멀티 비트 트리 비트맵 기반 패킷 분류

정회원 최 병 철*, 이 정 태**

A Multibit Tree Bitmap based Packet Classification

Byeong-cheol Choi*, Jeong-tae Lee** *Regular Members*

요 약

패킷 분류는 인터넷 망에서 QoS(Quality of Service)보장, VPN(Virtual Private Network)등과 같은 사용자들의 다양한 서비스를 수용하기 위한 중요한 요소이다. 패킷 분류는 기본적으로 IP(Internet Protocol) 패킷 헤더 내의 목적지 주소뿐만 아니라 발신지 주소, 프로토콜, TCP(Transmission Control Protocol)포트 번호 등 여러 필드들을 조합하여 룰 테이블로부터 best matching 룰을 찾는 것이다. 본 논문에서는 멀티 비트 트라이 구조의 트리 비트맵을 이용하여 하드웨어적인 룰 검색이 가능한 패킷 분류 기법을 제안한다. 검색 대상 필드 및 패킷 분류 룰을 구성하는 프레픽스를 비교 단위가 되는 일정한 비트 크기의 멀티 비트로 나누고, 이와 같이 구분된 멀티 비트 단위로 트리 비트맵 기반의 룰 검색 기능을 수행한다. 제안한 기법은 프레픽스의 일정한 상위 비트들에 대해서는 인덱싱 키로 사용하여 룰 검색을 위한 메모리 액세스 횟수를 줄이도록 하였다. 또한 룰 검색시 성능 저하를 초래하는 백트랙킹이 발생하지 않도록 하기 위하여 룰 테이블 구축시 마커 프레픽스에 대한 처리 기법을 제안하였다. 그리고 본 논문에서는 IPMA(Internet Performance Measurement Analysis) 프로젝트에서 제공하는 라우팅 테이블의 프레픽스들을 이용하여 2차원 즉, 목적지 주소와 발신지 주소의 2필드로 구성되는 랜덤 룰 셋을 생성하고 제안한 기법에 대한 메모리 소요량 및 성능 비교를 하였다.

Key Words : packet classification; multi-field; tree bitmap; rule table.

ABSTRACT

Packet classification is an important factor to support various services such as QoS guarantee and VPN for users in Internet. Packet classification is a searching process for best matching rule on rule tables by employing multi-field such as source address, protocol, and port number as well as destination address in IP header. In this paper, we propose hardware based packet classification algorithm by employing tree bitmap of multi-bit trie. We divided prefixes of searching fields and rule into multi-bit stride, and perform a rule searching with multi-bit of fixed size. The proposed scheme can reduce the access times taking for rule search by employing indexing key in a fixed size of upper bits of rule prefixes. We also employ a marker prefixes in order to remove backtracking during searching a rule. In this paper, we generate two dimensional random rule set of source address and destination address using routing tables provided by IPMA Project, and compare its memory usages and performance.

Ⅰ. 서 론

전통적으로 IP 패킷에 대한 라우팅은 IP 헤더내의 목적지 주소(Destination Address)를 기반으로 하여 다음 홉으로 향하는 링크를 결정하고 패킷을 전달하는 것과 관련되어 있다. 목적지 주소 기반 패킷 전달은 동일한 목적지 주소를 가진 모든 패킷에 대해서는 동일하게 취급하는 것이다. 그러나 오늘날 사용자들의 차별화된 서비스에 대한 요구가 점점 증가함에 따라 목적지 주소만으로는 그 요구를 수용하기가 어렵게 되었다. 따라서 이와 같은 차별화된 서비스를 수용하기 위하여 IP 패킷에 대한 패킷

* 한국전자통신연구원 광가입자망연구그룹(bcchoi@etri.re.kr), ** 부산대학교 컴퓨터공학과
논문번호 : 040032-0119, 접수일자 : 2004년 1월 19일

분류가 필요하게 되었으며, 이것은 IP 패킷 헤더내의 목적지 주소뿐만 아니라 발신지 주소(Source Address), TCP 발신지 포트 번호 및 목적지 포트 번호, 프로토콜 등 여러 필드들을 조합하여 결정하게 된다. 패킷 분류는 여러 가지 필드들을 기반으로 하여 해당 패킷에 대하여 가장 적합한 룰을 찾아내는 것이다. 패킷 분류의 일반적으로 L4+ 스위칭이라고도 하며 응용으로는 방화벽을 위한 패킷 필터링, MPLS(Multi-Protocol Label Switching) 터널링을 위한 플로우 병합, QoS 라우팅, 플로우 보존 로드 밸런스 등이 있다.

본 논문에서는 IP 헤더의 멀티 필드를 이용한 패킷 분류를 위하여 멀티 비트 트리 비트맵 기반의 룰 검색 알고리즘을 제안한다. 목적지 주소만을 이용한 트리 비트맵 기반의 룩업 알고리즘은 [1]에 소개되어 있으며, 이 기법은 단일 비트 트라이 검색 알고리즘의 성능 향상과 하드웨어적인 기반의 검색이 가능하도록 제안된 것이다. 본 논문에서는 멀티 필드 중 필드 길이 및 검색 복잡도가 상대적으로 높은 목적지 주소와 발신지 주소의 2 필드로 구성되는 2차원 룰에 대하여 중점적으로 논의한다. 제안하는 패킷 분류 기법은 32 비트로 구성되는 목적지 주소와 발신지 주소의 각 프레픽스에 대하여 상위 특정 비트크기에 대해서는 서로 조합하여 인덱싱을 위한 키로 사용하고 나머지 비트들에 대해서는 일정한 비트 크기로 나누어 목적지 프레픽스와 발신지 프레픽스를 서로 교차하면서 계층적으로 비교하여 해당하는 룰을 찾아내는 기법이다. 이와 같이 제안하는 기법은 하드웨어적인 구현이 용이하고 룰 검색 횟수를 줄이므로 성능을 향상시킬 수 있다.

본 논문의 구성은 II장에서 기존의 패킷 분류 알고리즘들의 장단점을 알아보고, III장에서는 멀티 비트 트리 비트맵 기법에 관하여 살펴본다. IV장에서는 멀티 비트 트리 비트맵을 이용한 패킷 분류 알고리즘을 제안하고, V장에서는 랜덤 룰 셋을 발생시켜 메모리 소요량과 룰 검색 성능을 알아본다. 마지막 VI장에서 결론을 맺는다.

II. 관련 연구

최근에 인터넷 사용자들의 차별화된 요구를 수용하기 위한 멀티 필드 패킷 분류를 위한 룰 테이블 구조 및 룰 검색 기법이 많이 제안되었다.

선형 검색 기법은 패킷 분류를 위한 가장 간단한 방법으로 모든 룰에 대하여 선형적으로 순차적 방법으로 검색을 수행하는 것이다. 이 방법은 $O(N)$의 메모리 공간과 $O(N)$의 룰 검색 시간을 요구하므로서 룰 수가 많은 경우에는 수용하기 힘들다. 여기서 N은 룰 수를 나타내고 있다. 캐쉬 기반의 패킷 분류 기법은 선형 검색 기법의 성능 향상을 위하여 동일한 패킷 헤더를 가지고 있는 플로우에 대하여 해당 룰들을 캐쉬 테이블에 저장하는 것이다. 이와 같은 방법은 캐쉬 테이블이 플로우에 대한 룰을 충분히 수용할 경우에는 룰에 대한 캐쉬 히트로서 성능 향상이 가능하나 그렇지 않을 경우에는 캐쉬 테이블에 대하여 룰 검색 미스가 발생하므로 성능의 급격한 저하를 초래하는 단점이 있다[2].

하드웨어 기반 패킷 분류 기능을 제공하는 TCAM (Ternary Contents Addressable Memory)은 룰 테이블을 구성하고 있는 룰 엔트리에 대하여 동시에 비교가 가능하게 함으로서 O(1)의 빠른 검색 시간을 제공하여 룰 검색을 효과적으로 수행할 수 있다. 그러나 CAM 방식은 룰을 구성하는 필드 수가 많을 경우 모든 필드들을 수용할 수 있도록 충분한 비트 크기를 제공하기 어려운 경우가 있다. 또한 현재까지의 기술로 CAM이 수용할 수 있는 룰 엔트리 용량에 한계가 있으며 가격이 비싸고 전력 소모가 많은 단점을 가지고 있다[3,4]. 또한 Laksman에 의해 제안된 다차원 range matching 기반 기법은 wide memory 액세스를 요구한다. 이 기법은 N개의 룰 각각이 d 필드로 구성된 경우 미리 계산된 N 비트의 비트맵을 이용하여 best matching 프레픽스를 계산한다[5]. 따라서 이 기법은 각 필드의 best matching 프레픽스에 해당하는 dN 비트 크기를 메모리로부터 읽고 상호 교차 영역을 구하여 best matching 룰을 찾는다. 이와 같은 기법의 메모리 요구는 $O(N^2)$이며 dN 비트 크기를 읽어야 한다. 이와 같은 하드웨어 기반의 룰 검색 기법은 하드웨어 비용이 매우 비싸다는 것과 융통성과 확장성에 제한이 있다는 단점을 가진다.

Srinivasan에 의하여 제안된 트라이 기반의 Grid-of Trie 검색 기법은 2차원 필드로 구성된 룰에 대하여 적용 가능하며 W를 룰을 구성하는 하나의 필드가 가지는 프레픽스 비트 길이로 정의할 때, $O(NW)$의 메모리 요구량과 $2W$-1의 룰 검색 시간을 요구하고 있다. 그리고 또 다른 2차원 룰 검색 기법으로 [6]에서 제안된 Cross-Product기법은 각 필드에 대한 best matching 검색을 수행하고 이 결과를 조합한 선행 계산 테이블을 이용하여 해당하는 룰을 찾아낸다. 이 방법은 d 필드 룰에 대해서는

O(dN)의 많은 메모리량이 요구된다. 한편 Gupta에 의해 제안된 RFC(Recursive Flow Classification)기법은 best matching 검색을 수행한 후 계층적 구조에 따라 반복적인 방법으로 Cross-Product 기법의 검색을 수행함으로서 일반화된 Cross-Product기법으로 간주할 수 있다[7]. 이 기법은 d 필드로 구성된 룰에 대하여 worst case에 대해서는 O(N^d)의 많은 메모리량을 요구하고 있다. 그리고 튜플 공간 검색 기반의 Rectangle 검색을 이용한 2차원 룰 검색 기법은 2W-1번의 헤쉬 기능을 요구하고 있다[8].

Ⅲ. 트리 비트맵 기법

본 장에서는 본 논문에서 제안하는 멀티 비트 트리 기반 패킷 분류 알고리즘의 기반이 되는 트리 비트맵 구조에 대하여 논의한다.

트리 비트맵 검색 기법은 효율적인 메모리 사용으로 IP 포워딩을 위한 룩업 기능을 수행하는 멀티 비트 트라이 데이터 구조를 적용하는 하드웨어 기반의 알고리즘이다. CIDR(Classless Inter Domain Routing)의 사용으로 IP 패킷 포워딩을 위한 룩업은 주어진 32비트 IPv4 목적지 주소에 대하여 포워딩 테이블에 저장되어 있는 가장 길게 매치되는 LPM (Longest Prefix Match) 프레픽스를 찾아내어 관련된 포워딩 정보를 검색하는 것이다[9]. 표 1은 16개의 프레픽스 엔트리의 한 예를 나타내고 있으며, 그림 1은 표 1의 엔트리에 대한 이진 트리 구조를 나타내고 있다. LPM 룩업은 IP 주소의 MSB(Most Significant Bit)를 시작으로 저장된 프레픽스 엔트리들과 한 비트씩 비교한다. 수신한 패킷의 IP 주소가 129.254.73.30 (10000001 11111110 01001001 00011110_2)에 대한 테이블 검색 결과로서 *, 1*, 1000*, 100000011* 등 네 개의 프레픽스 엔트리가 매치되며, 위 검색 결과의 프레픽스 중에 네번째 프레픽스인 P14가 가장 길게 매치되므로 해당하는 다음 홉 경로 정보는 포트 5임을 알 수 있다. 이와 같은 룩업 결과로서 얻어진 포워딩 정보를 이용하여 패킷은 헤더 변경 과정을 거쳐 지정된 다음 홉으로 전달된다.

트리 비트맵 알고리즘은 이와 같은 룩업 기능을 하드웨어 기반으로 효과적으로 수행하기 위하여 이진 트라이 구조로 프레픽스들을 저장하는 것으로 시작한다[10]. 이진 트라이 구조에서 검색은 IP 주소의 MSB로부터 한 비트씩 트라이를 따라 수행된다. 이와 같은 검색 과정의 속도 향상을 위하여 IP 주소의 다수개의 비트를 한번에 비교하는 기법이 제안되었다[11].

표 1. 프레픽스 엔트리 예

Pi	Prefix	Next hop	Pi	Prefix	Next Hop
P1	*	1	P9	0101 0*	5
P2	1*	2	P10	1000 1*	7
P3	010*	5	P11	0001 10	5
P4	011*	6	P12	1100 10*	1
P5	101*	4	P13	1100 110*	10
P6	1000*	1	P14	1000 0001 1*	5
P7	1111*	3	P15	1000 0111 1*	7
P8	0001 0*	9	P16	1000 0001 01*	11

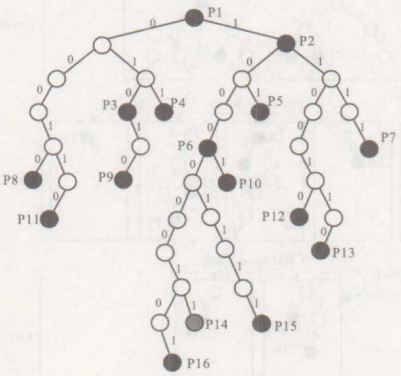


그림 1. 표 1에 대한 프레픽스의 이진 트라이 구조

멀티 비트 트라이 구조는 이진 트라이 구조의 상위비트로부터 일정한 멀티 비트를 하나의 노드 그룹으로 묶고 연속되는 프레픽스에 대해서는 다음 하위 레벨에서 다시 멀티 비트를 노드 그룹으로 처리하는 구조이다. 이와 같은 구조는 룩업을 수행함에 있어 한 비트씩 비교하는 구조에 비하여 검색에 소요되는 메모리 액세스 수를 줄일 수 있다. 하나의 멀티 비트 트라이 노드를 구성하는 서브트리의 깊이를 stride라 정의하며, 그림 2는 표 1에 주어진 프레픽스들에 대하여 이진 트라이를 기반으로 4비트 stride를 사용하여 변경된 멀티 비트 트라이 구조를 나타내고 있다. 이와 같은 경우 목적지 주소 4비트 단위로 멀티 비트 트라이를 따라 검색에 이용된다. 즉, 예에서 목적지 주소 상위 4비트 1000_2은 루트 노드에서 비교를 위하여 사용되고 다음 4비트 0001_2은 다음 하위 레벨 2의 노드에 비교를 위하여 사용된다. 이와 같이 4비트씩 하나의 그룹으로 하위 레벨로 룩업을 연속적으로 수행한다.

멀티 비트 트라이 구조에서 하위 레벨의 노드들은 상위 레벨의 노드로부터 하위 노드 어레이 포인터를 이용하여 해당 포인터를 알 수 있다. 임의의 노드로부터 하위 레벨 노드로의 확장은 4비트씩의 노드 그룹화를 수행할 경우 프레픽스가 계속되는 엔트리에 대하여 하위 노드 포인터를 이용하여 프레픽스가 종료되는 레벨까지 생성된다.

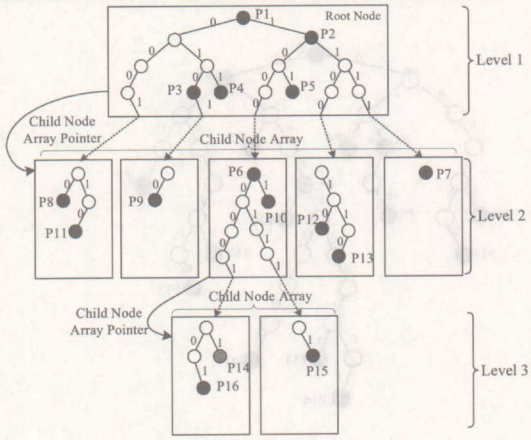


그림 2. 4비트 stride를 사용한 멀티 비트 트라이 구조

트리 비트맵 알고리즘은 멀티 비트 트라이로 구성된 각 노드와 관련된 프레픽스 정보를 비트맵을 사용하여 코드화 한다. 비트맵은 IPB(Internal Prefix Bitmap)와 EPB(Extending Paths Bitmap)로 구분할 수 있다. IPB는 내부 프레픽스 비트맵으로 멀티 비트 노드의 이진 서브트리에서 저장된 프레픽스 정보를 나타내고, EPB는 확장 경로 비트맵으로서 계속되는 프레픽스 비트의 유무에 대한 정보를 나타낸다. 각 노드는 IPB, EPB외에 다음 홉 테이블 포인터와 하위 노드 어레이 포인터 정보를 가진다. 이와 같은 정보를 이용하여 특정 상위 노드에 속한 하위 노드들은 어레이를 구성하여 연속적으로 저장되도록 한다. 이와 같은 구조는 임의의 상위 노드에 속한 하위 노드들의 저장 위치를 하위 노드 어레이를 구성하는 첫 노드를 나타내는 어레이 노드 포인터와 EPB로부터 계산된 옵셋을 이용하여 계산할 수 있도록 한다. 이때 하위 노드에 대한 옵셋은 EPB의 비트맵에서 해당 프레픽스 위치까지의 '1'의 개수로서 계산된다. 그림 3은 그림 2에서 레벨 1의 루트 노드에 대한 IPB와 EPB의 비트맵 코드화 하는 예를 나타내고 있다.

또한 경로를 나타내는 다음 홉 정보를 알기 위해서는 해당 노드가 가지는 다음 홉 테이블 포인터와 IPB의 해당 프레픽스까지의 위치 옵셋을 이용한다. 이때 옵셋은 해당 IPB 비트맵에서 해당 프레픽스 위치까지의 '1'의 개수로서 계산된다.

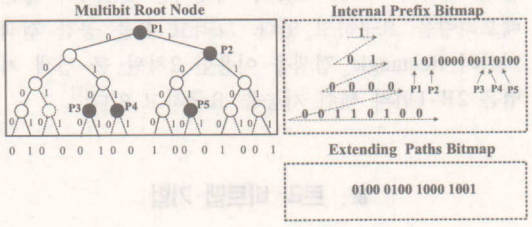


그림 3. 멀티 비트 트라이 노드의 비트맵 코드화

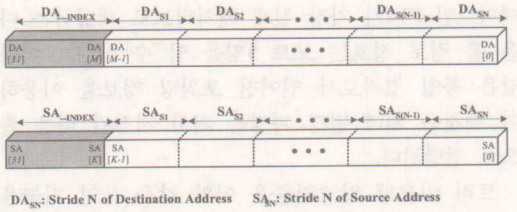
Ⅳ. 멀티 비트 트리 비트맵 기반 패킷 분류 기법

본 장에서는 IP 헤더의 멀티 필드를 이용하여 패킷 분류를 위한 새로운 룰 검색 기법을 제안한다.

1. 룰 검색을 위한 주소 분해

멀티 비트 트리 비트맵에 기반한 룰 검색을 수행하기 위하여 룰을 구성하고 있는 프레픽스들은 일정한 비트 크기의 고정 길이로 나누어진다. 이와 같은 고정 길이는 한번에 수행되는 검색 단위로서 stride로 나타낼 수 있다.

IP 헤더의 (DA, SA)의 두 튜플로 구성되는 룰 R은 목적지 주소 DA={DA_{INDEX}, DA_{S1}, DA_{S2}, ..., DA_{SN}}, 발신지 주소 SA={SA_{INDEX}, SA_{S1}, SA_{S2}, ..., SA_{SN}}등으로 각각 분해된 주소 구조를 가진다. 그림 4는 DA, SA에 대하여 멀티 비트 트리 룰 검색을 위한 인덱스 영역과 일정한 길이의 stride로 분해된 주소 구조를 나타내고 있다.



DA_{SN}: Stride N of Destination Address SA_{SN}: Stride N of Source Address

그림 4. 4비트 stride를 가지는 멀티 비트 비트맵 트리를 위한 주소 분해 구조

분해된 주소 구조에서 각 주소의 DA_{INDEX}와 SA_{INDEX} 영역은 인덱싱을 위한 키로 사용되고 DA_{S1}

.. DA_{SN}과 SA_{S1} .. SA_{SN}은 멀티 비트 트리 비트맵 기반의 룰 검색을 위하여 사용된다.

그림 5는 DA, SA로 구성되는 2차원 멀티 필드 패킷 분류를 위한 룰 검색 구조를 나타내고 있다. 수신한 패킷 Pi의 $DA(i)_{INDEX}$와 $SA(i)_{INDEX}$ 영역은 서로 조합하여 멀티 비트 트리 비트맵 룰 테이블 검색을 위하여 해당하는 루트 노드의 포인터를 획득한다. 이와 같은 인덱싱 기법의 사용은 멀티 비트 트라이 구조로만 구성되는 것보다 룰 검색에 소요되는 검색 시간을 줄일 수 있다.

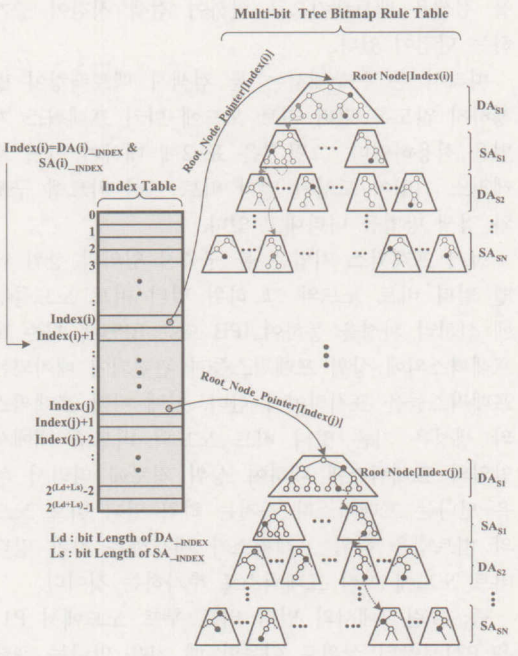


그림 5. 멀티 필드 패킷 분류를 위한 룰 검색 구조

예를 들어 DA, SA주소의 상위 8비트씩을 인덱스 영역으로 사용하고 나머지를 4 비트 stride를 사용하는 것은 인덱스 영역이 없이 4비트 stride로 구성되는 구조의 첫 4번 액세스가 한번의 인덱싱과 동일하다는 것을 알 수 있다. 이와 같이 인덱싱을 통하여 DA_{S1}의 루트 노드 포인터를 획득하고 루트 노드에 저장되어 있는 룰 프레픽스 정보인 IPB, EPB 정보를 페치한다. 그리고 수신한 패킷의 DA_{S1}에 해당하는 멀티 비트와 프레픽스 검색 기능을 수행하여 연속되는 다음 SA_{S1} 노드 포인터를 획득한다. SA_{S1}의 노드 포인터로부터 룰 프레픽스 정보인 IPB, EPB 프레픽스 비트맵 정보를 페치하여 수신한 패킷의 SA_{S1}에 해당하는 멀티 비트를 비교하여 연속되는 DA_{S2} 노드 포인터를 획득한다. 이와 같이 본 논문에서 제안하는 패킷 분류 기법은 수신 패킷의 DA와 SA 수신 주소와 멀티 비트 노드에 구축되어 있는 IPB, EPB의 룰 정보의 비교 검색 기능을 프레픽스가 더 이상 연속되지 않을 때까지 서로 교차하면서 상위 멀티 비트 노드 그룹부터 하위 멀티 비트 노드 그룹으로 수행하면서 best matching 룰을 찾아내는 것이다.

2. 룰 검색

멀티 비트 트리 비트맵 기반 룰 검색은 해당 멀티 비트 노드의 LPM를 선택하여 다음 하위 레벨 노드 포인터로 이동하여 멀티 비트 노드의 LPM을 찾아내는 방식이다. 이때 하위 레벨 노드에서 멀티 비트 비트맵 검색시 매치되는 경우가 존재하지 않을 수 있다. 이와 같은 경우에는 다시 상위 레벨 노드로 되돌아와서 LPM이 아닌 덜 매치되는 프레픽스가 지정하는 다음 노드 포인터로 이동하여 비트맵을 비교하여야 하는 백트랙킹이 발생한다. 백트랙킹이 발생하는 경우 룰 테이블 구축은 간단하지만 검색시 상위 레벨 노드로 되돌아가서 다시 하위 레벨 노드를 검색하여야 하기 때문에 부가적인 검색 시간이 요구된다.

표 2는 8비트 길이의 DA와 SA 프레픽스로 구성되는 2차원 룰의 예를 나타내고 있으며, 그림 6은 표 2에 있는 룰에 대하여 백트랙킹을 가지는 구조에 대한 구축된 룰과 검색의 한 예를 나타내고 있다. 이와 같은 경우 룰 테이블 구축은 4 비트 stride 별로 DA와 SA를 서로 교차하면서 구축하며 하나의 프레픽스 stride를 저장하는 멀티 비트 노드는 다음 프레픽스 stride가 저장되는 포인터 정보와 IPB, EPB 비트맵 정보를 저장한다. 그림 6에서 하위 레벨 해당 멀티 비트 노드로의 포인터 표시는 화살표로 나타내고 있다.

표 2. {DA,SA} 프레픽스로 구성되는 2차원 룰의 예

Rule	DA/SA	Flow	Next Hop
R1	0*** ****/11** **	100	1
R2	100* ****/10** **	101	3
R3	100* ****/1110 01	215	2
R4	0100 10**/1*** **	15	1
R5	0100 000*/00** **	4	5

그림 6에 구축된 2차원 룰 셋에 대하여 레벨 L1인 멀티 비트 루트 노드에 저장되어 있는 IPB=(0 01 0000 00001000)이고 EPB=(0000 1000 0000 0000)로 됨을 알 수 있다. 이때 수신한 패킷 P의 IP 헤더내의 주소 8비트 P=(DA,SA)=(1000 0101, 1100 1011)이면 수신 패킷 DA의 첫 stride에 해당하는 멀티 비트 1000 검색에 대하여 IPB 비트맵 (0 01 0000 00001000)에서 밑줄로 표시된 2개의 프레픽스가 매치되고 EPB 비트맵에 대해서는 매치되는 프레픽스가 없음을 알 수 있다. 따라서 첫 멀티 비트 1000 대하여 매치된 2개의 프레픽스 비트 중 우측에 위치한 '1'이 가장 길게 매치되는 LPM 프레픽스에 해당된다.

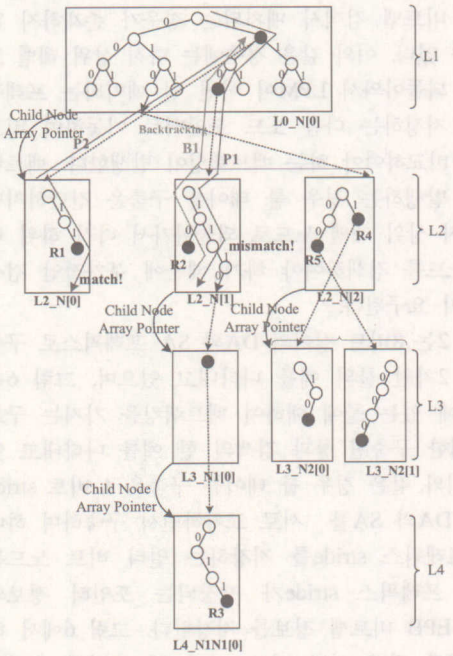


그림 6. 백트랙킹(backtracking)이 발생되는 경우 검색 방법

이때 다음 하위 레벨 L2에서 검색할 노드는 두번째 멀티 비트 노드인 L2_N[1]이 된다. L2_N[1]에서 수신 패킷 필드 SA의 첫 4비트 stride에 해당하는 1100에 대한 LPM 프레픽스 검색 수행시 IPB와 EPB 비트맵에서 매치되는 프레픽스가 존재하지 않는다. 이와 같이 미스매치가 발생되는 경우 상위 레벨 멀티 비트 노드인 레벨 L1 멀티 비트 루트 노드에서 LPM 프레픽스인 100* 보다 덜 매치되는 1*** 프레픽스로 검색의 백트랙킹이 발생한다. 1*** 프레픽스는 다음 하위 멀티 비트 노드로 L2_N[0]를 가리키고 있으며, 이 노드에서도 수신한 SA 첫 4비트에 해당하는 1100에 매치되는 프레픽스로 11**이 존재한다.

이와 같은 검색은 검색 경로에 따른 DA와 SA stride에 대한 EPB 비교 결과가 모두 0으로 될 때 검색은 종료되고 해당하는 룰 정보를 페치한다. 즉, P1 검색 경로에 따른 미스매치 발생으로 B1의 백트랙킹이 발생하고 P2의 검색 경로에서 매치된 룰을 찾아낸 것이다.

이상에서 살펴본 바와 같이 백트랙킹이 발생하는 룰 검색은 백트랙킹으로 인하여 검색 시간이 증가하는 단점이 있다.

따라서 본 논문에서는 룰 검색시 백트랙킹이 발생하지 않도록 멀티 비트 노드에 마커 프레픽스 기법을 적용하였다. 그림 7은 표 2에 대하여 마커 프레픽스 기법이 도입된 멀티 비트 트리 비트맵 구조와 검색 방법을 나타내고 있다.

마커 프레픽스 기법은 룰 구축시 임의의 상위 레벨 멀티 비트 노드와 그 하위 멀티 비트 노드들간에 선처리 과정을 통하여 IPB 또는 EPB에 기존 룰 프레픽스외에 상위 프레픽스들과 관련되어 매치되는 프레픽스들을 표시하여야 한다. 이때 마커 프레픽스의 생성은 기존 멀티 비트 노드의 비트맵 내에서 임의의 프레픽스에 대하여 상위 경로에 따라서 처음 만나는 프레픽스가 가지는 하위 멀티 비트 노드의 비트맵을 하위 프레픽스가 지정하는 하위 멀티 비트 노드에 마커 프레픽스로 추가하는 것이다.

즉, 그림 7에서의 멀티 비트 루트 노드에서 P11은 P21로부터 상위로 이동할 때 처음 만나는 프레픽스로 P21이 지정하는 하위의 멀티 비트 노드에 P11이 지정하는 하위 멀티 비트 노드의 프레픽스를 마커 프레픽스 M11로 추가한 것을 나타내고 있다.

이와 같이 마커 프레픽스 기반의 룰이 구축되어 있는 경우 레벨 L1인 멀티 비트 루트 노드에 저장되어 있는 IPB=(0 01 0000 00001000)이고 EPB=(0000 1000 0000 0000)이다. 이때 수신한 패킷 P의 IP 헤더내의 주소 8비트 P=(DA,SA)=(1000 0101, 1100 1011)이면 수신 패킷의 DA의 첫 stride에 해당하는 멀티 비트 1000의 대하여 IPB의 비교 결과 비트맵 (0 01 0000 00001000)에서 밑줄로 표시된 2개의 프레픽스가 매치되고 EPB 비트맵은 모두 0이 된다. 따라서 첫 멀티 비트 1000 대하여 가장 길게 매치되는 LPM 프레픽스는 IPB 비교 결과 비트맵에서 우측에 위치한 '1'인 100* 프레픽스가

이에 해당된다.

이때 다음 하위 레벨인 레벨 L2에서 검색할 노드는 두번째 멀티 비트 노드인 L2_N[1]가 된다. 이 멀티 비트 노드의 IPB=(0 00 0011 00000000), EPB=(0000 0000 0000 0010)이며 수신 패킷 필드 SA의 첫 4비트 stride에 해당하는 1100에 대한 IPB 비교 결과 비트맵은 (0 00 0001 00000000)를 가진다. 따라서 IPB에서 매치되는 프레픽스가 존재함을 알 수 있으며 이것은 마커 프레픽스의 생성으로 인한 것이며, 마커 프레픽스가 지정하는 하위 멀티 비트 노드는 비트맵 정보가 아닌 *switch pointer*를 저장하고 있다. 이 *switch pointer*는 순방향의 검색이 계속되도록 다음 노드를 지정하는 역할을 한다. 그림 7에서는 *switch pointer*가 지정하는 노드로부터 R1에 대한 룰 정보를 페치하여 룰 검색을 종료한다. 이와 같이 마커 프레픽스를 도입한 경우 백트랙킹으로 인하여 발생되는 검색 시간 증가를 없앨 수 있다.

그림 8은 백트랙킹이 발생하지 않는 검색 알고리즘을 나타내고 있다.

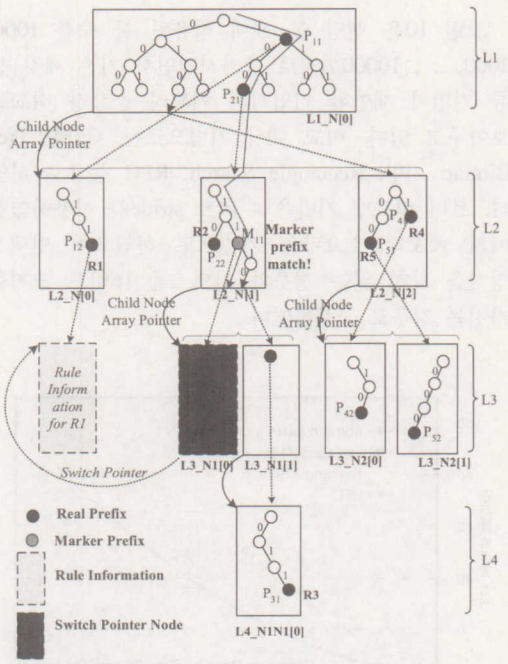


그림 7. 백트랙킹이 없는 경우 노드 비트맵 및 룰 검색 구조

```
Rule Search

get DA and SA of packet P
Index = DA_index & SA_index;
if (IndexData.attribute = null)
   get RuleInformationPointer
else
   get MutibitRootNodePointer
   PrefixFlag[i,1]=1;
   PrefixFlag[i,2]=1;
  for(i=1; i<N; i++)
  {
    for(j=1;j=<2;j++)  /* 1: DA stride node, 2:SA stride node */
    {
          get MutibitNodeInformation(NextNodePointer)
          if(NextNode.type==Marker)
              get SwitchPointer for Prefix Node[i,j]
          else
              perform treeFunction(node.IPB, node.EPB, DA_stride[i,j])
              if(node.EPB(DA_stride[i,j]) == null)
                    PrefixFlag[i,j] = 0;
              else
                    NextNodePointer=node.NextPointer+
                                        CountOnes(IPB, EPB)
              end if;
          end if;
    }
    if (PrefixFlag[i,1]==PrefixFlag[i,2]==0)
    {
          RuleInformationPointer=node.NextPointer+
                                        CountOnes(IPB, EPB)
          break;
    }
end if;
return(RuleInformationPointer);
```

그림 8. 패킷 분류 룰 검색 알고리즘

V. 메모리 소요 및 검색 성능 분석

본 논문에서 제안하는 패킷 분류를 위한 룰을 구성하는 테이블은 인덱스 테이블과 프레픽스에 대한 비트맵 정보 및 다음 stride 노드 배열 포인터를 저장하는 멀티 비트 트리 비트맵 정보 테이블로 구성된다.

1. 인덱스 테이블

인덱스 테이블은 룰을 구성하는 DA와 SA의 상위 비트에 속하는 각 인덱스 영역 DA_{INDEX}와 SA_{INDEX} 영역이 서로 결합된 주소를 인덱스 테이블 주소로 사용하여 룰 검색에 대한 속성 및 검색을 위한 멀티 비트 루트 노드를 찾아내도록 포인터를 저장하고 있다.

```
struct index_node
{
      attribute; 2
      multibit root node pointer (or rule information
                                        pointer);16
}
```

attribute는 룰의 DA_{INDEX}, SA_{INDEX}가 결합된 인덱스가 가지는 속성, 즉 룰 구성 프레픽스 길이가 인덱

스 길이 이하인 경우와 그렇지 않은 경우를 나타내고 있다. 예를 들어, DA_{INDEX}, SA_{INDEX}가 각각 8비트로 구성되는 경우, 룰을 구성하는 각 프레픽스가 4비트, 8비트인 룰 인덱스에 해당하는 주소 영역은 멀티 비트 트리 비트맵 정보 테이블에 대한 프레픽스 검색이 불필요하다. 이와 같은 경우에는 인덱스 테이블 자체에서 해당 룰 정보가 저장된 포인터를 나타내고 있다. 그렇지 않은 경우 비교 검색을 수행해야하는 다음 stride에 대한 멀티 비트 루트 노드 포인터를 나타내고 있다. 따라서 DA_{INDEX}, SA_{INDEX}의 조합으로 $2^{(DAINDEX+SAINDEX)}$*18bits에 해당하는 인덱스 테이블을 위한 메모리 용량이 필요하다.

2. 멀티 비트 트리 비트맵 정보 테이블

멀티 비트 트리 비트맵 정보 테이블은 한번에 비교되는 멀티 비트의 비트 수에 해당하는 일정한 stride별로 해당 노드의 속성, DA와 SA 프레픽스에 대한 IPB 비트맵 정보와 EPB 비트맵 정보, 그리고 다음 비교 stride에 해당하는 하위 노드에 대한 포인터 정보, 룰 정보 포인터 또는 마커 프레픽스가 지정하는 switch pointer 등을 저장한다. 이때 룰 정보 포인터와 switch pointer를 저장할 경우 IPB, EPB는 null값을 가진다.

```
struct bitmap_node
{
        attribute;  3
        internal prefixes bitmap; 15
        extending paths bitmap; 16
        next child array node pointer(or rule
         information pointer, switch pointer); 18
}
```

3. 랜덤 룰 셋 생성 및 메모리 사용

룰 셋은 일반적으로 상업적인 보안 데이터로 간주되어 획득하기가 쉽지 않다. 따라서 본 논문에서는 IPMA 프로젝트에서 제공하는 라우팅 테이블을 이용하여 랜덤 룰 셋을 생성하여 분석 자료로 사용하였다[12]. 랜덤 룰 셋 생성은 라우팅 테이블을 구성하는 엔트리들에 대하여 두개의 랜덤 함수를 이용하여 하나는 목적지 주소에 대한 프레픽스를 생성하고 다른 하나는 발신지 주소에 대한 프레픽스를 생성하도록 하여 생성된 순서대로 서로 조합하여 룰 엔트리를 발생시켰다.

그림 9는 4비트 stride를 적용하여 10,000개의 룰을 생성하였을 때 (DA, SA)로 구성되는 룰에 대하여 stride 단위 길이의 프레픽스 조합인 Stride Length (DA, SA) 분포도를 나타내고 있다. 예를 들어 R=(DA,SA)=(129.254.73.*/24, 110.20.*.*/16)로 룰이 구성되는 경우 4비트 stride를 사용시 Stride 길이 (6,4)의 프레픽스 조합의 분포를 가짐을 알 수 있다. 랜덤 룰 셋의 프레픽스 길이별 분포를 보면 라우팅 테이블에서 24비트 유효 프레픽스를 가지는 주소가 많기 때문에 생성된 룰 분포도 (6,6)에서 현저하게 많은 룰 수가 존재함을 보이고 있다.

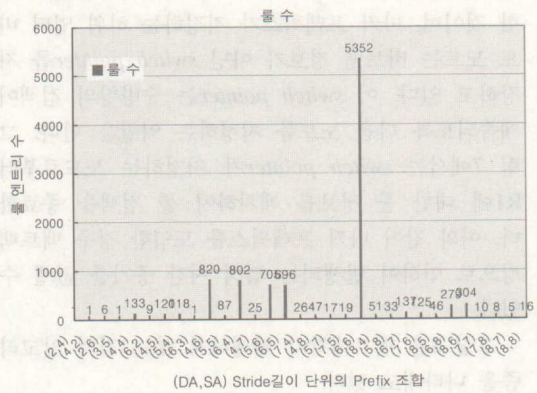


그림 9. 랜덤 룰 셋의 프레픽스 길이별 분포(4bit stride, 룰 수 : 10,000)

그림 10은 랜덤 룰 셋에 적용된 룰 수를 1000, 2000, .. , 10000까지로 증가시키면서 기존 패킷 분류 기법과 제안한 기법과의 메모리 소요량 비교를 보여주고 있다. 비교 대상 기법으로는 제안한 Tree Bitmap 기반, Rectangle Search, RFC 등으로 하였다. 트리 비트맵 기반은 4 비트 stride를 사용하였을 때의 메모리 소요량을 기준으로 하였으며 인덱스 영역을 갖지 않는 경우와 인덱스를 16비트 크기를 가지는 경우로 구분하였다.

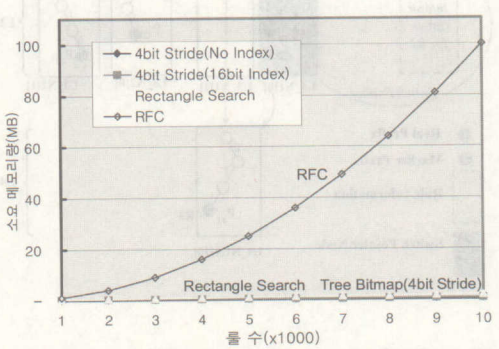


그림 10. 룰 수의 증가에 따라 룰 구축에 소요되는 메모리량

그림 10에서는 RFC 기법이 룰 수가 증가함에 따라 메모리 소요량이 현저하게 증가함을 보이고 있으며, 트리 비트맵 기반 검색과 Retangle 기법 검색은 미미하게 증가하고 있다.

그림 11에서는 메모리 소요량 스케일을 달리하여 Rectangle 검색과 트리 비트맵 기반 검색에서 인덱스 길이가 4 비트 stride를 적용한 경우 0, 4, 8, 12, 16 비트 인덱스 길이에 대하여 메모리 소요량을 보여주고 있다.

Rectangle 검색이 트리 비트맵 기반의 룰 검색 기법보다 룰 수가 증가함에 따라 메모리 증가 비율이 보다 큼을 알 수 있다. 또한 트리 비트맵 기반 검색 기법에서 인덱스 영역이 0, 4, 8, 12 비트인 경우에는 메모리 소요량이 인덱스 영역 크기에 무관하게 거의 비슷한 용량이 소요되나, 16 비트 크기인 경우에는 임의의 룰 수에 대하여 거의 120KB 정도의 인덱스 테이블 크기만큼 그 소요량이 많음을 알 수 있다.

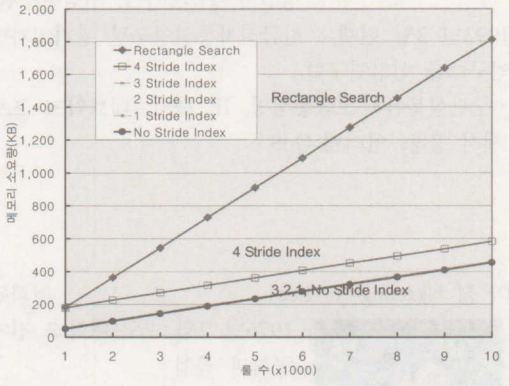


그림 11. 트리 비트맵 기반 검색 기법의 인덱스 영역 길이별 룰 수 증가에 따라 소요되는 메모리량

4. 룰 검색 성능

본 논문에서 제안한 룰 검색 기법은 수신한 IP 패킷의 헤더를 검출하여 룰 엔트리들이 저장된 룰 테이블의 비트맵들을 비교하고 미스매치 또는 best matching 룰을 찾을 때까지 계층적으로 수행하는 것이다. 따라서 멀티 비트 트리 비트맵 기반의 룰 검색 성능은 룰을 구성하는 프레픽스 길이, 즉 멀티 비트 노드의 검색 레벨에 따라 룰 테이블을 구성하는 메모리 액세스 횟수에 의존하며, 표3은 2차원으로 구성되는 룰에 대한 worst case 액세스 시간에 대한 비교를 보여주고 있다.

여기서 W는 룰을 구성하는 한 필드의 프레픽스 길이, L_{index}는 인덱싱을 위하여 사용될 키 길이, L_{stride}는 트리 비트맵 노드의 멀티 비트를 구성하는 단위 비트 크기, d는 필드 수를 각각 나타내고 있다. 표3으로부터 RFC 기법이 액세스 횟수는 적으나 룰 수가 많은 경우 메모리량이 기하급수적으로 많이 소요되는 단점이 있다. 따라서 멀티 비트 트리 비트맵 기반의 패킷 분류는 Rectangle 검색 기법보다 소요되는 메모리량과 액세스 횟수의 비교에서 모두 우수함을 보이고 있다.

표 3. Worst case 액세스 시간

룰 검색 기법	Worst case time complexity
Linear Search	N
Hierachical tries	W^d
Rectangle	$2W-1$
RFC	d
Mutibit tree bitmap	$\lceil (2W-L_{index})/L_{stride} \rceil + 1$

Ⅵ. 결 론

본 논문에서는 멀티 비트 트라이 기반의 트리 비트맵을 이용하여 하드웨어적인 룰 검색이 가능한 패킷 분류 기법을 제안하였다. 제안한 패킷 분류 기법은 검색 대상 필드 및 패킷 분류 룰을 구성하는 프레픽스를 비교 단위가 되는 멀티 비트로 나누고, 일정한 크기의 멀티 비트 단위로 트리 비트맵 기반의 룰 검색을 수행한다. 제안한 기법은 일정한 상위 비트들에 대해서는 인덱싱 키로 사용하여 검색을 위한 메모리 액세스 횟수를 줄이도록 하여 성능 향상을 꾀하였다. 하위 레벨 노드에 대한 포인터를 초기 배열 포인터에 대한 정보와 비트맵 정보로부터의 옵셋을 사용하므로서 메모리의 사용 효율을 향상시켰다. 또한 검색 성능 향상을 위하여 노드 내부의 비트맵에 마커 프레픽스 표시를 하므로서 백트랙킹이 발생하지 않도록 하였다.

본 논문에서는 기존 패킷 분류 기법들과 비교를 위하여 IPMA 프로젝트에서 제공하는 라우팅 테이블의 프레픽스들을 이용하여 2차원 즉, 목적지 주소와 발신지 주소의 2필드로 구성되는 랜덤 룰 셋을 생성하고, 제안한 기법에 대한 메모리 소요량 및 성능 비교를 하였다. 인덱싱 키를 16비트, 멀티 비트 단위를 4비트로 한 경우 2필드 룰 셋에 대한 worst case 검색은 메모리 액세스 횟수가 13회가 되며, 랜

덤 룰 셋에서의 메모리 소요량은 10,000개의 룰 엔트리 테이블 구축시 약 582KB 정도임을 알 수 있었다. 따라서 제안한 기법은 수 Gbps급의 차별화된 서비스 및 QoS를 보장하는 IP 패킷 포워딩을 위하여 활용 가능하다.

참 고 문 헌

[1] D. E. Taylor, J. W. Lockwood, T. S. Sproull, J. S. Turner, and D. B. Parlour, "Scalable IP lookup for programmable routers," Proc. of IEEE INFOCOM'02 pp. 562-571, 2002.

[2] P. Newman, G. Minshall, and L. Huston, "IP switching and gigabit routers," IEEE Communication Magazine, vol. 35, pp. 64-69, Jan. 1997.

[3] T. B. Pei and C. Zukowski, "Putting Routing Tables in Silicon," IEEE Network Magazine, pp. 42-50, Jan. 1992.

[4] A. McAuley and P. Francis, "Fast Routing Table lookup using CAMs," Proc. of IEEE INFOCOM'93, pp. 1382-1391, Mar. 1993.

[5] T. V. Laksman and D. Stidialis, "High speed policy-based packet forwarding using efficient multi-dimensional range matching," Proc. of SIGCOMM'98, pp. 203-214, Aug. 1998.

[6] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," Proc. of SIGCOMM'98, pp. 191-202, Aug. 1998.

[7] P. Gupta and N. McKeown, "Packet classification on multiple fields," Proc. of SIGCOMM'99, pp. 147-160, Aug. 1999.

[8] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," Proc. of SIGCOMM'99, pp. 135-146, Aug. 1999.

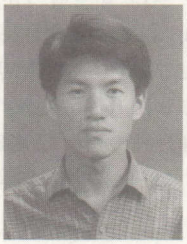
[9] S. Fuller, T. Li, J. Yu, and K. Varadhan, S. Keshav and Rosen Sharma, "Classless inter-domain routing(CIDR): an address assignment and aggregation strategy," RFC 1519, Sep. 1993.

[10] M. Walvogel, G. Varhghese, J. Turner, and B. Plattner, "Scalable High Speed IP Routing Lookups," Proc. of ACM SIGCOMM'97, pp. 25-36, Sep. 1997.

[11] V. Srinivasan and G. Varghese, "Fast IP Lookups using Controlled Prefix Expansion," Proc. of ACM SIGMATICS'98, pp.1-10, June 1998.

[12] Internet Routing Statistics http://nic.merit.edu/ipma/routing_table

최 병 철(Byeong-cheol Choi) 정회원



1987년 2월 : 한양대학교 전자공학과 졸업
1997년 8월 : 한남대학교 전자공학과 석사
1997년 9월~현재 : 부산대학교 컴퓨터공학과 박사 과정
1987년 3월 ~1993년 2월 : 삼성전자 통신연구소 주임연구원
1993년 2월~현재 : 한국전자통신연구원 광가입자망 연구그룹 선임연구원
<주관심분야> 고속통신망, IP 룩업, 네트워크 프로세서 응용, 인터넷 QoS

이 정 태(Jeong-tae Lee) 정회원

1976년 2월 : 부산대학교 전자공학과 졸업
1983년 8월 : 서울대학교 컴퓨터공학과 석사
1989년 2월 : 서울대학교 컴퓨터공학과 박사
1977년 12월~1985년 2월 : 한국전자통신연구소 선임연구원
1985년3월~1988년2월 :동아대학교 공과대학 조교수
1992년8월~1993년7월 :일본 NTT 연구소 초빙 연구원
1988년3월~현재 : 부산대학교 컴퓨터공학과 교수
<주관심분야> 고속TCP/IP, Mobile IP, IPsec, IPv6