DBpia

논문번호 12-37B-09-15

한국통신학회논문지 '12-09 Vol.37B No.09
http://dx.doi.org/10.7840/kics.2012.37B.9.849

개선된 Elephant Flows 발견 알고리즘

정 진 우◆°, 최 윤 기*, 손 성 훈**

An improved algorithm for Detection of Elephant Flows

Jinoo Joung◆°, Yunki Choi*, Sunghoon Son**

요 약

본 논문에서는 빠르고 정확하게 elephant flow를 발견할 수 있는 알고리즘을 제시한다. 최근 인터넷 사용자의 증가와 다양한 응용 프로그램의 등장으로 인하여, 네트워크 트래픽의 대규모화가 급속히 진행되고 있는 추세이다. 이러한 변화에 따라 네트워크 대역의 상당 부분을 점유하는 elephant flow 가 자주 발생하게 되었다. Elephant flow는 인터넷 트래픽의 관리 (management) 및 서비스 측면에서 네트워크 대역 (network bandwidth)의 불공평한 사용 문제를 유발한다. 본 논문에서는 Elephant flow를 발견하는 방법들 중 하나인 기존 Landmark-LRU 기법에 간단한 메커니즘을 추가시켜, 발견율을 크게 증가시키는 방법을 제시하였다. 그리고 제안하는 개선안을 실제 네트워크에서 추출한 트레이스 (network traces)에 적용하는 시뮬레이션을 통하여 평가하였다. 그 결과로 우리가 제시하는 개선 알고리즘이 효율적인 메모리 비용을 유지하면서 Landmark-LRU 기법보다 더 정확하게 elephant flow를 발견하는 것을 확인할 수 있었다.

Key Words : Elephant Flows, Flow Detection, Traffic Managements, Algorithm, QoS

ABSTRACT

We proposed a scheme to accurately detect elephant flows. Along the ever increasing traffic trend, certain flows occupy the network heavily in terms of time and network bandwidth. These flows are called elephant flows. Elephant flows raises complicated issues to manage for Internet traffics and services. One of the methods to identify elephant flows is the Landmark LRU cache scheme, which improved the previous method of Least Recently Used scheme. We proposed a cache update algorithm, to further improve the existing Landmark LRU. The proposed scheme improves the accuracy to detect elephant flow while maintaining efficiency of Landmark LRU. We verified our algorithm by simulating on Sangmyung University's wireless real network traces and evaluated the improvement.

Ⅰ. 서 론

최근 인터넷의 급속한 발전 및 보급으로 인하여, 네트워크 트래픽 (network traffic)의 대규모화가 급속히 진행되고 있는 추세이다. 또한, 기존 최선형 (best effort)의 서비스에서 벗어나, 품질지향형 서비스로의 진화를 요구하는 실정이다.

이러한 변화에 따른 QoS (quality of service) 지원을 위하여 네트워크 측정 (network measurement) 기술을 필수적으로 요구하고 있다. 네트워크 측정은 네트워크 설계 및 운영, 과금, QoS 제어, 보안 등의 제공을 위해서도 중요한 역할을 한다. 인터넷 서비

※ 본 연구는 한국연구재단 기초연구사업-일반연구자지원사업-기본연구지원사업(2012-0001351) 지원으로 수행되었습니다.
◆° 주저자 및 교신저자 : 상명대학교 컴퓨터과학부 부교수, jjoung@smu.ac.kr, 정회원
* 상명대학교 일반대학원 컴퓨터과학과 석사과정, filterk@sangmyung.kr, 준회원
** 상명대학교 컴퓨터과학부 조교수, shson@smu.ac.kr
논문번호 : KICS2012-03-154, 접수일자 : 2012년 3월 30일, 최종논문접수일자 : 2012년 9월 3일

www.dbpia.co.kr

스에 대한 QoS를 지원하기 위한 다양한 노력이 지속적으로 시행되고 있으나, 인터넷 사용자의 지속적인 증가로 인한 트래픽의 대규모화와 다양한 응용 프로그램의 등장은, 인터넷 트래픽의 특성을 보다 복잡하게 만드는 요인이 되었다. 특히 P2P (peer to peer), 웹 하드 등의 고용량 파일을 전송하는 서비스는 large data, high traffic을 유발하는 경향이 있다. 이는 특정 사용자가 전체 네트워크 대역의 일부분을 홀로 특정 시간 동안 점유하고 있는 현상이 발생하는 요인이 되었다.

Elephant Flow는 한 네트워크 링크에서 특정 시간동안 전체 대역 (bandwidth)의 일부를 홀로 차지하고 있는 네트워크 플로우 (network flow)이다. 즉, 극도로 많은 바이트 수를 가지고 있는 네트워크 플로우를 가리킨다. Elephant flow는 다른 용어로 heavy-hitter flow, large flow라고 불리며, 이와 반대되는 속성을 가진 플로우를 mice flow 또는 small flow라 한다. 그리고 실제 네트워크 트래픽은 elephant flow와 mice flow가 다양하게 공존하고 있는 형태이다.

Elephant flow는 전체 대역 공유의 불균형을 유발하는 문제점을 가지고 있다. 이 문제점은 대역 관리 및 과금 측면에서 문제를 발생 시키므로, 관리해야 할 네트워크 트래픽 요소 중 하나이다. 예를 들면, elephant flow들을 구성하고 있는 플로우 개수는 전체 플로우 수의 0.02%에 불과하지만, 대역을 차지하고 있는 비율은 60%나 되는 특성을 가지고 있는 사례가 있었다[9].

이러한 이유로 Elephant flow를 발견하기 위해 다양한 방법들이 제시되었다. 물론 단순 전수 조사 후 측정은 가장 정확한 결과를 보여준다. 하지만, 확장성 (scalability) 및 실시간 지원에서 비효율적이다. 이는 인터넷 트래픽 양의 폭발적인 증가와 메모리 (memory)를 대표로 한 샘플링 장비들의 자원들은 제한되어 있기 때문이다.

따라서, 위의 문제들을 해결하기 위해 효율적인 알고리즘을 도입하여, 트래픽을 측정하는 기법들이 다양하게 연구되고 있다. 이 중 elephant flow를 효율적으로 발견하기 위해 자원을 적게 사용하고, 높은 정확도를 가지는 측정 방법들이 다양하게 제시되고 있다.

이러한 방법들 중 하나인 Least Recently Used (LRU) 캐시 (cache) 기법[4]은 구현이 쉽고, 제한된 메모리에서 빠른 속도로 elephant flow 발견이 가능한 장점이 있다. 하지만 다양한 mice flow 들이 자주 캐시로 들어오게 되면, 저장하고 있는 실제 elephant flow 정보가 처리되기 전에 빠르게 삭제될 수 있는 단점이 있다. 이 단점은 정확한 elephant flow 검출을 어렵게 만드는 상황을 유발한다[1].

Landmark-LRU 기법은, 이러한 LRU의 단점을 극복하기 위해 캐시 안에 Landmark라 명명된 공간을 따로 설정하여, 기존 LRU 기법에서 빠르게 손실되는 elephant flow 정보를 보존하기 위해 새로이 제안된 방법이다. 이는 LRU 기법의 간단함을 유지하면서, LRU 사용 시 자주 들어오는 mice flow에 의해 elephant flow 정보가 빠르게 삭제되는 경우를 방지함으로써, LRU 기법의 단점을 극복하는 방법을 제시한다[1].

하지만 캐시 내 Landmark의 크기를 어느 정도로 할당하는 가에 따라 elephant flow를 발견할 확률이 크게 변하는 단점이 있으며, Landmark 크기를 너무 작게 할당하면, 기존 LRU와 크게 다르지 않다는 문제점이 발생하며, 너무 크게 할당하면 대량의 elephant flow 정보 저장이 어려워지는 가능성이 발생한다.

본 논문에서는 이러한 문제점을 극복하기 위해, Landmark-LRU의 간단한 구조를 그대로 유지하며, 기존 알고리즘보다 elephant flow의 발견 확률을 높이는 알고리즘을 제안한다. 그리고 우리가 제시하는 알고리즘을 실제 추출한 무선 네트워크 트레이스 (network traces)에 적용하여 기존 기법들과 성능을 비교 및 분석하였다.

Ⅱ. 관련 연구

2.1. Definition of Elephant Flows

인터넷 트래픽을 구성하고 있는 패킷들은 protocol type, source address, destination address, port number, length 등의 속성들을 포함하고 있다. 네트워크 플로우 (network flow)는 1) 특정 측정 장소에서 2) 특정 시간동안 3) 측정되어, 공통된 속성을 가진 패킷들의 집합으로 분류한 것을 나타낸다. 플로우를 정의하는 패킷의 속성들의 집합을 flow id라 부른다[7]. 플로우의 크기는 패킷의 카운트 수 또는 바이트 카운트 수로서 정의할 수 있으며, 크기를 기준으로 두 종류의 플로우: mice flows (short lived flows)와 elephant flows (long lived flows)로 구분 지을 수 있다[8].

링크 내 elephant flows의 수는 매우 적으나, 전체 트래픽을 점유하는 크기 (volume)는 매우 크다.

www.dbpia.co.kr

연구에 따르면 elephant flows는 전체 플로우 수의 0.02%만 차지하고 있으나, 전체 대역의 60%를 차지하고 있는 사례가 있었다[9]. Elephant flow의 기준은 네트워크 운영 측의 기준에 따라 결정된다[8]. 이 논문에서, 우리는 [9]와 같게 전체 트래픽의 0.1% 이상을 차지하고 있는 플로우를 elephant flow로 정의한다.

2.2. Sample & Hold (SH) Scheme

Elephant flows 식별을 위해서 일반적으로 패킷 카운터 (counter)를 사용하여 측정하는 기법들이 제시되었다. 패킷 카운터는 패킷의 수 또는 패킷 페이로드의 바이트 (bytes) 수를 카운트하는 것을 말한다. 패킷 카운트 도중, 패킷 카운트 수가 지정된 threshold를 초과하면 해당 플로우는 elephant flow라고 정의할 수 있다. 이 중 Sample & Hold(SH) 기법[3]은 flow id와 해당하는 패킷 카운트를 이용하여 elephant flow를 찾아내는 방법을 제시하였다.

SH 기법에서는 패킷 전수 조사를 하지 않고 확률 p에 따른 트래픽의 일부만 샘플링을 하여 elephant flow를 발견하는 방법을 제시하였다. 즉, 트래픽에 속한 패킷들은 전부 샘플링 되는 것이 아니라, 지정된 $1/p$의 확률로 샘플링이 된다. 샘플링된 패킷들은 샘플링 모듈의 메모리에 각 flow id에 따른 엔트리를 생성하여 flow id와 패킷 카운트 두 가지 정보로 저장된다. 지속적인 샘플링을 통해 일정 시간이 흐른 후, 기록된 패킷 카운트가 지정된 threshold T를 초과하게 되면 해당 플로우는 elephant flow로 결정된다. 이 기법은 패킷 전수 조사를 하지 않고 elephant flow를 발견하는 방법을 제시하였다. 이에 따라 네트워크 측정에 사용되는 메모리 비용을 절감하는 효과를 얻을 수 있다. 하지만, 확률적인 샘플링을 수행하므로, 정확한 측정을 위한 샘플링 후 추정 방법이 요구된다.

또한 [3]에서는 Sample & Hold 외 필터 (filter) 기반의 parallel multistage filter 기법을 제시하였다. [3]에서 제시하는 필터링 모듈은 여러 단계의 hashing stage를 가진다. 각 hashing stage는 각각 다른 hash function에 따라 플로우를 분류하며, 각 bucket에 hash function에 따른 플로우의 카운트를 유지한다. 각 stage는 병렬 구조로서 동작한다. 패킷이 들어올 때, 각 hashing stage에서 hash function에 따라 플로우가 구분되고, 해당 패킷의 flow id에 매칭되는 bucket에 패킷 카운트가 저장된다. 이 후, 각 stage에 저장된 패킷 카운트의 크기가 모두 지정한 threshold T를 초과하게 되면 해당 플로우는 elephant flow로 결정된다.

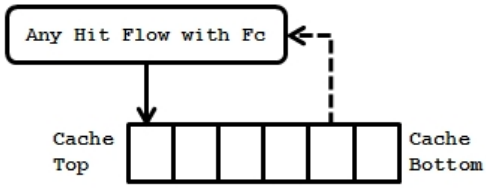


Fig. 1. The LRU Scheme.

Parallel multistage filter 기법은 패킷 전수 조사에 사용할 수 있으며 또한 좋은 측정 결과를 보인다[2,3]. 하지만 이 기법은 특수 하드웨어 기반으로 구현해야 한다[2]. 하드웨어 기반의 방법은 뛰어난 성능을 보이나, 소프트웨어 기반 방식에 비하여 구현 비용이 높고 다양한 상황에서 유연성이 부족하다[2]. 본 논문에서 제시하는 알고리즘은 간단한 소프트웨어 기반 방식으로, 트래픽 측정 시 적은 비용을 목표로 연구를 진행하였다.

2.3. Least Recently Used (LRU) Scheme

Least Recently Used (LRU) 캐시 기법은 대표적인 캐시 교체 기법 중 하나이다. LRU 캐시 기법은 저장된 시점과 관계없이, 가장 적게 사용된 엔트리를 교체한다. LRU 캐시 기법을 elephant flow 발견 사용에 적용하면[4], 맨 처음 들어온 플로우는 캐시 top에 들어오게 된다. 이 후 들어온 플로우에 대하여 cache miss가 발생하면, 캐시 top에 들어오게 되며, 기존 top에 있는 플로우 엔트리는 한 블록 아래로 내려가 위치하게 된다. 그리고 들어온 플로우가 이미 캐시 내에 존재하여 cache hit가 발생하게 되면, 해당 플로우 엔트리의 패킷 카운트를 업데이트 하여, 캐시 top으로 이동시킨다. 그리고 캐시가 가득 채워질 경우, bottom에 있는 플로우 엔트리 정보는 캐시에서 버려지게 된다. 그림 1은 이러한 과정을 보여준다.

위의 과정을 계속 반복하면 elephant flow는 cache hit의 빈도가 높아지게 되므로, 캐시 내 저장되는 시간이 증가하게 된다. 반면에 mice flow는 빠르게 캐시에서 삭제되어 캐시 내에는 elephant flow 정보가 유지되는 시간이 높아진다. 하지만 mice flow의 점유율이 높은 네트워크 환경인 경우, 어느 특정 플로우가 실제 elephant인 플로우임에도 불구하고, 패킷 카운트가 elephant flow로 인식되기 전에 캐시에서 빠르게 방출될 수 있는 단점이 존재하게 된다.

www.dbpia.co.kr

2.4. Landmark-LRU (L-LRU) Scheme

Landmark LRU (L-LRU) 캐시 기법은 LRU 기법의 mice flow가 많은 경우, 캐시 내 elephant flow가 빠르게 삭제되는 단점을 극복하기 위하여 제안된 방법이다[1].

LRU와 동일한 캐시 내에 "Landmark"라 명명된 공간을 따로 설정하는 차이점이 있으며, 새로 들어오는 플로우는 Landmark top으로 들어오게 된다. 그리고 Landmark 내에서 동작은 기본적으로 기존 LRU와 동일하다. 하지만 샘플링 도중 Landmark 안의 패킷 카운트 Fc가 지정된 elephant 크기 E 이상일 경우, Landmark가 아닌 캐시의 top으로 해당 플로우 엔트리를 옮긴다. 그림 2는 이러한 Landmark-LRU의 동작을 나타낸 그림이다.

따라서 Landmark가 아닌 구역의 캐시는 elephant flow 엔트리만 저장하게 되며, 이는 LRU 기법과 비교하여 상대적으로 정확한 elephant flow 발견이 가능하게 된다. Landmark가 아닌 캐시도 내부적으로 LRU 기법으로 동작한다.

하지만, 캐시 안에 Landmark 크기를 얼마나 설정할 것인지에 대한 문제점이 존재하며, Landmark의 크기에 따라 기존 LRU의 단점이 발생할 수 있다. Landmark의 크기가 너무 작으면, mice flow가 많은 네트워크 환경일 경우 elephant flow 정보도 빠르게 캐시에서 삭제되는 단점이 발생한다.

또한, Landmark 크기가 너무 크면, 많은 양의 elephant flow를 오랫동안 저장하지 못하는 단점이 발생한다.

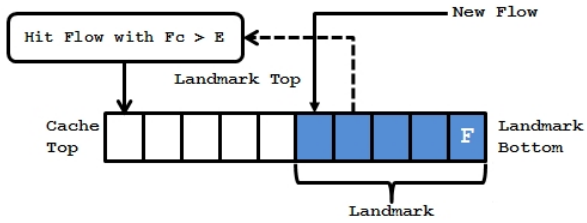


Fig. 2. The Landmark-LRU Scheme.

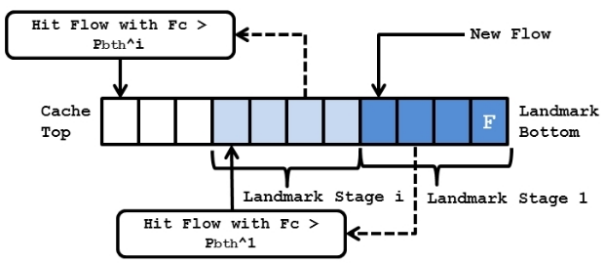


Fig. 3. The Multi-stage Landmark-LRU Scheme.

2.5. Multistage Landmark-LRU (ML-LRU) Scheme

Multi-Stage Landmark-LRU (ML-LRU) 기법은 Landmark-LRU의 후속 연구로 진행된 기법이다[2]. ML-LRU 기법의 기본 동작 구조는 Landmark LRU와 동일하다. 기존 기법과의 차이점은, 한 캐시 내에 Landmark 구역을 한 개가 아닌, 여러 개의 Landmark를 할당하여, elephant flow의 발견율을 향상시켰다. 구조는 다음과 같다. 제일 하단에 위치한 Landmark를 1 stage라 가정하고, 한 캐시 안에 i stage의 Landmark 구역들을 가지는 구조이다. 새로 들어오는 플로우는 1 stage Landmark의 top으로 들어오게 된다. 그리고 Landmark 내에서의 동작은 기본적으로 기존 LRU와 동일하다. 패킷 샘플링 도중 1 stage Landmark 안의 패킷 카운트 Fc가 지정된 threshold P_{bth}^{1} 이상일 경우, 다음 단계인 2 stage Landmark top으로 이동하게 된다. 샘플링 도중 2 stage Landmark 안의 패킷 카운트 Fc가 stage에 지정된 threshold P_{bth}^{2} 이상이 되면, 다음 stage의 Landmark top으로 이동한다.

즉, 캐시 하나는 i 개의 stage로 구성되어 있으며, 각 stage들은 각 threshold P_{bth}^{i} 를 가진다. 이러한 과정을 통과하여, 결국 최종 stage i에서 패킷 카운트 Fc 가 지정된 elephant flow 크기 E 이상이 되면, 해당 플로우는 elephant flow로 판정되고, Landmark가 아닌 캐시의 top으로 해당 플로우 엔트리를 옮긴다. 그림 3은 이러한 Multi-stage Landmark-LRU의 구조를 보여주고 있다.

이러한 다단계 stage를 가짐으로써, 실제 elephant flow 정보를 오랫동안 보존할 수 있기 때문에, 기존 L-LRU에 비하여 elephant flow의 발견율을 대폭 상승시킬 수 있다. 하지만 기존 L-LRU가 가지고 있는 문제점인 Landmark의 크기를 어떻게 설정할 것인지에 대한 문제점이 다중으로 유발되어 복잡한 구조를 가지게 된다. 그리고 정확한 발견을 위하여 메모리의 비용이 더 많이 요구되는 상황을 유발할 수 있다. 또한, 실제 실험 결과 LRU 구조에서는 stage 수보다 threshold에 따른 영향이 더 크게 나타난다. 이는 많은 stage 수는 영향이 크지 않다는 것을 보여준다.

www.dbpia.co.kr

Ⅲ. 제안하는 알고리즘

3.1. 제안하는 알고리즘

우리가 제시하는 개선 기법의 동작은 기본적으로 Landmark-LRU (L-LRU) 기법과 동일하다. L-LRU 기법과 동일하게, 캐시 내 하나의 Landmark 영역을 가진다. 하나의 플로우 엔트리는 하나의 플로우와 관련된 정보를 가진다. 이 정보는 flow id와 해당 플로우의 패킷 카운트 (packet count) Fc 이다.

알고리즘의 동작은 다음과 같다. 1) 새로운 플로우 F (miss flow)가 Landmark top에 들어온다. 그리고 플로우 엔트리에 해당 flow id와 패킷 카운트를 기록한다. 2) 이 후, 또 새로운 플로우가 들어오게 되면, 기존에 존재하는 플로우 F의 엔트리는 한 블록 아래로 옮겨지게 된다. 그리고 들어온 새로운 플로우는 Landmark top에 위치하며, 해당 플로우의 flow id와 패킷 카운트를 기록한다. 3) 만약 플로우 F에 대하여 flow hit가 발생하게 되면, 플로우 F의 엔트리는 Landmark top으로 옮겨진다. 이 동작은 LRU의 구조와 동일하다. 4) Flow hit이 발생할 때, 해당 flow의 패킷 카운트 Fc가 elephant flow의 지정된 크기인 E 이상이면, 해당 플로우 엔트리는 캐시 top으로 옮겨진다. 따라서 elephant flow들의 정보는 Landmark 구역이 아닌 캐시에 저장되게 된다.

5) 만약 flow miss가 지속적으로 발생하게 되면, 결국 특정 플로우 엔트리는 Landmark bottom에 위치하게 된다. 이후, 한 번 더 flow miss가 발생하면, 해당 플로우 엔트리는 캐시에서 삭제되게 된다. 여기까지의 과정은 L-LRU 기법과 동일하다.

하지만, 우리의 개선안은 기존 기법들 보다 더 정확한 elephant flow 발견을 위해, 두 가지 동작을 추가적으로 가진다. 첫 번째는 Landmark bottom에 위치한 플로우 크기를 분석하는 것이다. 즉, 캐시에서 삭제되기 직전의 플로우 정보를 분석하는 방법을 추가시켰다. 두 번째는 플로우 엔트리에 cycle flag를 추가적으로 두어 분석하는 횟수에 제한을 둔다. 두 가지 추가 동작을 정리하면 다음과 같다. 6) 플로우가 Landmark bottom에 위치하면, 해당 플로우의 패킷 카운트를 검사한다. 7) 만약 검사한 플로우가 elephant flow가 될 가능성이 있다고 판단되면, 해당 플로우를 Landmark top으로 옮긴다. Elephant flow가 될 수 있는 가능성에 대한 지표로는 파라미터 TH_E 를 사용하여 표현한다. 플로우의 패킷 카운트인 Fc가 가능성 지표인 TH_E 이상이면, 해당하는 플로우를 Landmark top으로 옮기고, cycle flag인 $curN$에 1을 더하여 기록한다. 만약 Fc가 TH_E 미만이면, 해당 플로우는 캐시에서 삭제한다. 8) 이후, 기존 L-LRU와 동일한 방법으로 샘플링을 하게 된다. 다시 Landmark top으로 옮겨진 플로우가, 만약 elephant flow가 아니라면, 다시 Landmark bottom에 위치하게 될 것이다. 이 플로우 (버려지기 직전의 플로우)의 패킷 카운트가 E보다 작고, TH_E 이상이면, $curN$을 체크한다. 9) 만약 $curN$이 지정한 파라미터인 N과 같으면 해당 플로우는 elephant flow가 아니라고 결정하고, 해당 플로우의 엔트리를 캐시에서 삭제한다.

표 1은 우리가 제시하는 알고리즘을 나타낸 것이다. 즉, 우리의 개선안은 Landmark bottom에 위치한 플로우 엔트리의 패킷 카운트가 TH_E 이상이면, 캐시에서 플로우 정보를 유지하고 샘플링 할 기회를 준다. N은 캐시에서 플로우가 유지될 수 있는 횟수를 나타내게 된다. 이 개선안을 OsF L-LRU(Give Opportunities to Survive Flows in Landmark LRU)라고 명명한다.

Table 1. Proposed Algorithm.

```
Input: F_i, E, β, N
Output: EF_i
Total Cache Reset
Initialize Each Entry's F_c = 0, curN = 0 in Landmark
Initialize TH_E = β * E
//Each Cache Entry has an info (F_i, F_c)
//Each Landmark Entry has an info (F_i, F_c, curN)
while not terminated Flow Sampling do
  LookupCache(F_i)
  if F_i is not in Cache and Landmark is not full then
    Insert Landmark Top (F_i, F_c)
  else if F_i is not in Cache and Landmark is full then
    if F_c at Landmark Bottom ≥ TH_E and
       curN < N then
      curN = curN + 1
      Move to Landmark Top (F_i, F_c, curN)
      Insert Landmark Top (F_i, F_c)
    else
      RemoveEntry at Landmark Bottom
      Insert Landmark Top (F_i, F_c)
    end if
  else
    UpdateEntry (F_i, F_c)
    if F_c ≥ E then
      Define as Elephant Flow EF_i ← F_i
      Move to Cache Top (EF_i, F_c)
    end if
  end if
end while
```


www.dbpia.co.kr

3.2. Potential to become Elephant: TH_E

파라미터 TH_E는 elephant flow가 될 수 있는 잠재력을 나타낸다. 이 논문에서는 TH_E는 elephant flow의 크기 E의 백분율로 세팅한다. 그러므로 TH_E는 수식 1처럼 나타낼 수 있다.

$$TH_E = \beta \cdot E,\ \beta \in (0,\ 1) \quad (1)$$

β는 백분율로서 표현되므로 0에서 1사이의 값을 가진다. 이는 네트워크 관리자에 의해 조정되는 값이다.

현재 Landmark bottom에 위치한 플로우 F의 패킷 카운트 Fc가 다음과 같은 상황이라 가정하자. 만약 Fc가 TH_E 이상이면, 플로우 F의 엔트리는 Landmark top으로 옮겨진다. 그리고 cycle flag $curN$은 $curN = curN + 1$ 으로 세팅되어진다. 반면에, Fc가 TH_E보다 작은 플로우 엔트리는 캐시에서 삭제한다. 그러므로 특정 플로우 F가 elephant flow가 될 수 있는 가능성을 가지는 조건은 아래 조건 (2), (3)으로 표현할 수 있다.

$$\begin{aligned} &if\ E > Fc \geq TH_E, \\ &then\ Move\ to\ Landmark\ Top \end{aligned} \quad (2)$$

$$\begin{aligned} &if\ Fc < TH_E, \\ &then\ Drop\ by\ Landmark\ Bottom \end{aligned} \quad (3)$$

Elephant flow가 될 가능성이 있다고 판단된 플로우는 곧바로 지워지지 않고, Landmark top으로 옮겨져서, 다시 플로우를 샘플링하게 된다. 샘플링 도중, 패킷 카운트가 E 이상이 되면, 해당 플로우는 elephant flow로 결정되고, 캐시 top으로 옮겨진다. 이러한 규칙에 따라, 실제 elephant flow이지만, mice flow들에 의해 버려지게 되는 현상을 방지할 수 있게 된다.

3.3. Inner Landmark Cycle: N

하지만 다음과 같은 경우가 발생할 수 있다. 실제 플로우 크기가 E보다는 작고, 파라미터 TH_E보다 큰 mice flow가 존재할 수 있다. 이러한 경우가 발생하면, 캐시 내에 실제 mice flow를 오랫동안 보관하는 상황이 발생한다. 이 상황은 저장 공간 사용에 비효율적인 상황을 유발할 수 있다. 또한, 너무 오랫동안 Landmark에 보관하게 되면, 실제 elephant flow가 아닌데도 불구하고, 차지하고 있는 플로우 엔트리는 계속 유지된다. 이 상황에서 캐시 리셋 간격이 길어지는 경우, 최종적으로 elephant flow라고 가정하는 false positive가 발생할 수 있다.

따라서 우리가 제시하는 개선안은 이러한 경우를 방지하기 위하여, 파라미터 N을 제시한다. 파라미터 N은 애매한 크기의 플로우가 Landmark 안을 순환하는 횟수이며, 이는 플로우가 살아남을 기회를 표현한다.

$curN$과 N은 양의 정수이며, N은 지정되는 상수이다. 최초 샘플링 시작 시, 각 엔트리의 $curN$은 0으로 초기화된다. 위에 언급한대로, 플로우 F가 elephant flow가 될 가능성 TH_E를 가지면, 해당 플로우 F는 cycle flag $curN$에 1을 더하여 표시하게 된다. 즉, $curN = curN + 1$이 되며, 플로우 엔트리는 Landmark top으로 이동한다. 이후, 만약 플로우 F가 다시 Landmark bottom에 위치하게 되어 플로우 카운트 Fc가 여전히 E보다 작고, $curN$이

지정된 cycle 횟수인 N과 동일하면, 플로우 F는 elephant flow가 아니라고 정의하고, Landmark에서 방출된다. 최종적으로, 우리가 제안하는 알고리즘에

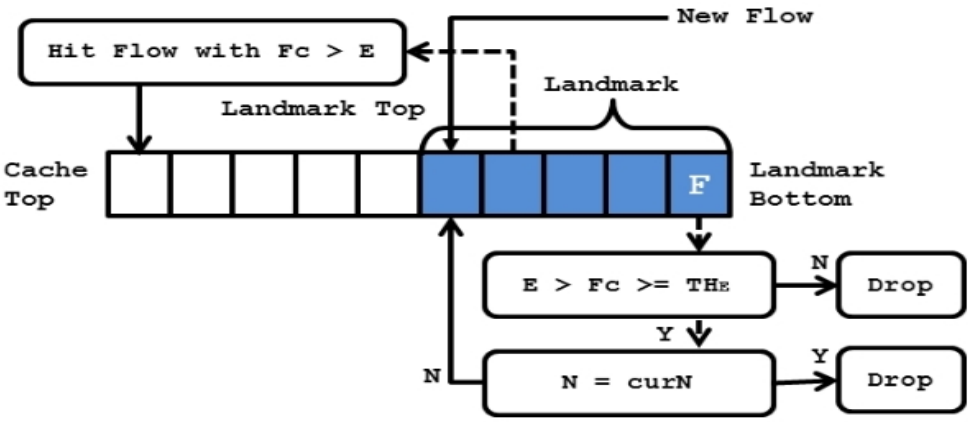


Fig. 4. The OsF L-LRU Scheme.

www.dbpia.co.kr

서 샘플링 될 기회를 주는 조건과 캐시에서 방출되는 조건은 아래 조건 (4), (5)와 같이 표현될 수 있다.

$$\begin{aligned} &\textit{if } E < Fc \leq TH_E \textit{ and } N > curN, \\ &\quad \textit{then Move to Landmark Top} \qquad (4) \\ &\quad \textit{if } Fc < TH_E \textit{ and } N = curN \\ &\textit{then Drop by Landmark Bottom} \qquad (5) \end{aligned}$$

일반적으로, 실제 하나의 플로우에 속하는 패킷은 연속적으로 인입되는 경향이 있으며, N번 후에 들어온 플로우는 flow id의 속성들이 동일해도, 이전과 동일한 플로우라고 정의할 수 없다.

그림 4는 우리가 제안한 개선안이 어떻게 동작하는지를 표현한 그림이다.

Ⅳ. 성능 평가

Table 2. Real Network Traffic Trace Information.

Measurement Interval	Total Bytes (KB)	Number of Flows	Feature
2011.10.19. 13:58~14:08	5,589,698	114,452	break time
2011.10.19. 14:30~14:41	6,154,234	148,386	lectures
2011.10.19. 15:10~15:18	6,070,684	148,998	end of lectures

본 장에서는 제시한 알고리즘에 대한 성능 평가 방법 및 결과에 대하여 서술한다. 성능 평가를 위해, 우리는 실제 네트워크 트레이스 (network traces)를 사용하였다. 실험에 사용한 네트워크 트레이스는 상명대학교 서울캠퍼스 망의 패킷들을 수집한 것이며, 시간을 다르게 하여 총 3가지 시간대의 네트워크 트레이스를 수집하였다. 각 기법들은 소프트웨어로 구현하였다. 실제 네트워크 트레이스를 입력 파일로 사용하여, 각 알고리즘을 통과시키고 새로이 기록된 결과를 통하여 성능을 비교 및 평가하였다. 네트워크 플로우는 5-tuple flow keys: {protocol, source IP address, source port number, destination IP address, destination port number}로서 정의하였으며, 위의 5가지 속성이 모두 일치하면 하나의 플로우로 정의하였다[7]. 각 플로우 엔트리는 오직 하나의 플로우 정보만을 저장한다. 표 2는 수집된 네트워크 트래픽 트레이스 정보이다.

4.1. 각 기법들의 성능 비교 실험

각 기법들의 성능 비교를 위해, 공통적으로 전체 캐시 크기 CS = 1024, Landmark 크기 L = 512로 설정하였다. 그리고 캐시 리셋 주기는 1분으로 설정하였다. 휴식 시간 트레이스를 예로 들면, 각 측정시간 1분 동안 샘플링된 평균 패킷 바이트 크기는 558,969KB였으며, 패킷 하나의 크기를 1000 bytes라고 가정하였다. 대역의 0.1%를 차지하는 플로우를 elephant flow로 지정하면, 558개 이상의 패킷을 가진 플로우를 elephant flow라고 가정할 수 있다. 즉, 세 개의 트레이스에 관하여 E = {558, 559, 758}로 지정할 수 있다. 이와 같이 대역의 0.1% 이상을 차지하는 플로우를 elephant flow로 지정하고 실험을 진행하였다.

Table 3. Average Detection Ratio when Using Maximum Memory.

Algorithm	Entries / Memory Usage (Maximum)	Ac (%)
SH (p = 0.01)	1,993 / 33,881 bytes	78.52%
SH (p = 0.05)	1,090 / 18,530 bytes	68.97%
SH (p = 0.001)	366 / 6,222 bytes	48.52%
OsF L-LRU	967 / 17,406 bytes	87.27%
	1,024 / 18,432 bytes	89.87%
LRU	512 / 8,704 bytes	23.37%
	1,024 / 17,408 bytes	33.34%
	2,048 / 34,816 bytes	51.80%
L-LRU	1,024 / 17,408 bytes	76.30%
	1,084 / 18,428 bytes	77.66%
2Stage ML-LRU	1,024 / 17,408 bytes	78.23%
	1,084 / 18,428 bytes	78.59%
3Stage ML-LRU	1,024 / 17,408 bytes	83.66%
	1,084 / 18,428 bytes	84.02%
4Stage ML-LRU	1,024 / 17,408 bytes	84.84%
	1,084 / 18,428 bytes	85.15%

$$Ac(\%) = \frac{Num\,of\,Detected\,E}{Num\,of\,Real\,E} \times 100 \quad (6)$$

Elephant flow 발견율은 (6)을 이용하여 계산하였다. 우리가 제시하는 알고리즘의 파라미터들은 $TH_E = E \cdot 0.5$, N = 3으로 세팅하였다. ML-LRU의 경우, 1stage L = 256, 그 외 Landmark stage들의 크기는 L = 128로 설정하였으며, threshold $P^1_{bth} = E \cdot 0.125$, $P^2_{bth} = E \cdot 0.25$, $P^3_{bth} = E \cdot 0.5$로 설정하였다. Sample & Hold 기법(이하 SH)의 경우, 측정 시 생성되는 엔트리의 개수에 제한을 두

www.dbpia.co.kr

지 않고 실험을 진행하였으며, 다른 기법들은 설정된 엔트리 수 만큼 캐시 크기를 할당하고 실험을 진행하였다.

표 3은 각 기법들의 실제 최대 메모리 사용 시 해당하는 발견율의 평균을 나타낸 것이다. 실험에서 요구하는 엔트리의 크기는 다음과 같다. Protocol = 1byte, Source IP address = 4bytes, Destination IP address = 4bytes, Source Port = 2bytes, Destination Port = 2bytes, Packet Counts = 4bytes로 SH, LRU, L-LRU 그리고 ML-LRU 기법 모두 1개의 엔트리 당 최소 17bytes를 유지하여야 한다. 우리가 제시하는 알고리즘에 사용되는 엔트리의 크기는 cycle flag 1byte를 추가적으로 요구하므로, 1개의 엔트리 당 18bytes의 용량을 요구한다. 따라서, 동일한 캐시 바이트 환경에서의 비교 실험은 Land-mark에 추가 엔트리를 할당하여 실험하였다.

Table 4. Average Detection Ratio with Varying Size of *E*.

Algorithm	Avg Ac (%)		
	E > 1%	E > 0.1%	E > 0.01%
SH (p = 0.01)	88.85%	76.01%	55.86%
SH (p = 0.05)	87.00%	67.78%	42.38%
SH (p = 0.001)	84.61%	43.02%	18.00%
OsF L-LRU	95.33%	89.06%	97.67%
LRU	85.78%	46.80%	29.26%
L-LRU	90.71%	75.76%	73.43%
2Stage ML-LRU	91.33%	77.96%	79.31%
3Stage ML-LRU	93.85%	83.36%	81.59%
4Stage ML-LRU	93.85%	84.62%	84.61%

결론적으로, 우리가 제시한 OsF L-LRU와 기존 L-LRU와 비교하여, 동일한 엔트리를 가지고 있는 환경 하에 평균적으로 13.57%의 발견율 향상이 있었다. 또한, 비슷한 메모리 사용량을 보이는 결과에 대하여 L-LRU 기반 기법과 OsF L-LRU의 elephant flow 발견율이 LRU와 SH보다 더 높으며, 이것은 L-LRU 기반 기법들이 LRU나 SH에 비해 효율적으로 메모리 절약이 가능하다는 증거이다.

표 4는 각 기법들마다 *E*의 크기에 따른 발견율 결과를 보여준다. 실험을 위하여, 각 기법들의 *CS*는 SH와 LRU를 제외하고 *CS* = 1,024로 통일하였다. LRU는 *CS* = 2,048로 설정하였다. 전체적으로 *E*를 1% 이상으로 설정하면 최소 80% 이상의 발견율을 보인다. 이는 1% 이상의 크기를 가진 플로우에 속하는 패킷이 매우 많기 때문에 샘플링 되는 확률도 높아지기 때문이다. 하지만, *E*가 작아질수록 SH 기법의 발견율이 크게 감소하는 것을 확인할 수 있다. 이는 패킷 카운트가 높지 않은 플로우는 저장될 확률이 낮기 때문이다. 반면에 우리가 제시한 알고리즘은 *E*가 매우 작으면 발견율이 상승하는 것을 확인할 수 있다. 제시한 알고리즘의 특성상, Landmark 내 패킷 카운트가 작은 엔트리가 유지되는 시간이 길어지기 때문이다.

4.2. 설계 파라미터 변화 실험

그림 5와 그림 6은 OsF L-LRU에서 제시하는 파라미터 값의 변화에 따른 실험 결과를 보여준다. 파라미터 변동에 따른 성능을 평가하기 위하여, 수집한 트레이스 중 실제 elephant flow가 가장 많은 구간을 추출하여 실험을 진행하였다. *E*는 대역의 0.1% 로 설정하였으며, 실험에서의 *E* = 560이다.

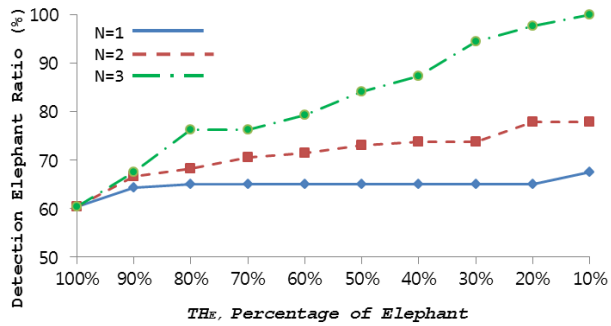


Fig. 5. Detection Ratio with Varying TH_E of OsF L-LRU.

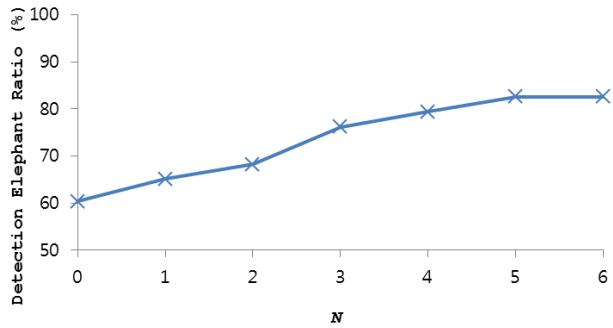


Table 6. Detection Ratio with Varying *N* of OsF L-LRU.

TH_E 값이 높을수록, 현재 Landmark bottom에 위치한 플로우는 elephant가 될 기회가 적어지는 것을 의미한다. 따라서 TH_E 값이 작아질수록, 플로우는 Landmark 안에서 살아남을 기회를 많이 가지게 되고, 오랫동안 Landmark 안에 남게 된다.

그림 5는 파라미터 TH_E 값의 변화에 따른 결과

www.dbpia.co.kr

를 보여주고 있으며, 성능 평가를 위해 CS = 1024, L = 256, N = {1, 2, 3}으로 변화를 주며 실험하였다. 만약 TH_E가 E의 100%와 동일한 경우는 L-LRU 기법과 동일하다. 전체적으로 TH_E값이 낮아질수록 elephant 발견율이 증가하는 것을 볼 수 있다.

N = 3인 경우, TH_E가 560 (E의 100%)에서 504 (E의 90%)로 변동할 때, 발견율은 60.31%에서 67.46%로 증가하였으며, 7.15%의 발견율 향상을 보였다. 이 결과는 우리가 elephant flow가 될 잠재력을 매우 낮게 평가하더라도, 상당한 발견율 향상이 있음을 보여준다.

TH_E 값이 작아질수록 발견율이 계속하여 향상되는 현상을 발견할 수 있다. 이는 알고리즘의 특성상 TH_E 값이 작아지고 N값이 증가할수록, Landmark내 패킷 카운트가 작은 플로우 엔트리가 유지되는 시간이 길어지기 때문이다. 따라서 샘플링 되는 시간이 길어지게 되고 발견율이 상승하게 된다. 하지만 그림 5에서 N = 3이고, TH_E가 448(E의 80%)에서 392(E의 70%)로 변동할 때, 발견율은 76.19%로 변화가 없음을 확인할 수 있다. 이는 실험 트레이스 내 패킷 카운트가 392 ~ 482인 플로우 수가 거의 동일하기 때문이다. 즉, 트래픽에 따라 발견율 향상이 발생하지 않는 현상이 있음을 확인할 수 있다. 또한 N = 1 로 설정하였을 때, TH_E 값 변동에 따른 발견율 향상이 크지가 않음을 보여준다.

그림 6은 파라미터 N 값에 따른 변화를 보여준다. N 값이 증가할수록, 플로우는 살아남을 기회를 많이 가지게 된다. 평가를 위해, CS = 1024, L = 256, E = 560, TH_E = 448 (E의 80%)으로 설정하였으며, N 값을 변경하며 실험하였다. N = 0이면 L-LRU와 동일하다. 그림 5에서 N 값이 증가할수록, elephant flow 발견율이 증가하는 것을 확인할 수 있다. 하지만 N 값이 5 이상인 경우, 더 이상의 성능 향상은 발생하지 않았다.

V. 결 론

우리가 제시한 OsF L-LRU 기법은 기존 LRU와 Landmark-LRU 기법에서 나타나는 문제점인, mice flows가 많은 네트워크 환경에서, 기존 기법보다 더 정확하게 elephant flow를 발견할 수 있는 해결책을 제시한다.

OsF L-LRU 기법은 기존 Landmark LRU 기법의 구조를 그대로 유지한다. 차이점은 1) Landmark bottom에서 버려지기 직전의 플로우의 정보를 다시 한 번 살펴보고, 2) 해당 플로우의 패킷 카운트가 지정한 TH_E 값 보다 높으면 3) 해당 플로우의 엔트리를 Landmark top으로 옮겨, 다시 샘플링 할 수 있는 기회를 준다. 따라서 실제 elephant flow이지만 측정된 패킷 카운트가 애매한 플로우를 수집하지 못하고 버려지는 일을 상당히 방지할 수 있다. 그리고 실제 elephant flow가 아닌 애매한 크기의 플로우가 캐시를 오랫동안 점유하는 것을 방지하기 위하여, cycle flag N 을 설정하여 측정시의 오류를 방지하였다. 따라서 우리가 제시한 기법은 기존 Landmark LRU 기법의 단순한 구조를 유지하며, 더 정확하게 실제 elephant flow를 발견할 수 있다.

시뮬레이션 결과는, 기존 기법들과 OsF L-LRU 기법의 elephant flow 발견율을 비교함으로써, elephant flow 발견율이 향상되는 것을 보여주고 있다. 더욱이, Multistage Landmark-LRU보다 간단한 구조를 가지며, Landmark-LRU와 동일한 메모리 절약 효율이 있음을 입증하고 있다. 우리가 제안한 알고리즘은 Multistage Landmark-LRU에 적용시킬 수 있으며, 이는 기존 Multistage Landmark-LRU 기법의 elephant flow 발견율을 더욱 향상시킬 수 있을 것으로 예상된다.

하지만, 네트워크 트래픽이 매우 많은 mice flow들을 가지고 있고, elephant flow를 구성하는 각 패킷의 간격이 매우 긴 경우에서는 우리가 제시한 개선안의 발견율이 향상되지 않는 경우도 발생하였다. 이는 기존 Landmark-LRU의 구조를 그대로 유지하는 것에 따른 한계점으로 판단된다.

향 후 연구로, 보다 다양한 네트워크 트래픽 상황들을 극복할 수 있는 구조적인 개선이 필요하며, 실제 장비에 적용 시 예상되는 프로세싱 오버헤드를 감소시키는 방법이 필요하다.

References

[1] L.Che, B.Qui. "Landmark LRU: an Efficient Scheme for the Detection of Elephant Flows at Internet Routers", *Communication Letters, IEEE*, vol.10, no.7, pp.567-569, 2006.

[2] L.Che, B.Qui. "Traffic Measurement for Large Internet Flows Using Multi-stage Landmark

www.dbpia.co.kr

LRU Cache", *TENCON 2005 IEEE Region 10*, pp.1-6, 2005.

[3] C.Estan, G.Varghese. "New Directions in Traffic Measurement and Accounting", in *Proc. ACM SIGCOMM'02*, pp.323-336, 2002.

[4] A.Smitha, I.Kim, and A.L.N.Reddy. "Identifying Long-term High-bandwidth Flows at a Router", in *Proc. High Performance Computing*, pp.361-371, 2001.

[5] L.Guo, I.Matta. "The War between Mice and Elephants", in *Proc. Network Protocols, Ninth International Conference on IEEE*, pp. 180-188, 2001.

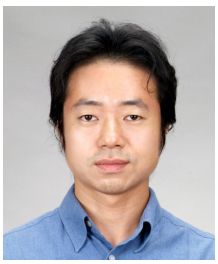
[6] WikiPedia, Elephant Flow(2011), Retrieved Dec., 15, 2011, from http://en.wikipedia.org/wiki/Elephant_flow.

[7] J.Quitteek, T.Zseby, B.Claise, and S.Zander, RFC 3917: Requirements for IP Flow Information Export(2004), Retrieved Aug., 11, 2011, from http://www.ietf.org/rfc/rfc3917.txt.

[8] R.Platenkamp. "Early Identification of Elephant Flows in Internet Traffc", in Proc. *6th Twente Student Conference on IT,* 2007.

[9] T.Mori, M.Uchida, C.M. "Identifying Elephant Flows Through Periodcally Sampled Packets", in Proc. *IMC'04, 4th ACM SIGCOMM conference on Internet measurement*, pp.115-120, 2004.

정 진 우 (Jinoo Joung)



1992년 KAIST 전기전자공학과 공학사

1997년 Polytechnic Univ., NY, USA, 공학박사(Ph.D.)

1997년~2001년 삼성전자 중앙연구소

2001년~2005년 삼성종합기술원

2005년~현재 상명대학교 컴퓨터과학부 부교수

<관심분야> 네트워크, 유무선 통신, 임베디드 시스템

최 윤 기 (Yunki Choi)

2011년 상명대학교 컴퓨터과학전공 학사

2011년~현재 상명대학교 일반대학원 컴퓨터과학과 석사과정

<관심분야> 네트워크, 임베디드 시스템, 트래픽 측정 및 분류

손 성 훈 (Sunghoon Son)

1991년 서울대학교 계산통계학과 학사

1993년 서울대학교 전산과학과 석사

1999년 서울대학교 전산과학과 박사

1999년~2004년 한국전자통신연구원 선임연구원

2004년~현재 상명대학교 컴퓨터과학부 조교수

<관심분야> 임베디드 시스템, 가상화

www.dbpia.co.kr