Packet Classification 을 위한 Cross-Product 알고리즘 구현과 성능평가

강길수*, 최경희*, 정기현**
* 아주대학교 정보 통신 전문 대학원
** 아주대학교 전자공학부
e-mail : monsterdarlf@hotmail.com

Cross-Product Algorithm Implementation and Performance Evaluation for Packet Classification

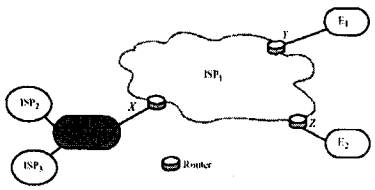
Kil-Soo Kang*, Kyunghee Choi*, Gihyun Jung**
* Graduate School for Information and Communication, Ajou University
** Division of Electronics Engineering, Ajou University

요 약

본 연구는 룰들의 각 필드들을 index 하여 곱한 cross-product 테이블을 이용한 packet classification 알고리즘에 대해 연구하고 그 것의 성능을 평가하고 분석한다. 현재 Packet Classification 은 Packet Filtering, Policy Routing, Accounting & Billing, Traffic Rate Limiting, Traffic Shaping, 등등의 서비스를 위한 가장 핵심적인 작업이다. 그러나 이들을 빠르게 서비스하는 알고리즘은 아직 존재하지 않는다. 단지 하드웨어 TCAM 을 이용해서 작은 룰들에 대한 처리만이 어느 정도 가능한 실정이다. 이에 본 연구는 소프트웨어를 이용한 cross-product 알고리즘의 효용성을 가늠하고자 연구하고 이를 실제 구현해 평가하고자 한다.

1. 서론

과거 라우터들은 단순 IP 포워딩 작업만을 처리하였다. 그러나 사용자의 네트워크 요구와 보안 요구의 증가 등으로 그 역할이 단순 포워딩 뿐만 아니라 packet filtering, policy routing, accounting & billing, traffic rate limiting, traffic shaping 등 다양해지고 복잡한 작업을 요구하고 있다. 아래 그림과 표는 이러한 서비스들을 그림과 예로 설명하고 있다.[6]



(그림 1) 네트워크 구성 예

서비스	예제
Packet Filtering	ISP3 에서 E2로 가는 모든 패킷은 Deny 시킨다.
Policy Routing	E1 에서 오는 모든 VoIP 트래픽은 ATM망으로 분리되어 있는 E2로 보낸다.
Accounting & Billing	E1으로 오는 모든 비디오 관련 트래픽은 가장 높은 우선순위를 가지고 accounting을 처리한다.
Traffic Rate Limiting	ISP2 email 트래픽은 10Mbps를 넘지 못하고 인터페이스 X에서 오는 총 트래픽은 50Mbps를 넘지 못하게 한다.
Traffic Shaping	ISP2의 인터페이스 X가 받는 웹 트래픽은 50Mbps 이상이 되지 않도록 한다.

(표 1) 서비스종류와 예

여기서 이런 서비스들은 packet classification 작업을 기본적으로 수반하고 있는데 현재 네트워크 트래픽의 처리양은 하드웨어의 성능만으로 이런 작업을 성능저하 없이 수행하기엔 역부족이다.[6]

이에 본 연구는 packet classification 의 여러 알고리즘들 중 가장 단순하고 빠른 성능을 내는 cross-product 방법에 대해 구현하고 성능을 예측한 후 이미

이론적으로 증명된 알고리즘의 한계가[2] 현실에 어떻게 적용될 수 있는지 설명하겠다. 앞으로 2 장에서 classification 문제에 대해 설명하고 cross-product 알고리즘에 대해 간단히 살펴본 뒤, 3 장과 4 장을 통해 실험 구현에 대한 설명과 성능 측정 결과를 보고 결론을 맺겠다.

2. 관련연구

이번 장은 packet classification 문제에 대한 설명과 cross-product 알고리즘의 동작에 대해 예를 들어 간략히 설명한다.

2.1 Packet Classification Problem

Packet classification 은 정의된 룰(Rule) 테이블을 탐색하여 패킷 헤더가 속하는 룰 중 가장 적합한 룰 하나를 선별하는 작업을 말한다. 여기서 적합한 룰이라 함은 우선, 룰에 정확히 매칭되어야 하고, 매칭된 룰들 중 두 가지의 선택기준(가장 지역적으로(specific) 정의한 필터를 선택하는 방법, 우선순위(priority)가 높은 필터를 선택하는 방법)중 하나에 의해 선별된 룰을 매칭 룰로 정의 한다. 룰 테이블은 하나 이상의 룰들로 구성된 룰 집합으로서, 각 룰들은 정의에 따라 서로간에 부분집합 관계를 가질 수 있고, 이들은 선별 기준에 따라 우선순위도 가질 수 있다.

하나의 룰은 여러 개의 필터들로 구성되고, 하나의 룰에 매칭되었다 함은 모든 필터들에 매칭되었음을 의미한다. 여기서 필터에 매칭되었다 함은 다음의 정의를 따른다. 각 필터는 비트 문자열로 패턴(pattern)과 마스크(mask)의 쌍으로 정의 되고 {0, 1, *}로 구성된 삼진문자(ternary-string)로도 정의 된다. (단, *는 don't care bit 을 의미 한다) 필터를 4bit 문자열로 예를 들면 패턴 '1001', 마스크 '1010' 으로 정의된 필터는 삼진문자로 '1*0*' 로서 비교하고자 하는 비트 문자열의 첫 번째와 세 번째 비트가 각각 1, 0 으로 정의된 비트 문자열을 필터에 매칭하였다고 말하게 된다. 여기서 don't care bit 로 정의된 두 번째와 네 번째는 고려하지 않게 된다.[7]

따라서 위의 정의를 바탕으로 Packet Classification 문제를 기술하면, 입력된 패킷으로부터 몇 가지 정보 필드(Field)를 이용하여 룰 테이블에서 가장 적합한 룰을 선택하는 작업이라고 말 할 수 있다.

일반적으로 Packet Classification 작업은 프로토콜 스택 중 4 계층(Layer 4; Session Layer)에서 수행되고 packet filtering, policy routing, accounting & billing, traffic rate limiting, traffic shaping 등의 서비스를 위해 이용되고 있다. 이 경우 주로 발신지/목적지 주소 (Source/Destination IP), 발신지/목적지 포트 (Source/Destination Port), 프로토콜(Protocol) 등의 5 개의 필드를 이용하여 탐색을 하며, 설정되는 룰의 개수는 적용되는 망의 크기, 속도, 방식에 따라 달라지지만 보통 수 백~수 만개 사이 이며 계속 늘어나는 추세이다. 그러므로 룰과의 비교시간이 성능에 많은 영향을 주게 된다.

본 논문에서는 여러 알고리즘 중 Cross-Product 룰 탐색 알고리즘의 실제 적용에 대해 알아 보겠다.[2]

2.2 Cross-Product 룰 탐색 알고리즘

Cross-Product 알고리즘은 Pankaj Gupta[2] 에 의해 제안 된 방식으로 룰의 탐색이 지역적(Geometric) 방식으로 이루어 진다. 아래와 같은 룰 테이블이 있을 경우 이들의 종류를 열(Column)별로 나누어서 그 내에 있는 모든 항목을 나열하고 Wildcard(*) 는 String Default 로 모든 것을 정의하게 된다. 아래의 테이블은 이를 기호로서 간략히 표현한 표이다.

Dest IP	Src IP	Dest Port	Src Port	Prot	Act
A	*	25	*	*	Allow
A	*	23	*	TCP	Allow
B	T1	55	*	UDP	Block
B	*	*	125	*	Block
B	T2	*	*	TCP	Allow
*	*	*	*	*	Block

(표 2) 룰 테이블

위 예에서는 5 개의 필드에 대해서 각각의 필드 값을 가지고 조합하여 검색을 하게 되는데, 우선, Prefix 테이블이라 하여 각 필드들의 값을 독립적으로 분리 검색 할 수 있는 테이블을 생성하고, 다음으로 이들의 인덱스 값을 곱한 cross-product 테이블을 생성하게 된다. 위의 경우 아래와 같은 Prefix 테이블과 3*3*4*2*3 = 216 개의 cross-product 테이블이 생성되게 된다.

3 • 3 • 4 • 2 • 3 = 216

Index	Dest IP	Src IP	Dest Port	Src Port	Prot
0	A	T1	25	125	TCP
1	B	T2	23	def	UDP
2	def	def	55		def
3			def		

(표 3) Prefix 테이블

NUM	Cross Product	Matching Filter
0	A,T1,25,125,TCP	Filter 1
1	A,T1,25,125,UDP	Filter 1
2	A,T1,25,125,def	Filter 1
3	A,T1,25,def,TCP	Filter 1
4	A,T1,25,def,UDP	Filter 1
5	A,T1,25,def,def	Filter 1
...	...	...
214	def,def,def,def,UDP	Filter 6
215	def,def,def,def,def	Filter 6

(표 4) Cross-Product 테이블

이렇게 생성된 prefix 테이블과 cross-product 테이블을 이용해 패킷의 룰 매칭작업을 수행하게 된다. 예를 들어 (A, T2, 33, 125, UDP) 의 패킷 헤더를 가진 패킷은 우선 prefix 테이블의 각 필드 별로 매칭 작업을 수행하여 각 필드들의 인덱스를 뽑아낸다. 이렇게 뽑아낸 인덱스는 (0, 1, 3, 0, 1) 의 인덱스를 가지게 된다. 이 인덱스 값들로 다음의 식을 통해 cross-product 테이블 인덱스를 구할 수 있다.

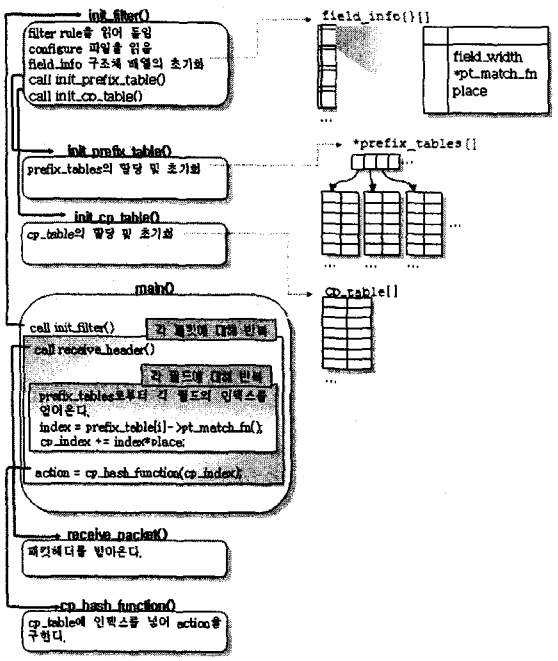
0 * (3*4*2*3) + **1** * (4 * 2 * 3) +**3** * (2 * 3) + **0** * (3) + **1** = 43

이렇게 구해진 43 인덱스는 cross-product 테이블의 44 번째 레코드를 가리키고 이를 참조해서 수행 해야 될 룰을 얻을 수 있다.

이렇게 구성되는 cross-product 는 이론상 50 개 이상의 룰을 적용하기에 메모리 오버헤드가 많이 걸리는 알고리즘이다. 만약 필드를 4 개 사용하고 50 개의 룰들의 각 필드들이 서로 다른 것을 정의 한다면 메모리 공간은 최대 50^4 개를 필요로 하고 한 레코드를 1byte 만 사용한다면 실제로 최대 50^4 byte = 6250000 byte = 6Mbyte 의 용량을 필요로 해 메모리 낭비가 다른 알고리즘에 비해 심하게 된다.

3. 설계 및 구현

본 논문에서 Cross-product 알고리즘은 C 로 작성되었고 portability 를 높이기 위해 어떠한 기본 라이브러리도 사용하지 않았다. 아래 도식은 cross-product 모듈과 동작 플로우에 대한 설명을 하고 있다.



(그림 2) 프로그램 플로우

이렇게 구현한 프로그램은 리눅스나 다른 임베디드 시스템에 손쉽게 적용되어 사용될 수 있다. 다음 장에서는 이렇게 구현한 cross-product 알고리즘의 성능측정에 대해 알아본다.

4. 실험평가

실험은 두 가지를 측정하는데 초점을 맞추었다. 첫째, cross-product 알고리즘의 성능을 측정하는 것과 둘째, 메모리 사용량을 측정하는 것이다. 실험방법은 시뮬레이션과 실제 테스트 모두 수행하였고 시뮬레이션을 통해 알고리즘의 성능을 예상하고 메모리 사용량을 측정하였다. 그리고 실제 환경에서의 성능을 측정하기 위해 real 망에서 패킷을 생성하여 stress 테스트 하였다.

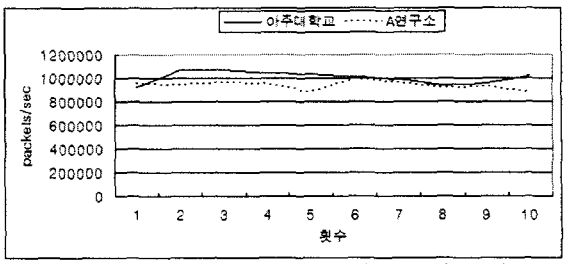
4.1 성능 측정 시뮬레이션

시뮬레이션은 일반 리눅스 PC 를 사용하였고 아주대학교 네트워크 망 방화벽 앞 단에서 캡쳐한 패킷 헤더 500M 가량을 실험에 사용하였다. 다음은 실험에 사용된 두 종류의 PC 사양을 나타내고 있다.

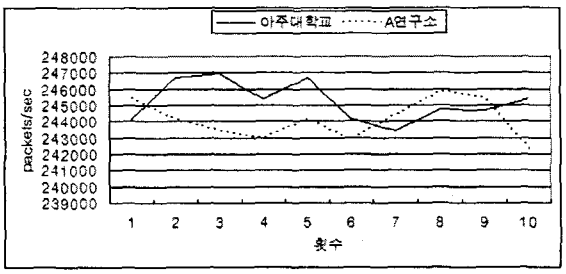
종류	사양
고성능 서버급	Red Hat Linux 7.0 Linux Kernel 2.4.18 CPU : Pentium III 1G * 2, 256Kb cache size RAM : 1G
저사양 일반 PC	Wow Linux 7.1 Linux Kernel 2.4.2 CPU : K6-2 300Mhz, 64Kb cache size RAM : 64Mb

(표 5) 시뮬레이션 PC 사양

룰 집합은 캡쳐한 패킷을 위해 아주대학교 측에 양해를 구해 네트워크 방화벽의 설정된 실제 룰 172 개와 A 연구소의 약 700 개 가량의 룰을 사용하여 실험에 사용하였다. 성능측정은 위 두 종류의 컴퓨터를 두어 단위 시간당 패킷 처리량을 계산하였고, 그 결과 다음과 같다.



(그림 3) 고성능 서버급에서의 패킷 처리량 비교



(그림 4) 저사양 PC 에서의 패킷 처리량 비교

그림에서 살펴볼 수 있듯이 cross-product 의 성능은 일반 PC 에서 룰의 개수와 관계없이 초당 20~100 만 개의 처리능력을 보여줌으로써 그 성능이 우수함을 증명하고 있다.

4.2 메모리 측정

Packet Classification 알고리즘들의 성능측정에 있어 중요한 사항으로 메모리 오버헤드를 들 수 있다. 본 논문에서 사용한 cross-product 의 가장 큰 단점으로 메모리 제약을 들 수 있는데 이를 측정하기 위해 임의의 랜덤한 룰을 발생 시키게 되면 4Mbyte 의 메모리 공간 제약을 두었을 때 40 개를 넘기 어렵다는 것을 시뮬레이션을 통해 측정하였는데 이는 cross-product 의 이론적 한계와 거의 일치하게 됨을 알 수 있다.

그러나 보통의 룰 집합은 사용자가 정의하고 이미 한계가 명확한 자신의 네트워크를 중심으로 룰을 정의 하기에 이를 현실적인 관점에서 살펴볼 필요가 있다. 이에 본 논문은 현실적인 메모리 측정을 위해 사용되고 있는 실제 방화벽들의 룰을 이용하여 사용량을 비교해 보았다. 그 결과, 실제 방화벽 룰 집합의 특징을 사용한 cross-product 테이블이 차지하는 prefix 테이블과 cross-product 테이블의 메모리 공간이 아래와 같이 매우 줄어들었음을 알 수 있다.

룰 개수	172 개 (A 대학교)	3741 개 (C 인터넷)
메모리 양	39Kbyte	2.6Mbyte

(표 6) 메모리 측정 결과

4.3 임베디드 시스템에서의 성능 측정

마지막으로 real 망에서의 stress 테스트를 하여 현실 적용 가능성을 살펴본다. 본 실험은 100Mbps 네트워크 상에서 테스트를 하였고 실험을 위해 본 연구실에서 만들고 있는 임베디드 시스템에 본 모듈을 적용하였다.[9] 이론상 100Mbps 의 망에서 요구하는 패킷 처리량은 (최소의 패킷 헤더만을 가진다면: 64byte) 약 144,000 개이다. 여러 가지 네트워크 상황과 한계를 고려해 stress 테스트를 통해 발생시킬 수 있는 패킷양은 초당 70,000 개 정도이다. 본 실험 결과 본 논문의 cross-product 알고리즘 모듈은 룰의 개수에 의존하지 않고 delay 없이 모두 룰 매칭을 수행하여 그 성능의 실현 가능성을 입증하고 있다.

5. 결론

본 연구에서는 cross-product 알고리즘을 이용해 실제 패킷 필터링을 수행하는 모듈을 설계 구성하였고, 이를 시뮬레이션 측정과 실제 측정을 모두 수행하였다. 실험 결과 cross-product 알고리즘은 룰의 개수에 상관 없이 일정한 성능을 보장 받을 수 있고, 그 성능 또한 빠름을 보여준다. 또한 이론상 cross-product 알고리즘이 가진 메모리 한계 역시 현실에 적용된 방화벽 룰들의 경우 약 700 개의 룰을 사용하여도 크게 문제가 되지 않음을 실험을 통해 확인할 수 있었다.

결론적으로 cross-product 알고리즘은 현실에서 정의된 방화벽 룰들로 검증한 결과 성능뿐만 아니라 메모리 공간에 있어서의 제약도 특수한 경우 사용 가능할 수 있음을 확인 할 수 있었다.

참고문헌

[1] P. Tsuchiya. "A search algorithm for table entries with non-contiguous wildcarding," unpublished report, Bellcore.

[2] V. Srinivasan, S. Suri, G. Varghese, and M. Waldvogel. "Fast and Scalable Layer four Switching," Proceedings of ACM Sigcomm, pages 203-14, September 1998

[3] T.V. Lakshman and D. Stiliadis. "High-Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Rage Matching", Proceedings of ACM Sigcomm, pages 191-202, September 1998.

[4] M.M. Buddhikot, S. Suri, and M. Waldvogel. "Space decomposition techniques for fast layer-4 switching,"Proceedings of Conference on Protocols for High Speed Networks, pages 25-41, August 1999.

[5] A Feldman and S. Muthukrishnan. "Tradeoffs for packet classification," Proceedings of Infocom, vol. 3, pages 1193-202, March 2000.

[6] Pankaj Gupta and Nick McKeown, Packet Classification on Multiple Fields, Proc. Sigcomm, Computer Communication Review, vol. 29, no. 4, pp 147-60, September 1999, Harvard University.

[7] Pankaj Gupta and Nick McKeown, Packet Classification using Hierarchical Intelligent Cuttings, Proc. Hot Interconnects VII, August 99, Stanford. This paper is also available in IEEE Micro, pp 34-41, vol. 20, no. 1, January/February 2000.

[8] V. Srinivasan, S. Suri, and G. Varghese. "Packet Classification using Tuple Space Search", Proceedings of ACM Sigcomm, pages 135-46, September 1999.

[9] Jongwook Moon,Hakbong Cho,Changseok Choi,Gihyun Jung, Younkwang Jung,Kyungehee Choi "Accelerating Firewall" IC'03