공학과 졸업
1999년 3월~2000년 7월 (주)인광정보통신
2001년 8월~2002년 8월 비트컴퓨터 Network 전문가 과정
2002년 10월~2004년 11월 (주)퓨전소프트 CDMA 개발팀
2005년 9월~현재 부산대학교 컴퓨터공학과 석사과정
<관심분야> 이동통신, RFID, BcN

김 종 덕 (Jong-Deok Kim) 정회원

1994년 2월 서울대학교 계산통계학과 졸업
1996년 2월 서울대학교 전산과학과 석사
2003년 2월 서울대학교 컴퓨터공학과 박사
2004년 2월~현재 부산대학교 정보컴퓨터공학과, 조교수
부산대학교 컴퓨터 및 정보통신연구소 정보기술연구원
<관심분야> 무선통신, 이동통신망, RFID/USN