듀얼 해쉬 구조를 이용한 대용량 트래픽 처리 방법에 관한 연구

박태영[0], 윤성호, 김명섭

고려대학교 컴퓨터정보학과

{no14on, sungho_yoon, tmskim}@korea.ac.kr

A study on a resolution of the massive traffic using the dual hash structure

요 약

급격한 인터넷의 발전으로 효율적인 네트워크 관리를 위해 트래픽 데이터 분석의 중요성이 강조되고 있다. 트래픽 분석은 트래픽 데이터를 메모리에 적재하고 조건에 맞는 데이터를 검색하는 과정을 반복한다. 하지만, 트래픽 양이 증가함에 따라 데이터 적재 및 검색 시간이 증가하기 때문에 대용량 트래픽의 실시간 분석을 위해서는 효과적인 데이터 적재 및 검색 방법이 필요하다. 이러한 요구에 일반적으로 고정된 크기의 싱글 해쉬 구조를 사용하고 있지만, 증가하는 트래픽 데이터 양에 따른 해쉬 성능저하 및 테이블 확장 시 해쉬 함수를 재정의 해야 하는 한계점이 있다. 본 논문에서는 기존의 트래픽 데이터 적재 방법의 한계점을 보완하기 위해 듀얼 해쉬 구조를 제안한다. 듀얼 해쉬 구조는 트래픽 데이터의 양에 따라 확장이 용이하고, 해쉬 성능을 유지 할 수 있도록 구축 되었다.

1. 서론

최근, 인터넷 사용자의 증가와 고속 네트워크의 보급으로 인해 네트워크 트래픽이 급증하였다. 이것은 단순히 WWW, FTP, e-mail 과 같은 전통적인 인터넷 서비스뿐만 아니라 멀티미디어 스트리밍, P2P(peer-to-peer) 파일 공유, 게임과 같은 다양한 멀티미디어 서비스의 대중화에 원인이 있다. 급증하는 인터넷 트래픽을 효과적으로 관리하기 위해 다양한 트래픽 분석 방법이 제안되고 있다.

트래픽 분석은 분석 대상 트래픽 데이터를 메모리에 적재하고 조건에 맞는 데이터를 검색하여 트래픽의 원천(응용, 프로토콜 등)을 명명하는 과정을 반복한다. 하지만, 트래픽 양의 증가에 따라 데이터 적재 및 검색 시간이 증가하기 때문에 대용량의 트래픽 분석을 위해서는 효과적인 데이터 구조와 적재 및 검색 방법이 필요하다.

일반적으로 트래픽 데이터를 분석하기 위한 관리 방법으로, 싱글 해쉬 구조를 사용하고 있지만 많은 한계점을 가지고 있다. 싱글 해쉬 구조는 고정된 크기의 해쉬 테이블을 사용하기 때문에 대용량 트래픽 발생 시 충돌로 인해 링크드 리스트가 길어짐으로서 해쉬 성능의 저하를 초래한다. 즉, 늘어난 링크드 리스트에 따라 메모리 검색 횟수 또한 늘어나게 되어 오버헤드가 일어날 수 있다. 뿐만 아니라, 유동적인 트래픽 양에 적절한 해쉬 테이블 크기 선정 및 해쉬 테이블 확장 시 해쉬 함수 재정의 문제 등이 있다.

본 논문에서는 대용량 트래픽 처리 시 기존의 싱글 해쉬 구조의 한계점을 보완 하기 위한 듀얼 해쉬 구조를 제안한다. 듀얼 해쉬는 1 단계 해쉬 테이블의 각 인덱스에, 2 단계 해쉬 테이블을 연결 시키는 구조이다. 2 단계 해쉬 테이블의 크기는 고정시키고, 1 단계 해쉬 테이블의 크기는 트래픽 데이터를 메모리에 로드 하기 전 트래픽 양을 계산하여 적합한 크기로 선정함으로써 해쉬의 성능을 유지 할 수 있도록 하였다. 뿐만 아니라 해쉬 테이블 확장 시 최적의 2 단계 해쉬 함수를 중복 사용하기 때문에, 해쉬 함수의 재정의가 필요 없게 된다.

본 논문은 다음과 같은 순서로 기술한다. 2 장에서는 네트워크 트래픽 분석을 위한 기존의 데이터 적재 방법 및 유사연구에 대해 살펴보고, 3 장에서는 듀얼 해쉬 구조를 정의한다. 4 장에서는 제안한 듀얼 해쉬 구조의 성능을 증명 하기 위한 실험 결과를 기술하고, 마지막으로 5 장에서 결론과 향후 연구를 언급한다.

2. 관련연구

트래픽 분석의 중요성이 증가함에 따라, 증가하는 트래픽의 관리 또한 중요시 되고 있다. 본 장에서는 트래픽 데이터 저장 시 일반적으로 많이 사용하고 있는 고정된 크기의 싱글 해쉬 구조와[1], 유사 연구에서의 데이터 저장 방법에 대해 알아본다[2].

* 이 논문은 정부(교육과학기술부)의 재원으로 2010 년도 한국연구재단-차세대정보컴퓨팅기술개발사업(20100020728) 및 2012 년도 한국연구재단(2012R1A1A2007483)의 지원을 받아 수행된 연구임.

싱글 해쉬 구조는 고정된 크기의 해쉬 테이블을 생성하고 데이터에서 추출한 Key 값으로, 해쉬 함수를 통해 index 를 계산하여 해당 index 에 데이터를 적재한다. 만약 동일한 index 에 둘 이상의 데이터를 적재해야 하는 경우(충돌: collision), 링크드 리스트 데이터 구조로 연결한다.

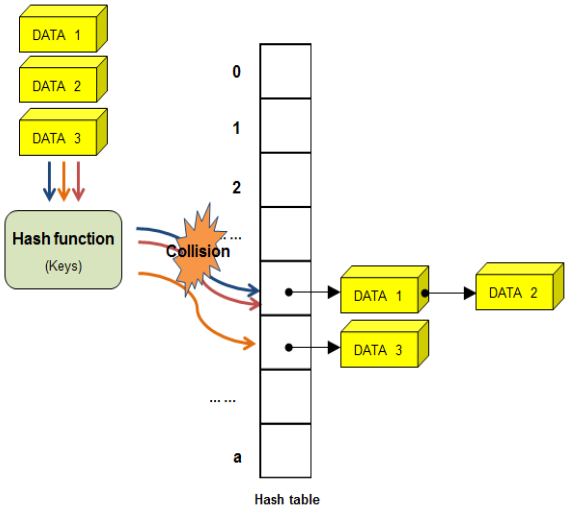

그림 1. 싱글 해쉬 구조

그림 1 은 트래픽 분석을 위한 일반적인 데이터 구조를 나타낸다. 응용의 종류가 다양해지고 사용자가 증가함에 따라 일반적으로 사용하는 싱글 해쉬 구조는 데이터 적재 및 검색 시 처리 시간이 증가하여 트래픽 분석에 있어 큰 오버헤드로 작용한다. 이는 해쉬 테이블의 크기는 고정되어 있지만, 트래픽 데이터의 양은 계속 증가하기 때문이다. 또한, 해쉬 테이블 사용률과 동일 index 에 링크드 리스트로 적재된 데이터의 개수가 증가하여, 적재 및 검색 시 데이터 비교 횟수가 증가하기 때문이다.

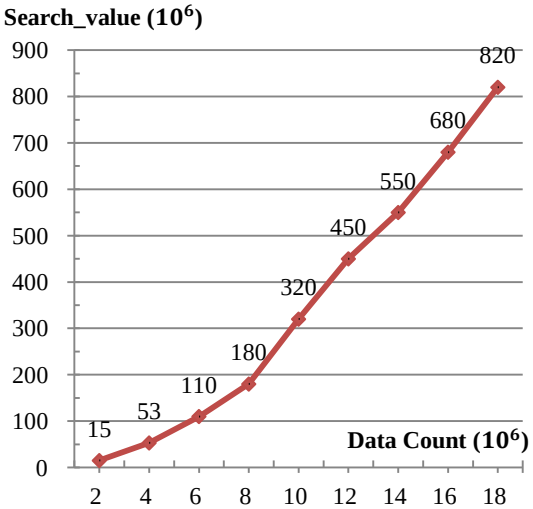

그림 2. 고정된 크기의 싱글 해쉬 구조에서 트래픽 데이터 증가에 따른 Search_value

Search_value 는 해쉬 테이블에 저장된 모든 트래픽 데이터를 1 회씩 비교 검색할 때 소요되는 총 비교 횟수로 해쉬 테이블의 성능을 평가하기 위해 사용된다. 그림 2 는 고정된 크기의 싱글 해쉬 구조(테이블 크기: 65,536(2^{16}))에서 트래픽 데이터 양의 변화에 따른 모든 데이터를 검색하기 위한 비교횟수(Search_value)를 나타낸 것이다. 고정된 크기의 싱글 해쉬 구조에서는 트래픽 데이터 양이 8×10^6이 되면 Search_value 가 급격하게 증가함을 알 수 있다. 이는 8×10^6이상의 데이터를 2^{16}의 테이블 크기에 적재 하고자 하면, 링크드 리스트가 길어지게 될 수밖에 없어 검색 횟수가 증가한다는 뜻이다. 이러한 문제를 해결하기 위한 가장 간단한 방법은 해쉬 테이블의 크기를 증가시키는 것이다. 하지만, 분석 시스템에서 사용할 수 있는 메모리의 크기는 제한이 있기 때문에 무한정 해쉬 테이블의 크기를 증가시키는 것은 불가능할 뿐만 아니라, 적절한 크기의 해쉬 테이블을 찾는 것 또한 쉽지 않다. 또한, 한시적으로 트래픽 양에 적절한 해쉬 테이블 크기를 찾더라도, 그에 적합한 해쉬 함수를 재정의 해야 할뿐만 아니라 트래픽 데이터의 양은 지속적으로 변하기 때문에 반복된 작업을 계속해야 하는 어려움이 있다.

데이터를 저장하는 방법에 있어 유사 연구로, 효율적인 라우팅 테이블의 검색 방법에 관한 연구가 진행되고 있다. [2] 연구는 여러 개의 해쉬 함수를 사용하는 것이 하나의 해쉬 함수를 사용하는 것보다 성능이 크게 개선되는 것을 분석을 통해 예측하고 실험하였다. 해쉬 테이블은 해쉬 함수의 수만큼 테이블을 생성하고, 하나의 데이터가 여러 개의 해쉬 함수(multiple hash)를 통해 얻은 index 값을 각 테이블의 포인터로 하여, 테이블 중 로드가 적은 쪽의 테이블에 채워지는 방식이다. 따라서, 데이터가 테이블에 균등하게 분포되는 방식이다. 그러나, [2] 에서 제안하는 방법은 충돌에 대한 고려가 없고, “multiple hash” 함수를 사용하기 위해서 데이터의 분포에 따라 적합한 해쉬 함수를 찾아내야 한다. 또한, 해쉬 테이블 업데이트를 위한 과정에서, 해쉬 테이블이 꽉 차 있는 경우 현재의 데이터 분포에 맞는 “multiple hash” 함수를 찾아내는 과정이 다시 수행되어야 하는 단점이 있다.

따라서 효율적인 데이터 관리를 위한 데이터 구조와 적재 및 검색 방법이 필요하다.

3. 듀얼 해쉬 구조

본 장에서는 효과적인 트래픽 분석을 위한 듀얼 해쉬 구조와 적재 및 검색 방법을 기술한다. 본 논문에서 제안하는 방법은 기존의 싱글 해쉬 구조의 한계점을 보완한 듀얼 해쉬 구조 이다. 1 단계 해쉬 테이블의 길이는 저장되는 트래픽 데이터의 양(N)에 따라 변동 가능한 α로 설정하고, 2 단계 해쉬 테이블의 길이는 β로 고정하여 설정한다.

그림 3 은 본 논문에서 제안하는 듀얼 해쉬 데이터 구조를 나타낸다. 트래픽 데이터의 양(N)은 데이터를 메모리에 로드 하기 전에 트래픽 데이터의 파

일 크기를 데이터 크기로 나누어 구한다. 트래픽 데이터 양(N)에 따라, 알고리즘 1 을 통해 계산한 α 길이로 1 단계 해쉬 테이블(First_Hash table)을 생성하고, 각 index 에 β 길이의 2 단계 해쉬 테이블(Second_Hash table)을 생성한다. 데이터를 적재할 경우, 알고리즘 2 의 1 단계 해쉬 함수(First_Hash function)와 알고리즘 3 의 2 단계 해쉬 함수(Second_Hash function)를 통해 각 단계의 해쉬 테이블 index 값을 얻어 해당 위치에 데이터를 적재한다. 만약 서로 다른 데이터가 동일한 index 값을 가지는 경우 (충돌: collision), 링크드 리스트 데이터 구조로 연결한다. 데이터를 검색하는 방법은 데이터를 적재하는 방법과 동일하게 동작한다.

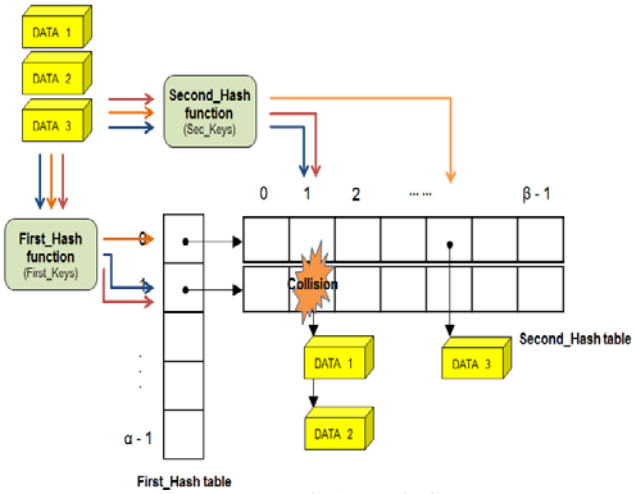

그림 3. 듀얼 해쉬 데이터 구조

```
Input : N (the number of data)

Output : α (size of First_Hash table)

BEGIN
int FindAlpha(int N)
                // (amounts of data N = File size / Data size)
{
        β = 65,536      // size of Second_Hash table
        int k ;

        find k such that  2^k ≤ N / (1.2 × ß) ≤ 2^(k+1)
        if ( | 2^k − N / (1.2 × ß) | ≤ | 2^(k+1) − N / (1.2 × ß) | )
        then   α = 2^(k+1)
        else   α = 2^k
        return α;
}
END
```

알고리즘 1. 1 단계 해쉬 테이블 크기

알고리즘 1 은 1 단계 해쉬 테이블의 크기 α를 계산하는 과정을 나타낸다. 해쉬 사용률이 80% 이상 되면 성능 저하가 시작 되기 때문에[3] 해쉬 사용률이 80% 이하가 되는 최대값 α 를 선정하고, β 는 IP_Address 의 앞, 뒤 2byte 씩 Folding 하여 Second_Hash 함수의 Key 로 사용하기 때문에 2byte, 즉 65,536(2^{16})길이로 정의한다.

해쉬 함수의 Key 로 사용할 데이터는 Flow 이고, Flow 의 정의는 다음과 같다. 단방향 Flow(One-way-flow)는 5-tuple(Source Ip, Source Port, Destination IP, Destination Port, Layer4-protocol)이 동일한 단일 방향의 연속된 패킷의 집합으로 정의하고, 양방향 Flow(Two-way-flow)는 단방향 Flow 와 반대 방향의 단 방향 Flow 의 집합으로 정의하였다. 본 논문에서는 양방향 Flow 를 사용하고, Key 값으로 5-tuple 을 전달한다. 해쉬 함수의 성능은 충돌을 최소화 시키는 것이 관건이며, 임의의 입력 집합에 대하여 균일한 출력 값을 가져야 한다[4].

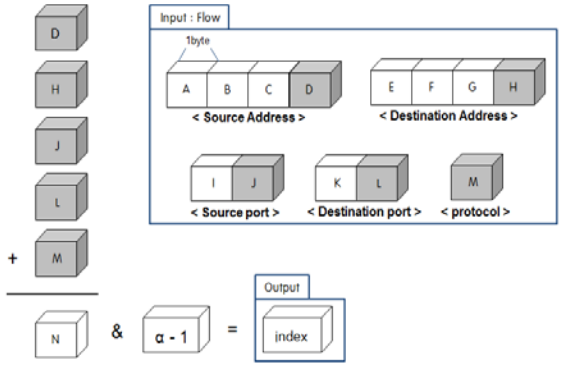

그림 4. 1 단계 해쉬 함수

그림 4 는 1 단계 해쉬 함수로, Source Address 와 Destination Address 의 뒤 1byte 가 다양한 값을 가지고 있기 때문에 각 1byte 씩 0 ~ 255 사이 수를 합하고, Source port 와 Destination port 그리고 protocol 의 뒤 1byte 를 모두 합한 값을 1 단계 해쉬 함수(First_Hash table)의 길이 α − 1과 AND 연산을 함으로써 First_Hash table 의 index 값(0 ~ α − 1) 을 얻는다.

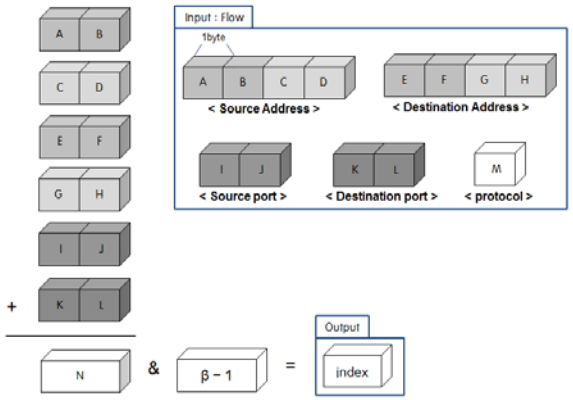

그림 5. 2 단계 해쉬 함수

그림 5 는 2 단계 해쉬 함수로써, Source Address 와 Destination Address 의 앞, 뒤를 Folding 방법[5]으로 2byte 씩 더하고, Source port 와 Destination port 의 2byte 를 더한 값에 2 단계 해쉬 함수(First_Hash table) β − 1(65,535, 0xFFFF)값과 AND 연산을 함으로써 인덱스 값 (0 ~ 65,535)을 얻는다.

표 1. 싱글 해쉬 & 듀얼 해쉬 실험 결과

Data / Hash	3		6		10		13		16		20	
	S_v	size	S_v	size	S_v	size	S_v	size	S_v	size	S_v	size
S1	89	0.25	291	0.25	865	0.25	1,474	0.25	2,119	0.25	3,179	0.25
S2	46	0.5	148	0.5	438	0.5	744	0.5	1,068	0.5	1,600	0.5
S3	25	1	77	1	224	1	379	1	542	1	810	1
S4	14	2	42	2	117	2	196	2	279	2	415	2
D	6	8	11	16	17	32	25	32	25	64	33	64

S1: 싱글(65,536), **S2**: 싱글(65,536×2), **S3**: 싱글(65,536×4), **S4**: 싱글(65,536×8), **D**: 듀얼(65,536× α), **S_v**: Search_value (단위 : 10^6)

듀얼 해쉬 구조는 기존의 싱글 해쉬 구조와 달리,트래픽 데이터를 메모리에 로드 하기 전에 트래픽 데이터 양을 확인하고, 그에 적절한 크기의 해쉬 테이블로 확장가능 하다. 그 이유는 테이블이 확장 될 시, 2 단계 해쉬 함수가 중복 사용되는 구조이므로, 해쉬 함수를 재정의 할 필요 없이 성능유지가 가능하기 때문이다. 네트워크 트래픽 양은 유동적이기 때문에, 그에 적합한 가변적인 해쉬 테이블을 사용 함으로써 효과적인 데이터 메모리 적재 및 검색이 가능하다.

4. 실험 및 평가

본 장에서는 3 장에서 제안한 듀얼 해쉬 구조의 성능을 증명 하기 위해 증가하는 트래픽 데이터 양에 따른 싱글 해쉬 구조와 듀얼 해쉬 구조의 성능 비교 결과를 기술한다.

표 1 은 트래픽 데이터 양에 따른 Search_value 값과 해쉬 테이블 크기에 대한 비교 결과이다. 싱글 해쉬 구조는 해쉬 테이블 길이(2^{16}, $2^{16} \times 2$, $2^{16} \times 4$, $2^{16} \times 8$)를 서로 다르게 고정 시키고, 듀얼 해쉬 구조는 트래픽 양의 변화에 알고리즘 1 을 통해 가변적으로 해쉬 테이블 길이($2^{16} \times \alpha$)를 정한다.

그림 6 과 같이 싱글 해쉬 구조는 해쉬 테이블의 길이가 고정되어 있기 때문에, 트래픽 양이 증가할수록 충돌 횟수 또한 증가하고, 따라서 링크드 리스트가 길어지게 되어 Search_value 가 급격하게 증가한다. 반면에, 듀얼 해쉬 구조는 트래픽 데이터 양의 증가에 적절한 크기의 해쉬 테이블이 생성 되기 때문에 점진적으로 Search_value 가 증가하는 것을 확인할 수 있다.

그림 7 과 같이 듀얼 해쉬는 해쉬 테이블이 확장되어 메모리를 많이 사용하였지만 해쉬 성능에서 트래픽 양의 변화와 관련 없이 좋은 성능을 유지할 수 있다. 뿐만 아니라, 저장 될 데이터의 크기와 비교하면 적은 메모리를 사용하는 것이고, 성능 유지를 위한 최소한의 메모리를 사용하기 때문에 효율적이라고 할 수 있다. 그리고, 트래픽 양의 변화에 대해, 해쉬 함수의 재정의 없이 자동적으로 해쉬 테이블을 확장 시켜줌으로써 효과적인 분석이 가능하다.

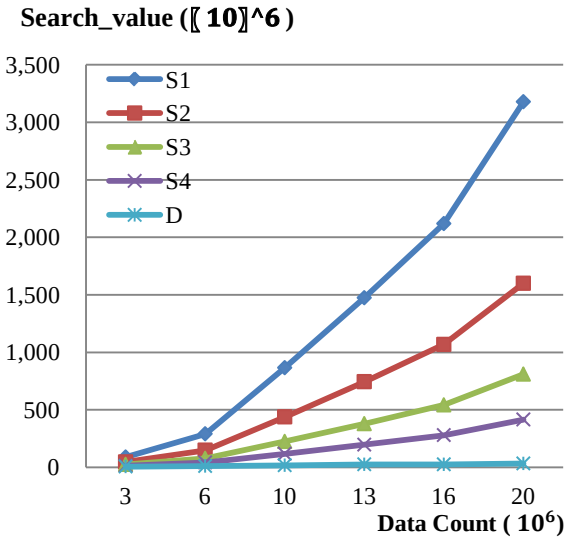

그림 6. 싱글 해쉬 & 듀얼 해쉬 구조 성능 비교

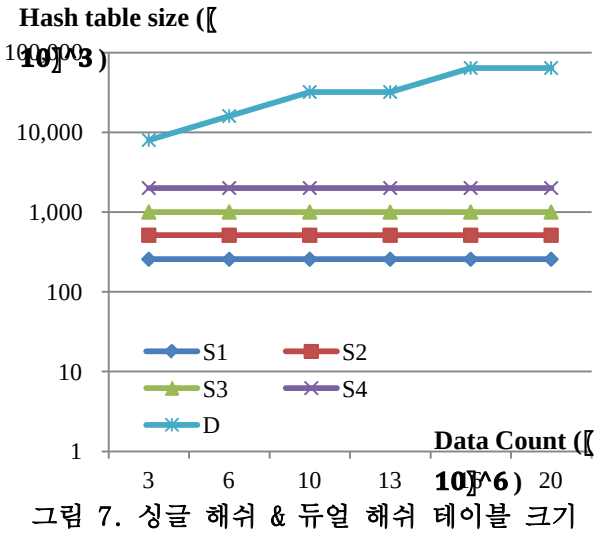

그림 7. 싱글 해쉬 & 듀얼 해쉬 테이블 크기

5. 결론 및 향후 연구

네트워크에서 발생하는 트래픽이 복잡 다양해지고 있고, 대용량의 트래픽이 유동적으로 발생하게 되어 기존의 싱글 해쉬 구조로 데이터의 관리가 어려워졌다.

본 논문에서는 대용량 트래픽 데이터의 메모리 적재 및 검색 방법에 대해 싱글 해쉬 구조의 한계점

을 보완한 듀얼 해쉬 구조를 제안하였다. 고정된 싱글 해쉬 구조와 달리, 유동적으로 변하는 트래픽 양에 대해 적절한 해쉬 테이블 크기를 선정하고, 확장하여 트래픽 양의 변화에도 우수한 성능을 가질 수 있다. 향후 연구로 제안하는 방법은 1 단계 해쉬 테이블의 크기뿐만 아니라, 2 단계 해쉬 테이블 크기 또한 가변적인 구조로 바꿔 다양한 양의 데이터를 처리 함에 있어, 성능 향상을 이끌어 낼 계획이다.

참고문헌

[1] 김명섭, 박상훈, 박진완, “Flow 기반 실시간 트래픽 수집 및 분석 시스템”, 제 28 회 한국정보처리학회 추계학술발표대회, 제 14 권 제 2 호, Nov. 2007.

[2] A. Broder and M. Mitzenmacher, “Using Muliple Hash Functions to Improve IP Lookups”, IEEE INFOCOM, pp. 1454-1463,2001.

[3] 박상현, “뇌를 자극하는 알고리즘”, 한빛미디어, pp. 343, 2009.

[4] Rich Seifert, “The Switch book”, wiley, 2000.

[5] MAUER, W., AND LEWIS, T. “HASH TABLE METHODS”, Comput. Surv. 7, 1, 5-19, March, 1975.