Journal of The Institute of Electronics and Information Engineers Vol.52, NO.6, June 2015 http://dx.doi.org/10.5573/ieie.2015.52.6.021
ISSN 2287-5026(Print) / ISSN 2288-159X(Online)

논문 2015-52-6-4

인터넷 라우터에서의 패킷 분류를 위한 2차원 이진 검색 트리

(Two-dimensional Binary Search Tree for Packet Classification at Internet Routers)

이 고 은*, 임 혜 숙**

(Goeun Lee and Hyesook Lim©)

요 약

현재의 인터넷 사용자들은 실시간으로 다양한 멀티미디어 서비스를 제공 받길 원한다. 이에 네트워크 트래픽의 속도는 매우 빨라지고 있으며, 처리하여야 하는 데이터의 양은 해마다 기하급수적으로 증가하고 있다. 데이터는 '패킷'이라는 단위의 데이터 형식으로 전송되며, 패킷분류는 인터넷 라우터의 가장 어려운 기능 중 하나로 모든 패킷에 대하여 선속도로 처리되어야 한다. 다양한 패킷 분류 알고리즘 중, 영역분할 패킷분류 알고리즘은 5개의 패킷 헤더 필드 정보를 동시에 검색할 수 있는 효율적인 알고리즘이다. 영역 분할 사분 트라이는 가장 대표적인 영역분할 패킷분류 알고리즘으로 메모리 요구량이 적은 알고리즘이지만, 빠른 검색성능을 보장하지 못하는 단점이 있다. 본 논문에서는, 영역 분할 사분 트라이의 단점을 이진 검색 트리를 사용해 보완하는 새로운 알고리즘을 제안한다. 실험을 통하여 제안하는 알고리즘은 입력과 비교되는 룰의 수에 있어 영역 분할 사분 트라이 보다 검색 성능이 향상됨을 보았다.

Abstract

The Internet users want to get real-time services for various multi-media applications. Network traffic rate has been rapidly increased, and data amounts that the Internet has to carry have been exponentially increased. A packet is the basic unit in transferring data at the Internet, and packet classification is one of the most challenging functionalities that routers should perform at wire-speed. Among various known packet classification algorithms, area-based quad-trie (AQT) algorithm is one of the efficient algorithms which can lookup five header fields simultaneously. As a representative space decomposition algorithm, the AQT requires a small amount of memory in storing classification rules, but it does not provide high-speed classification performance. In this paper, we propose a new packet classification algorithm by applying a binary search for the codewords of the AQT to overcome the issue of the AQT. Throughout simulation, it is shown that the proposed algorithm provides a better performance than the AQT in the number of rule comparisons with each input packet.

Keywords : Packet classification; Area-based quad-trie; Binary search tree; Leaf-pushing AQT; 2D BST

Ⅰ. 서 론

현재의 인터넷 사용자들은 다양한 네트워크 응용 프로그램의 발전에 맞춰 실시간으로 향상된 서비스를 제공 받길 원하고 있으며, 다양한 형태의 기기에서 안정적인 인터넷 사용을 원하고 있다. 이에 발맞추어 네트워크 트래픽의 속도는 매우 빨라지고 있으며, 인터넷 네트워크 및 장비에서 처리하여야 하는 데이터의 양은

* 학생회원, ** 정회원, 이화여자대학교 전자공학과
(Dept. of Electronics Engineering, Ewha Womans University)
© Corresponding Author(E-mail: hlim@ewha.ac.kr)
※ 본 연구는 한국연구재단(NRF)의 중견연구자지원사업 도약과제(2014R1A2A1A11051762)와 정보통신산업진흥원의 대학 IT연구센터 지원사업(IITP-2015-H8501-15-1007)의 연구비 지원으로 수행하였습니다.
Received ; January 28, 2015 Revised ; May 18, 2015
Accepted ; May 31, 2015

해마다 기하급수적으로 증가하고 있다[1]. 인터넷 사용자들이 전송하는 데이터는 '패킷'이라는 단위의 데이터 형식으로 전송된다. 데이터 전송에 있어 품질 보장 서비스를 제공하기 위해 라우터에서는 패킷을 미리 정의된 플로우(flow)에 따라 분류하는 "패킷분류" 작업을 수행해야 한다[2].

라우터는 패킷 헤더의 여러 필드(field)에 대한 일치 여부를 판단함[3]으로 패킷 분류를 수행하게 된다. 패킷 분류에 사용되는 패킷 헤더는 목적지 IP 주소, 근원지 IP 주소, 목적지 포트 넘버, 근원지 포트 넘버, 프로토콜 정보로 구성된 5개의 필드를 포함한다[4]. 패킷 분류를 위한 필드 검색은 각 필드의 특성에 따라 IP 주소 필드는 프리픽스 일치(prefix match)[5], 포트넘버 필드는 영역 일치(range match), 프로토콜 타입 필드는 확정 일치(exact match)의 방법을 사용하여 일치 여부를 판단하게 된다. 이러한 검색을 통해 라우터에 미리 저장된 테이블의 룰들 중 다양한 필드 값이 모두 일치하는 룰들을 찾고, 그 중에서도 가장 높은 우선순위를 갖는 룰의 클래스에 따라 입력 패킷을 처리하는 과정을 "패킷 분류"라고 한다.

인터넷 라우터에 입력되는 패킷의 속도에 대응하는 빠른 검색 속도를 얻기 위해 여러 필드를 동시에 검색하는 패킷분류 알고리즘이 많이 연구되어 왔다[6]. 그 중 패킷 헤더의 필드 중 플로우의 다양성이 가장 큰 IP 주소 필드를 다차원 검색하는 방법이 효율적이라는 연구결과가 보고[7]된 바 있으며, 이에 근원지 IP 프리픽스와 목적지 IP 프리픽스를 결합한 코드워드를 사용하여 이차원 트리를 만들어 검색하는 영역 분할 사분 트라이(Area based Quad Trie, AQT)가 연구되었다[8]. 영역 분할 사분 트라이는 모든 룰이 한번만 저장되기에 매우 적은 메모리를 사용한다는 장점이 있으나 빠른 검색성능을 보장하지 못하는 단점이 존재한다[9].

본 논문에서는 트라이와 트리에 대해 설명하고 있다. 트라이와 트리의 가장 큰 차이점은 트라이는 트라이 구축에 있어 기준이 된 필드를 한 비트씩 검사하여 진행하고, 트리는 트리 구축에 있어 기준이 된 필드 값에 대한 대소 비교를 통하여 진행한다.

트라이 구조는 기본적으로 트라이의 패스(path)에 빈 노드가 존재함으로 인해 비효율적인 메모리 사용을 보이고[10], 이로 인해 검색 성능이 저하되는 단점이 존재한다. 하지만 이진 검색 트리 알고리즘[11]을 통해 패스에 존재하는 빈 노드를 완벽하게 제거함[12]으로써 메모리 요구량을 낮추며, 동시에 메모리 접근 횟수를 줄여 검색 성능을 향상시킬 수 있다.

본 논문에서는 영역 분할 사분 트라이의 단점을 이진 검색 트리[13]를 사용해 보완하여 검색 성능을 향상 시키는 알고리즘을 제안하고자 한다. 제안하는 알고리즘의 성능을 기존의 알고리즘인 영역 분할 사분 트라이와 비교 평가하였다.

본 논문은 다음과 같은 순서로 구성되었다. Ⅱ장에서 기존의 패킷 분류 구조에 대해 간략하게 설명하고 Ⅲ장에서 본 논문에서 제안하는 2차원 이진 검색 트리(Two-dimensional Binary Search Tree, 2D-BST)와 리프 푸시된 2차원 이진 검색 트리(Leaf Pushed 2D-BST)에 대해 상세히 설명한다. Ⅳ장에서는 본 논문에서 제안하는 패킷 분류 구조와 Ⅱ장에서 설명한 기존의 패킷 분류 구조와의 성능 비교 결과를 보이며, 마지막 Ⅴ장에서 결론을 맺는다.

Ⅱ. 관련 연구

1. 영역 분할 사분 트라이 (Area based Quad Trie, AQT)

AQT(Area - based Quad Trie), 즉 영역 분할 사분 트라이는 패킷 헤더의 검색 필드를 영역으로 해석하여 패킷 분류를 수행하는 가장 기본이 되는 패킷 분류 알고리즘이다. F1 필드인 근원지 프리픽스 필드와 F2 필드인 목적지 프리픽스 필드를 사용하여 영역을 분할하게 되지만, 근원지와 목적지 프리픽스 필드를 직접 사용하지 않고 코드워드(codeword)를 생성하여 영역을 분할한다. 근원지 프리픽스 필드와 목적지 프리픽스 필드를 각각 x축, y축으로 갖는 정사각형 평면을 전체 검색 영역으로 갖게 되는데, 근원지 프리픽스 길이, 목적지 프리픽스 길이가 각각 i, j이고 최대 프리픽스 길이가 인 룰이 차지하는 영역은 넓이의 사각형으로 표현된다.

(1) 구축과정

영역 분할 사분 트라이 알고리즘을 구축하기 위해서는 두 가지 방법을 사용할 수 있다. 첫 번째, 룰이 저장되는 영역의 코드워드를 경로로 가지는 트라이를 그리는 것이다. 먼저 룰 각각의 근원지와 목적지 프리픽스

필드를 결합하여 코드워드를 생성한 후 이를 이용하여 사분 트라이를 구성한다. 근원지 또는 목적지 프리픽스 필드의 길이 중 짧은 프리픽스 필드의 길이를 n이라고 가정하면 각 프리픽스 필드의 MSB(Most Significant Bit)부터 한 비트 씩 결합하여 n 디지트 (digit)의 코드워드를 생성한다. 이때, 근원지·목적지 프리픽스 필드가 항상 동일한 길이를 갖는다는 것은 불가능하기에 코드워드는 짧은 프리픽스 길이를 갖는 필드에 맞추어 생성되게 된다. 생성된 코드워드는 십진수 또는 이진수로 값을 나타낸다. 이렇게 생성된 코드워드에 해당하는 룰을 n번째 레벨의 노드에 저장하게 된다.

영역 분할 사분 트라이 구조는 각각의 노드에 대하여 최대 4개의 자식 노드를 생성할 수 있으며, 각각의 자식 노드들은 해당 노드의 영역을 동등하게 4분할 한 영역 중 하나를 갖는다. 이와 같은 방법으로 영역을 분할하여 구축하는 두 번째 과정을 살펴보면, 우선 전체 검색 영역부터 시작하여 균등 4분할하여 생긴 각각의 영역은 각각의 코드워드를 가지게 된다. 원점(0,0)을 기준으로 하여 왼쪽 혹은 아래쪽은 0, 오른쪽 혹은 위쪽은 1의 비트 값을 의미한다고 할 때 두 필드의 비트 값을 결합한 00, 01, 10, 11이 4분할된 영역들이 가지는 디지트이다. 분할된 영역을 다시 4분할한다면 분할되기 전 영역의 디지트에 위와 같은 방법을 통해 얻어진 00, 01, 10, 11 중 하나의 값을 추가한 디지트를 가지는 방법으로 계속해서 분할되는 영역들의 코드워드가 생성되게 된다.

이때 생성된 코드워드가 선택된 영역의 필드 중 어느 한 필드라도 모두 차지하고 있다면, 그 룰은 선택된 영역의 크로싱 필터 셋(crossing filter set, CFS)으로 정의한다. 이로 인해 하나의 크로싱 필터 셋에 여러 개의 코드워드 즉, 룰이 포함될 수 있다. 이는 앞서 설명한, 프리픽스의 길이가 다른 근원지 프리픽스와 목적지 프리픽스를 결합하여 코드워드를 만들 때 둘 중 프리픽스 필드의 길이가 짧은 프리픽스 필드 길이에 맞춰 결합하는 것이 해당 룰의 코드워드가 되는 것으로 설명할 수 있다.

아래의 표 1은 본 논문에서 사용하게 될 간단한 예시 룰 셋이다. 표 1의 프리픽스 항목에 존재하는 '*' 는 와일드카드(wildcard)를 의미하며 0 또는 1의 어떤 값이나 가질 수 있다.

표 1. 예시 룰 셋
Table 1. example rule set.

Rule No.	Source prefix	Destination prefix	Source Port	Destination Port	Protocol
R_0	010*	011*	0,65535	1704,1704	6
R_1	01100*	0110*	161,161	1711,1711	6
R_2	0110*	1001*	1024,1024	1521,1521	6
R_3	1010*	1101*	119,119	1717,1717	6
R_4	1*	10*	53,53	2110,2110	6
R_5	00*	0*	1024,1024	11717,1717	6
R_6	*	110*	80,80	1221,1221	6
R_7	000*	*	0,65535	0,65535	0

표 2. 영역 분할 사분 트라이 코드워드
Table 2. Area based quad trie's codeword.

Rule No.	Source prefix	Destination prefix	Codeword
R_0	010*	011*	00_11_01 (031)
R_1	01100*	0110*	00_11_11_00 (0330)
R_2	0110*	1001*	01_10_10_01 (1221)
R_3	1010*	1101*	11_01_10_01 (3121)
R_4	1*	10*	11 (3)
R_5	00*	0*	00 (0)
R_6	*	110*	*
R_7	000*	*	*

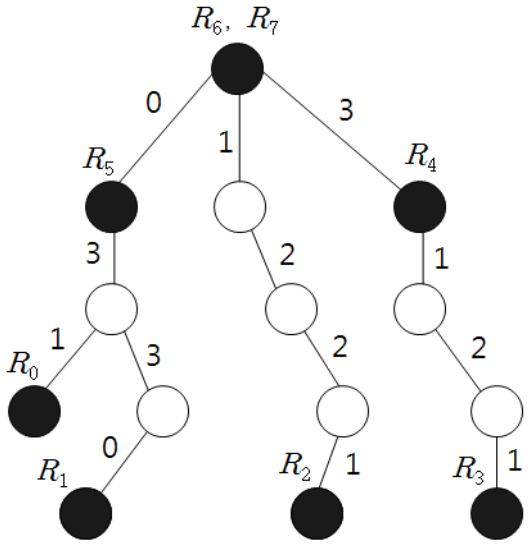

그림 1. 예시 룰 셋에 대한 영역 분할 사분 트라이
Fig. 1. Area based Quad Trie for example rule set.

표 2는 표 1의 예시 룰 셋을 가지고 영역 분할 사분 트라이 구축을 위한 코드워드를 생성한 것을 보여준다. 코드워드는 이진수와 십진수로 표현될 수 있으며, 이진

수 디지트 00은 십진수 0으로 01은 1, 10은 2, 11은 3으로 표현할 수 있다. 예를 들어 (0110*, 011*)이 속한 크로싱 필터 셋의 영역은 MSB부터 코드워드로 00_11_01을 가지므로 십진수로 표현하면 031이고 루트 노드부터 시작하여 해당 경로에 위치하는 노드에 가 저장된다.

그림 1은 표 1의 예시 룰 셋에 대하여 표 2의 코드워드를 활용하여 해당 경로에 룰을 저장시켜 최종적으로 구축한 영역 분할 사분 트라이이다.

(2) 검색과정

입력된 패킷의 근원지와 목적지 IP 주소의 프리픽스 필드의 비트를 결합하여 코드워드를 생성한 후 각 레벨에 접근하여 검색을 진행할 수 있다. 즉, 검색 과정은 입력된 패킷의 근원지 IP 주소와 목적지 IP 주소를 사용하여 구축 과정과 같은 방식으로 코드워드를 생성하고, 루트부터 순차적으로 그 경로에 존재하는 노드를 따라가면서 진행하게 된다. 만약 검색 진행 과정 중 룰이 저장된 노드를 만나게 되면, 입력 패킷과 해당 노드에 저장된 룰들이 일치하는지 판단하는 룰 비교를 수행한다. 이 때, 일치하는 룰들이 발생하게 되면, 일치하는 룰들 중 가장 높은 우선순위를 가지는 룰이 최우선 일치 룰(best matching rule, BMR)로 정해진다.

나. BST (Binary Search Tree, 이진 검색 트리)

기존의 알고리즘들은 각 필드로 구성된 트라이의 내부에 빈 노드가 존재하게 되므로 메모리 요구량이 증가하고, 이로 인해 메모리 효율성이 떨어지게 된다. 이러한 빈 노드로 인한 메모리 비효율성의 단점을 완전히 제거하여 메모리의 효율성을 높인 구조가 이진 검색 트리이다. 이진 검색 트리는 불필요한 메모리 접근을 제거함으로써 검색 기능을 향상 시킨 알고리즘이다.

(1) 구축과정

이진 검색 트리 알고리즘을 구축하기 위해서는 프리픽스 필드들 사이의 크기 비교를 통한 정렬을 정의하고, 프리픽스 필드의 네스팅 관계에 대한 개념을 정의해야 한다. 먼저, 크기 비교를 통한 정렬을 정의함에 있어서 각각의 프리픽스 필드가 고정된 길이를 갖는 것이 아니고 서로 다른 길이를 갖기 때문에 어려움이 있다. 이러한 프리픽스 필드의 가변성은 같은 길이를 갖는 프리픽스 필드의 경우에는 프리픽스 필드의 값 비교를 통해 크기를 결정하고, 서로 다른 길이의 프리픽스 필드를 갖는 경우에는 짧은 쪽 프리픽스 필드 길이까지의 값을 비교하여 크기를 비교함으로써 해결 할 수 있다.

다음으로 네스팅 관계를 해결하기 위해, 디스조인트(disjoint), 인클로져(enclosure), 인클로즈드(enclosed)의 개념을 정리한다. 디스조인트 프리픽스는 인클로져 또는 인클로즈드가 아닌 프리픽스이며, 인클로져 프리픽스는 자신을 프리픽스로 갖는 또 다른 프리픽스가 존재하는 프리픽스이고, 인클로즈드 프리픽스는 인클로져 프리픽스를 포함하는 프리픽스이다. 인클로져 프리픽스와 인클로즈드 프리픽스들 간의 크기 비교는 다음과 같이 정의한다. 인클로즈드 프리픽스의 다음 비트가 0이면 인클로즈드 프리픽스가 인클로져 프리픽스보다 작고, 1이면 인클로져 프리픽스보다 크다.

크기 비교 및 네스팅 관계 정의가 완료 되었다면, 크기 비교를 통한 이진 검색 트리를 만들되 인클로져 프리픽스는 인클로즈드 프리픽스보다 먼저 비교되어야 올바른 검색 결과를 얻을 수 있다. 그러므로 인클로져 프리픽스가 인클로즈드 프리픽스보다 트리의 상위레벨에 위치할 수 있도록, 먼저 디스조인트 프리픽스 필드와 인클로져 프리픽스를 크기순으로 정렬하여 이진 검색 트리를 만들되 인클로져 프리픽스가 노드에 위치한 후 인클로져 프리픽스를 하위 레벨 노드에 위치시키는 방법을 사용한다. 이진 검색 트리는 필드별로 각 한 개씩의 테이블을 갖는다. 테이블의 엔트리에는 근원지·목적지 주소 뿐 만 아니라 근원지 포트, 목적지 포트, 프로토콜 필드 등 룰의 상세 정보들이 저장된다.

011*, 00*, 0110*, 1*, 1010*의 작은 예시 프리픽스로 이진 검색 트리를 구축해보면, 먼저 프리픽스들 사이의 네스팅 관계를 파악한다. 그림 2는 예시 프리픽스들의 네스팅 관계, 즉 인클로져와 인클로즈드 프리픽스를 표현한 그림이다.

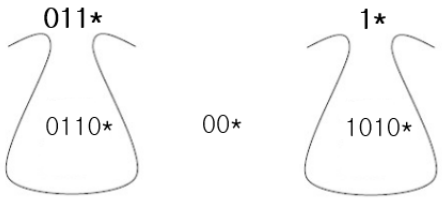

그림 2. 예시 프리픽스들에 대한 네스팅 관계
Fig. 2. Nesting relationship for example prefix.

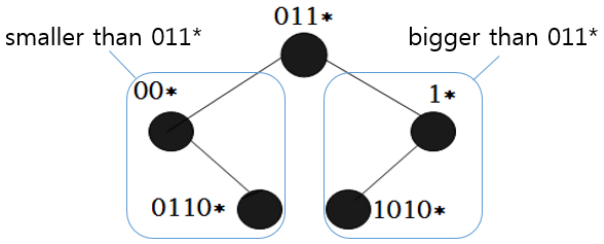

그림 3. 예시 프리픽스들에 대한 이진 검색 트리
Fig. 3. Binary search tree for example prefix.

011*는 0110*의 인클로저 프리픽스이며, 1*는 1010* 프리픽스의 인클로저이다. 그러므로 서로 디스조인트한 관계인 00*, 011*, 1*를 크기 순으로 정렬한다. 011* 프리픽스가 루트에 저장되고, 011*보다 작은 00*가 왼쪽, 011*보다 큰 1* 가 오른쪽에 저장된다. 011*의 인클로즈드 프리픽스인 0110*는 011*보다 작으므로 루트의 왼쪽 서브-트리에 포함되어야 하고 00*보다 크므로 00*의 오른쪽에 위치하게 된다. 1*의 인클로즈드 프리픽스인 1010*은 1* 보다 작으므로 1* 노드의 왼쪽 서브 트리에 위치하여야 한다. 그림 3은 예시 프리픽스들에 대한 이진 검색트리이다.

다. AQT – Leaf Pushing

리프 푸싱(Leaf pushing)[14]이란 트라이 내부 노드에 위치한 모든 프리픽스 노드들을 리프 노드로 보내 유효한 프리픽스 노드가 마지막 리프(잎 노드)에만 존재하도록 만드는 방법이다[15]. 리프 푸싱을 하게 되면 마지막 리프는 유효한 프리픽스 노드들로만 구성되고, 이들은 네스팅 관계에서 서로 디스조인트 관계가 된다.

(1) 구축 과정

먼저 앞서 설명한 과정으로 영역 분할 사분 트라이 알고리즘을 구축한 후, 완성된 영역 분할 사분 트라이를 리프 푸싱 해준다. 즉, 기존의 룰이 갖고 있는 코드워드가 아닌 리프 푸싱 된 새로운 코드워드로 영역 분할 사분 트라이를 구축하는 것이다. 리프 푸싱 된 코드워드를 생성하는 방법은 근원지와 목적지 프리픽스 필드 중 프리픽스 길이가 긴 쪽을 기준으로 두고 짧은 프리픽스 필드를 한 비트 씩 확장시켜가며 리프 노드에 존재하게 되는 코드워드를 만드는 것이다. 마지막 리프 노드에 존재 할 수 있게 되는 코드워드가 리프 푸싱 된 새로운 코드워드이고 이렇게 생긴 새로운 코드워드를 사용하여 영역 분할 사분 트라이를 구축한다.

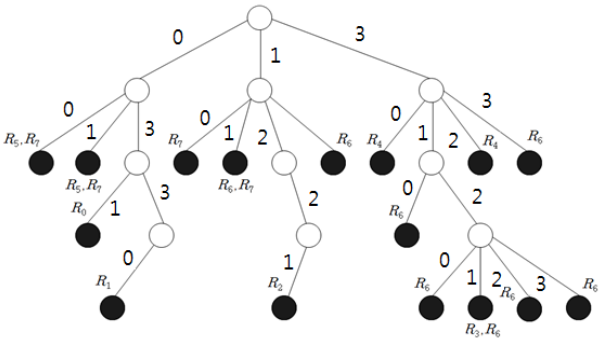

그림 4. 예시 룰 셋에 대한 리프 푸싱 영역 분할 사분 트라이
Fig. 4. Leaf pushing Areabased quad trie for example rule set.

이후 과정은 기존의 영역 분할 사분 트라이 구축과정과 동일하다.

앞서 예시 룰 셋의 영역 분할 사분 트라이(그림2)에서 보면 R_4, R_5, R_6, R_7는 룰이 리프에 존재하지 않고 트라이의 내부에 존재한다. 이렇게 되면 앞서 설명한 방법과 같이 리프 푸싱을 해줘야 한다. 예를 들어 R_4는 근원지 프리픽스는 1*이고 목적지 프리픽스는 10*이다. 근원지 프리픽스는 1비트이고 목적지 프리픽스는 2비트이므로, 목적지 프리픽스의 길이가 더 길기 때문에 근원지 프리픽스를 확장시킨다. 먼저, 근원지 프리픽스와 목적지 프리픽스 각각의 첫 번째 비트를 결합하여 3의 코드워드를 생성한다. 3은 리프 노드가 아니므로, 길이가 짧은 근원지 프리픽스를 확장시켜 새로운 리프 푸싱 된 코드워드를 생성한다. 근원지 프리픽스를 목적지 프리픽스에 맞춰 2비트로 확장 시켜보면, 10*와 11*가 된다. 확장시킨 근원지 프리픽스와 목적지 프리픽스를 결합하여 코드워드를 생성하게 되면 30 과 32의 코드워드를 갖게 된다. 기존의 영역분할 사분트리(그림2)에서 30과 32의 경로는 리프 노드에 해당되는 코드워드이므로, 더 이상 프리픽스를 확장하지 않고 해당 노드에 룰을 저장한다. 그림 4는 리프 푸싱한 코드워드로 구축한 영역 분할 사분트리이다.

(2) 검색 과정

검색 과정은 기존의 영역 분할 사분 트라이와 동일하다. 즉 입력된 패킷의 근원지와 목적지 IP 주소의 프리픽스 필드를 한 비트 씩 결합하여 코드워드를 생성한 후, 루트 노드부터 시작하여 경로에 존재하는 노드를 순차적으로 따라가면서 검색을 진행한다. 검색 진행 과

정 중 룰이 저장된 마지막 리프 노드를 만나게 되면, 코드워드 일치여부를 판단한 후 입력 패킷과 해당 노드에 저장된 룰들이 일치하는지를 판단하는 룰 비교를 수행한다. 이 때, 일치하는 룰이 여러 개 발생하게 되면, 가장 높은 우선순위를 가지는 룰이 최우선 일치 룰(best matching rule, BMR)로 정해지고 검색 결과가 된다.

Ⅲ. 제안하는 구조

영역 분할 사분 트라이의 단점을 이진 검색 트리를 사용해 보완하여 검색 성능을 향상 시키는 알고리즘을 제안한다. 아래 그림 5는 제안하는 구조의 도식화이다.

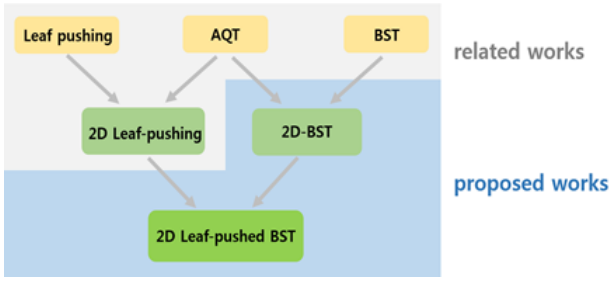

그림 5. 제안하는 알고리즘
Fig. 5. Proposed Algorithms.

가. 2차원 이진 검색 트리 (2D-BST)

기존의 트라이 알고리즘은, 근원지와 목적지 프리픽스 필드 값을 결합하여 하나의 코드 워드로 이차원 검색을 가능하게 한 영역 분할 사분 트라이와 달리 근원지 프리픽스 필드와 목적지 프리픽스 필드를 계층적으로 연결하였다. 이를 통해 기존 영역 분할 사분 트라이와 비교하여 수많은 선형 검색을 피하고, 빠른 검색 성능을 제공한다. 하지만, 기존 트라이 알고리즘은 프리픽스 필드로 구성된 각각의 트라이 내부에 빈 노드가 존재하게 되고, 이로 인해 메모리 효율성이 급격히 떨어지게 된다. 본 논문에서는 영역 분할 사분 트라이 알고리즘에 이진 검색 트리 알고리즘을 적용하여 빈 노드를 완벽히 제거하여 메모리 효율성을 높인 이진 검색 트리를 구성하여 계층적 접근이 아닌 한번에 2차원 접근이 가능한 이차원 이진검색 트리(2D-BST)를 제안하고자 한다.

(1) 구축 과정

먼저, 근원지 IP 주소의 프리픽스와 목적지 IP 주소의 프리픽스를 한 비트 씩 결합하여 코드워드를 생성한다. 코드워드를 만들 때에는 두 IP 주소의 프리픽스 필드 중 길이가 짧은 프리픽스 필드 길이에 맞춰 결합한다. 이진 검색 트리를 구축하기 위해 코드워드들을 정렬하여야 하는데, 이때 앞서 설명한 프리픽스들간의 크기 비교에 관한 규칙을 활용한다. 코드워드들 사이의 네스팅 관계 즉 인클로져, 인클로즈드, 디스조인트 코드워드를 정의하고, 인클로져 코드워드와 인클로즈드 코드워드의 크기 비교는 다음과 같이 정의할 수 있다. 인클로즈드 코드워드의 다음 디지트가 0 혹은 1이면 인클로져 코드워드보다 작고, 2 혹은 3이면 인클로져 코드워드보다 크다.

먼저 정의된 인클로져와 디스조인트 코드워드를 크기순으로 정렬한다. 이 후 인클로저에 포함된 인클로즈드 코드워드가 존재하면, 인클로즈드 코드워드를 정렬 리스트에 포함하여 재 정렬을 하고 이렇게 정렬된 리스트를 최종 리스트로 활용하여 이진검색 트라이를 구성한다. 이진 검색 트라이를 구축한 후, 근원지·목적지 프리픽스 두 필드 모두가 와일드카드인 룰들은 일반적으로 우선순위가 매우 낮으므로 검색의 효율성을 위해 선형 검색이 가능하도록 별도로 저장해 준다.

앞서 표 1의 예시 룰 셋을 사용하여 제안하는 구조를 구축해보겠다. 먼저 표 2를 활용하여 코드워드들 간의 네스팅 관계를 정의한다. *는 나머지 모든 코드워드의 인클로져이다. 즉, *라는 인클로져 안에 0, 1221, 3, 031, 0330, 3121 라는 인클로즈드 코드워드가 존재한다. 하지만 이 코드워드들 사이에서도, 0은 031, 0330의 인클로져 코드워드가 되고 3은 3121의 인클로져가 된다. 1221 코드워드는 인클로즈드 코드워드가 없으므로, 0, 1221, 3는 서로 디스조인트한 코드워드가 된다. 이렇게 네스팅 관계가 정의된 코드워드들 간의 크기 정렬을 통해

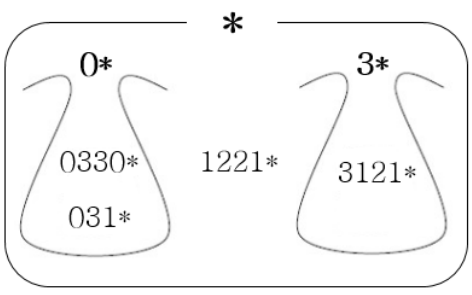

그림 6. 예시 룰 셋에 대한 네스팅 관계
Fig. 6. Nesting relationship for example rule set.

이진 검색 트리를 구축한다. 그림 6은 예시 룰 셋에 대한 네스팅 관계를 표현한 그림이다.

먼저 *는 모든 프리픽스의 인클로저 코드워드이므로 루트 노드에 저장하고, 그 아래 0, 1221, 3을 크기 정렬을 통해 0, 1221는 *보다 작으므로 *노드의 왼쪽에 위치하고 3은 *보다 크므로 오른쪽에 위치해야 한다. 0과 3이 노드에 위치되면 이들의 인클로즈드 코드워드들도 노드에 위치시킬 수 있다. 0을 루트로 하는 서브 트리에는 0의 인클로즈드 코드워드인 031, 0330과 아직 노드를 차지하지 않은 1221이 들어가게 되는데 이들은 모두 0보다 큰 코드워드이므로 0의 오른쪽 서브 트리에 포함되어야 한다. 이 들 세 개 코드워드들은 서로 디스조인트하므로 031, 0330, 1221 의 순으로 정렬하여 0의 오른쪽 서브 트리를 구성한다.

마찬가지로, 3의 인클로즈드인 3131은 3보다 크기가 작으므로 3노드의 왼쪽 자식노드에 저장된다. 아래 그림 7은 이러한 코드워드들을 사용하여 2차원 이진검색 트리를 구축한 제안하는 구조이다.

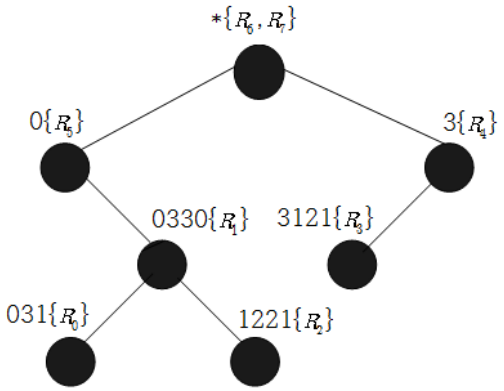

그림 7. 예시 룰 셋에 대한 2차원 이진 검색 트리
Fig. 7. Two dimensional Binary search tree for example rule set.

(2) 검색 과정

검색을 진행하기 위해서는 먼저 이차원 검색이 가능하게하기 위하여 입력패킷의 코드워드를 생성하여야 한다. 코드워드를 만드는 과정은 구축과정의 코드워드 생성과정과 동일하다. 근원지 IP 주소와 목적지 IP 주소를 한 비트 씩 결합하여 코드워드를 생성한다. 이렇게 생성된 코드워드가 대소 비교를 위해 사용된다. 검색은 입력 패킷의 대소 비교를 통하여 입력 패킷의 코드워드가 작을 경우 왼쪽포인터를 따라 검색을 진행하고, 입력 패킷의 코드워드가 클 경우에는 오른쪽 포인터를 따라 검색을 진행하게 된다. 만약 코드워드가 일치하는 노드를 만나게 되면, 선형 검색을 진행한다. 즉, 나머지 룰의 상세 정보들도 비교하여 모든 정보가 정확히 일치하는 룰을 검색 결과로 찾아낸다. 이러한 이진 검색은 마지막 잎 노드까지 수행하여 일치하는 룰들을 찾아낸다. 검색 결과로 얻은 룰들 중 우선순위가 가장 높은 룰을 최종 검색 결과로 얻게 된다. 이진 검색을 통해 얻은 최종 검색 결과를 프리픽스 필드가 모두 와일드 카드인 룰들과 우선순위 비교를 통해 선형검색을 한 번 더 진행하게 된다. 선형 검색까지 모두 마친 결과가 최종 결과로 도출된다.

나. Leaf pushed - 2D BST

앞서 제안한 2 차원 이진 검색 트리 (2D-BST)는 빈 노드를 완전히 제거한 이진 검색 트리로서 검색 성능 향상을 이루었다. 하지만 구축하는 과정에서 인클로즈드 프리픽스는 항상 인클로져 프리픽스 노드의 하위 레벨에 존재하여야 한다. 즉, 코드워드들 간의 네스팅 관계에 따라 트리의 깊이가 매우 깊어질 수 있는 단점이 존재하게 된다. 또 코드워드를 생성하는 과정에서 같은 코드워드를 갖는 룰들은 하나의 같은 노드에 존재하게 되는데, 이렇게 되면 검색 과정에서 선형 검색해야 할 룰들이 많아져 검색 성능이 저하된다. 이러한 단점들을 개선하기 위해 복잡한 네스팅 관계를 정리할 수 있는 리프 푸싱을 활용하여 균형 맞는(balanced) 이진 검색 트리를 제안한다.

(1) 구축 과정

Leaf pushed - 2D BST 구축 과정의 핵심은 AQT - Leaf pushing 알고리즘을 통해 구축된 트리의 마지막 리프 노드에 존재하는 프리픽스들을 떼어 오는 과정이다. 마지막 리프 노드들을 떼어 올 때에는 룰 번호, 리프 푸쉬 된 노드 위치의 코드워드, 근원지 IP 주소, 목적지 IP 주소, 근원지 포트, 목적지 포트, 프로토콜 등의 룰 상세 정보를 모두 가지고 와야 한다. 이렇게 얻은 마지막 리프 노드들의 룰 정보를 가지고 이진 검색 트리를 구축하게 된다. 앞선 2D-BST와 다르게 코드워드를 따로 생성하지 않고 AQT - Leaf push에서 가져온 코드워드를 가지고 정렬을 하여 이진 검색 트리를 구축한

다. 이 때 코드워드들은 모두 디스조인트 관계이기 때문에 따로 네스팅 관계를 정의하지 않고 크기별로 정렬해 주면 된다. 크기순으로 정렬이 완료되면 이진 검색 트리를 완성하는 것으로 이진 검색 트리구축이 끝난다. 구축 과정에서 주의해야 할 점은 AQT - Leaf push 알고리즘에서 새로운 리프 푸싱된 코드워드를 생성하는 과정에서 근원지·목적지 프리픽스가 모두 * 인 룰들을 리프 푸싱하는 경우 과도한 룰 복사가 일어나는 문제점이 있다. 또한 이러한 룰들은 대부분 우선순위가 낮은 룰들이므로, 별로로 분리하여 검색하는 것이 유리하다. 따라서 이러한 룰들은 리프 푸싱하지 않고, 따로 정보를 가져와 이진 검색 트리 구축이 끝난 후, 선형 검색을 할 수 있도록 별도로 저장해 준다.

앞서 표 1의 예시 룰 셋을 사용하여 두 번째 제안하는 구조인 Leaf pushed - 2D BST를 구축해보겠다. 먼저 Leaf pushed - 2D BST를 구축하기 위해 AQT-Leaf push에서 필요한 정보들을 모두 정리해야 한다. 표 3에 리프 푸싱 된 새로운 코드워드 및 룰 정보들을 정리해 보았다.

이렇게 얻은 표 3을 활용하여 이진 검색 트리를 구축한다. 이진 검색 트리를 구축하는 방법은 앞서 첫 번째 제안하는 구조와 동일하다. 다만 차이점은 룰의 코드워드가 아닌, 리프 푸싱 된 새로운 코드워드를 사용한다

표 3. 리프 푸시 된 새로운 영역 분할 코드워드
Table 3. Area based quad trie's Leaf-pushed codeword.

Rule No.	Source prefix	Destination prefix	Codeword
R_0	010*	011*	00_11_01 (031)
R_1	01100*	0110*	00_11_11_00 (0330)
R_2	0110*	1001*	01_10_10_01 (1221)
R_3	1010*	1101*	11_01_10_01 (3121)
R_4	1*	10*	11_00 (30)
			11_10 (32)
R_5	00*	0*	00_00 (00)
			00_01 (01)
R_6	*	110*	01_01 (11)
			01_11 (13)
			11_11 (33)
			11_01_00 (310)
			11_01_10_00 (3120)
			11_01_10_01 (3121)
			11_01_10_10 (3122)
			11_01_10_11 (3123)
R_7	000*	*	00_00 (00)
			00_01 (01)
			01_00 (10)
			01_01 (11)

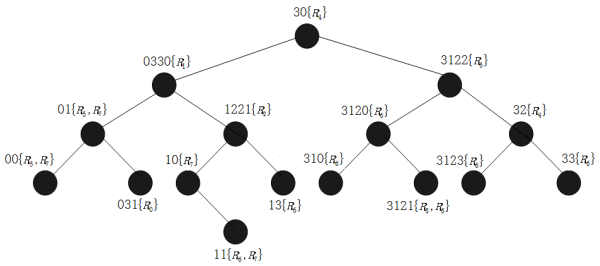

그림 8. 예시 룰 셋에 대한 Leaf pushed - 2D BST
Fig. 8. Leaf pushed - 2D BST for example rule set.

는 점이다. 즉, 리프 푸싱 된 새로운 코드워드들은 네스팅 관계가 없으므로, 크기 정렬을 통해 이진검색 트리를 구축하는 것이다. 그림 8은 위와 같은 방법을 사용하여 두 번째 제안하는 구조를 구축한 결과이다.

(2) 검색 과정

검색 과정은 기본적으로 2D BST 검색 과정과 같다. 즉, 이진 검색을 모두 마친 후, 두 개 프리픽스 필드가 모두 와일드카드인 룰들에 대한 선형검색이 추가된다. 그러나 Leaf pushed - 2D BST의 이진검색은 앞서 제안한 이차원 이진 검색 트리와 이진검색 방법이 조금 다르다. 즉, Leaf pushed - 2D BST는 리프 푸싱을 통해 모든 코드워드들이 디스조인트한 관계이므로, 어떤 노드에서 일치한 코드워드를 만나는 경우 트리의 리프까지 검색하지 않고 바로 검색을 종료하게 된다. 선형 검색은 이진검색 트리에서 얻은 현재까지의 최우선순위 룰 (current BMR)과 선형 검색을 위해 저장된 룰들의 최우선순위 룰의 우선순위 비교를 통해, 이진 검색 트리에서 검색된 현재까지의 BMR보다 우선순위가 높은 룰만 비교하면 된다. 이렇게 선형 검색까지 마친 결과가 최종 결과로 도출된다.

Ⅳ. 성능 평가

본 논문에서는 보다 효율적인 패킷 분류를 위해 근원지·목적지 프리픽스를 코드워드로 결합하여 한 번에 이차원 검색이 가능하게 하고, 프리픽스들 간의 네스팅 관계를 정리하여 균형 맞는 트리를 만들어 빠른 검색 성능을 갖는 이차원 검색 트리 알고리즘을 제안하였다. 제안하는 두 알고리즘의 성능 평가를 위해 실제 라우터의 유사한 룰 특성을 반영하는 클래스벤치의 룰 셋인

2D BST B_s, the number of rules: 9735				Leaf pushed - 2D BST L_s, the number of rules: 195783			
Entry width				Entry width			
node (119bits = 14.875bytes)		Rule entry (174bits = 21.75 bytes)		node (121bits = 15.125bytes)		Rule entry (182bits = 22.75bytes)	
Field	No. of bits	Field	No. of bits	Field	No. of bits	Field	No. of bits
node threshold	1	rule number	14	node threshold	1	rule number	18
codeword length	6	source prefix length	6	codeword length	6	source prefix length	6
codeword	64	source prefix	32	leaf-pushed codeword	64	source prefix	32
left pointer valid	1	destination prefix length	6	left pointer valid	1	destination prefix length	6
left pointer	16	destination prefix	32	left pointer	16	destination prefix	32
right pointer valid	1	source port wild	1	right pointer valid	1	source port wild	1
right pointer	16	source port start	16	right pointer	16	source port start	16
rule pointer	14	source port end	16	rule pointer	18	source port end	16
		destination port wild	1			destination port wild	1
		destination port start	16			destination port start	16
		destination port end	16			destination port end	16
		protocol wild	1			protocol wild	1
		protocol type	3			protocol type	3
		next rule pointer	14			next rule pointer	18

그림 9. 데이터 구조
Fig. 9. Data structure.

ACL (access control list), FW (firewall), IPC (IP chain)의 룰 셋[16~17]과 인풋을 이용하여 C언어 코딩 실험을 진행했다. 성능 평가를 위해 사용된 룰 셋은 타입별로 각각 1k, 5k, 10k 의 룰 개수를 갖는다.

그림 9는 각각 2D BST, Leaf-pushed 2D BST의 데이터 구조를 나타내고 있다. B_s, L_s는 2D BST, Leaf-pushed 2D BST 각각의 데이터 구조에서 룰 메모리에 저장되는 총 룰의 수를 의미한다. 2D BST와 Leaf-pushed 2D BST의 데이터 구조는 동일한 형식을 가지고 있으나, node에서 2D BST는 룰의 코드워드가 저장되지만 Leaf-pushed 2D BST에서는 리프 푸시를 통해 새롭게 생성된 leaf-pushed 코드워드가 저장되는 것이 차이점이다.

표 5에서는 2D BST, Leaf pushed - 2D BST의 룰 메모리 요구량을 비교하였다. B_s, L_s는 각 알고리즘 데이터 구조에서 설명했던 것과 같이 복사되어 저장된 룰의 개수까지 포함한 총 저장된 룰의 개수를 의미한다. M_b, M_l는 룰의 개수에 데이터 구조의 룰 엔트리 크기를 곱하여 룰 메모리 요구량을 KB 단위로 나타낸 것이다.

표 5. 룰 메모리 요구량
Table 5. Rule's memory.
(Memory requirement for rule database)

	2D BST		Leaf pushed - 2D BST	
	B_s	M_b(KB) (B_s ×174bit)	L_s	M_l(KB) (L_s ×182bit)
ACL1k	958	20.34	7378	163.47
FW1k	871	18.50	9206	204.53
IPC1k	988	20.99	11134	247.36
ACL5k	4660	98.98	39362	874.50
FW5k	3067	65.14	195783	4349.67
IPC5k	4468	94.90	43064	956.74
ACL10k	9735	206.77	39473	876.96
FW10k	4351	92.42	56508	1255.43

가. 노드 접근

그림 10에서는 AQT, 2D BST, Leaf-pushed 2D BST 간의 최대 노드 접근 횟수와 평균 노드 접근 횟수를 비교하였다. ACL, FW, IPC 세가지 타입의 룰셋에

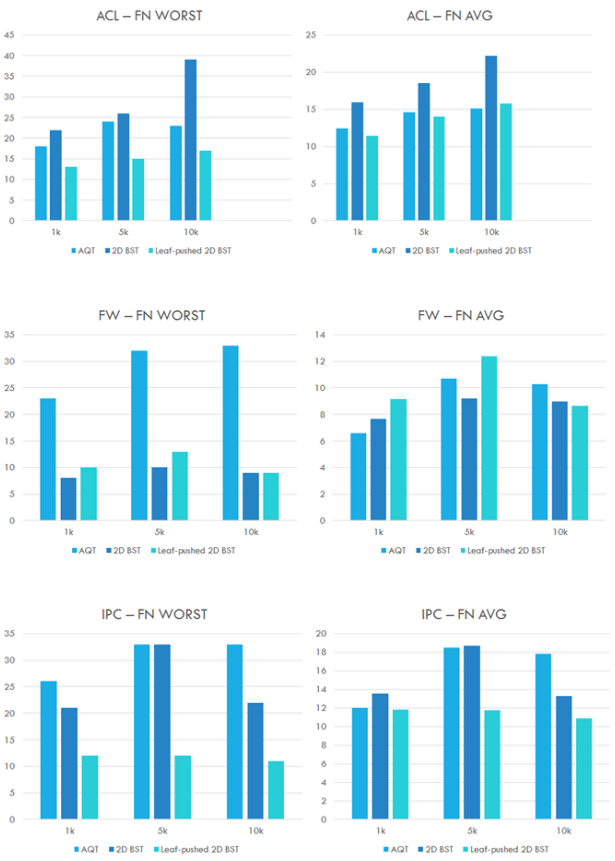

그림 10. 평균 및 최대 노드 접근 횟수
Fig. 10. The average and the worst-case number of node accesses.

대하여 각각 1k, 5k, 10k 개수에 따라 노드 접근 횟수를 y-축에 나타내었다. 기본적인 검색 과정에 있어 AQT는 하나의 노드에 자식노드가 4개씩 존재하고, 제안하는 나머지 두 2D BST, Leaf-pushed 2D BST는 하나의 노드에 자식 노드가 2개씩 존재하기 때문에 룰 특성에 따라 다르지만 노드 접근 횟수가 증가할 수 있는 가능성이 존재한다. 따라서 기존 알고리즘인 AQT에 비하여 노드 접근 횟수에 있어서는 월등히 뛰어난 검색 성능을 보여주지는 않는다. 실제 라우터에서의 검색 성능은 룰 검색 성능에 의해 좌우 되므로 노드 접근 횟수에서의 작은 손해는 검색 성능에 큰 영향을 미치지 않는다.

(2) 룰 접근

그림 11에서는 AQT, 2D BST, Leaf-pushed 2D BST 간의 최대 룰 접근 횟수와 평균 룰 접근 횟수를 비교하였다. 제안하는 알고리즘 중 첫 번째 2D BST는 기존의 알고리즘인 AQT와 비교하였을 때 거의 같은 룰 접근 성능을 보여주고 있다. 10k에서 AQT에 비해

그림 11. 평균 및 최대 룰 메모리 접근 횟수
Fig. 11. The average and the worst-case number of rule memory accesses.

성능이 조금 떨어진 것을 확인할 수 있는데, 이는 이차원 이진 트리를 검색하는 과정에 있어서 코드워드가 일치하는 노드를 만날 때마다 그 노드에 연결리스트로 존재하는 모든 룰과의 선형적인 비교가 일어나기 때문이다. 두 번째 제안하는 알고리즘인 Leaf-pushed 2D BST는 기존 알고리즘 및 2D BST와 비교하여 룰 검색 성능이 월등하게 좋은 것을 확인 할 수 있다. 이는 리프 푸싱을 통해 복잡했던 네스팅 관계를 정리하여 모든 코드워드들은 디스조인트한 관계가 되므로 어떤 노드에서 일치한 코드워드를 만나는 경우, 트리의 리프까지 검색함 없이 바로 검색을 종료할 수 있기 때문에 룰 비교 횟수가 급격히 줄어들어 검색 성능의 향상이 이루어진 것이다.

V. 결 론

본 논문에서는 기존 패킷분류 알고리즘보다 빠른 패킷 분류 성능을 제공하는 두 가지의 이진 검색 트리를 제안하였다. 첫째, 기존 트라이 알고리즘 보다 효율적인 패킷 분류를 제공하기 위해 두 프리픽스 필드를 결합하여 코드워드를 생성하여 한 번에 이차원 검색이 가능한 이차원 이진 검색 알고리즘을 제안하였다. 둘째, 기존의 영역 분할 알고리즘이 룰의 특성이 잘 고려되지 못하여 여러 개의 룰이 하나의 노드에 저장되어 검색 속도가 저하되는 단점을 리프 푸싱을 통해 해결 한 알고리즘을 제안하였다. 제안한 알고리즘은 많은 선형 검색으로 검색 성능이 떨어지는 기존의 영역 분할 알고리즘의 단점을 보완하였다. 제안하는 알고리즘은 성능평가 검증 결과 룰 접근 횟수 즉, 검색 성능이 기존의 패킷 분류 알고리즘과 비교하여 월등히 좋아짐을 확인하였다.

REFERENCES

[1] H. Song, J.W. Lockwood, “Efficient Packet Classification for Network Intrusion Detection Using FPGA,” in Proc. ACM SIGDA FPGA, pp.238-245, 2005
[2] W. Lu, S. Sahni, “Succinct Representation of Static Packet Classfiers,” IEEE ACM Transactionson Networking, vol.17, Iss.3, pp.803 - 816, June. 2009.
[3] M.de Berg, M.Van Kreveld,M. Overmars, and

O.Schwarzkopf, “Computational Geometry Algorithms and Applications,” Springer-Verlag, 2000.
[4] P. Gupta, N. Mckeown, “Algorithms for packet classification,” IEEE Network, vol.15, no.2, pp.24-32, Mar. Apr. 2001
[5] H. Lim and N. Lee, “Survey and Proposal on Binary Search Algorithms for Longest Prefix Match,” in Proc. IEEE Communications Surveys and Tutorials , vol.14, no.3, Third Quarter, pp.681-697, 2012
[6] V. Srinivasan, S. Suri, G. Varghese, “Packet classification using tuple space search,” in Proc. of ACM SIGCOMM Computer Communication Review, vol.29, no.4, pp.135-146, 1999
[7] F. Baboescu, S. Singh, G. Varghese, “Packet classification for core router: is there an alternative to CAM”, in Proc. IEEE INFOCOM 2003, vol.1, pp.53-63, Mar. 2003
[8] M. M. Buddhikot, S. Suri, M. Waldvogel, “Space decomposition techniques for fast layer-4 switching,” in Proc. Conf. Protocols for High Speed Networks, pp.25-41, Aug. 1999
[9] H.Lim, M.Kang, and C.Yim, “Two-dimensional packet classification algorithm using a quad-tree, ”Computer communications, Science, vol.30, no6, pp.1396-1405, Mar.200
[10] H.Lim, H.Chu, and C.Yim, “Hierarchical Binary Search Tree for Packet Classfication,” IEEE Communications Letters, vol.11, no8, pp.689-691, Aug.2007.
[11] N. Yazdani and P.S Min, “Fast and Scalable Schemes for the IP Address Lookup Problem,” Proc.IEEE HPSR2000, pp83-92,2000.
[12] H.Lim, W.Kim, and B.Lee, “Binary Search in a Balanced Tree for IP Address Lookup,” Proc.IEEE HPSR 2005, May2005
[13] V.Srinuvasan and G.Varghese, “Fast Address Lookups Using Controlled Prefix Expansion”, ACM Transactions on Computer Systems, Vol.17, No,1, pp.1-40, Feb.1999.
[14] J. Mun, H.Lim and C.Yim, “Binary Search on Prefix Lengths for IP Address Lookup,” IEEE communications Letters, vil.10, no.6, pp.492-494, June,2006
[15] C.Yim, B.Lee, and H.Lim, “Efficient Binary Search for IP Address Lookup,” to be appeared on IEEE Comminications Letters, Jul.2005.
[16] D.E. Taylor, J.S. Turner, J.S, “ClassBench: a packet classification benchmark,” Proc. IEEE INFOCOM 2005, pp.2068-2079,March 2005.
[17] D.E. Taylor, J.S. Turner. The source code of Packet Classficatio Bench.

저 자 소 개

이 고 은(학생회원)
2013년 이화여자대학교 전자공학과 학사
2015년 이화여자대학교 전자공학과 석사

임 혜 숙(정회원)-교신저자
1986년 서울대학교 공과대학 제어계측공학과 학사
1991년 서울대학교 공과대학 제어계측공학과 석사
1996년 The University of Texas at Austin 전자공학 박사
1996년～2000년 Bell Labs at Lucent Technologies, Murray Hill, NJ, Member of Technical Staff
2000년～2002년 Cisco Systems, San Jose, CA, Hardware Engineer
2002년～현재 이화여자대학교 전자공학과 교수
1991년 국비유학장학생 선정
2008년 LG 연암재단 해외연구교수
2013년 이화여자대학교 연구우수교수
2014년 미래창조과학부 장관상 올해의 여성기술자상 수상