최적화된 영역 분할을 이용한 패킷 분류 알고리즘

(Optimum Range Cutting for Packet Classification)

김 형 기 † 박 경 혜 † 임 혜 숙 ††
(Hyeonggee Kim) (Kyonghye Park) (Hyesook Lim)

요 약 현재 패킷 분류에 대한 다양한 알고리즘들이 연구되어 오고 있다. 그 중 HiCuts와 HyperCuts와 같은 디시젼(decision) 트리에 기초한 패킷 분류 알고리즘은 룰의 각 필드가 가지는 영역에 따른 기하학적 구조를 이용한 방법으로 잘 알려져 있다. 그러나 이 알고리즘들은 분할(cutting)을 수행할 필드(Field)를 선택하거나 디시젼 트리의 각 노드에서 컷(cut)의 수를 결정해야 하는 등의 비교적 복잡한 작업을 요구하므로 현실적으로 구현하기 어려운 점을 가진다. 또한 각 룰이 차지하는 영역의 특성을 고려하지 않고 일정한 크기의 영역으로 커팅이 이루어지므로 효과적인 커팅을 하지 못하는 단점이 있다.

본 논문에서는 새로운 영역 분할을 사용한 효과적인 패킷 분류 알고리즘을 제안한다. 제안하는 알고리즘은 먼저 프리픽스를 가지는 두 필드를 이용하여 각 룰이 차지하는 영역들을 찾아내 이들을 이용해 영역 분할을 수행한다. 따라서 제안된 알고리즘은 보다 효율적인 디시젼 트리를 구성한다. 즉, 디시젼 트리의 각 노드에서는 HiCuts이나 HyperCuts와 같은 복잡한 작업없이 최적화된 커팅을 수행할 수 있다. 클래스 벤치에서 제공된 데이타베이스에 대하여 시뮬레이션을 수행한 결과, 제안된 알고리즘은 평균 검색 속도에서 기존의 알고리즘들보다 훨씬 향상되었고 메모리 요구량에서는 기존의 커팅 알고리즘과 비교하여 대략 3~300배까지 크게 줄어드는 효과를 보였다.

키워드 : 패킷 분류, 영역 분할, HiCuts, HyperCuts, 디시젼(decision) 트리

Abstract Various algorithms and architectures for efficient packet classification have been widely studied. Packet classification algorithms based on a decision tree structure such as HiCuts and HyperCuts are known to be the best by exploiting the geometrical representation of rules in a classifier. However, the algorithms are not practical since they involve complicated heuristics in selecting a dimension of cuts and determining the number of cuts at each node of the decision tree. Moreover, the cutting is not efficient enough since the cutting is based on regular interval which is not related to the actual range that each rule covers.

In this paper, we proposed a new efficient packet classification algorithm using a range cutting. The proposed algorithm primarily finds out the ranges that each rule covers in 2-dimensional prefix plane and performs cutting according to the ranges. Hence, the proposed algorithm constructs a very efficient decision tree. The cutting applied to each node of the decision tree is optimal and deterministic not involving the complicated heuristics. Simulation results for rule sets generated using class-bench databases show that the proposed algorithm has better performance in average search speed and consumes up to 3-300 times less memory space compared with previous cutting algorithms.

Key words : Packet classification, Range cutting, HiCuts, HyperCuts, Decision tree

· 본 연구는 지식 경제부 및 정보통신 연구 진흥원의 대학 IT연구 센터(홈네트워크 연구 센터) 육성, 지원 사업의 연구 결과로 수행되었습니다.

† 학생회원 : 이화여자대학교 대학원 전자정보통신학과
calmsea@ewhain.net
khpark09@ewhain.net
†† 비 회 원 : 이화여자대학교 전자정보통신학과 교수
hlim@ewha.ac.kr

논문접수 : 2008년 5월 9일
심사완료 : 2008년 8월 31일

정보과학회논문지: 정보통신 제35권 제6호(2008.12)

1. 서 론

인터넷의 급속한 성장과 더불어 인터넷 서비스의 질을 향상시키기 위하여 라우터에서의 패킷 포워딩이 매우 중요한 작업이 되고 있다. 패킷 포워딩이란 라우터 내로 들어온 패킷의 헤더 정보를 이용하여 최종 목적지 네트워크를 향해 패킷을 내 보내어 주는 작업을 말한다. 인터넷에서의 다양한 서비스의 요구는 패킷 처리를 더욱 어렵게 만들고 있는데, 예를 들어 오디오나 비디오 스트리밍과 같이 전달 속도에 민감한 응용 프로그램의 등장은 quality-of-service(QoS)를 요구하며, 방화벽(firewall protection)이나 load balancing, 웹 스위칭, intrusion detection and security(IDS) 등의 다양한 서비스가 요구되고 있다. 이러한 서비스들을 제공해주기 위하여 라우터에서는 패킷 분류(packet classification)가 선행되어야 한다.

패킷 포워딩에 있어 패킷들을 입력되는 링크 속도에 맞춰 처리하여 내보내 주어야 하는데, 패킷 분류는 단순히 목적지 주소만 확인하여 처리하는 IP 주소 검색과는 달리, 패킷 헤더 정보의 모든 필드를 검사하여야 하므로 패킷 처리 속도에 더 민감한 작업이라 할 수 있다[1,2]. 따라서 패킷 분류를 효율적으로 수행하기 위한 여러가지 패킷 분류 알고리즘 및 구조에 관한 연구가 활발하게 수행되어 오고 있다. 패킷 분류 알고리즘 및 구조들의 성능 평가에 있어 가장 중요한 기준은 패킷 처리 속도이고, 패킷 처리 속도는 메모리 접근 횟수에 의하여 결정된다. 그 다음으로 중요한 기준은 패킷 분류 테이블을 저장하기 위해 요구되는 메모리의 크기이다. 요구되는 메모리의 크기는 패킷 분류 테이블이 갖는 데이타 구조에 의존한다.

기존의 패킷 분류 관련 알고리즘들은 여러 가지가 있으나 본 논문에서는 커팅(cutting)을 이용한 알고리즘들을 분석하여 기존의 문제점들을 개선시키는 새로운 알고리즘을 제안한다. 커팅을 이용한 알고리즘은 들어온 패킷 헤더 정보가 일정 영역에 매치하는지 판단하여 맞는 최종 영역을 찾아가는 알고리즘으로 검색속도가 매우 우수한 방법이다. 그러나 이를 이용한 기존의 알고리즘들 즉, HiCuts[3], HyperCuts[4] 등은 일정한 간격으로 영역을 분할하여 패킷을 분류해 가므로 룰셋의 패턴에 맞춰 효과적으로 패킷을 분류하지 못하는 단점이 있다.

따라서 본 논문에서는 룰 패턴에 맞추어 커팅하는 효과적인 커팅 알고리즘을 제안한다. 이는 기존 방식인 일정한 간격으로 영역을 나누는 것이 아니라, 이진 영역분할 방식(BSR)[5]을 적용하여 룰의 패턴에 맞도록 룰의 영역을 커팅한다.

본 논문은 다음과 같이 구성된다. 2장에서는 기존의 커팅을 이용한 패킷 분류 알고리즘들의 문제점을 분석한다. 3장에서 제안하는 알고리즘의 이해를 돕기 위해 기존의 영역 분할 방식을 사용한 IP 주소 검색 알고리즘인 binary search on range (BSR)[5]을 설명한다. 4장에서 제안하는 알고리즘의 기본 구조를 설명하고, 5장에서는 제안하는 알고리즘의 최적화 구조를 설명한다. 5장에서는 제안하는 알고리즘의 성능을 기존의 알고리즘들과 비교한다. 마지막으로 6장에서 결론을 맺는다.

2. 기존의 패킷 분류 알고리즘

커팅을 이용한 패킷 분류 알고리즘은 룰 셋의 각 필드(field)를 영역(range)으로 표현하고 영역을 줄여가면서 패킷이 속하는 가장 구체적인 영역을 찾는 방법이다. 즉, 매 단계에서 각각의 필드에 대해 각 필드 별로 전체 영역을 적당한 간격으로 나누고 각각의 영역 내에 해당하는 룰들을 찾아 최종 영역의 엔트리에 저장하는 식으로 라우팅 테이블을 만든다. 따라서 들어오는 패킷에 맞는 룰을 검색할 때에도 검색의 매 단계에서 검색 영역을 줄여가면서 맞는 영역을 찾아 해당 룰들을 얻고, 최종적으로 해당 영역에 저장된 룰들을 검색하여 정확히 일치하면서 가장 순위가 높은 룰로 결정하면 된다.

표 1에 5개의 필드로 이루어진 룰 셋의 예시를 보였다. 처음 두 필드인 F1, F2 필드는 근원지 주소와 목적지 주소에 관한 필드로서 프리픽스 매칭이 이루어져야 하는 필드이다. 여기서 프리픽스 매치란 입력된 패킷의 해당 헤더 필드를 F1 혹은 F2 필드에 주어진 프리픽스 길이까지만을 비교하는 것을 의미한다. F3와 F4 필드는 근원지 및 목적지의 포트번호에 관한 필드로서 영역 매칭이 이루어져야 하는 필드이다. 여기서 영역 매치란 입력된 패킷의 해당 헤더가 F3 혹은 F4 필드에 주어진 영역에 포함되는 지를 확인하는 것을 의미한다. F5 필드는 프로토콜 타입에 관한 필드로서 확정 매칭(exact matching)이 이루어져야 하는 필드이다. 확정 매치란 입력된 패킷의 해당 헤더가 F5 필드에 주어진 값과 정확하게 일치하는지를 확인하는 것을 의미한다. 설명의

표 1 룰 셋

Rule #	F1	F2	F3	F4	F5	action
R1	0111	011*	0-255	23	TDP	act0
R2	010*	011*	0-255	23	UDP	act1
R3	*	111*	0-255	0-255	TCP	act0
R4	000*	*	15	23	UDP	act2
R5	1*	1*	0-255	0-255	TCP	act1
R6	0111	*	15	213	UDP	act2
R7	001*	0*	0-255	0-255	UDP	act1
R8	*	011*	15	23	UDP	act0

표 2 영역으로 표현한 룰 셋

Rule #	F1	F2	F3	F4	F5	action
R1	7	6-7	0-255	23	1	act0
R2	4-5	6-7	0 255	23	0	act1
R3	0-15	14 15	0-255	0-255	1	act0
R4	0-1	0-15	15	23	0	act2
R5	8-15	8-15	0-255	0-255	1	act1
R6	7	0-15	15	213	0	act2
R7	2-3	0-7	0-255	0-255	0	act1
R8	0-15	6-7	1115	23	0	act2

편의를 위하여 F1 및 F2 필드의 프리픽스 값에 대하여 4 비트를 최대 길이로 가정하여 전체 영역은 0-15로 표현하였고, 필드 3와 필드 4의 전체 영역은 0 255라고 가정하였다. 표 2는 표 1의 룰들을 십진수의 영역으로 표현한 결과이다.

커팅을 이용하여 영역을 분할하는 기존의 방식으로 HiCuts[3]과 HyperCuts[4]을 살펴본다.

2.1 HiCuts

HiCuts은 매 분할 과정에서 룰들의 특성을 파악하여 영역을 분할할 필드와 몇 개의 영역(일정한 간격)으로 분할할 것인지를 정한다. 영역을 분할할 필드를 선정하는 기준은 유니크(unique)한 값의 종류가 많은 필드를 선택한다. [6]에 따르면 실제 백 본 라우터에서 사용되는 분류 테이블을 분석하여 보았을 때, 근원지 및 목적지 프리픽스의 종류가 가장 다양하여 이러한 두 개의 필드를 사용하여 일치 가능한 룰을 거르는 경우, 95%이상의 입력 패킷이 5개 이하의 룰과 일치 가능한 것으로 보고하고 있다. 그러므로, 먼저 근원지 및 목적지 프리픽스를 사용하여 영역을 분할하는 것이 효율적이다.

다음 단계는 분할된 영역을 바탕으로 디시전 (decision) 트리를 구성하는데, 디시전 트리의 리프(leaf) 노드에는 미리 정해진 수 이하의 룰을 가지고 있게 한다. 여기서 리프 노드에 포함될 미리 정해진 룰의 개수를 빈스(binth)라고 부른다. 검색 과정은 디시전 트리를 따라 내려가면서 리프 노드에 도달하면 그 노드에 저장된 룰에 대하여 순차적으로 비교한다. HiCuts는 리프에 저장된 룰의 수를 일정한 수(빈스) 이하로 하였으므로, 검색 시간을 조절할 수 있다. 다시 말하면 어떠한 필드에 대해 영역을 분할하여 룰들을 분류하였을 때, 특정 영역에 해당하는 룰들의 개수가 빈스를 넘으면 그 영역은 다른 필드에 대하여 다시 분할한다. 표 1에서 보여진 룰 셋을 예로 하여 HiCuts을 설명하면 다음과 같다.

그림 1은 표 2를 토대로 HiCuts를 실행하는 과정을 영역으로 표현하였다. 이를 토대로 빈스를 2로 가정하여 HiCuts를 실행하면 다음과 같다. 이 디시전 트리를 구성하는 과정 및 완성된 디시전 트리를 그림 2에 보였다. 먼저 유니크(unique) 프리픽스가 가장 많은 필드 1에 대해 4개의 영역으로 분할하면 영역 [0,3], [4,7], [8,11], [12,15]로 나누어지고, 각각에 4, 5, 3, 3개의 룰이 속하게 된다. 따라서 4개의 영역에 속한 룰의 개수는 모두 빈스를 초과하므로 이들에 대해 각각 필드 2의 프리픽스들을 이용해 영역을 분할한다. 필드 2에 대하여 영역 분할의 결과, 여전히 빈스를 넘는 개수의 룰을 포함하는 영역 F1[0,3], F2[4,7]과 F1[4,7], F2[4,7]이 있어, 이 영역을 다시 필드 4를 사용하여 분할한다. 그 결과, 다시 F1[0,3], F2[4,7], F4[0,127]과 F1[4,7], F2[4,7], F4[0,127]에 해당하는 룰의 개수는 빈스를 초과한다. 따라서 각각 필드 3과 필드 5로 영역을 분할한다. 그 결과, F1[0,3], F2[4,7], F4[0,127], F3[0,127]영역은 빈스를 초과하는 룰들이 해당하므로 또 다시 룰을 분류해야 한다. 마지막으로 필드 5를 사용해 이 룰들을 분류하면 각 영역에 여전히 빈스를 초과하는 룰들이 해당한다. 그러나

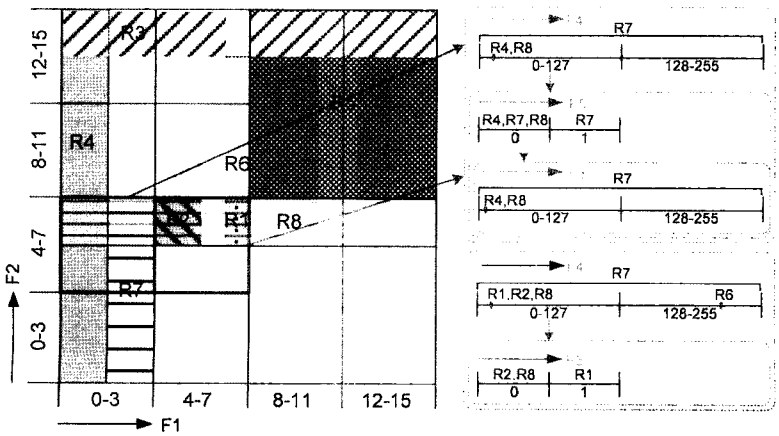
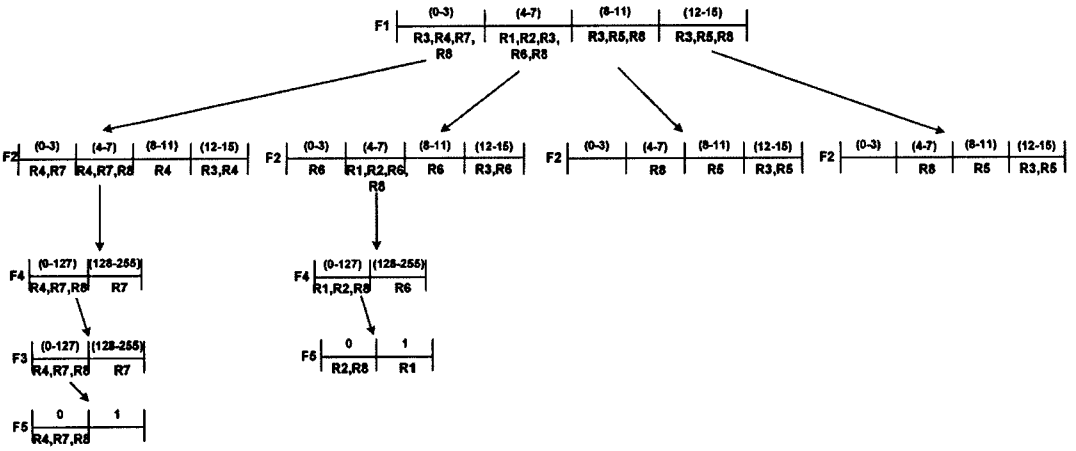
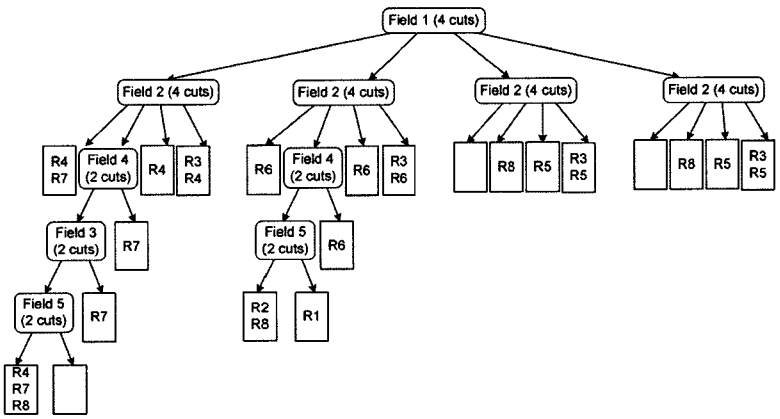


그림 1 HiCuts의 실행 과정을 영역으로 표현한 예



(a) HiCuts를 실행하는 과정



(b) 디시젼(Decision) tree

그림 2 HiCuts의 디시젼 트리를 구성한 예

이 룰들에 대하여 이미 모든 필드를 사용하여 분류하였으므로 더 이상 분류를 진행하지 못하고 리프에 이 해당 룰들을 저장해야 한다. 이와 같이 모든 필드를 사용하여 룰들을 분류하였으나 효과적인 영역 분할을 하지 못하면 트리의 높이는 커지는 데 반해, 빈스를 초과하는 리프가 생성될 수 있다.

각 필드 별 정보가 (0111, 0110, 100, 23, UDP)와 같은 패킷이 입력되었음을 가정하면, 검색 방법은 다음과 같다. 먼저, 디시젼 트리에서 필드 1에 대하여 전체 영역을 4개로 분할하였으므로 패킷의 필드 1 헤더인 0111은 [4,7]의 영역에 포함된다. 따라서 두 번째 가지를 따라간다. 다음, 필드 2에 대하여 4개 영역으로 분할하였으므로 패킷의 필드 2의 헤더인 0110은 [4,7]영역에 포함되므로 현재 엔트리에서 두 번째 가지를 따라간다. 다음, 패킷의 필드 4에 대하여 전체 영역[0,255]을 2개로 분할하였고 패킷의 필드 4이 값인 23은 첫번째 영역에 속하므로, 첫번째 가지를 따라간다. 다음, 필드 5에 대하여 2개 영역으로 분할하였으므로 패킷의 필드 5 헤더인 UDP는 두 번째 가지를 따라간다. 리프 노드를 만났으므로, 리프 노드에 속한 룰들과 비교한다. 먼저, 패킷이 우선 순위가 높은 룰 R2과 매치하므로 다음 룰 R8에 대하여 검색을 진행하지 않고 최종 best matching rule(BMR)은 R2가 된다.

2.2 HyperCuts

HiCuts의 경우, 룰 셋의 분포에 따라 디시젼 트리의 높이가 커져 검색을 위해 메모리 접근 횟수가 커질 수 있다는 단점이 있다. HyperCuts의 경우에는 한번에 여러 필드의 영역을 동시에 검사하여 높이를 2로 제한하는 디시젼 트리를 구성한다. 즉, HyperCuts에서는 먼저 각 필드별로 유니크 프리픽스 영역의 개수 및 그의 평

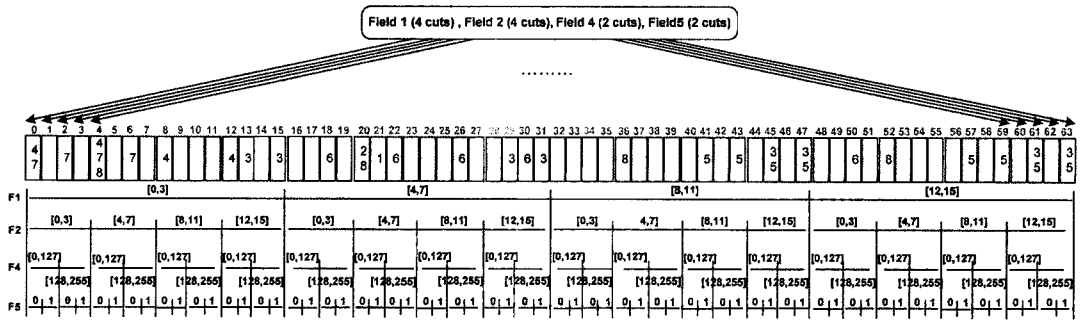


그림 3 HyperCuts의 디시젼 트리를 구성한 예

균값을 찾고, 각 필드 별로 유니크 값의 개수가 전체 평균의 이상인 필드들을 찾는다. 이 필드들을 영역을 분할할 필드로 결정하고 각각에 대해 몇 개의 영역으로 분할할 것인지를 결정한다. 그 다음, 이 필드들의 각 분할 영역들로 조합 가능한 모든 영역에 대해 룰들을 분류한다. 즉, $nc(i)$를 필드 i의 컷의 수라 할 때, 가능한 모든 영역의 수는 $\prod_{i \in D} nc(i)$이다. 따라서 한번에 하나의 필드로 룰들을 분류해 나가는 HiCuts의 경우와 달리 HyperCuts은 여러 필드들로 동시에 룰들을 분류하므로 검색 속도가 더 향상된다. 그러나 HyperCuts의 경우에는 여러 필드들을 동시에 사용하여 영역을 분할하므로 엔트리 수가 훨씬 늘어날 수 있다는 단점이 있다. 표 2를 토대로 HyperCuts를 수행하여 디시젼 트리를 구성한 결과를 그림 3에 보였다. 디시젼 트리를 구성함에 있어 HiCuts와 마찬가지로, F1 및 F2에 대하여 각각 4개의 컷을, F4에 대하여 2개의 컷을 적용하였다.

그림 2와 그림 3을 비교하여 보면, HyperCuts는 HiCuts와 비교하여 트리의 높이가 2로 제한되어 메모리 접근 횟수를 훨씬 줄일 수 있다는 장점이 있다. 그러나 이 예에서 보듯이 HiCuts의 엔트리 수가 30인데 비하여 HyperCuts의 경우, 엔트리 수가 64개이며 이 차이는 컷 수가 커질수록 더 커질 것이다. 또한 룰을 갖지 않는 빈 엔트리가 생성되어 메모리 낭비가 심해진다.

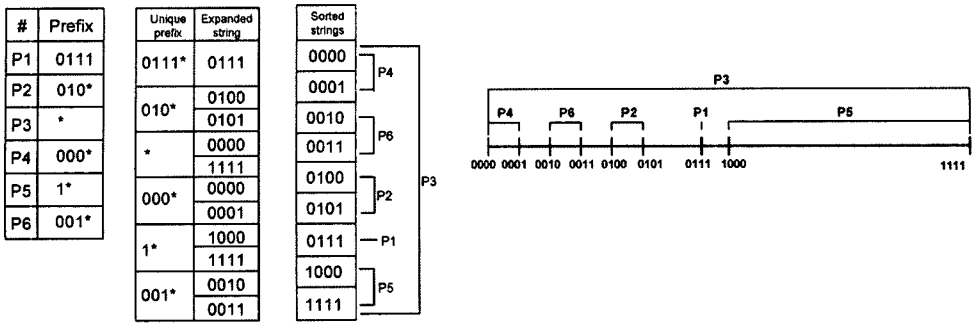
검색 과정은 HiCuts와 마찬가지로 간단하다. HiCuts에서와 같이 각 필드 별 정보가 (0111, 0110, 100, 23, UDP)인 패킷이 들어왔다고 가정하면, 검색 방법은 다음과 같다. 디시젼 트리에서 필드 1, 필드 2, 필드 4, 필드 5를 이용하여 영역을 분할하였고 각 필드의 전체 영역을 각각 4, 4, 2, 2개로 분할하였으므로 이를 이용하여 패킷을 분류하여 트리를 따라간다. 즉, F1필드에 대하여 1번 가지, F2필드에 대하여 1번 가지, F4필드에 대하여 0번 가지이므로 이 영역들을 모두 합치면 패킷이 속하는 영역은 (1*4*2*2)+(1*2*2)+(0*2)+0=20가 되어 20번 가지를 따라가면 된다. 그 결과, R2과 R8이 저장되어 있으므로 이들에 대하여 비교한다. 먼저, 패킷이 우선순위가 높은 룰 R2와 매치하는지 검사한다. 그 결과, R2와 매치하므로 R8에 대해 검색을 하지 않고 최종 BMR은 R2로 결정한다. 이 예에서 볼 수 있듯이 HiCuts의 검색 과정과 비교하여 HyperCuts는 한번의 수학적 계산으로 트리의 리프에 도달하므로 HiCuts보다 검색 속도가 향상된다.

3. 기존의 IP 주소 검색을 위한 영역 분할 방식

제안하는 알고리즘을 보기 전에 먼저 영역 검색에 기초한 방식으로 binary search on range (BSR)[5] 알고리즘을 살펴본다. IP 주소 검색은 라우팅 프리픽스로 주어진 프리픽스 셋에 대하여 입력된 패킷의 목적지 IP 주소와 가장 길게 일치하는 프리픽스를 찾아 그 프리픽스에서 지정된 출력 포트로 입력된 패킷을 전달하여 주는 일련의 과정을 말한다. 입력된 패킷의 목적지 주소와 일치하는 여러 개의 프리픽스를 찾고, 그 중 가장 길게 일치하는 프리픽스를 찾아야 한다는 점에 그 어려움이 있다.

BSR은 IP 주소 검색을 위한 알고리즘으로 라우팅 테이블의 프리픽스들을 영역으로 표현하되, 원래 프리픽스의 뒤에 '0'을 붙여 그 최대 길이로 확장시킨 것을 시작점으로, '1'을 붙여 최대 길이로 확장시킨 것을 끝점으로 갖는 영역으로 표현한다.

예를 들어, 001*를 길이 4로 확장시킨다면 이 프리픽스는 전체 영역 [0000, 1111]에서 [0010, 0011]에 해당된다. 그림 4에 영역 분할 과정의 예를 보였다. 그림 4(a)의 프리픽스들에 대하여 각 프리픽스에 해당하는 영역을 시작점과 끝점으로 표현하면 그림 4(b)와 같다. 확장된 스트링들을 크기 순으로 나열하여 전체 영역을 분할하는데, 중복된 스트링이 있을 시에는 하나만 취한다. 결과적으로 전체 영역은 스트링들을 영역의 분할점으로 갖는 서로 겹침이 없는 작은 영역들로 분할되었다. 예를



(a) Prefix Set (b) 유니크 프리픽스들을 확장시킨 결과 (c) 영역 분할점 (d) 분할된 각 영역에 속하는 프리픽스

그림 4 영역을 분할하는 과정

들어, 위에서 확장된 스트링들을 크기 순으로 나열하면 그림 4(c)와 같다. 그림 4(d)는 이와 같이 분할된 영역에 속하는 프리픽스들을 나타내었다. BSR 알고리즘은 이와 같이 크기순으로 나열된 분할점들을 사용하여 라우팅 테이블을 만드는데, 라우팅 테이블의 각 엔트리에는 초 과(>)포인터와 일치(=) 포인터를 미리 계산하여 저장한다. 초과 포인터에는 엔트리의 값 보다 큰 프리픽스 중 가장 긴 프리픽스를, 일치 포인터에는 현재 엔트리의 값과 가장 길게 매치하는 프리픽스를 각각 저장한다. 그림 4의 과정을 거쳐 라우팅 테이블을 구성한 예를 표 3에 보였다.

검색 과정은 입력된 패킷의 목적지 주소에 대하여 이진 검색을 진행하며 best matching prefix (BMP)를 업데이트하여 나가되, 입력된 목적지 주소와 엔트리의 값과 일치하면 일치 포인터에 저장된 값을 출력하고 검색을 종료한다. 입력된 목적지 주소가 엔트리의 값보다 크면 초과 포인터에 저장된 프리픽스로 BMP를 업데이트한다. 더 이상 진행될 엔트리가 없을 때, 마지막으로 업데이트된 BMP를 출력하고 검색을 종료한다.

BSR은 모든 프리픽스를 같은 길이로 확장하기 때문에 가장 길게 일치하는 프리픽스를 찾는 IP주소 검색에서 길이 차원을 완전히 없애고 이를 이용해 균형 이진 검색을 할 수 있다는 점에서 매우 우수한 알고리즘이다. 그러나 예에서 보듯이 BSR의 경우 라우팅 테이블의 엔트리의 수는 9이다. 즉, BSR은 라우팅 테이블의 엔트리 수가 원래 프리픽스의 수보다 최대 2배로 늘어날 수 있다는 한계를 가진다.

표 3 BSR의 라우팅 테이블 예시

영역 시작점	>	=
0000	P4	P4
0001	P6	P4
0010	P6	P6
0011	P2	P6
0100	P2	P2
0101	P3	P2
0111	P5	P1
1000	P5	P5
1111	-	P5

4. 제안하는 알고리즘

HiCuts은 한번에 한 필드의 분할 영역으로만 룰들을 분류하므로 룰들의 분포에 따라 트리의 높이가 커질 수 있다는 단점이 있다. HyperCuts은 이러한 트리의 높이를 제한하기 위해 동시에 여러 필드를 사용해 더 구체적인 영역으로 나눈다. 그러나 이는 엔트리의 개수가 훨씬 늘어날 수 있는 단점이 있다. 이 장에서는 HiCuts과 같이 엔트리 수를 적게 하면서 HyperCuts과 같이 트리의 높이를 줄이는 구조를 제안한다.

먼저 두 경우 모두 주목할 점은 룰셋의 분포 영역과는 상관없이 일정한 간격으로 영역을 분할한다는 점이다. HiCuts의 경우, 매 단계마다 영역을 줄여가면서 룰들을 분류해 나가지만 효율적으로 분류하지 못하기 때문에 트리의 높이가 커지게 된다. 따라서 한번의 메모리 접근으로 다음 검색할 룰의 개수를 효율적으로 줄이지 못하게 된다. HyperCuts의 경우는 이 트리의 높이를 줄이기 위해 한번에 여러 필드를 사용해 룰들을 분류하였으나 이 역시 일정한 간격으로 영역을 분할하는 방식을 사용하며 엔트리 수가 비효율적으로 커지는 단점이 있다.

본 논문에서는 BSR에서 사용된 방식인 프리픽스들이 표현하는 영역을 이용하여 전체 영역을 분할하는 새로운 알고리즘을 제안한다. 그러나 BSR 방식은 엔트리의 수가 최대 2배로 늘어날 수 있다는 한계가 있는데, 본 논문에서는 BSR 방식에서의 시작점 만을 사용하여 전

체 영역을 분할하는 방식을 제안하여, 엔트리의 수를 증가시키지 않는다.

4.1 라우팅 테이블 구성(Build)

룰들을 분류하여 라우팅 테이블을 만드는 방식은 HiCuts의 방식과 유사하나 제안하는 구조에서는 좀 더 효율적인 영역 분할 방식을 사용한다.

4.1.1 영역을 분할할 필드 선택

영역을 분할할 필드를 선택할 때 고려할 점은 한번의 분할로서 각 영역에 속하지 않는 최대한 많은 룰을 제거해야 한다는 점이다. 따라서 아직 선택되지 않은 필드들 중에서 유니크(unique) 정보가 가장 많은 필드를 영역을 분할할 필드로 선택한다. 여기서 유니크하다는 것은 정보가 서로 같지 않다는 것을 의미한다. 일반적으로 실제의 룰 셋에서는 필드 1과 필드 2, 즉 목적지 주소와 근원지 주소 정보를 갖는 두 필드에서의 유니크 정보는 그 외 필드(근원지 포트 범위, 목적지 포트 범위, 프로토콜 타입)보다 매우 많다. 결과적으로 위의 조건 하에서 필드를 선택하면 일반적으로 트리의 높이 2까지는 필드 1와 필드 2가 선택된다.

4.1.2 선택된 필드의 영역 분할

위에서 본 커팅 알고리즘들은 룰셋의 분포 영역과는 상관없이 일정한 간격으로 영역을 분할한다. HiCuts의 경우를 보면 매 단계마다 영역을 줄여가면서 룰들을 분류해 나가지만 비효율적으로 분류하기 때문에 트리의 높이가 커지게 되어 한번의 메모리 접근으로 다음 검색할 룰의 수를 효율적으로 제거하지 못한다. HyperCuts의 경우는 이 트리의 높이를 줄이기 위해 한번에 여러 필드를 사용해 룰들을 분류하였으나 이 역시 일정한 간격으로 영역을 분할하는 방식을 사용하며 엔트리 수가 비효율적으로 커지는 단점이 있다.

제안하는 알고리즘에서는 룰들의 분포 영역 특성을 적용하는 방법으로 BSR의 영역 분할 방식을 적용한다. BSR에서의 하나의 프리픽스는 원래 프리픽스의 뒤에 각각 '0'과 '1'을 붙여 그 최대 길이로 확장시킨 두 개의 스트링으로 인코딩한다. 이 스트링들은 중복되는 스트링을 제외하고 분할된 영역의 경계점이 되어 라우팅 테이블에 저장된다. 따라서 엔트리의 수가 원래 프리픽스들의 수보다 최대 2배로 증가하는 단점이 있다. 제안하는 알고리즘에서는 BSR과 같은 영역분할 방식을 적용하되, 경계점의 수 혹은 엔트리의 수를 최소화하는 방법을 제안한다. 제안하는 방법은 다음과 같다.

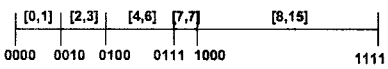
영역을 분할할 필드가 필드 3이나 필드 4와 같이 길이가 일정한 수로 구성된다면 현재 분류할 룰들에 대하여 선택된 필드의 유니크 영역만을 가져와 크기 순으로 정렬하여 분할 영역으로 사용한다.

영역을 분할할 필드가 필드 1이나 필드 2와 같이 프리픽스로 구성되는 경우, 엔트리의 수를 최소화하기 위하여 제안하는 방법은 BSR에서의 시작점만을 사용하여 영역을 분할한다. 다시 말하면, 현재 분류할 룰들에 대하여 선택된 필드의 유니크 프리픽스들에 대하여, '0'을 붙여 확장시키되, 현재 분류할 룰들 중의 최대 길이에 맞추어 확장시킨다. 예를 들어 표 1에서 필드 1을 첫번째 분류 필드로 선택하였다고 한다면, 첫번째 필드에서 유니크 프리픽스만을 가져오면 0111, 010*, *, 000*, 1*, 001*이 된다. 프리픽스 최대 길이는 4이므로 각 프리픽스에 '0'을 붙여 4비트로 확장시키면 그림 5(a)와 같이 0111, 0100, 0000, 0000, 1000, 0010이 된다. 제안하는 알고리즘은 이 확장된 스트링들을 이용해 현재 필드의 전체 영역을 분할하는데, 즉 인코딩된 스트링들을 크기 순으로 나열하며 중복된 스트링이 있을 시에는 하나만 취한다. 결과적으로 분할된 영역은 이 나열된 스트링들을 영역의 시작점으로 갖는 영역들이다. 예를 들어, 위에서 인코딩된 스트링들을 크기 순으로 나열하면 0000, 0010, 0100, 0111, 1000이며 이들은 각 영역의 시작점이므로, 전체 영역 [0,15]에 대하여 분할된 영역은 [0,1], [2,3], [4,6], [7,7], [8,15]이다. 이 결과를 그림 5(b)에서 보였다. 위의 BSR 예시에서 현재 룰 셋의 필드 1과 같은 프리픽스를 예를 사용하였으므로 제안한 알고리즘으

Unique prefix	Expanded string
0111	0111
010*	0100
*	0000
000*	0000
1*	1000
001*	0010

(a) 유니크 프리픽스들을 확장시킨 결과

Sorted string	Range
0000	[0,1]
0010	[2,3]
0100	[4,6]
0111	[7,7]
1000	[8,15]



(b) 분할된 영역

그림 5 필드 1을 처음 분류로 시작할 시 필드 1의 분할된 영역 예시

로 영역을 분할하면, 분할된 영역의 수를 최소화할 수 있음을 볼 수 있다. 즉, 제안하는 구조에서는 분할 영역의 수 즉, 라우팅 테이블의 엔트리 수가 많아야 원래 유니크 프리픽스 수와 같아 매우 효율적임을 알 수 있다.

4.1.3 디시젼(Decision) 트리 구성

디시젼 트리를 구성하는 방식은 기존의 커팅 알고리즘 HiCuts의 방식과 유사하다. 즉, 선택된 필드로 룰을 분류한 다음, 빈스보다 많은 룰이 속하는 영역은 다시 영역을 분할할 다음 필드를 선택하여 룰을 분류한다. 표 1을 룰 셋에 대하여 제안하는 알고리즘으로 디시젼 트리를 구성하면 그림 6과 같다. 그림에서와 같이 룰을 분류한 결과, 그림 2와 비교하여 제안한 알고리즘은 트리의 높이 4만으로도 각 엔트리가 갖는 룰의 수가 빈스를 넘지 않아, 효과적으로 룰을 분류하였음을 볼 수 있다.

제안한 알고리즘은 기존의 커팅 알고리즘들의 비효율적인 커팅 개념을 사용하지 않고 BSR의 영역 분할 방식을 개선시켜 효과적으로 룰을 분류한다. 또한 분할 영역의 수를 최소화시켜 HyperCuts과 같이 엔트리 수를 크게 증가시키지 않으면서도 디시젼 트리의 높이를 감소시킨다. 표 1의 룰 셋에 대하여 표 4에 HiCuts, HyperCuts, 그리고 제안하는 알고리즘의 디시젼 트리의 높이 및 엔트리의 개수를 비교하였다. 이 결과에서 보듯이 HiCuts는 트리의 높이가 커지면서 엔트리 수도 많아지며, HyperCuts는 트리의 높이를 2로 제한시키나 엔트리 수가 HiCuts보다 훨씬 많아진다. 반면 제안한 알고리즘은 트리의 높이를 줄이면서도 효과적으로 룰을 분류하며 엔트리 수를 최소화한다.

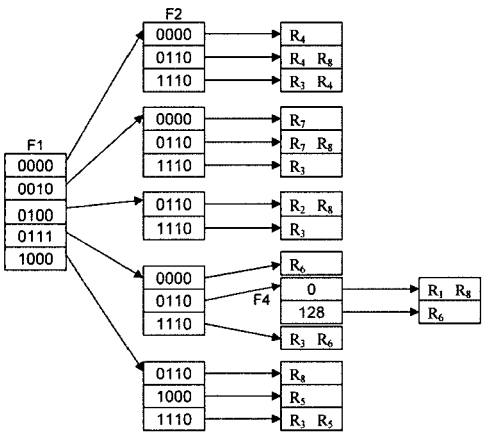


그림 6 제안된 알고리즘의 디시젼 트리

표 4 커팅 결과 비교

커팅 알고리즘	디시젼 트리의 높이	엔트리 수
HiCuts	6	30
HyperCuts	2	64
제안한 알고리즘	4	22

4.2 검색(Search)

제안하는 알고리즘의 검색 과정은 다음과 같다. 먼저 트리의 루트로부터 검색을 시작한다. 제안하는 알고리즘은 HiCuts와 HyperCuts의 경우와 같이 전체 영역을 일정한 간격으로 분할하지 않고 디스조인트한 영역으로 분할한다. 그러므로 BSR에서와 같이 이진 검색을 통하여 패킷이 속하는 영역을 찾아 해당하는 가지를 따라 아래 레벨로 진행한다. 즉, 현재 위치에서 선택된 필드를 패킷에서 취하여 이진 검색으로 패킷이 속하는 영역을 찾는다. 찾아간 영역이 리프가 아니면 이 과정을 반복하고 리프라면 저장된 룰들에 대하여 비교 검색을 진행한다. 리프에서 룰들은 우선 순위로 저장되어 있으므로 차례대로 비교 검색을 하되, 현재의 룰과 매치하면 다음 룰들에 대하여 검색을 하지 않고 현재 매치한 룰을 BMR로 결정하고 검색을 종료한다.

그림 6의 제안하는 알고리즘의 디시젼 트리의 경우 와일드 프리픽스가 많은 룰 셋에 대하여 많은 영향을 받음을 볼 수 있는데, 와일드 프리픽스가 많을수록 룰 복사가 많아져 엔트리 수가 많아진다. 따라서 필드 1의 정보가 와일드 프리픽스인 R3, R8이 필드 1의 모든 분할 영역에 속하게 되어 빈스를 초과하는 노드가 많아진다. 이는 필드 2에서 다시 한번 영역을 분할해 주어야 하며 그 결과, 또 다시 빈스를 초과하는 영역이 발생하게 된다. 그림에서와 같이 이 해당 영역에 대하여 필드 4로 다시 한번 영역을 분할해 주었다. 따라서 트리의 높이가 커지고 엔트리 수도 많아지는 것을 알 수 있다.

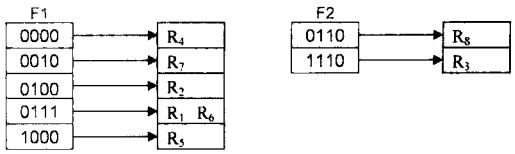
5. 제안하는 알고리즘의 최적화 구조

앞서 제안한 알고리즘에서는 유니크한 프리픽스로 서로 겹침이 없는 영역을 구성하여, 그 영역에 해당하는 룰을 저장하게 된다. 하지만 프리픽스의 길이가 짧을수록 그 프리픽스가 해당하는 영역이 많아지게 되어 빈스를 넘는 영역이 발생할 확률이 높아진다. 따라서 다음 필드를 선택하여 영역 분할을 계속해야 하는 경우 또한 많아진다. 즉, 짧은 프리픽스를 가진 룰은 여러 영역에 속할 확률이 커, 여러 영역에 반복하여 속하게 되므로 메모리 효율면에서 좋지 않은 영향을 미친다. 특히, 프리픽스의 길이가 0인 '*'(와일드 프리픽스)인 경우, 항상 모든 분할 영역에 속하게 되므로 가장 큰 영향을 준다. 이 장에서 제안하는 알고리즘의 최적화 구조는 짧은 프리픽스들 중에서도 와일드 프리픽스만을 따로 처리하는 방법을 사용하여 하나의 룰이 여러 영역에 속하게 되는 경우를 줄인다.

5.1 디시전 트리 구성

제안하는 알고리즘의 최적화 구조는 메인 트리와 와일드 카드 트리의 두 개의 트리를 생성한다. 메인 트리는 처음 영역 분할을 위해 선택되는 필드(필드 1 또는 필드 2)의 정보가 와일드 프리픽스(*)가 아닌 룰들로 구성되는데, 구성하는 방식은 전장에서 설명한 제안하는 알고리즘에서와 같다. 그 과정을 그림 7에 보였다. 즉, 각 영역에 해당하는 룰의 개수가 빈스를 초과한다면 다음 선택되는 필드에 대하여 영역 분할을 진행한다. 그렇지 않다면, 그 영역에 속하는 룰들을 저장한다. 메인 트리는 첫 단계에서 와일드 프리픽스들을 제외시키고 영역 분할을 진행하기 때문에 이 프리픽스들이 여러 영역에 복사되는 것을 막아 빈스를 초과하는 영역의 수도 그만큼 줄어들게 된다. 와일드 트리를 구성하는 방식은 그림 8에 보였다. 현재 선택된 필드의 정보가 모두 와일드인 프리픽스만으로 트리를 구성하므로 이 필드로는 영역을 분할하지 않고 두 번째 선택된 필드에 대하여 앞서 제안한 방법과 같은 방법으로 트리를 구성한다.

제안하는 알고리즘의 최적화 구조에서는 첫 분할 단계에서 와일드 프리픽스를 제외시킴으로써 와일드 카드를 갖는 룰들이 모든 리프에 복사되는 것을 막는다. 그림 9는 제안하는 알고리즘의 최적화에 따른 디시전 트리로서, 그림 9(a)는 메인 트리이고 그림 9(b)는 와일드 트리이다. 그림 6과 비교하여 그림 9(a)는 필드 2로의 영역 분할이 필요없게 되어 엔트리 수도 훨씬 줄어드는



(a) 메인 트리 (b) 와일드카드 트리

그림 9 제안된 최적화 기법

효과를 볼 수 있다. 또한 제안하는 구조는 프리픽스들 간의 분포 영역 특성을 적용하여 영역을 분할하기 때문에 그림 9(b)에서와 같이 [0,5]영역에 아무런 룰도 해당하지 않는 경우에 대하여 엔트리를 생성하지 않는다. 따라서 빈 엔트리가 생겨 메모리를 낭비하게 되는 문제를 제거하는 효과를 볼 수 있다.

5.2 검색

메인 트리에서의 검색 과정은 먼저, 메인 트리에서 검색을 진행하여 메인 트리에서의 BMR을 찾는다. 이를 메인 트리 BMR이라 칭한다. 일반적으로 와일드 프리픽스를 갖는 룰들은 낮은 우선 순위를 갖는다는 사실에 기초하여, 메인 트리 BMR을 찾은 후에는 와일드 트리에서의 검색을 추가적으로 진행할 것인지를 결정한다. 다시 말하면 와일드 트리에서의 가장 높은 우선순위를 갖는 룰을 임계값으로 정하여, 만약 메인 트리 BMR이 이 임계값보다 높은 우선순위를 갖는다면 와일드 트리에서의 검색을 진행할 필요가 없으며, 그렇지 않은 경우

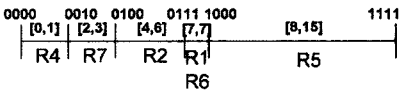
#	Prefix
R1	0111
R2	010*
R4	000*
R5	1*
R6	0111
R7	001*

(a) 와일드 프리픽스를 제외한 룰(필드 1)

Unique prefix	Expanded string
0111	0111
010*	0100
000*	0000
1*	1000
001*	0010

(b) 유니크 프리픽스들을 확장시킨 결과

Sorted string	Range
0000	[0,1]
0010	[2,3]
0100	[4,6]
0111	[7,7]
1000	[8,15]



(c) 분할된 영역

그림 7 메인 트리 분할영역 예시

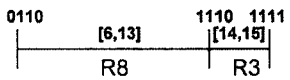
#	Prefix
R3	111*
R8	011*

(a) 룰 셋(필드 2) <필드 1이 와일드 프리픽스>

Unique prefix	Expanded string
111*	1110
011*	0110

(b) 유니크 프리픽스들을 확장시킨 결과

Sorted string	Range
0110	[6,13]
1110	[14,15]



(c) 분할된 영역

그림 8 와일드 트리 분할영역 예시

에만 와일드 트리에서의 검색을 진행한다. 와일드 트리의 리프에서의 비교 검색 또한 메인 트리 BMR 보다 우선순위가 높은 룰들에 대해서만 수행한다.

6. 시뮬레이션 결과 및 성능 평가

본 논문에서는 제안된 기본 알고리즘과 최적화 알고리즘의 성능을 분석하기 위해 클래스 벤치 [7]에서 제공된 데이타베이스 ACL1k, ACL5k, FW1k, FW5k, IPC1k, IPC5k에 대하여 시뮬레이션을 수행하였다. 그 결과를 비교 분석한 결과는 표 5와 같다. 표에서 Tavg는 평균 메모리 접근 횟수, Tmax는 최대 메모리 접근 횟수, Tmin은 최소 메모리 접근 횟수를 의미한다. 또한 와일드 수는 처음 룰 분류를 시작하기 위하여 선택한 필드에 대하여 와일드 프리픽스를 가진 룰의 수이다.

여기서 빈스는 4로 하였으며 메모리 요구량은 그림 10과 같은 엔트리 구조를 갖는 라우팅 테이블에 대하여 계산한 수치이다. 그림에서 보듯이 분할된 영역과 다른 엔트리로의 포인터를 저장하는 엔트리와 룰을 저장하는 엔트리의 두가지가 있다. 첫번째 엔트리에서 영역(Range) 부분은 현재 선택된 필드로 분할된 각 영역을 나타내고 시작 포인터(Start_ptr)와 끝 포인터(End_ptr)가 의미하는 것은 두가지 경우에 해당한다. 즉, 현재 엔트리가 의미하는 영역에 속하는 룰의 개수가 빈스를 초과하는 경우라면 그 영역에 속하는 룰들에 대하여 다음 선택된 필드로 다시 영역을 분할해야 한다. 따라서 이 경우에 시작과 끝 포인터는 이 엔트리의 영역에 속하는 룰들에 대하여 분할된 영역들을 저장하는 엔트리들을 가르킨다. 즉, 이 엔트리들의 시작 인덱스(index)와 끝 인덱스를 나타낸다. 그렇지 않은 경우, 즉 현재 엔트리가 나타내는 영역에 속하는 룰들의 개수가 빈스를 초과하지 않는다면, 더 이상 룰들을 분류하지 않고 여기에 속하는 룰들을 저장해야 한다. 따라서 이 엔트리의 시작과 끝 포인터들은 현재 엔트리의 영역에 속하는 룰들이 저장된 엔트리의 시작 인덱스와 끝 인덱스를 나타낸다. 두번째 엔트리는 앞에서 말한 룰들을 저장해야 하는 경우, 그 영역에 속하는 룰들을 저장한다. 따라서 룰들을 저장해야 하는 경우에만 첫번째 엔트리에서 두번째 엔트리로 포인터를 가진다.

표 5에서 보듯이 제안하는 알고리즘의 최적화 구조는 전체 룰의 개수에 대하여 와일드 프리픽스를 가진 룰의 비율이 커질수록 기본 구조보다 메모리 측면에서의 성능이 우수하나, 검색 속도 측면에서의 성능은 저하된다. 최적화 구조는 처음 룰 분류를 시작할 시 선택된 필드에 대하여 와일드 프리픽스를 가진 룰들을 따로 분류하기 때문에 엔트리 수를 최소화시키는 장점을 가진다. 그러나 최악의 경우, 두 개의 디시젼 트리에 대하여 모두 검색을 진행해야 하므로 두 번째 디시젼 트리를 구성하는 룰의 수 즉, 처음 분류를 시작하기 위해 선택된 필드

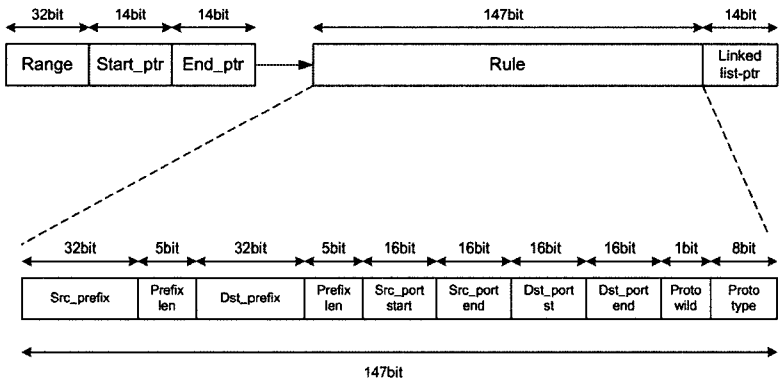


그림 10 제안하는 알고리즘의 엔트리 구조

표 5 제안된 알고리즘의 시뮬레이션 결과

Rule set	Rule #	Input #	Wild #	제안하는 기본 구조			제안하는 최적화 구조		
				Tavg	Tmax	Mem(Kbyte)	Tavg	Tmax	Mem(Kbyte)
Acl1k	968	4156	3	12.60	36	32.81	12.33	36	29.12
Acl5k	4660	13980	8	16.62	50	339.57	15.68	52	209.80
Fw1k	871	2613	32	15.21	29	185.39	16.93	35	85.29
Fw5k	3067	9201	114	19.03	49	337.19	20.62	57	162.73
Ipc1k	988	2971	27	14.28	18	216.29	17.93	24	89.87
Ipc5k	4468	20610	76	14.44	30	461.38	16.62	40	270.12

의 와일드 프리픽스 수가 많아질수록 이 트리가 커져 검색 속도에서의 저하를 보일 수 있다.

표 6은 기존의 커팅 알고리즘들 HiCuts와 HyperCuts와 제안하는 알고리즘에서 디시젼 트리를 구성하기 위하여 필요한 노드의 수 즉, 분할 영역의 수를 비교한 표이다. 여기서 HiCuts과 HyperCuts의 경우는 빈스와 각 선택된 필드에서 몇 개의 영역으로 분할하는가에 따라 그 성능이 좌우된다. 따라서 제안하는 알고리즘의 성능을 최대한 동등한 조건 하에 평가하기 위하여 HiCuts과 HyperCuts의 시뮬레이션 조건을 다음과 같이 주었다. 빈스는 제안하는 구조와 같이 4로 주되, 각 선택된 필드에서 분할할 영역의 수는 현재 선택된 필드의 유니크한 정보(예를 들어 필드 1의 경우, 유니크한 프리픽스의 수)의 수를 세어 표현 가능한 가장 작은 2의 배수로 결정하였다. 그 결과, 표에서 보듯이 제안하는 알고리즘은 기존의 커팅 알고리즘들보다 훨씬 적은 수의 분할 영역을 가짐에도 불구하고 아래 표 7과 표 8에서 보듯이 검색 속도와 메모리 요구량에서 훨씬 우수한 성능을 보인다. 이는 제안하는 알고리즘에서는 디스조인트한 영역을 사용하여 룰들을 효과적으로 분류하기 때문이다. 또한 제안하는 최적화 구조가 기본 구조보다 더 적은 분할 영역의 수를 가짐을 볼 수 있는데 이것은 최적화 구조에선 여러 영역에 룰이 복사되는 문제를 줄였기 때문이다.

표 6 커팅 알고리즘들의 분할 영역 수 비교

Rule set	제안하는 기본 구조	제안하는 최적화 구조	HiCuts	HyperCuts
Acl1k	986	868	1388	131072
Acl5k	10387	6420	1057202	4194304
Fw1k	4399	2653	84290	65536
Fw5k	8434	4192	229440	131072
Ipc1k	6239	3469	140938	524288
Ipc5k	7803	6366	458880	262144

표 7과 표 8은 제안된 알고리즘의 성능을 평가하기 위하여 기존의 패킷 분류알고리즘들과 시뮬레이션 결과를 비교한 표이다. 비교를 위하여 시뮬레이션한 기존의 패킷 분류 알고리즘들로는 위에서 언급한 HiCuts, HyperCuts 외에 Hierarchical trie(H-trie)[8], Area-Based Quad-tree(AQT)[9], Priority-Based Quad-tree (PQT)[10], Bit Vector(BV)[11]을 추가하였다. 간략하게 이 알고리즘들을 소개하면 H-trie는 각 필드 별로 트라이를 구성하고 이 트라이들을 계층적으로 연결한 구조이다. AQT와 PQT는 룰의 각 필드를 영역으로 표현하고 영역을 줄여가면서 패킷이 속하는 가장 구체적인 영역을 찾는 방법이다. AQT는 프리픽스로 표현되는 두 필드를 동시에 보면서 전체 검색 범위를 1/4로 줄여가며 패킷이 해당하는 영역 내의 룰들을 얻는 알고리즘이다. PQT는 AQT를 개선하기 위하여 룰의 우선 순위를 고려하여 구성한 트리 구조이다. BV는 프리픽스 형태로 표현되는 두 필드를 이용하여 독립적인 이진 트라이를 구성하고, 각각의 필드 트라이의 프리픽스 노드마다 주어진 룰 셋의 사이즈에 해당하는 비트 벡터를 갖는 구조이다.

표 7은 제안된 알고리즘과 다른 알고리즘들의 시뮬레이션 결과, 평균 메모리 접근 횟수를 나타내며 그림 11과 그림 12는 기존의 커팅 알고리즘들과 비교한 그래프로 나타낸 것이다. 표에서 보듯이 제안된 알고리즘은 HiCuts와 HyperCuts 뿐만 아니라, 다른 여러 알고리즘들에 대하여 평균 메모리 접근 횟수가 훨씬 작다. 이는 라우터 내로 들어온 패킷 처리 속도 면에서 우수한 성

표 7 여러 알고리즘들과의 평균 메모리 접근 횟수 (Tavg) 비교

Rule Set	기본 구조	최적화 구조	HiCuts[3]	HyperCuts[4]	H trie[8]	AQT[9]	PQT[10]	BV[11]
Acl1k	12.6	12.3	19.2	15.4	77.2	38.6	35.6	66.0
Acl5k	16.6	15.7	51.8	47.8	84.0	50.1	59.6	64.1
Fw1k	15.2	16.9	37.8	33.8	52.1	369.3	197.9	196.6
Fw5k	19.0	20.6	142.9	138.9	69.2	660.5	571.1	738.8
Ipc1k	14.3	17.9	16.7	12.8	71.9	94.5	73.6	63.6
Ipc5k	14.4	16.6	52.9	48.9	85.6	344.8	202.1	151.9

표 8 여러 알고리즘들과의 메모리 요구량 (Kbyte) 비교

Rule set	기본 구조	최적화 구조	HiCuts	HyperCuts	H trie	AQT	PQT	BV
Acl1k	32.8	29.1	98.9	3029.4	82.9	56.4	29.9	153.3
Acl5k	339.6	209.8	42467.0	167403.6	401.5	200.2	145.6	2793
Fw1k	185.4	85.3	7369.6	9300.9	39.4	35.2	27.2	111.9
Fw5k	337.2	162.7	45572.6	50900.2	119.1	479.8	136.0	2340
Ipc1k	216.3	89.9	2249.2	4119.9	121.6	71.2	30.9	154.3
Ipc5k	461.4	270.1	39936.2	43089.4	224.7	234.3	139.9	2351

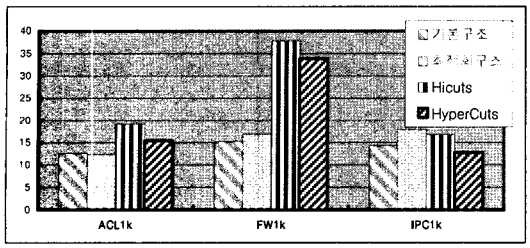


그림 11 1k 룰 셋에 대한 평균 검색 성능 비교

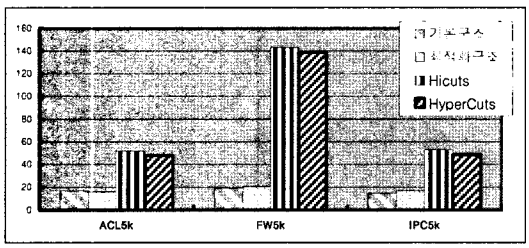


그림 12 5k 룰 셋에 대한 평균 검색 성능 비교

능을 보인다고 할 수 있다. 또한 데이타베이스 1k(Acl1k, Fw1k, Ipc1k)에서 5k(Acl5k, Fw5k, Ipc5k)로 룰의 수가 4~5배로 늘어남에도 불구하고 평균 메모리 접근 횟수는 크게 증가하지 않는다. 이에 반해 다른 알고리즘들은 룰의 수가 늘어남에 따라 이 수치도 크게 증가하는 점을 볼 수 있다. 이러한 점에서 볼 때, IPv6로의 확장에서도 우수한 패킷 처리 속도를 보일 것으로 예상된다.

표 8은 제안된 알고리즘과 다른 알고리즘들의 시뮬레이션 결과, 메모리 요구량을 나타낸다. 표에서 보듯이 제안하는 알고리즘의 메모리 요구량은 H-trie, AQT, PQT, BV와 같은 알고리즘들과 비교하여 유사하거나 약간 크지만 기존의 커팅 알고리즘들보다 훨씬 적은 메모리 요구량을 가짐을 볼 수 있다.

7. 결 론

패킷 분류는 단순히 목적지 주소만 확인하여 처리하는 IP 주소 검색과는 달리, 패킷 헤더 정보의 모든 필드를 검사하여야 한다. 따라서 패킷 처리 속도에 더 민감한 작업이라 할 수 있다. 따라서 패킷 분류를 효율적으로 수행하기 위한 여러가지 패킷 분류 알고리즘 및 구조에 관한 연구가 활발하게 수행되어 오고 있다. 본 논문에서는 패킷 분류 알고리즘 및 구조들 중 가장 효율적으로 알려진 커팅 알고리즘에 대하여 연구하였다.

본 논문에서는 기존의 커팅 알고리즘들의 한계점을 개선하여 효과적으로 룰을 분류하는 새로운 알고리즘을 제안하였다. 즉, HiCuts, HyperCuts 등에서 일정한 간격으로 영역을 두어 패킷을 분류해 가므로 룰셋의 룰들을 룰셋의 패턴에 맞춰 효과적으로 분류하지 못한다. 제안하는 알고리즘은 기존의 무조건 일정한 간격으로 영역을 나누는 방법을 이용하지 않고 룰 패턴을 적용하여 최소한의 디스조인트한 영역을 사용하여 매우 우수한 패킷 분류 속도를 보임을 확인하였다.

참 고 문 헌

[1] F. Baboescu and G. Varghese, "Scalable Packet Classification," in Proc. ACM SiGCOMM, Aug. 2001.

[2] A. Feldmann and S. Muthukrishnan, "Tradeoffs for packet Classification," IEEE INFOCOM, 2000.

[3] P. Gupta and N. Mckeown, "Classification using hierarchical intelligent cuttings," IEEE Micro, Vol.20, No.1, pp. 34-41, Jan./Feb., 2000.

[4] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," in Proc. SiGCOMM, 2003.

[5] B. Lampson, B. Srinivasan, and G. Varghese, "IP lookups using multiway and multicolumn search," IEEE/ACM Tans. Networking, Vol.7, pp. 324-334, June 1999.

[6] F. Baboescu, S. Singh, G. Varghese, "Packet classification for core router: is there an alternative to CAMs?," IEEE INFOCOM 2003, Vol.1, pp. 53-63, Mar. 2003.

[7] D. E. Taylor, J. S. Turner, J.S, "ClassBench: a packet classification benchmark," Proc. IEEE INFOCOM 2005, pp. 2068-2079, Mar. 2005.

[8] P. Gupta and N. Mckeown, "Algorithms for packet classification," IEEE Network, Vol.15, No. 2, pp. 24-32, Mar./Apr. 2001.

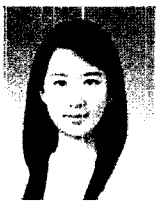
[9] M. M. Buddhikot, S. Suri, and M. Waldvogel, "Space decomposition techniques for fast layer-4 switching," in Proc. Protocols for High Speed Networks, pp. 25-41, Aug. 1999.

[10] Hyesook Lim, Min Young Kang, and Changhoon Yim, "Two-dimensional packet classification algorithm using a quad-tree," Computer Communications, Elsevier Science, Vol.30, No.6, pp. 1396-1405, Mar. 2007.

[11] T. V. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," in Proc. ACM SIGCOMM, Sep. 1998.

김 형 기

2007년 이화여자대학교 정보통신학과 졸업(학사). 2007년~ 이화여자대학교 전자공학과 석사과정. 관심분야는 라우터나 스위치 등의 네트워크 관련 SoC 설계



박 경 혜

2007년 이화여자대학교 정보통신학과 졸업(학사). 2007년~ 이화여자대학교 전자공학과 석사과정. 관심분야는 라우터나 스위치 등의 네트워크 관련 SoC 설계

임 혜 숙

1986년 서울대학교 제어계측공학과 졸업(학사). 1986년 8월~1986년 2월 삼성휴렛 팩커드 연구원. 1991년 서울대학교 제어계측공학과 졸업(석사). 1996년 The University of Texas at Austin, Electrical and Computer Engineering 졸업(박사). 1996년 11월~2000년 7월 Lucent Technologies Member of Technical Staff. 2000년 7월~2002년 2월 Cisco Systems, Hardware Engineer. 2002년 3월~ 이화여자대학교 공과대학 전자정보통신학과 부교수. 관심분야는 라우터나 스위치 등의 네트워크 관련 SoC 설계, TCP/IP 관련 하드웨어 설계