(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2019-0001871
(43) 공개일자 2019년01월07일

(51) 국제특허분류(Int. Cl.)
H04L 29/06 (2006.01) *H04L 12/851* (2013.01)
(52) CPC특허분류
H04L 69/22 (2013.01)
H04L 47/2441 (2013.01)
(21) 출원번호 10-2017-0082062
(22) 출원일자 2017년06월28일
심사청구일자 2017년06월28일

(71) 출원인
계명대학교 산학협력단
대구광역시 달서구 달구벌대로 1095, 계명대학교 산학협력관 201호(신당동)
(72) 발명자
박우길
대구광역시 달서구 월서로 51 101동 1806호 (진천동, 월배역 포스코 더 샵 아파트)
위재형
대구광역시 동구 입석로1길 56, 2층 안쪽집 (입석동)
(74) 대리인
김건우

전체 청구항 수 : 총 20 항

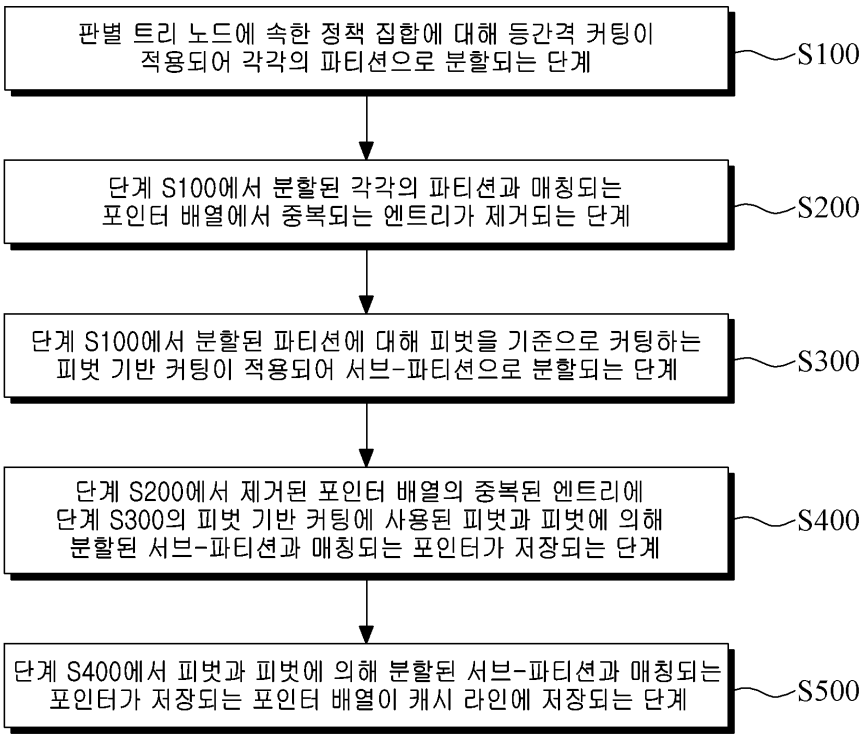
(54) 발명의 명칭 **하이브리드 커팅을 이용한 고속 패킷 분류 방법 및 시스템**

(57) 요 약

본 발명은 하이브리드 커팅을 이용한 고속 패킷 분류 방법에 관한 것으로서, 보다 구체적으로는 하나 이상의 차원에 대해 커팅 방식을 이용하여 패킷을 분류하는 패킷 분류 방법에 있어서, (1) 판별 트리 노드에 속한 정책 집합에 대해 등간격 커팅이 적용되어 각각의 파티션으로 분할되는 단계; (2) 상기 단계 (1)에서 분할된 각각의 파

(뒷면에 계속)

대 표 도 - 도2



티션과 매칭되는 포인터 배열에서 중복되는 엔트리가 제거되는 단계; (3) 상기 단계 (1)에서 분할된 파티션에 대해 피벗을 기준으로 커팅하는 피벗 기반 커팅이 적용되어 서브-파티션으로 분할되는 단계; (4) 상기 단계 (2)에서 제거된 상기 포인터 배열의 중복된 엔트리에 상기 단계 (3)의 상기 피벗 기반 커팅에 사용된 피벗과 상기 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 저장되는 단계; 및 (5) 상기 단계 (4)에서 상기 피벗과 상기 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 저장되는 포인터 배열이 캐시 라인에 저장되는 단계를 포함하는 것을 그 구성상의 특징으로 한다.

또한, 본 발명은 하이브리드 커팅을 이용한 고속 패킷 분류 시스템에 관한 것으로서, 보다 구체적으로는 커팅 방식을 이용하여 패킷을 분류하는 패킷 분류 시스템에 있어서, 판별 트리 노드에 속한 정책 집합에 대해 등간격 커팅을 적용하여 각각의 파티션으로 분할하는 등간격 커팅 모듈; 상기 등간격 커팅 모듈에 의해 분할된 각각의 파티션과 매칭되는 포인터 배열에서 중복되는 엔트리를 제거하는 중복 엔트리 제거 모듈; 상기 등간격 커팅 모듈에 의해 분할된 파티션에 대해 피벗을 기준으로 커팅하는 피벗 기반 커팅을 적용하여 서브-파티션으로 분할하는 피벗 기반 커팅 모듈; 상기 중복 엔트리 제거 모듈에 의해 제거된 상기 포인터 배열의 중복된 엔트리에 상기 피벗 기반 커팅 모듈의 상기 피벗 기반 커팅에 사용된 피벗과 상기 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터를 저장하는 포인터 배열 저장 모듈; 및 상기 피벗과 상기 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 저장되는 포인터 배열을 캐시 라인에 저장하는 캐시 라인 저장 모듈을 포함하는 것을 그 구성상의 특징으로 한다.

본 발명에서 제안하고 있는 하이브리드 커팅을 이용한 고속 패킷 분류 방법 및 시스템에 따르면, 2가지 커팅 방식을 동시에 사용함으로써, 정책 집합을 더욱 효율적으로 커팅할 수 있고, 이로 인해 정책 집합의 분포와 무관하게 더욱 빠른 패킷 분류 속도를 얻을 수 있으며, 판별 트리의 깊이와 생성되는 테이블의 크기 또한 크게 감소시킬 수 있다.

또한, 본 발명에 따르면, 메모리 액세스 시 함께 캐싱되는 메모리 영역 내에 필요한 엔트리들이 배치됨으로써, 메모리 액세스 수를 크게 감소시킬 수 있다.

이 발명을 지원한 국가연구개발사업
과제고유번호 2014R1A1A1038306
부처명 교육부
연구관리전문기관 한국연구재단
연구사업명 (구)신진연구지원사업
연구과제명 지속적 화상 관제를 위한 태양광 발전 기반 초저전력 무선 사물지능통신기술 연구
기 여 율 1/1
주관기관 계명대학교 산학협력단
연구기간 2016.07.01 ~ 2017.06.30

명 세 서

청구범위

청구항 1

커팅 방식을 이용하여 패킷을 분류하는 패킷 분류 방법에 있어서,

(1) 판별 트리 노드에 속한 정책 집합에 대해 등간격 커팅이 적용되어 각각의 파티션으로 분할되는 단계;

(2) 상기 단계 (1)에서 분할된 각각의 파티션과 매칭되는 포인터 배열에서 중복되는 엔트리가 제거되는 단계;

(3) 상기 단계 (1)에서 분할된 파티션에 대해 피벗을 기준으로 커팅하는 피벗 기반 커팅이 적용되어 서브-파티션으로 분할되는 단계;

(4) 상기 단계 (2)에서 제거된 상기 포인터 배열의 중복된 엔트리에 상기 단계 (3)에서 상기 피벗 기반 커팅에 사용된 피벗과 상기 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 저장되는 단계; 및

(5) 상기 단계 (4)에서 상기 피벗과 상기 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 저장되는 포인터 배열이 캐시 라인에 저장되는 단계를 포함하는 것을 특징으로 하는, 하이브리드 커팅을 이용한 고속 패킷 분류 방법.

청구항 2

제1항에 있어서, 상기 단계 (3)은,

상기 등간격 커팅이 적용되어 가장 많은 수로 커팅된 파티션을 가지는 차원에 대해서 상기 피벗 기반 커팅이 적용되는 것을 특징으로 하는, 하이브리드 커팅을 이용한 고속 패킷 분류 방법.

청구항 3

제2항에 있어서, 상기 단계 (3)은,

상기 단계 (1)에서 분할된 각각의 파티션에 대해 중복되는 정책 집합의 수를 최소화하는 피벗을 기준으로 상기 피벗 기반 커팅이 적용되는 것을 특징으로 하는, 하이브리드 커팅을 이용한 고속 패킷 분류 방법.

청구항 4

제1항에 있어서, 상기 단계 (3)은,

상기 단계 (1)에서 분할된 각각의 파티션에 대해 피벗 기반 커팅이 적어도 한 번 이상 적용되는 것을 특징으로 하는, 하이브리드 커팅을 이용한 고속 패킷 분류 방법.

청구항 5

제1항에 있어서, 상기 단계 (2)는,

상기 포인터 배열에 중복되지 않은 파티션과 매칭되는 고유 포인터만 저장되는 것을 특징으로 하는, 하이브리드 커팅을 이용한 고속 패킷 분류 방법.

청구항 6

제5항에 있어서, 상기 포인터 배열은,

각각의 엔트리가 인덱스가 저장되는 인덱스 필드와, 포인터 또는 피벗이 저장되는 공유 필드로 구성되는 것을 특징으로 하는, 하이브리드 커팅을 이용한 고속 패킷 분류 방법.

청구항 7

제6항에 있어서, 상기 포인터 배열은,

상기 공유 필드에 상기 피벗 기반 커팅에 사용된 피벗과 상기 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 저장되는 것을 특징으로 하는, 하이브리드 커팅을 이용한 고속 패킷 분류 방법.

청구항 8

제6항에 있어서, 상기 단계 (4)는,

상기 포인터 배열에서 상기 고유 포인터가 첫 엔트리부터 연속적으로 배치된 후, 상기 피벗 기반 커팅에 사용된 피벗과 상기 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 배치되는 것을 특징으로 하는, 하이브리드 커팅을 이용한 고속 패킷 분류 방법.

청구항 9

제6항에 있어서, 상기 포인터 배열은,

상기 피벗에서 상기 파티션 또는 서브-파티션의 시작값을 뺀 오프셋을 저장하여 32bit 피벗을 28bit의 공유 필드에 저장하는 것을 특징으로 하는, 하이브리드 커팅을 이용한 고속 패킷 분류 방법.

청구항 10

제1항에 있어서, 상기 단계 (5)는,

상기 포인터 배열이 하나의 캐시 라인에 저장되되, 상기 포인터 배열의 크기가 상기 하나의 캐시 라인 사이즈를 초과하는 경우, 하나의 캐시 라인 사이즈마다 새로운 헤더가 사용되는 것을 특징으로 하는, 하이브리드 커팅을 이용한 고속 패킷 분류 방법.

청구항 11

커팅 방식을 이용하여 패킷을 분류하는 패킷 분류 시스템에 있어서,

판별 트리 노드에 속한 정책 집합에 대해 등간격 커팅을 적용하여 각각의 파티션으로 분할하는 등간격 커팅 모듈;

상기 등간격 커팅 모듈에 의해 분할된 각각의 파티션과 매칭되는 포인터 배열에서 중복되는 엔트리를 제거하는 중복 엔트리 제거 모듈;

상기 등간격 커팅 모듈에 의해 분할된 파티션에 대해 피벗을 기준으로 커팅하는 피벗 기반 커팅을 적용하여 서브-파티션으로 분할하는 피벗 기반 커팅 모듈;

상기 중복 엔트리 제거 모듈에 의해 제거된 상기 포인터 배열의 중복된 엔트리에 상기 피벗 기반 커팅 모듈의 상기 피벗 기반 커팅에 사용된 피벗과 상기 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터를 저장하는 포인터 배열 저장 모듈; 및

상기 피벗과 상기 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 저장되는 포인터 배열을 캐시 라인에 저장하는 캐시 라인 저장 모듈을 포함하는 것을 특징으로 하는, 하이브리드 커팅을 이용한 고속 패킷 분류 시스

템.

청구항 12

제11항에 있어서, 상기 피벗 기반 커팅 모듈은,

상기 등간격 커팅이 적용되어 가장 많은 수로 커팅된 파티션을 가지는 차원에 대해서 상기 피벗 기반 커팅이 적용되는 것을 특징으로 하는, 하이브리드 커팅을 이용한 고속 패킷 분류 시스템.

청구항 13

제12항에 있어서, 상기 단계 피벗 기반 커팅 모듈은,

상기 등간격 커팅 모듈에서 분할된 각각의 파티션에 대해 중복되는 정책 집합의 수를 최소화하는 피벗을 기준으로 상기 피벗 기반 커팅이 적용되는 것을 특징으로 하는, 하이브리드 커팅을 이용한 고속 패킷 분류 시스템.

청구항 14

제11항에 있어서, 상기 피벗 기반 커팅 모듈은,

상기 피벗 기반 커팅 모듈에서 분할된 각각의 파티션에 대해 피벗 기반 커팅이 적어도 한 번 이상 적용되는 것을 특징으로 하는, 하이브리드 커팅을 이용한 고속 패킷 분류 시스템.

청구항 15

제11항에 있어서, 상기 중복 엔트리 제거 모듈은,

상기 포인터 배열에 중복되지 않은 파티션과 매칭되는 고유 포인터만 저장하는 것을 특징으로 하는, 하이브리드 커팅을 이용한 고속 패킷 분류 시스템.

청구항 16

제15항에 있어서, 상기 포인터 배열은,

각각의 엔트리가 인덱스가 저장되는 인덱스 필드와, 포인터 또는 피벗이 저장되는 공유 필드로 구성되는 것을 특징으로 하는, 하이브리드 커팅을 이용한 고속 패킷 분류 시스템.

청구항 17

제16항에 있어서, 상기 포인터 배열은,

상기 공유 필드에 상기 피벗 기반 커팅에 사용된 피벗과 상기 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 저장되는 것을 특징으로 하는, 하이브리드 커팅을 이용한 고속 패킷 분류 시스템.

청구항 18

제16항에 있어서, 상기 포인터 배열 저장 모듈은,

상기 포인터 배열에서 상기 고유 포인터를 첫 엔트리부터 연속적으로 배치한 후, 상기 피벗 기반 커팅에 사용된 피벗과 상기 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터를 배치하는 것을 특징으로 하는, 하이브리드 커팅을 이용한 고속 패킷 분류 시스템.

청구항 19

제16항에 있어서, 상기 포인터 배열은,

상기 피벗에서 상기 파티션 또는 서브-파티션의 시작값을 뺀 오프셋을 저장하여 32bit 피벗을 28bit의 공유 필드에 저장하는 것을 특징으로 하는, 하이브리드 커팅을 이용한 고속 패킷 분류 시스템.

청구항 20

제11항에 있어서, 상기 캐시 라인 저장 모듈은,

상기 포인터 배열이 하나의 캐시 라인에 저장되되, 상기 포인터 배열의 크기가 상기 하나의 캐시 라인 사이즈를 초과하는 경우, 하나의 캐시 라인 사이즈마다 새로운 헤더가 사용되는 것을 특징으로 하는, 하이브리드 커팅을 이용한 고속 패킷 분류 시스템.

발명의 설명

기 술 분 야

[0001] 본 발명은 패킷 분류 방법 및 시스템에 관한 것으로서, 보다 구체적으로는 하이브리드 커팅을 이용하여 고속으로 패킷을 분류하는 방법 및 시스템에 관한 것이다.

배 경 기 술

[0002] 현대의 네트워크는 IoT(Internet Of Things)로 인해 크기가 대형화되고 있다. 이에 따라 정책 집합들이 복잡해지고, 그 크기와 개수 또한 빠르게 증가하고 있다. 이러한 대용량 보안 정책들로 인해, 높은 분류 속도와 작은 테이블 크기를 동시에 가질 수 있는 새로운 패킷 분류 알고리즘에 대한 요구가 높아지고 있다. 그러나 일반적으로 패킷 분류 속도와 테이블의 크기는 Trade-off 관계를 가진다. 따라서 기술적으로 패킷 분류 알고리즘의 속도와 테이블 크기를 동시에 개선하는 것은 매우 어려운 실정이다.

[0003] 대부분의 고속 패킷 분류 알고리즘은 판별 트리(Decision Tree)를 이용하고 있다. 이때, 각각의 판별 트리의 노드(Node)는 다양한 기준에 따라 자식 노드(Child Note)를 선택한다. 자식 노드를 선택하는 방법으로는 특정 필드의 일정 부분을 인덱스(Index)로 하거나, 특정 위치의 비트값들의 조합으로 해쉬키(Hash Key)를 만드는 방법이 주로 사용된다. 이러한 방법으로 부모 노드(Parent Node)에 포함된 정책 집합들을 효과적으로 작은 크기의 정책 집합들로 분할할 수 있다. 그러나 정책 집합의 특성에 따라서는 특정 크기 이하의 서브 집합으로 분할하는데 많은 단계가 요구되며, 이에 따라 테이블 크기가 급격히 커지는 문제가 발생하기도 한다. 이처럼 대형 테이블을 사용하게 되는 패킷 분류 알고리즘은 테이블 생성 시간 및 업데이트에 불리하다는 단점이 있다.

[0004] 도 1은 HyperCuts 알고리즘에 의해 커팅이 수행되어 8개의 파티션으로 분할된 모습을 도시한 도면이다. HyperCuts 알고리즘은 패킷 분류 알고리즘 중 하나로, 하나 이상의 차원(필드)에 대해 등간격으로 커팅을 수행한다. 도 1에 도시된 바와 같이, 판별 트리 노드는 HyperCuts 알고리즘에 의해 X 차원에서 4개, Y 차원에서 2개로 분할되어 총 8개의 파티션으로 분할될 수 있다. HyperCuts 알고리즘은 매우 간단하여, 기존의 패킷 분류 알고리즘에 비해 훨씬 높은 성능을 얻을 수 있다. 그러나 이러한 HyperCuts 알고리즘은 커팅에 의해 중복된 파티션들이 많이 생길 수 있다는 문제점이 있다. 도 1에 도시된 바와 같이, C1, C2, C4, C5 및 C6은 동일한 정책을 포함하여 중복된 파티션에 해당하므로 사실상 하나의 파티션에 해당한다. 따라서 HyperCuts 알고리즘에 의해 커팅되어 실제로 분할된 파티션은 4개에 해당한다. HyperCuts 알고리즘과 같이 등간격으로 커팅을 수행하는 경우, 해당 차원 상에서 정책 집합의 분포에 대한 정보를 고려하지 않고, 단순히 커팅할 범위와 커팅 수에 의해 분할되는 위치가 결정된다는 문제점이 있다. 따라서 최적으로 분할이 이루어지지 않게 되며, 정책 집합이 각각의 차원에 균일하게 분산되어 있지 않으면 중복된 파티션들이 더욱 많이 발생하게 된다. 이러한 특성으로 인해, HyperCuts 알고리즘에 의한 패킷 분류는 검색 속도 저하와 테이블 크기 증가 문제가 빈번하게 발생하고

있다.

[0005] 관련된 선행 기술로서, 한국 등록특허 제10-0965552호 '영역분할을 이용한 패킷 분류 테이블 생성 방법 및 패킷분류 방법과 장치' 등이 제안된 바 있다.

발명의 내용

해결하려는 과제

[0006] 본 발명은 기존에 제안된 방법들의 상기와 같은 문제점들을 해결하기 위해 제안된 것으로서, 2가지 커팅 방식을 동시에 사용함으로써, 정책 집합을 더욱 효율적으로 커팅할 수 있고, 이로 인해 정책 집합의 분포와 무관하게 더욱 빠른 패킷 분류 속도를 얻을 수 있으며, 판별 트리의 깊이와 생성되는 테이블의 크기 또한 크게 감소시킬 수 있는, 하이브리드 커팅을 이용한 고속 패킷 분류 방법 및 시스템을 제공하는 것을 목적으로 한다.

[0007] 또한, 본 발명은, 메모리 액세스 시 함께 캐싱되는 메모리 영역 내에 필요한 엔트리들이 배치됨으로써, 메모리 액세스 수를 크게 감소시킬 수 있는, 하이브리드 커팅을 이용한 고속 패킷 분류 방법 및 시스템을 제공하는 것을 목적으로 한다.

과제의 해결 수단

[0008] 상기한 목적을 달성하기 위한 본 발명의 특징에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법은,

[0009] 커팅 방식을 이용하여 패킷을 분류하는 패킷 분류 방법에 있어서,

[0010] (1) 판별 트리 노드에 속한 정책 집합에 대해 등간격 커팅이 적용되어 각각의 파티션으로 분할되는 단계;

[0011] (2) 상기 단계 (1)에서 분할된 각각의 파티션과 매칭되는 포인터 배열에서 중복되는 엔트리가 제거되는 단계;

[0012] (3) 상기 단계 (1)에서 분할된 파티션에 대해 피벗을 기준으로 커팅하는 피벗 기반 커팅이 적용되어 서브-파티션으로 분할되는 단계;

[0013] (4) 상기 단계 (2)에서 제거된 상기 포인터 배열의 중복된 엔트리에 상기 단계 (3)에서 상기 피벗 기반 커팅에 사용된 피벗과 상기 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 저장되는 단계; 및

[0014] (5) 상기 단계 (4)에서 상기 피벗과 상기 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 저장되는 포인터 배열이 캐시 라인에 저장되는 단계를 포함하는 것을 그 구성상의 특징으로 한다.

[0015] 바람직하게는, 상기 단계 (3)은,

[0016] 상기 등간격 커팅이 적용되어 가장 많은 수로 커팅된 파티션을 가지는 차원에 대해서 상기 피벗 기반 커팅이 적용될 수 있다.

[0017] 더욱 바람직하게는, 상기 단계 (3)은,

[0018] 상기 단계 (1)에서 분할된 각각의 파티션에 대해 중복되는 정책 집합의 수를 최소화하는 피벗을 기준으로 상기 피벗 기반 커팅이 적용될 수 있다.

[0019] 바람직하게는, 상기 단계 (3)은,

[0020] 상기 단계 (1)에서 분할된 각각의 파티션에 대해 피벗 기반 커팅이 적어도 한 번 이상 적용될 수 있다.

[0021] 바람직하게는, 상기 단계 (2)는,

[0022] 상기 포인터 배열에 중복되지 않은 파티션과 매칭되는 고유 포인터만 저장될 수 있다.

[0023] 더욱 바람직하게는, 상기 포인터 배열은,

[0024] 각각의 엔트리가 인덱스가 저장되는 인덱스 필드와, 포인터 또는 피벗이 저장되는 공유 필드로 구성될 수 있다.

[0025] 더욱더 바람직하게는, 상기 포인터 배열은,

[0026] 상기 공유 필드에 상기 피벗 기반 커팅에 사용된 피벗과 상기 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 저장될 수 있다.

[0027] 더욱더 바람직하게는, 상기 단계 (4)는,

[0028] 상기 포인터 배열에서 상기 고유 포인터가 첫 엔트리부터 연속적으로 배치된 후, 상기 피벗 기반 커팅에 사용된 피벗과 상기 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 배치될 수 있다.

[0029] 더욱 바람직하게는, 상기 포인터 배열은,

[0030] 상기 피벗에서 상기 파티션 또는 서브-파티션의 시작값을 뺀 오프셋을 저장하여 32bit 피벗을 28bit의 공유 필드에 저장할 수 있다.

[0031] 바람직하게는, 상기 단계 (5)는,

[0032] 상기 포인터 배열이 하나의 캐시 라인에 저장되되, 상기 포인터 배열의 크기가 상기 하나의 캐시 라인 사이즈를 초과하는 경우, 하나의 캐시 라인 사이즈마다 새로운 헤더가 사용될 수 있다.

[0033] 상기한 목적을 달성하기 위한 본 발명의 특징에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 시스템은,

[0034] 커팅 방식을 이용하여 패킷을 분류하는 패킷 분류 시스템에 있어서,

[0035] 판별 트리 노드에 속한 정책 집합에 대해 등간격 커팅을 적용하여 각각의 파티션으로 분할하는 등간격 커팅 모듈;

[0036] 상기 등간격 커팅 모듈에 의해 분할된 각각의 파티션과 매칭되는 포인터 배열에서 중복되는 엔트리를 제거하는 중복 엔트리 제거 모듈;

[0037] 상기 등간격 커팅 모듈에 의해 분할된 파티션에 대해 피벗을 기준으로 커팅하는 피벗 기반 커팅을 적용하여 서브-파티션으로 분할하는 피벗 기반 커팅 모듈;

[0038] 상기 중복 엔트리 제거 모듈에 의해 제거된 상기 포인터 배열의 중복된 엔트리에 상기 피벗 기반 커팅 모듈의 상기 피벗 기반 커팅에 사용된 피벗과 상기 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터를 저장하는 포인터 배열 저장 모듈; 및

[0039] 상기 피벗과 상기 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 저장되는 포인터 배열을 캐시 라인에 저장하는 캐시 라인 저장 모듈을 포함하는 것을 그 구성상의 특징으로 한다.

[0040] 바람직하게는, 상기 피벗 기반 커팅 모듈은,

[0041] 상기 등간격 커팅이 적용되어 가장 많은 수로 커팅된 파티션을 가지는 차원에 대해서 상기 피벗 기반 커팅이 적용될 수 있다.

[0042] 더욱 바람직하게는, 상기 단계 피벗 기반 커팅 모듈은,

[0043] 상기 등간격 커팅 모듈에서 분할된 각각의 파티션에 대해 중복되는 정책 집합의 수를 최소화하는 피벗을 기준으로 상기 피벗 기반 커팅이 적용될 수 있다.

[0044] 바람직하게는, 상기 피벗 기반 커팅 모듈은,

[0045] 상기 피벗 기반 커팅 모듈에서 분할된 각각의 파티션에 대해 피벗 기반 커팅이 적어도 한 번 이상 적용될 수 있다.

[0046] 바람직하게는, 상기 중복 엔트리 제거 모듈은,

[0047] 상기 포인터 배열에 중복되지 않은 파티션과 매칭되는 고유 포인터만 저장할 수 있다.

[0048] 더욱 바람직하게는, 상기 포인터 배열은,

[0049] 각각의 엔트리가 인덱스가 저장되는 인덱스 필드와, 포인터 또는 피벗이 저장되는 공유 필드로 구성될 수 있다.

[0050] 더욱 바람직하게는, 상기 포인터 배열은,

[0051] 상기 공유 필드에 상기 피벗 기반 커팅에 사용된 피벗과 상기 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 저장될 수 있다.

[0052] 더욱 바람직하게는, 상기 포인터 배열 저장 모듈은,

[0053] 상기 포인터 배열에서 상기 고유 포인터를 첫 엔트리부터 연속적으로 배치한 후, 상기 피벗 기반 커팅에 사용된 피벗과 상기 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터를 배치할 수 있다.

[0054] 더욱 바람직하게는, 상기 포인터 배열은,

[0055] 상기 피벗에서 상기 파티션 또는 서브-파티션의 시작값을 뺀 오프셋을 저장하여 32bit 피벗을 28bit의 공유 필드에 저장할 수 있다.

[0056] 바람직하게는, 상기 캐시 라인 저장 모듈은,

[0057] 상기 포인터 배열이 하나의 캐시 라인에 저장되되, 상기 포인터 배열의 크기가 상기 하나의 캐시 라인 사이즈를 초과하는 경우, 하나의 캐시 라인 사이즈마다 새로운 헤더가 사용될 수 있다.

발명의 효과

[0058] 본 발명에서 제안하고 있는 하이브리드 커팅을 이용한 고속 패킷 분류 방법 및 시스템에 따르면, 2가지 커팅 방식을 동시에 사용함으로써, 정책 집합을 더욱 효율적으로 커팅할 수 있고, 이로 인해 정책 집합의 분포와 무관하게 더욱 빠른 패킷 분류 속도를 얻을 수 있으며, 판별 트리의 깊이와 생성되는 테이블의 크기 또한 크게 감소시킬 수 있다.

[0059] 또한, 본 발명에 따르면, 메모리 액세스 시 함께 캐싱되는 메모리 영역 내에 필요한 엔트리들이 배치됨으로써, 메모리 액세스 수를 크게 감소시킬 수 있다.

도면의 간단한 설명

[0060] 도 1은 HyperCuts 알고리즘에 의해 커팅이 수행되어 8개의 파티션으로 분할된 모습을 도시한 도면.

도 2는 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법의 구성을 도시한 도면.

도 3은 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법에서 도 1에서 분할된 파티션과 매칭되는 포인터 배열을 도시한 도면.

도 4는 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법에서 도 1에서 분할된 파티션에 피벗 기반 커팅이 적용되어 서브-파티션으로 분할되는 모습을 도시한 도면.

도 5는 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법에서 도 3의 포인터 배열을 재구성한 결과를 도시한 도면.

도 6은 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법에서 포인터 배열이 캐시 라인에 저장되는 모습을 도시한 도면.

도 7은 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법에서 정책 종류에 따른 정책 집합 크기별 상대적인 메모리 액세스 양을 도시한 도면.

도 8은 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법에서 캐시를 고려한 정책 종류에 따른 정책 집합 크기별 상대적인 메모리 액세스 양을 도시한 도면.

도 9는 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법에서 정책 종류에 따른 정책 크기별 테이블의 상대 크기를 도시한 도면.

도 10은 본 발명의 다른 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 시스템(100)의 세부 구성을 도시한 도면.

발명을 실시하기 위한 구체적인 내용

[0061] 이하에서는 첨부된 도면을 참조하여 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자가 본 발명을 용이하게 실시할 수 있도록 바람직한 실시예를 상세히 설명한다. 다만, 본 발명의 바람직한 실시예를 상세하게 설명함에 있어, 관련된 공지 기능 또는 구성에 대한 구체적인 설명이 본 발명의 요지를 불필요하게 흐릴 수 있다고 판단되는 경우에는 그 상세한 설명을 생략한다. 또한, 유사한 기능 및 작용을 하는 부분에 대해서는 도면 전체에 걸쳐 동일 또는 유사한 부호를 사용한다.

[0062] 덧붙여, 명세서 전체에서, 어떤 부분이 다른 부분과 '연결' 되어 있다고 할 때, 이는 '직접적으로 연결' 되어 있는 경우뿐만 아니라, 그 중간에 다른 소자를 사이에 두고 '간접적으로 연결' 되어 있는 경우도 포함한다. 또한, 어떤 구성요소를 '포함' 한다는 것은, 특별히 반대되는 기재가 없는 한 다른 구성요소를 제외하는 것이 아니라 다른 구성요소를 더 포함할 수 있다는 것을 의미한다.

[0063] 도 2는 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법의 구성을 도시한 도면이다. 도 2에 도시된 바와 같이, 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법은, 하나 이상의 차원에 대해 커팅 방식을 이용하여 패킷을 분류하는 패킷 분류 방법에 있어서, (1) 판별 트리 노드에 속한 정책 집합에 대해 등간격 커팅이 적용되어 각각의 파티션으로 분할되는 단계(S100); (2) 단계 S100에서 분할된 각각의 파티션과 매칭되는 포인터 배열에서 중복되는 엔트리가 제거되는 단계(S200); (3) 단계 S100에서 분할된 파티션에 대해 피벗을 기준으로 커팅하는 피벗 기반 커팅이 적용되어 서브-파티션으로 분할되는 단계(S300); (4) 단계 S200에서 제거된 포인터 배열의 중복된 엔트리에 단계 S300의 피벗 기반 커팅에 사용된 피벗과 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 저장되는 단계(S400); 및 (5) 단계 S400에서 피벗과 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 저장되는 포인터 배열이 캐시 라인에 저장되는 단계(S500)를 포함할 수 있다. 이하에서는 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법의 각각의 구성에 대해 상세히 설명하기로 한다.

[0064] 본 명세서에서 하이브리드 커팅은, 등간격 커팅과 피벗 기반 커팅을 함께 사용하는 커팅을 의미한다. 여기서,

피벗 기반 커팅은, 특정 값을 기준으로 작거나 같은 정책 집합과 큰 정책 집합으로 구성된 파티션들로 커팅하는 분할 방법을 말한다. 이때, 커팅에 사용되는 특정 값을 피벗이라 부르기로 한다. 또한, 판별 트리 노드가 분할되어 생성되는 파티션은 자식 노드를 의미하며, 서브-파티션은 자식 노드를 다시 분할하여 생성되는 자식 노드를 의미한다.

[0065] 단계 S100에서는, 판별 트리 노드에 속한 정책 집합에 대해 등간격 커팅이 적용되어 각각의 파티션으로 분할될 수 있다. 등간격 커팅은, HyperCuts 알고리즘과 유사한 방식으로, 한 개 이상의 차원(필드)에 대해서 등간격으로 커팅을 수행한다.

[0066] 단계 S200에서는, 단계 S100에서 분할된 각각의 파티션과 매칭되는 포인터 배열에서 중복되는 엔트리가 제거될 수 있다. 도 3은 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법에서 도 1에서 분할된 파티션과 매칭되는 포인터 배열을 도시한 도면이다. 도 3을 참조하면, 총 파티션의 수는 8개이지만, 4개의 p_1이 중복되므로, 실제로는 단 4개의 파티션만을 얻게 된다. 이처럼 등간격 커팅으로 인한 커팅 방식으로는 중복되는 파티션이 생성될 수 있으며, 패킷 분류에 대한 성능 저하의 가장 큰 요인이 된다. 이를 해결하기 위해서, 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법은, 후술하는 바와 같이, 포인터 배열에 중복된 포인터 대신 고유 포인터만 저장하고, 빈 엔트리들에 피벗과 각각의 피벗에 의해 분할된 2개의 파티션에 대한 포인터들을 저장하는 방법을 사용한다. 이를 위해서는 먼저 단계 S200에서 중복된 포인터들을 제거하는 단계가 선행되어야 한다. 즉, 단계 S200에서는, 포인터 배열에 중복되지 않은 파티션과 매칭되는 고유 포인터만 저장될 수 있다.

[0067] 단계 S300에서는, 단계 S100에서 분할된 파티션에 대해 피벗을 기준으로 커팅하는 피벗 기반 커팅이 적용되어 서브-파티션으로 분할될 수 있다. 여기서, 피벗 기반 커팅은, 등간격 커팅이 적용되어 가장 많은 수로 커팅된 파티션을 가지는 차원에 대해서 피벗 기반 커팅이 적용될 수 있다. 도 4는 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법에서 도 1에서 분할된 파티션에 피벗 기반 커팅이 적용되어 서브-파티션으로 분할되는 모습을 도시한 도면이다. 예를 들어, 도 1을 참조하면, 등간격 커팅으로 인해 X 차원에 대해서 4개, Y 차원에 대해서 2개의 파티션으로 커팅되었으므로, X 차원에 대해서 피벗 기반 커팅이 적용된다. 이때, 도 4에 도시된 바와 같이, 피벗 기반 커팅은, 단계 S100에서 분할된 각각의 파티션에 대해 중복되는 정책 집합의 수를 최소화하는 피벗(V_1, V_2)을 기준으로 피벗 기반 커팅이 적용될 수 있다. 이 경우, 고유 포인터의 수를 u, 전체 컷의 수를 N이라고 하면, 하나의 피벗으로 2개의 컷을 생성하므로, 총 컷의 수는 u+[(N-u)*2/3]를 올림(Ceiling)한 수가 된다.

[0068] 단계 S400에서는, 단계 S200에서 제거된 포인터 배열의 중복된 엔트리에 단계 S300에서 피벗 기반 커팅에 사용된 피벗과 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 저장될 수 있다. 이하에서는 설명의 편의를 위해, 기존 HyperCuts 알고리즘 및 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법에 의해 분할된 서브-파티션과 매칭되는 포인터 배열의 k번째 엔트리를 각각 c[k] 및 C[k]로 표현한다.

[0069] 포인터 배열은, 각각의 엔트리가 인덱스가 저장되는 인덱스 필드와, 포인터 또는 피벗이 저장되는 공유 필드로 구성될 수 있다. 기존 HyperCuts에서 c[k]는 서브-파티션과 매칭되는 32bit 포인터 값을 가진다. 파티션의 크기를 HyperCuts와 동일하게 유지하면서도 추가적인 정보를 제공하기 위해서, C[k]는 4bit 인덱스 값과 28bit 포인터 값 또는 피벗으로 구성될 수 있다. 이를 각각 C[k].idx와 C[k].ptr 또는 C[k].pivot으로 표현한다. 즉, 인덱스 필드에는 4bit 인덱스 값이 저장되며, 공유 필드에는 28bit 포인터 값 또는 피벗이 저장될 수 있다. 따라서 공유 필드에는, 피벗 기반 커팅에 사용된 피벗과 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 저장될 수 있다. 이를 쉽게 구별하기 위해서, 파티션에는 공유 필드가 포인터 또는 피벗을 포함하는지 여부를 구별하기 위한 비트벡터를 더 포함할 수 있다. 여기서, 비트벡터의 길이는 해당 파티션의 서브-파티션의 수와 동일하다. 비트벡터는, LSB(Least Significant Bit)가 엔트리 0번, 다음 비트가 엔트리 1번으로 매핑된다. 즉, 비트벡터에서 해당 비트가 1이면, 대응되는 포인터 배열의 엔트리에는 피벗 값을, 0이면 포인터가 저장될 수 있

다.

[0070] 도 5는 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법에서 도 3의 포인터 배열을 재구성한 결과를 도시한 도면이다. 도 5에 도시된 바와 같이, 단계 S400에서는, 포인터 배열에서 고유 포인터가 첫 엔트리부터 연속적으로 배치된 후, 피벗 기반 커팅을 위한 피벗과 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 배치될 수 있다. 즉, 도 3을 참조하면, 0, 1, 3, 4, 5번째 엔트리가 서로 동일하게 p_1을 가리키므로, 중복인 1, 3, 4, 5번째 엔트리의 공유 필드는 다른 용도로 사용할 수 있다. 대신 해당 엔트리의 인덱스 필드는 0을 가리키게 설정하는데, 이는 C[0].ptr가 p_1을 가리키는 포인터이기 때문이다. 실시 예에 따라서는, 고유 포인터는 포인터 배열의 첫 엔트리부터 연속적으로 배치될 수 있다. 그 후에 피벗 기반 커팅을 위한 피벗과 포인터가 배치될 수 있다. 따라서 뒤쪽 엔트리는 C[k].idx를 제외하고, 중복된 엔트리의 수만큼 비어있게 된다. 그리고 이러한 엔트리에는 피벗과 피벗으로 분할된 두 서브-파티션들의 포인터가 저장될 수 있다. 즉, 도 5에서 회색 엔트리는 2개의 피벗(V_1, V_2)과 피벗으로 분할된 각각의 2개의 서브-파티션(F_3' 및 F_3'', F_4' 및 F_4'')들의 포인터가 저장되어 있는 엔트리이다.

[0071] 한편, C[0].idx는 첫 피벗이 저장되어 있는 엔트리의 인덱스 필드(k_{pivot})로 설정된다. 따라서 이 엔트리를 헤더(Header)라 부른다. 도 5를 참조하면, C[0].idx는 2에 해당하며, C[2].pivot에는 첫 번째 피벗인 V_1이 배치된다. C[3].ptr 및 C[4].ptr에는 V_1을 기준으로 한 피벗 기반 커팅이 적용되어 분할된 각각의 서브-파티션에 대한 포인터인 F_3' 및 F_3''이 배치된다. 또한, C[5].pivot에는 두 번째 피벗인 V_2가 배치되며, C[6].ptr 및 C[7].ptr에는 V_2를 기준으로 한 피벗 기반 커팅이 적용되어 분할된 각각의 서브-파티션에 대한 포인터인 F_4' 및 F_4''이 배치된다.

[0072] 도 5를 참조하면, C[0] 및 C[1]에는 고유 포인터가 저장된다. 이때, 0<k<C[0].idx인 C[k]에 대해서, C[k].idx 값은 원래 저장되어야 하는 포인터 값이 존재하는 엔트리의 인덱스로 설정된다. 즉, C[1].idx 값은 도 3을 참조하면, 원래 포인터 p_1이 저장되어 있는 엔트리에 해당하므로 0의 값을 가진다. 그리고 C[0].ptr은 F1이므로, C[1]의 공유 필드의 값이 변경되어도 C[1]에 해당하는 p_1 포인터를 알 수 있다. 반면, k≥C[0].idx인 경우에는, 피벗 기반 커팅을 위한 피벗과 해당 피벗보다 작거나 큰 값으로 구성된 2개의 파티션에 대한 포인터들이 순서대로 저장된다. 도 3 및 5를 참조하면, 도 3의 포인터 p_3 및 p_4에 대응하는 C[k].idx 값은 도 5에서 해당 포인터가 아닌 피벗이 저장된 엔트리를 가리키게 된다. 즉, C[6].idx는 원래 포인터 p_3을 가리켜야 하지만, 피벗으로 커팅되었으므로 커팅에 사용된 피벗 V_1이 저장된 엔트리의 인덱스인 2로 설정된다.

[0073] 한편, 피벗의 경우 포인터와 마찬가지로 32bit 값에 해당하나, 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법에서는, 피벗을 28bit 공유 필드에 저장하여야 한다. 이를 해결하기 위해서 포인터 배열은 피벗에서 서브-파티션의 시작값을 뺀 오프셋을 저장하여, 32bit 피벗을 28bit의 공유 필드에 저장할 수 있다. 피벗은, 현재 파티션에서 선택된 차원 D에 대해 모든 부모 노드로부터 현재 파티션까지 커팅한 수 N_c^D이 2^4, 즉 16을 넘는 경우와 넘지 않는 경우에 대해 다른 값으로 설정된다. 현재 파티션에서 피벗 기반 커팅을 할 컷에 대해 매칭되는 차원 D의 범위가 D_s, D_e라고 하면 C[k].pivot은 다음의 수학식 1에 의해 설정된다.

수학식 1

$$C[k].\text{pivot} = \begin{cases} P - D_s & if\ N_c^D \geq 16 \\ (P - D_s) \gg \lfloor log_2 N_c^D \rfloor & if\ N_c^D < 16 \end{cases}$$

[0074]

[0075] 여기서, P는 피벗, N_c^D는 현재 파티션에서 선택된 차원 D에 대해 모든 부모 노드로부터 현재 파티션까지 커팅한 수, D_s는 파티션의 시작값, D_e는 파티션의 종료값을 의미하고, '>>' 는 오른쪽 시프트 연산자, '$\lfloor \rfloor$' 는 올림 연산자를 의미한다. 즉, 피벗을 28bit 공유 필드에 저장하기 위해 피벗 대신 서브-파티션의 시작값을 뺀 오프셋을 저장하고, 이러한 경우에도 공유 필드에 저장할 수 없다면 저장을 할 수 있을 만큼 오른쪽으로 시프트(shift)한 값을 저장할 수 있다. 예를 들어, 피벗 P가 30,000이고, 이미 알고 있는 D_s의 값은 20,000이라고 하면, P-D_s는 훨씬 작은 값(10,000)에 해당되어 보다 적은 비트를 사용해 표현할 수 있는 가능성이 높다. 이때, 커팅에 의해 생성된 파티션의 수도 중요한데, 32bit 필드는 0 내지 0xFFFF_FFFF의 값을 갖게 된다. 예를 들어, 이를 4개의 파티션 p1 내지 p4로 분할했다고 하면, 파티션 p1은 0 내지 0x3FFF_FFFF, 파티션 p2는 0x4000_0000 내지 0x7FFF_FFFF, 파티션 p3는 0x8000_0000 내지 0xBFFF_FFFF, 파티션 p4는 0xC000_0000 내지 0xFFFF_FFFF가 된다. 이때, 파티션 p2만 4개의 서브-파티션으로 분할되었다면, 서브-파티션 p2_1은 0x4000_0000 내지 0x4FFF_FFFF, 서브-파티션 p2_2는 0x5000_0000 내지 0x5FFF_FFFF, 서브-파티션 p2_3는 0x6000_0000 내지 0x6FFF_FFFF, 서브-파티션 p2_4는 0x7000_0000 내지 0x7FFF_FFFF가 된다. 파티션 p2_1에 대한 총 파티션 수 N_c^D는 16이다. 파티션 p2의 시작값 D_s는 0x4000_0000이고, 피벗 P는 0x4000_0000 내지 0x4FFF_FFFF에 속하는 값이 된다. 따라서 P-D_s는 0x0000_0000 내지 0x0FFF_FFFF가 되고, 이 값은 28bit로 충분히 표현될 수 있다. 따라서 28bit 필드인 C[x].pivot에 저장될 수 있다. 반면, 파티션 p3에 대한 총 파티션 수 N_c^D는 4이다. 파티션 p3의 시작값 D_s는 0x8000_0000이고 피벗 P는 0x8000_0000 내지 0xBFFF_FFFF에 속하는 값이 된다. 이제 P-D_s는 0x0000_0000 내지 0x3FFF_FFFF에 속하는 값이 되나, 이 값은 28bit로 표현할 수 없다. 따라서 이 경우에는 정확한 피벗을 저장하는 대신 총 파티션수를 log2로 변환한 만큼 P-D_s를 우측으로 비트쉬프트를 수행한 값을 C[x].pivot에 저장한다. 즉, (P-D_s)>>2의 범위는 0x0000_0000 내지 0x0FFF_FFFF가 되어 28bit 필드인 C[x].pivot에 저장될 수 있다.

[0076] 한편, 단계 S300에서는, 단계 S100에서 분할된 각각의 파티션에 대해 피벗 기반 커팅이 적어도 한 번 이상 적용될 수 있다. 피벗 기반 커팅이 한 번 이상 수행되는 방법은 전술한 피벗 기반 커팅 방식과 동일하게 적용된다.

[0077] 도 6은 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법에서 포인터 배열이 캐시 라인에 저장되는 모습을 도시한 도면이다. 단계 S500에서는, 단계 S400에서 피벗과 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 저장되는 포인터 배열이 캐시 라인에 저장될 수 있다. 또한, 단계 S500에서는, 포인터 배열의 크기가 하나의 캐시 라인 사이즈를 초과하는 경우, 하나의 캐시 라인 사이즈마다 새로운 헤더가 사용될 수 있다. 단계 S100 내지 단계 S400을 통해 포인터 배열에서 중복된 포인터를 제거하여 포인터 배열을 컴팩트하게 구성하였으나, 파티션과 매칭되는 포인터를 얻기 위해서는 1개 이상의 엔트리에 액세스하여야 한다. 이 경우 메모리 액세스 양이 증가하고, 결과적으로 패킷 분류 속도가 저하될 수 있다. 이를 해결하기 위해서, 단계 S500에서는, 포인터 배열이 하나의 캐시 라인에 저장될 수 있다. 만일 포인터 배열의 크기가 캐시 라인 사이즈를 초과하면, 하나의 캐시 라인 사이즈마다 새로운 헤더를 사용할 수 있다. 즉, 캐시 라인 사이즈가 64Bytes인 경우, 엔트리의 수가 16개 이하이면 하나의 헤더만으로 충분하다. 그러나 엔트리의 수가 16개를 초과하는 경우, 16개의 엔트리마다 1개의 헤더가 존재하게 된다. 이를 통해 메모리 액세스 시 함께 캐싱되는 메모리 영역 내에 필요한 엔트리들이 배치됨으로써 메모리 액세스 양을 크게 감소시킬 수 있다.

[0078] 이하에서는 도 7 내지 도 9를 참조하여, 본 발명의 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법의 성능을 비교한다. 성능 평가에 사용되는 정책 집합은 ACL, FW 및 IPC의 총 3가지를 이용하였으며, 메모리 액세스 양, 캐시 라인 기준 메모리 액세스 양, 및 테이블 크기를 측정한다. 또한 정책 집합 수에 따라 성능을 측정함으로써, 정책 집합 크기에 따른 경향성을 비교한다.

[0079] 도 7은 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법에서 정책 종류에 따른 정책 집합 크기별 상대적인 메모리 액세스 양을 도시한 도면이다. 여기서, X 축은 정책 집합의 크기, Y 축은 메모리 액세스 양의 상대값, 즉, 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법에 따른 메모리 액세스 양을 기존 HyperCuts 방법에 따른 메모리 액세스 양으로 나눈 값이다. 일반적으로 메모리 성능이 전체 성능의 병목이 되므로, 메모리 액세스 양을 줄이는 것이 패킷 분류 성능 향상에 가장 중요하다. 도 7에 도시된 바와 같이, 메모리 액세스 양은 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법에서 기존 HyperCuts 대비 30% 이상 감소함을 알 수 있다. 비록, 하나의 파티션에서 메모리에 액세스하는 양이 증가하지만, 판별 트리의 높이 자체가 낮아지므로 전체적인 메모리 액세스 양은 오히려 감소하는 것을 알 수 있다. 따라서 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법이 기존 HyperCuts의 단점을 효과적으로 개선하고 있음을 알 수 있다,

[0080] 도 8은 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법에서 캐시를 고려한 정책 종류에 따른 정책 집합 크기별 상대적인 메모리 액세스 양을 도시한 도면이다. 여기서, X 축은 정책 집합의 크기, Y 축은 메모리 액세스 양의 상대값, 즉, 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법에 따른 메모리 액세스 양을 기존 HyperCuts 방법에 따른 메모리 액세스 양으로 나눈 값이다. 메모리 액세스 양은 실제로 캐시에 의한 영향을 고려하고 있지 않으므로, 정확한 성능을 반영하지 못한다. 따라서 정확한 성능 분석을 위해서, 캐시 라인 액세스 양을 측정하는 것이 필요하다. 도 8에 도시된 바와 같이, 메모리 액세스 양은 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법에서 기존 HyperCuts 대비 55% 이상 감소되어 2배 이상의 성능을 제공할 수 있다.

[0081] 도 9는 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법에서 정책 종류에 따른 정책 크기별 테이블의 상대 크기를 도시한 도면이다. 여기서, X 축은 정책 집합의 크기, Y 축은 테이블 상태 크기의 상대값, 즉, 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법에 따른 테이블 상태 크기를 기존 HyperCuts 방법에 따른 테이블 상태 크기로 나눈 값이다. 테이블 크기는 지원하는 정책의 수에 영향을 주기 때문에, 매우 중요한 성능 측정 요소이다. 도 9에 도시된 바와 같이, 테이블 크기는 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법에서 모든 정책 종류와 정책 크기에 대해 감소됨을 알 수 있다. 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법은, 하이브리드 커팅이 분할 방식을 크게 개선함으로써, 실제 필요한 파티션의 수를 크게 감소시킬 수 있다. 이로 인해 테이블 크기가 감소하게 된다.

[0082] 도 10은 본 발명의 다른 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 시스템(100)의 세부 구성을 도시한 도면이다. 도 10에 도시된 바와 같이, 본 발명의 다른 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 시스템은, 등간격 커팅 모듈(110), 중복 엔트리 제거 모듈(120), 피벗 기반 커팅 모듈(130), 포인터 배열 저장 모듈(140), 및 캐시 라인 저장 모듈(150)을 포함할 수 있다.

[0083] 등간격 커팅 모듈(110)은, 판별 트리 노드에 속한 정책 집합에 대해 등간격 커팅을 적용하여 각각의 파티션으로 분할할 수 있다. 중복 엔트리 제거 모듈(120)은, 등간격 커팅 모듈(110)에 의해 분할된 각각의 파티션과 매칭되는 포인터 배열에서 중복되는 엔트리를 제거할 수 있다. 피벗 기반 커팅 모듈(130)은, 등간격 커팅 모듈(110)에 의해 분할된 파티션에 대해 피벗을 기준으로 커팅하는 피벗 기반 커팅을 적용하여 서브-파티션으로 분할할 수 있다. 포인터 배열 저장 모듈(140)은, 중복 엔트리 제거 모듈(110)에 의해 제거된 포인터 배열의 중복

된 엔트리에 피벗 기반 커팅 모듈(130)의 피벗 기반 커팅에 사용된 피벗과 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터를 저장할 수 있다. 캐시 라인 저장 모듈(150)은, 피벗과 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 저장되는 포인터 배열을 캐시 라인에 저장할 수 있다. 각각의 모듈에서 수행되는 정보 처리는, 전술한 본 발명의 일 실시예에 따른 하이브리드 커팅을 이용한 고속 패킷 분류 방법의 각각의 단계에서 수행되는 정보 처리와 대응될 수 있다.

[0084] 전술한 바와 같이, 본 발명에서 제안하고 있는 하이브리드 커팅을 이용한 고속 패킷 분류 방법 및 시스템에 따르면, 2가지 커팅 방식을 동시에 사용함으로써, 정책 집합을 더욱 효율적으로 커팅할 수 있고, 이로 인해 정책 집합의 분포와 무관하게 더욱 빠른 패킷 분류 속도를 얻을 수 있으며, 판별 트리의 깊이와 생성되는 테이블의 크기 또한 크게 감소시킬 수 있다. 또한, 본 발명에 따르면, 메모리 액세스 시 함께 캐싱되는 메모리 영역 내에 필요한 엔트리들이 배치됨으로써, 메모리 액세스 수를 크게 감소시킬 수 있다.

[0085] 이상 설명한 본 발명은 본 발명이 속한 기술분야에서 통상의 지식을 가진 자에 의하여 다양한 변형이나 응용이 가능하며, 본 발명에 따른 기술적 사상의 범위는 아래의 청구범위에 의하여 정해져야 할 것이다.

부호의 설명

[0086] 100: 하이브리드 커팅을 이용한 고속 패킷 분류 시스템

110: 등간격 커팅 모듈

120: 중복 엔트리 제거 모듈

130: 피벗 기반 커팅 모듈

140: 포인터 배열 저장 모듈

150: 캐시 라인 저장 모듈

S100: 판별 트리 노드에 속한 정책 집합에 대해 등간격 커팅이 적용되어 각각의 파티션으로 분할되는 단계

S200: 단계 S100에서 분할된 각각의 파티션과 매칭되는 포인터 배열에서 중복되는 엔트리가 제거되는 단계

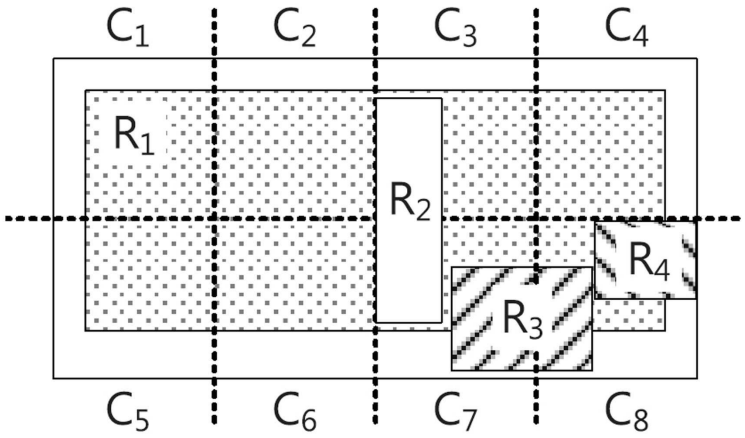
S300: 단계 S100에서 분할된 파티션에 대해 피벗을 기준으로 커팅하는 피벗 기반 커팅이 적용되어 서브-파티션으로 분할되는 단계

S400: 단계 S200에서 제거된 포인터 배열의 중복된 엔트리에 단계 S300의 피벗 기반 커팅에 사용된 피벗과 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 저장되는 단계

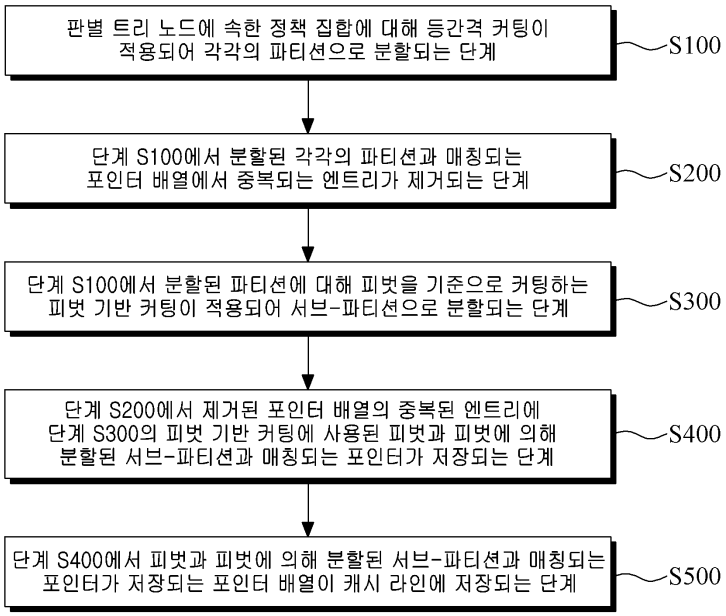
S500: 단계 S400에서 피벗과 피벗에 의해 분할된 서브-파티션과 매칭되는 포인터가 저장되는 포인터 배열이 캐시 라인에 저장되는 단계

도면

도면1



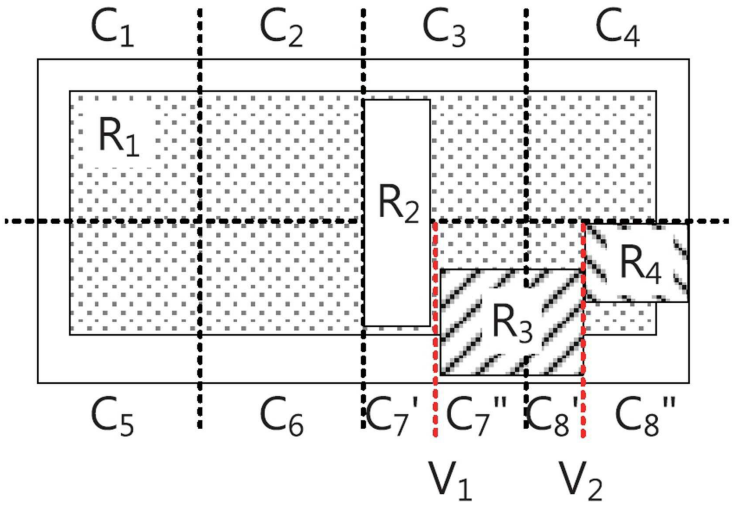
도면2



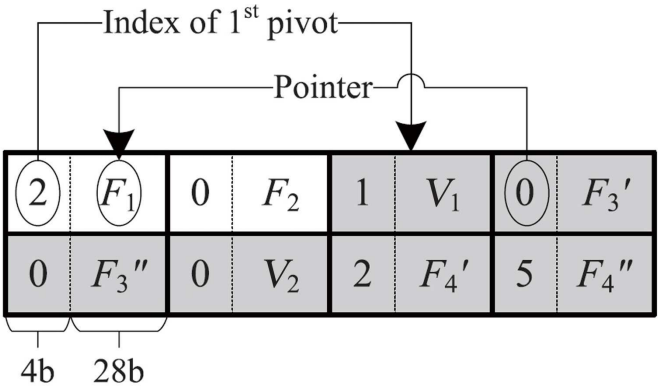
도면3

p_1	p_1	p_2	p_1
p_1	p_1	p_3	p_4

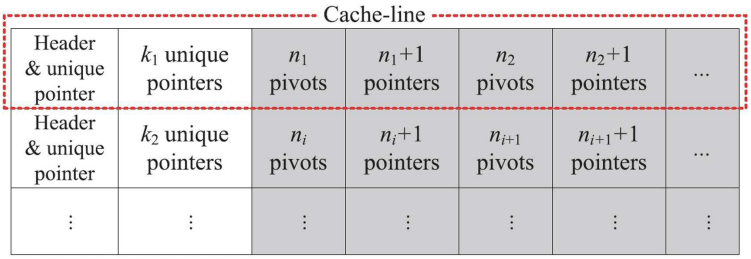
도면4


C1
C2
C3
C4
R1
R2
R4
R3
C5
C6
C7'
C7"
C8'
C8"
V1
V2

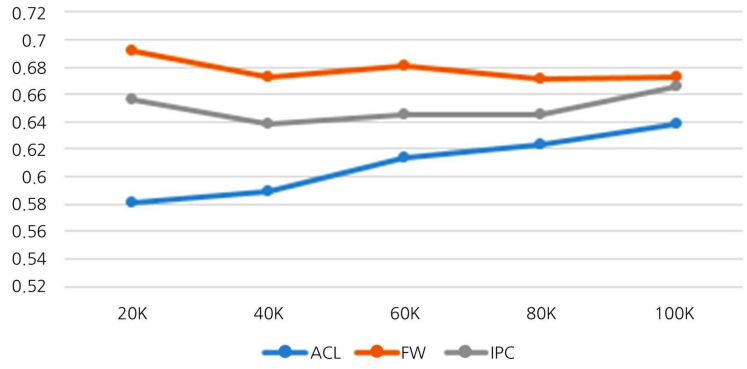
도면5


Index of 1st pivot
Pointer
2
F1
0
F2
1
V1
0
F3'
0
F3"
0
V2
2
F4'
5
F4"
4b
28b

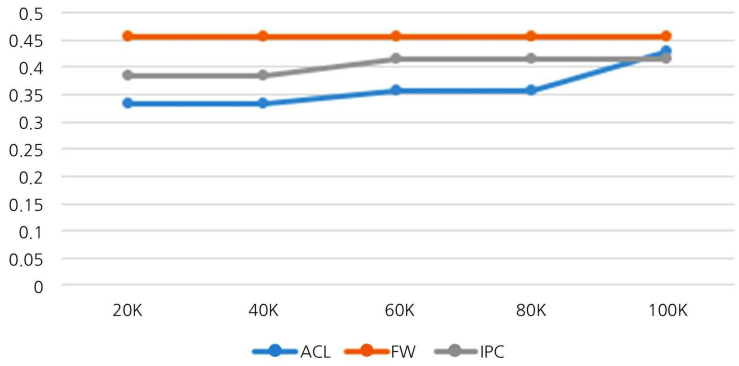
도면6


Cache-line
Header & unique pointer
k1 unique pointers
n1 pivots
n1+1 pointers
n2 pivots
n2+1 pointers
...
Header & unique pointer
k2 unique pointers
ni pivots
ni+1 pointers
ni+1 pivots
ni+1+1 pointers
...

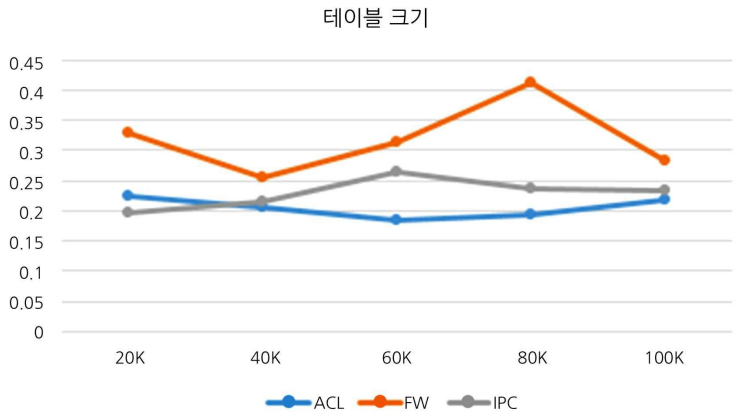
도면7


0.72
0.7
0.68
0.66
0.64
0.62
0.6
0.58
0.56
0.54
0.52
20K
40K
60K
80K
100K
ACL
FW
IPC

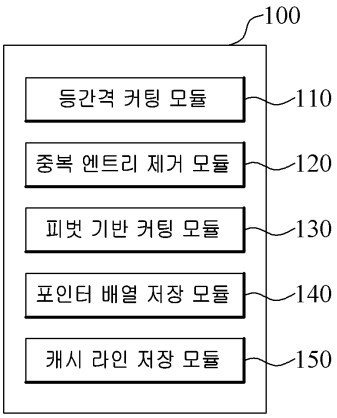
도면8


0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
20K
40K
60K
80K
100K
ACL
FW
IPC

도면9


테이블 크기
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
20K
40K
60K
80K
100K
ACL
FW
IPC

도면10


100
등간격 커팅 모듈
110
중복 엔트리 제거 모듈
120
피벗 기반 커팅 모듈
130
포인터 배열 저장 모듈
140
캐시 라인 저장 모듈
150