어플리케이션 레벨의 Signature 생성 시스템과 검증 방법

박병철 [1], 원영준 [1], 김명섭 [2], 홍원기 [1]

[1] 포항공과대학교 컴퓨터공학과

[2] 고려대학교 컴퓨터공학과

[1]{fates, yjwon, jwkhong}@postech.ac.kr, [2]tmskim@korea.ac.kr

요 약

현재까지 가장 전통적인 트래픽 분류는 미리 정의된 well-known 포트를 통해 이루어져왔지만 이러한 분석 방법은 높은 정확도를 제공하지 못하고 있다. 새로운 대체 방법으로 어플리케이션 레벨의 signature 를 통한 트래픽 분류 방법이 제안 되었지만 정확도가 높은 signature 를 생성하기 위해서는 트래픽에 대한 수작업의 철저한 탐색이 요구된다. 어플리케이션 프로토콜에 대한 정보가 사전에 제공된다면 높은 정확도의 signature 를 생성할 수 있지만 많은 어플리케이션들이 독자적인 프로토콜을 사용함에 따라 수작업을 통한 signature 생성을 피할 수 없게 되었다. 어플리케이션 signature 를 활용한 트래픽 분류에 관한 많은 연구가 진행되고 있음에도 트래픽 분류 결과의 정확도를 판단할 수 있는 효과적인 검증 방법의 부제 또한 해결해야 할 과제이다. 본 논문에서는 어플리케이션 프로토콜에 대한 정보가 없이도 어플리케이션 레벨의 signature 를 생성할 수 있는 LASER 알고리즘과 signature 를 통한 트래픽 분류 결과를 검증할 수 있는 방법을 제시하고 이를 통해 LASER 알고리즘이 높은 정확도의 signature 를 생성함을 보인다.

1. 서론

오늘날 인터넷 트래픽의 급격한 변화로 인하여 기존의 포트 기반의 트래픽 분류 방법은 더 이상 높은 정확도를 제공하지 못하게 되었다. 최근 들어 웹(인터넷) storage 와 Peer-to-Peer (P2P) 파일 공유 어플리케이션들의 종류가 급격하게 증가하였고, 이들이 발생시키는 트래픽의 량이 전체 인터넷 트래픽에서 매우 많은 부분을 차지하고 있다[1]. 특히 최신의 P2P 어플리케이션들은 동적인 port 할당이나 relay 노드를 사용하는 것과 같은 방법들을 통해 detection 과 filtering 을 피하고 있다. 포트 기반의 트래픽 분류 방법은 backbone 네트워크의 방대한 트래픽 양이나 computing 리소스의 제약으로 인해 현재에도 많이 사용되고 있지만 정확한 어플리케이션 분석이 이루어 질 수가 없다.

어플리케이션 signature 기반의 트래픽 분류 방법은 기존의 분류 방법들의 문제점들을 해결하기 위해 제안되었다[2][3]. 어플리케이션 signature 는 어플리케이션의 종류를 구분 지을 수 있는, packet payload 데이터상의 고정적으로 나타나는 string 이나 hex 값의 패턴을 의미한다. 어플리케이션 signature 는 현재까지 수작업을 통해 생성되어 왔다. 이러한 signature 생성은 어플리케이션 프로톨에 대한 조사나 패킷 payload 에 대한 사전 분석이 요구 되기 때문에 새로운 어플리케이션이 등장하였을 때 signature 를 만들기 위해서는 상대적으로 많은 시간이 소요된다. 이러한 문제점들을 해결하기 위해서는 효율적인 방법으로 signature 를 생성하는 체계적인 시스템이 필요하다. 전반적인 트래픽 분류 연구들에서는 트래픽 분류 결과에 대한 검증 방법을 제시하고 있지 못하고 있기 때문에 정확한 검증 방법의 제시 또한 필요하다.

본 논문에서는 선행적인 프로토콜 조사 없이 자동적으로 어플리케이션 signature 를 생성하는 새로운 방법으로 LCS (Longest Common Subsequence) 알고리즘 기반의 LASER (LCS-based Application Signature ExtRaction)알고리즘을 제안하고 LASER 알고리즘의 효용성을 검증하기 위하여 새로운 검증 방법 또한 제시하며 이를 통해 LASER 를 통해 생성된 signature 들의 정확도를 검증한다.

본 논문의 구성은 다음과 같다. 2 장에서는 관련 연구들과 우리의 선행 연구를 소개한다. 3 장에서는 자동으로 signature 생성할 수 있는 LASER 알고리즘을 소개한다. 4 장에서는 트래픽 분류 시스템의 정확도를 검증할 수 있는 방법을 제시하고 이를 통해 LASER 알고리즘으로 생성한 signature 의 정확도를 검증하며 마지막으로 5 장에서 결론과 향후 연구에 대하여 기술한다.

2. 관련 연구

어플리케이션 signature 를 이용한 트래픽 분류에 관한 기존의 연구들은 자체적으로 생성한 signature

본 연구는 두뇌한국 21 과 지식경제부 및 정보통신연구진흥원의 대학 IT 연구센터 지원사업의 연구결과로 수행되었음 (IITA-2008-C1090-0801-0045)

를 통해 어플리케이션 트래픽의 분석률을 제시하지만 signature 를 생성하는 방법이나 그 signature 자체의 정확도에 대한 검증을 중요하게 언급하고 있지 않다.

Gummadi 의 연구 [2]는 KaZaA 의 다운로드 트래픽의 분류를 위해 어플리케이션 레벨의 signature 를 사용하였지만, 다양한 signature 들 간의 충돌 등의 문제와 생성된 signature 에 대해 정확도 측정과 같은 검증 과정을 제시하지 않았다. Sen 의 연구 [4]에서는 어플리케이션의 프로토콜을 분석하여 P2P 어플리케이션의 signature 를 생성하였다. 어플리케이션 프로토콜에 대한 분석은 signature 의 정확도를 높일 수 있지만 수동적인 방법으로 signature 를 생성해야 하므로 상대적으로 효율성이 낮으며 실시간 시스템에 적용하기에는 부적절하다. 종합적으로 보았을 때에 이와 비슷한 연구들은 수작업을 통한 signature 생성과 signature 기반의 어플리케이션 트래픽의 분석률을 제시하지만 signature 를 생성하는 방법이나 그 signature 자체의 정확도에 대한 검증을 중요하게 언급하지 않고 있다.

Signature 기반의 트래픽 분류는 웜 detection 을 위한 Intrusion Detection System (IDS) 들에서 주로 사용되어왔다[5]. 이 연구들은 포트 스캐닝이나 웜의 전염 등과 같은 비정상 트래픽들의 signature 를 자동으로 생성할 수 있는 시스템들을 기술하고 있다. 이와 같은 목적이 일반적인 어플리케이션 signature 생성과 가장 큰 차이점은 각각의 어플리케이션들이 자신만의 유일한 signature 를 갖는 것과 달리 웜 signature 의 경우는 여러 개의 웜들이 하나의 signature 를 공유할 수 있다는 점이다. 이는 어떠한 의미에서 웜 signature 가 좀더 약한 제약사항을 갖는다고 볼 수 있다.

일반적으로 웜 signature 생성에 관련된 연구에서는 sliding-windows 알고리즘과 자신들이 정의 한 break point 를 사용하고 있다. Scheirer 의 연구 [8]에서는 이러한 break point 의 선정과 sliding-window 알고리즘의 종류에 관해 기술하고 있다. 이들 break point 는 웜들이 공통적으로 갖고 있는 특정 동작에 해당하는 instruction code 와 같은 hex 값들이다. 세 가지의 sliding-window 알고리즘은 Fixed Partition Sliding Window (FPSW), Variable-length Partition Sliding Window(VPSW), Variable-length Partition with Multiple Breakmarks (VPMB)로 window 가 패킷의 payload 를 따라 움직이며 특정한 비트 패턴을 찾는 방법으로 웜의 signature 를 생성한다. 하지만 일반적인 트래픽에서는 break point 라는 특정한 비트 패턴이 존지하지 않는다. 따라서 웜과 같은 비정상 트래픽과 특성이 다른 P2P 와 같은 일반적인 인터넷 어플리케이션 트래픽 분류에 break point 를 사용하는 sliding-windows 알고리즘을 직접 적용하는 것에는 어려움이 따른다. 이러한 점들이 기존의 웜 signature 생성에 관한 연구와 본 연구 사이의 차이점을 시사한다.

암호화된 트래픽의 signature 생성에 관한 연구도 일부 진행되었다. Ehlert 의 연구 [9]에서는 패킷의 암호화를 사용하는 Skype (v1.5, v2.0)의 초기 통신 설정과정의 트래픽에서 어플리케이션 signature 를 생성하였으며, 암호화된 SSL 연결에서의 signature 생성에 관한 연구[10] 또한 진행되었다.

우리는 선행 연구[11]에서 IDS 와 일반적인 트래픽 분류 시스템의 차이점을 지적하고 웜과 일반 어플리케이션의 signature 포맷을 정리하였다. 이러한 분석을 통해 LCS 알고리즘 [12]을 활용하여 가장 포괄적인 signature 의 포맷인 sequence of common substring 형태의 signature 를 생성할 수 있는 시스템을 개발하고 알고리즘의 효율성에 대한 검증을 제시하였다. 본 논문은 알고리즘의 효율성이 아닌 LASER 알고리즘이 생성한 signature 의 정확성 검증에 초점을 두고 있다.

3. LASER 알고리즘을 통한 Signature 생성

3.1 Signature 생성 과정

자동화된 어플리케이션 signature 의 생성은 signature 생성에 필요한 타깃 어플리케이션의 트래픽 수집과정과 signature 추출의 두 가지 단계로 나뉘어진다. 타킷 어플리케이션 트래픽은 signature 생성을 원하는 특정 어플리케이션에서 발생한 트래픽만을 포함한 것을 말한다. 만약 다양한 어플리케이션에서 발생한 트래픽이 섞이게 된다면 signature 추출 과정에서 생성된 signature 에 의미 없는 값이 포함되거나 signature 자체가 생성되지 않는 문제점이 발생할 수 있다. 따라서 타깃 어플리케이션의 트래픽만을 수집하는 것은 signature 생성의 효율성과 정확성을 보장하기 위해 필수적이다. 이처럼 특정 어플리케이션의 트래픽을 독립적으로 수집하기 위해 signature 생성 시스템에는 트래픽을 자동으로 수집할 수 있는 에이전트가 탑재된다. 트래픽 수집 에이전트는 windows 기반으로 개발 되었으며 에이전트가 탑재된 컴퓨터에서 발생하는 모든 트래픽을 어플리케이션 (또는 프로세스)별로 수집하는 기능을 수행한다. 이 때에 수집된 트래픽은 동일한 5-tuple (source IP, destination IP, source port, destination port, protocol) 정보를 갖는 flow 형태로 이루어진다.

트래픽 수집 에이전트는 Winpcap library [13]와 windows system call 을 이용하여 트래픽 수집과 저장을 수행한다. 패킷이 수집 되었을 때 새로운 TCP 의 Syn flag 가 설정된 경우에는 새로운 세션이 맺어진 것으로 판단하고, 어떠한 어플리케이션이 맺는 세션인지를 확인하기 위해 windows system call 을 통해 세션이 맺어질 때 사용한 네트워크 port 를 사용하고 있는 프로세스 정보를 운영체제로부터 받아오게 된다. 따라서 트래픽 수집 에이전트는 자동적으로 어떠한 패킷이 어떠한 어플리케이션에서 발생한 것인지를 판단할 수 있고 트래픽을 어플리케이션 별로 수집한다.

Signature 추출 과정을 위해 LCS 알고리즘[12]을 수정 및 적용하였다. LCS 알고리즘은 서로 다른 두개 이상의 DNA 간의 동일한 단백질 구조를 판단하고 이를 통해 DNA 간의 유사성을 계산하는 생명과학 분야의 DNA sequence matching 에 주로 사용되어 왔다. DNA 에서 동일한 sequence 를 찾는 것과 패킷의 payload 에서 공통적으로 나타나는 패턴을 찾는 것은 유사하지만 구성 요속가 4 개로 이루어진 DNA 와 패킷 payload 는 많은 면에서 다르기 때문에 LCS 알고리즘을 어플리케이션 signature 생성에 적합하도록 수정하여 적용하였다. 패킷 payload 가 DNA 를 대치하고 LASER 알고리즘이 동일한 어플리케이션에서 발생한 트래픽에서 공통적으로 나타나는 동일한 패턴을 추출하게 된다.

3.2 Signature 생성을 위한 제약사항

Signature 생성은 많은 수의 string 들을 비교하는 과정이 반복되기 때문에 좀더 효율적인 과정을 위해 기존의 LCS 알고리즘에 수정이 요구된다.

Flow 를 이루는 패킷의 수: flow 는 동일한 source IP, destination IP, source port, destination port 그리고 동일한 프로토콜을 갖는 패킷들의 집합을 의미한다. Sen 의 연구 [4]에서는 어플리케이션 signature 가 존재할 수 있는 패킷은 flow 를 이루는 패킷들 중 초반에 존재하는 몇 개의 패킷에 제한됨을 설명하였다. 이는 signature 생성을 위해 flow 상의 모든 패킷들의 내용들을 모두 조사할 필요가 없음을 의미한다.

Substring 의 길이: 생성된 signature 는 한 개 이상의 substring 들로 이루어지며, 각 substring 의 길이는 정확한 트래픽 분류를 위한 signature 로서의 신뢰도 및 정확도를 반영한다. 길이가 너무 짧은 substring 은 signature 로써의 의미를 갖기 힘들고 Signature 기반의 트래픽 분류 과정의 계산 비용은 signature 의 길이에 비례하기 때문에 [14] substring 의 길이에 대한 제약사항이 요구된다. 만약 하나의 문자가 signature 의 substring 들 중 하나로 선정되었을 경우, 그 문자가 동일한 offset 에서 반복적으로 나타나지 않는다면 실제 signature 의 일부로써 의미를 갖기 어렵다. 예를 들어 HTTP 프로토콜을 사용하는 많은 어플리케이션들의 payload 에는 '/'와 같은 문자가 많이 포함되는데 이러한 문자는 어플리케이션 signature 로써의 의미가 없다. Substring 의 최소 길이 제약 조건을 통해 이와 같은 하나의 문자가 signature 의 일부에 포함되는 것을 방지할 수 있다.

패킷 크기의 비교: 서로 비교되는 패킷들의 목적(예: signaling, downloading)과 트래픽의 특성이 비슷할 경우에 신뢰도가 높은 signature 를 찾을 수 있는 확률이 증가한다. 패킷의 크기는 여러 패킷들 간의 유사성을 판단할 수 있는 가장 기본적인 특성이다. 그림 1 은 서로 다른 네 가지 어플리케이션에서 파일을 다운로드할 때 발생하는 패킷들 중 초기 100 개 패킷의 크기 분포를 나타낸다. 초기의 몇 개 패킷은 파일 다운로드 요청을 위한 connection handshake 에 사용되는 패킷들로 실제 파일 전송을 위한 패킷들보다 상대적으로 작은 크기를 갖는다. 이러한 signaling 패킷들은 각 어플리케이션 프로토콜의 특징을 반영한 string 을 포함한다. 따라서 명확한 signature 는 초기의 signaling 패킷에 존재할 확률이 높다. 반면 downloading 패킷의 경우는 어플리케이션 프로토콜이 아닌 다운로드되는 컨텐츠의 내용에 해당하는 string 들을 포함하고 있기 때문에 signature 를 생성하는 과정에서 크기가 작은 signaling 패킷들과 크기가 큰 downloading 패킷들을 서로 비교하는 과정은 불필요하다. 결과적으로 불필요한 비교 과정을 제거하기 위해 비슷한 크기 범위 내에 있는 패킷들 사이에서만 비교가 이루어지도록 하는 제약조건이 필요하다. 실제적인 예로 Gnutella 계열의 P2P 어플리케이션인 LimeWire 의 경우, 서버 접속이나 컨텐츠 검색과 같은 signaling 패킷의 크기는 평균적으로 390byte 인 것에 반해 downloading packet 의 크기는 1460byte 이다[15]. 이러한 두 가지 종류의 트래픽들은 어플리케이션 내에서 수행하는 동작과 목적이 다르기 때문에 동일한 signature 를 포함할 확률이 적다.

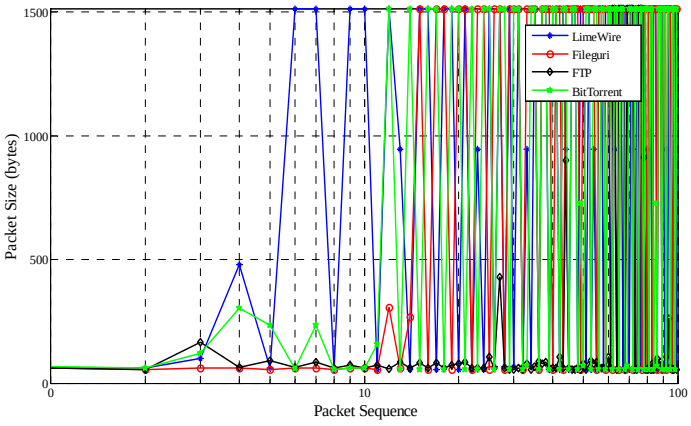

그림 1. 파일 다운로드시에 생성되는 초기 100 개 패킷들의 크기 분포
(1) LimeWire, (2) Fileguri, (3) FTP, (4) BitTorrent

3.3 LASER 알고리즘

알고리즘 1. LASER 를 이용한 signature 생성

```
1: procedure Signature_Generation ()
2:   Flow_Pool {F1[]…Fx[]} ← Santized_packet_collector
3:   F1[] ← Iterate, packet dump for Flow 1
4:   F2[] ← Iterate, packet dump for Flow 2
5:   while i from 0 to #_packet_constraint do
6:     while j from 0 to #_packet_constraint do
7:       if |F1[i].packet_size - F2[j].packet_size| < threshold
8:         result_LCS ← LASER (F1[i], F2[j])
9:         LCS_Pool {} ← Append result_LCS, end if
10:        j++, end while
11:  i++, end while
```

```
12:   S ←select the longest from LCS_Pool
13:   while i from 0 to # of rest flows of Flow_Pool do
14:      F_i ← select one from the rest of Flow_Pool
15:      result_LCS ← LASER (S, F_i)
16:      S ← select the longest from result_LCS
17:      i++, end while, end while
18:   return S

19: procedure LASER (Packet_A[1...m], Packet_B[1...n])
20:   Packet_A [m...1] ← Reverse byte stream
21:   Packet_B [n…1] ← Reverse byte stream
22:   Matrix [m][n]
23:   while i from 0 to m do
24:      while j from 0 n do
25:         if i = 0 or j = 0, then Matrix [i][j] ← 0
26:         else if Packet_A [i] = Packet_B [j], then
27:               Matrix [i][j] ← ‘ Diagonal’
28:         else if Matrix[i][j] != p[i][j-1], then
29:               Matrix[i][j] ← ‘ Up’
30:         else Matrix[i][j] ← ‘ Left’, end while
31:   end while
32:   i ← m-1; j ← n-1   //Tracking
33:   while Matrix[i][j] != 0 do
34:      if Matrix[i][j] = ‘ Left’, then j--
35:         else if Matrix[i][j] = ‘ Up’, then i--
36:         else if Matrix[i][j] = ‘ Diagonal’, then do
37:           Substring ← Append Packet_A[i]
38:         if Matrix[i-1][j-1] != ‘ Diagonal’, then
39:            Append special break point character (e.g. ‘/’)
40:   i--; j--, end while
41:   while tokenizing substring based on break point do
42:      if token_length > substring_length_constraint
43:      then, result_LCS ←Append token_substring,
44:   end while
45: return result_LCS
```

알고리즘 1 은 LASER 알고리즘과 signature 생성의 전체적인 과정을 기술하고 있다. Signature 생성을 위한 타깃 어플리케이션의 트래픽은 트래픽 수집 에이전트에 의해 동일한 5-tuple 정보를 갖는 패킷의 집합인 flow 형태로 수집된다. Signature 생성의 첫번째 반복 과정의 입력값으로 서로 다른 두 flow F1 과 F2 가 사용되며 LASER 알고리즘의 입력 값으로는 각각 F1 과 F2 에 속해있는 서로 다른 패킷 payload 의 byte stream 이 사용된다 (Line 8).

그림 2 는 하나의 flow 에 해당하는 packet 수에 대한 제약사항 (Line 5, 6)과 패킷 크기에 대한 제약사항 (Line 7)이 적용된 후의 과정을 나타내고 있다. 각각의 패킷은 LimeWire 에서 발생되었으며 Gnutella 계열의 클라이언트인 Morpheus 의 사용자가 LimeWire 의 클라이언트에게 보내는 byte stream 의 일부이다. Candidate signature 는 signature 생성 과정 중 아직 최종적으로 확정되지 않은 어플리케이션의 signature 를 의미하며 그림의 candidate signature 는 이 두 byte stream 중 불필요한 string 들이 LASER 알고리즘에 의해 제거된 결과를 다음과 같이 보여주고 있다: ‘HTTP 200 OK Server: * e * Content-type: * e *’

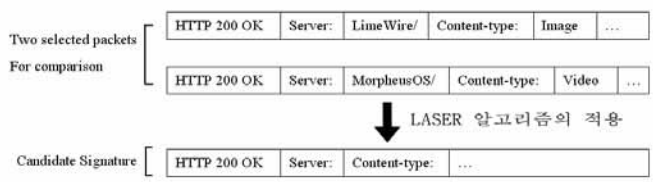

그림 2. 두개의 LimeWire 패킷에 대한 LASER 알고리즘의 적용

하지만 ‘e’와 같은 substring 은 signature 로써 의미를 갖기에는 길이가 짧기 때문에 substring 의 길이에 대한 제약사항 (Line42)를 통해 이와 같은 string 은 최종적인 signature 에서 제거된다. 한번의 반복을 통해 생성된 candidate signature 는 단 2 개의 flow 만을 이용하였기 때문에 (Line 1-12) 불필요한 string 들이 다수 포함되어 있을 수가 있다. 예를 들어 HTTP protocol 을 사용하는 application 들에서 자주 나타나는 ‘HTTP 200 OK’와 같은 string 은 signature 로써 변별력이 떨어지기 때문에 signature refinement 과정을 필요로 한다.

Signature refinement 과정은 더 많은 flow 의 sample set 을 입력으로 하여 LASER 알고리즘의 반복적인 적용을 통해 candidate signature 상의 IP 주소, URL 등과 같은 무의미하거나 불필요한 string 들을 제거하는 과정을 의미한다. 첫 번째 반복으로 생성된 candidate signature 는 새로운 flow 와 함께 LASER 알고리즘의 입력으로 다시 사용된다(Line 12-17). 알고리즘의 반복이 계속 됨에 따라 candidate signature 상의 불필요한 substring 들은 제거되며 이러한 반복은 candidate signature 가 더 이상 변화하지 않을 때까지 반복된다. 궁극적으로 많은 반복을 통해 좀 더 간략하고 정확한 signature 가 생성되며 이러한 과정은 다음과 같이 간략히 표현될 수 있다.

Candidate_signature_1 = Signature (Flow 1, Flow 2)
Candidate_signature_2 = Signature (Flow 3, Candidate_signature_1)
…
Candidate signature_n = Signature (Flow n+1, Candidate_signature_n-1)

If Candidate_signature_n = Candidate signature_n-1
For the certain iteration counts then
Candidate_signature_n is the final signature

위와 같은 signature refinement 과정의 최종 결과물로 어플리케이션 signature 가 결정된다.

4. Signature 의 정확도 검증

LASER 알고리즘으로 생성한 signature 의 정확도를 검증하기 위하여 3 가지의 P2P 어플리케이션 (LimeWire, BitTorrent, Fileguri)을 선정하여 signature 를 생성하고 Backbone 트래픽에 적용하여 트래픽 분류 결과를 확인하였다. LimeWire 와 BitTorrent 는 전세계적으로 매우 널리

사용되고 있는 파일공유 어플리케이션이며 Fileguri 역시 국내에서 가장 많이 사용되고 있는 파일공유 P2P 어플리케이션 중 하나이며, 이들은 signature 기반의 트래픽 분류 연구들[4],16]에서 또한 사용되었던 어플리케이션들이다.

3.3 정확도 판단을 위한 Metrics

Signature 의 정확도를 판단하기 위해 3 가지의 metric 을 정의하였다.

- ***False Positive (FP)***: application 의 signature 가 non-application traffic 을 application traffic 으로 분류하는 error 를 측정하는 metric

$$FP = \frac{\text{Non-application traffic classified as application traffic}}{\text{Total application traffic}}$$

- ***False Negative (FN)***: application 의 signature 가 application 의 traffic 을 non-application traffic 으로 분류하는 error 를 측정하는 metric

$$FN = \frac{\text{Application traffic classified as non-application traffic}}{\text{Total application traffic}}$$

- ***Overall Accuracy (OA)***: 하나의 signature 에 해당되는 정확도가 아닌 모든 signature 를 종합하여 분류된 결과의 전체적인 정확도를 판단하는 metric. OA ratio 는 FP, FN 비롯하여 signature 간의 충돌까지 판단할 수 있는 기준을 제공한다.

$$OA = \frac{\text{Total traffic} - (\text{Total FP traffic} + \text{Total FN traffic})}{\text{Total traffic}}$$

앞의 두 metric 들은 잘못된 분류 결과의 정도를 나타내며 OA 는 모든 signature 를 적용하여 분석한 분류 결과의 정확도를 판단하는 것으로 어플리케이션 signature 의 정확도가 높다는 결론을 내리기 위해서는 낮은 FP 와 FN 와 높은 OA 가 필요하다.

3.4 트래픽 수집과 검증 방법

우리는 POSTECH 의 backbone 에서 수집한 패킷 trace 를 분석하였다. 그림 3 은 POSTECH 의 backbone 네트워크를 도식화한 것이다. POSTECH 의 네트워크는 3000 여명의 학생과 교직원이 사용하는 Gigabit Ethernet 으로 어떠한 port 도 사용에 제한을 받고 있지 않다. 따라서 네트워크 상에는 다양한 P2P 어플리케이션 트래픽을 포함한 일반적인 Internet 트래픽들이 존재한다. 검증에 사용된 트래픽은 은 2007 년 8 월 16 일 목요일에 3 시간 동안 수집되었으며 전체 트래픽의 양은 약 450Gbtyes 였다.

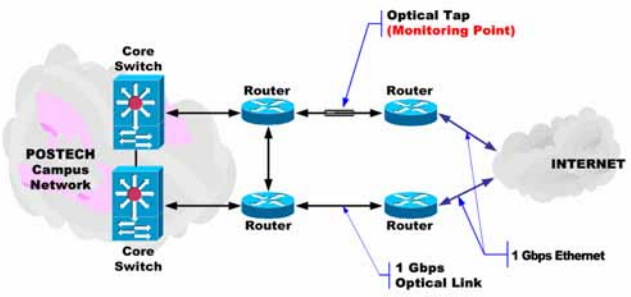

그림 3. POSTECH 의 Backbone 네트워크

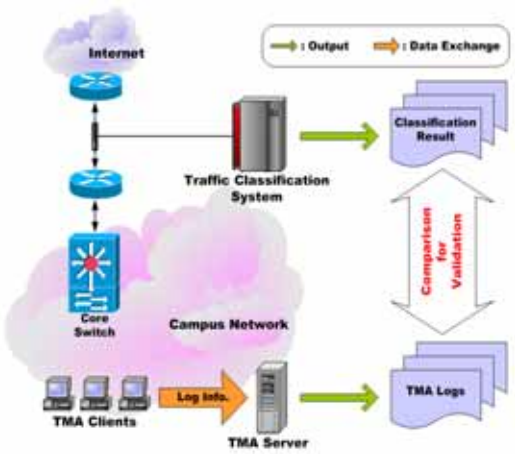

그림 4. TMA 의 활용

수집된 트래픽을 signature 기반의 트래픽 분류 시스템을 통해 분류하여도 어떠한 트래픽이 어떠한 어플리케이션에서 발생한 트래픽인지를 정확히 판단할 수 없다면 해당 분류 결과를 통해 얻은 통계는 신뢰할 수가 없다. 2 장에서 언급된 몇몇의 연구들에서는 FN 를 계산하기 위하여 수작업을 통해 패킷 내용을 하나하나 분석하거나 port 기반의 트래픽 분류 방법을 이용하여 얻은 결과를 ground truth 로 가정하였으며, 고립된 test bed 에서 타깃 어플리케이션을 실행하여 발생한 트래픽을 수집하였다. 또한 FP 값을 계산하기 위하여 P2P 트래픽이 존재하지 않을 것으로 가정되는 VPN (Virtual Private Network)의 트래픽을 사용하였다. 하지만 이러한 방법들은 기존에 정확도의 문제를 갖고 있는 port 기반의 분석 방법이나 가정에 의존하기 때문에 정확한 FP/FN 값을 얻기 힘들다.

본 연구에서는 이러한 문제점을 해결하고 좀더 정확한 수치를 확인할 수 있도록 TMA (Traffic Measurement Agent)를 개발하였다. TMA 는 Windows 기반의 클라이언트에 설치되어 클라이언트에서 발생되는 모든 트래픽에 대하여 로그를 남기는 기능을 수행한다. 3.1 장의 트래픽 수집 에이전트와 같이 Windows system call 을 사용하여 발생하는 모든 패킷에 대하여 어떠한 어플리케이션 또는 프로세스에서 생성된 패킷인지를 기록하기 때문에 트래픽 분류 시스템이 분류한 결과를 TMA 의 로그와 비교할 경우 정확한 FN/FP, OA 를 계산할 수 있다. 그림 4 는 TMA 의 역할을 간략히 보여주는 그림이다. 캠퍼스 네트워크 상의 edge 클라언트에

TMA 를 설치하여 각각의 클라이언트들이 발생하는 트래픽에 대한 로그를 남기고 이를 TMA 로그 서버로 전송한다. TMA 로그 서버는 각각의 TMA 클라이언트들로부터 전송된 모든 로그 데이터를 갖고 있으며, 이는 앞서 설명한 바와 같이 트래픽 분류 시스템이 분석한 분류 결과의 정확도를 판단할 수 있는 비교 자료로 활용된다. 이러한 TMA 는 본 논문의 연구 주제인 signature 기반의 트래픽 분류뿐만 아니라 다른 모든 방법의 트래픽 분류 방법들의 정확도를 측정하는데 활용되어 정확한 자료를 제공할 수 있다.

3.4 정확도 검증 결과

2 개의 클라이언에 TMA 를 설치하고 정확도 측정에 사용할 3 가지 어플리케이션 (Limewire, BitTorrent, Fileguri) 이외에도 다양한 네트워크 어플리케이션을 실행시킨 후 backbone 에서 트래픽을 수집하는 동시에 클라이언트에서 발생한 트래픽에 대한 TMA 로그를 남겼다. 그리고 backbone 에서 수집된 트래픽들 중 IP 필터링을 통해 TMA 가 설치된 두 개의 클라이언트에서 발생한 트래픽만을 따로 분류해낸 후 이 트래픽들에 signature 를 적용하여 트래픽 분류 작업을 수행하였다. TMA 가 설치된 클라이언트에서 직접 트래픽을 수집하지 않고 backbone 에서 수집한 이유는 다음과 같다. 많은 패킷들이 backbone 을 통과할 때 패킷 시퀀스의 변경이나 비대칭 라우팅 등의 영향을 받게 된다. 많은 트래픽 분류 시스템들이 backbone 네트워크의 링크에 설치 됨을 고려할 때에 종단 호스트에서 수집한 트래픽을 그대로 사용할 경우 앞에서 설명한 트래픽의 변화를 제대로 반영할 수 없기 때문이다. 표 1 은 해당 트래픽에 대한 분류 결과를 보여주고 있다.

- ***FP 분석***: TMA 로그를 통해 3 개의 타깃 어플리케이션을 제외한 트래픽중 3 가지 어플리케이션의 트래픽으로 분류된 트래픽은 0%였다. Signature 의 포맷이 substring 들이 시퀀스를 이루는 형태의 엄격한한 형태이기 때문에 다른 어플리케이션의 트래픽을 오분류 하지 않는 것을 확인할 수 있다. 한가지 주목할 점은 others 에 해당하는 트래픽에는 Internet explorer 에서 발생한 HTTP 트래픽이 많은 량 포함되어 있음에도 LimeWire 나 Fileguri 로 분류되지 않았다는 것이다. 물론 signature 에 포함된 'HTTP'라는 substring 이 공통적으로 포함되어 있지만 나머지 substring 들과 이들이 이루는 시퀀스의 유일함이 FP 를 낮게 함을 확인할 수 있는 결과이다.
- ***FN 분석***: 세가지 어플리케이션 모두 FN 을 보였다. 이는 해당 어플리케이션 트래픽 중 signature 로 탐지하지 못하는 트래픽이 일부 존재한다는 것을 의미한다. 이러한 오분류의 원인은 여러 가지가 있을 수 있으나 이에 해당되는 트래픽을 확인해본 결과 다음과 같은 원인에 의한 것이었다. 패킷에 어플리케이션 레벨의 payload 가 존재하지 않는 경우, 예를 들어 TCP connection 을 위한 SYN, SYN-ACK, ACK 등의 flag 만을 갖는 flow 는 제대로 분석이 되지 않았다. 또한 트래픽을 수집하는 구간에서 비대칭적인 라우팅 경로로 인하여 signature 가 존재하는 패킷이 소실된 경우에도 해당 flow 를 제대로 분류하지 못하였다. 그리고 어플리케이션 내부에 웹 브라우저가 내장된 경우 트래픽을 발생한 것은 해당 어플리케이션이지만 실제 트래픽의 내용은 순수한 웹 트래픽과 동일하기 때문에 어플리케이션 signature 로 트래픽을 분류하지 못하는 경우가 발생하였다.
- ***OA 분석***: OA 는 97.39%로 높았지만 BitTorrent 와 LimeWire 의 경우 FN 이 10%에 가까운 수치를 보였다. 이는 상당히 부정확한 정확도로 판단할 수 있지만 프로토콜 분석로 signature 를 생성한 다른 연구 [4]의 결과에서 Gnutella 와 BitTorrent 의 FN 이 각각 4.97%, 9.90%임을 감안할 때에는 수용 가능한 결과이다. LASER 알고리즘이 생성한 signature 는 수작업으로 어플리케이션의 모든 프로토콜을 분석하는 것과 달리 모든 과정이 자동으로 이루어지는 것을 생각할 때에 이러한 점은 더욱 확실해 진다. 더욱이 다른 연구에서 [4]에서 FN 의 기준으로 삼은 port 기반의 트래픽 분류 방법은 TMA 와 달리 어플리케이션에서 발생한 일반적인 웹 트래픽을 해당 어플리케이션 트래픽으로 구분하지 못하기 때문에 오히려 실제보다 더 낮은 FN 값을 가질 가능성이 있다는 점을 감안할 때 이 사실은 더욱 명확해진다.

표 1. 트래픽 분류 결과

Application	TMA Log (MB)	Classification Result (MB)	False Negative (%)	False Positive (%)
LimeWire	1223.36	1120.35	8.42	0
BitTorrent	4190.07	3754.30	10.40	0
Fileguri	3189.61	3177.17	0.39	0
Others	12482.69	13033.91	-	-
Total	21085.73MB		-	-
Overall Accuracy	97.39 (%)			

제한적이지만 정확성 검증을 통해 LASER 알고리즘을 통해 생성한 signature 는 수용 가능한 FN/FP, OA 의 정확도를 가지고 있음을 확인하였다. 더 많은 수의 TMA 클라이언트를 활용한다면 트래픽 분류 결과에 대한 좀더 정확한 수치를 확인할 수 있을 것이다.

3.5 LASER 알고리즘의 효율성

그림 5 는 3.3 절에서 설명한 signature refinement 과정에서 signature 의 길이 변화를 도식화한 것이다. Refinement 과정이 반복됨에 따라 candidate signature 의 길이는 점차 감소하게 되며 특정 시점을 기준으로 더 이상 길이의 변화가 없는 최종적인 signature 가 된다. 세가지 어플리케이션 모두에서 최종적인 signature 가 생성되기까지 필요한 refinement 반복과정은 10 회 이하로 나타났다. 이는 signature 생성을 위한 LASER 알고리즘 트래픽 입력 데이터의 양이 적다는 것을 의미한다.

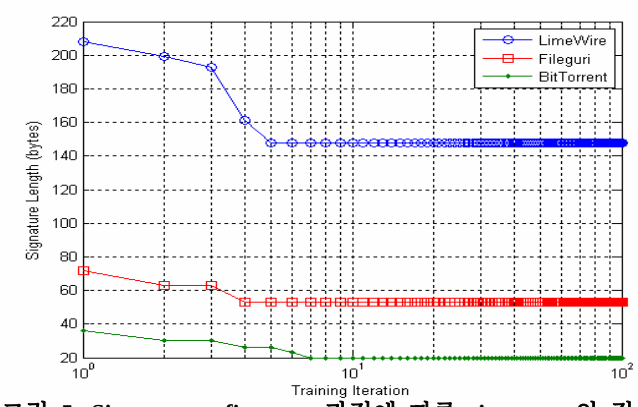

그림 5. Signature refinement 과정에 따른 signature 의 길이 변화

표 2. LASER 를 통해 생성한 어플리케이션 Signature

	Automated Signature Generation by LASER
LimeWire	"LimeWire" "Content-Type:" "Content-Length:" "X-Gnutella-Content-URN" "run:sha:1" "X-Alt" "X-Falt" "X-Create-Time:" "X-Features:" "X-Thex-URI"
BitTorrent	"0x13BitTorrent protocol"
Fileguri	"HTTP" "Freechal P2P" "User-Type:" "P2P-ErrorCode:" "Content-Length:" "Content-Type:" "Last-Modified"
FTP	"230 logged"
Afreeca TV	"0x02 02 21 CB 4E 02 00 00 6D DB 00 00", "0x00 00 00 00", "0x7E 00 00"
PDBOX	"0x00 00 01 03 16 05 00 00 08 00 00 00 1E 05 01 03 00 00 00 00 32 00 00 00 57 37 59 5D"
Skype (v3.0)	No signature can be found
KaZaA(v3.25)	"HTTP1.1" "Kazaa client" "X-Kazaa-Username:" "X-Kazaa-Network:" "X-Kazaa-IP:" "X-Kazaa-SupernodeIP:" "X-Kazaa"

표 2 는 LASER 알고리즘을 통해 생성한 어플리케이션 signature 의 리스트이다. 우리는 정확도 검증에 사용한 어플리케이션 이외에 전통적인 ftp 를 포함하여 VoIP, P2P, 인터넷 TV 등의 어플리케이션 카테고리에서 최소 1 개 이상의 어플리케이션을 선정하여 signature 를 생성해 보았다. Afreeca TV 와 PD-BOX 는 국내에서 많이 사용되고 있는 인터넷 TV 와 웹 스트로지의 대표적인 어플리케이션이다. 또한 암호화를 사용하는 어플리케이션에 대한 signature 생성 가능성을 판단하기 위해 KaZaA 와 Skype 의 signature 를 생성해 보았다. 기존의 Skype v1.5 와 v2.0 의 signature [9]는 더 이상 사용이 불가능 했으며 패킷의 암호화로 인하여 어떠한 signature 도 생성할 수 없었다. 대다수의 어플리케이션들이 암호화를 사용하고 있지 않지만 Skype 와 같은 어플리케이션을 분류하기 위해서는 signature 가 아닌 새로운 접근 방법이 필요할 것으로 판단된다.

5. 결론 및 향후 과제

본 논문에서는 LASER 알고리즘과 기반으로 선행적인 프로토콜 분석이나 수작업을 통한 payload 분석 없이 자동적으로 어플리케이션 signature 를 생성할 수 있는 시스템을 제시하였고, 트래픽 분류 시스템의 분류 결과에 대한 정확도 검증을 위한 새로운 방법을 제시하고 이를 통하여 LASER 알고리즘의 정확도를 검증하였다. 또한 적은 양은 flow 를 시스템의 입력 데이터로 사용하여도 충분히 정확한 signature 를 생성할 수 있음을 확인하였다.

향후 과제로 주기적으로 트래픽을 수집하고 이를 활용하여 실시간으로 어플리케이션 signature 를 자동으로 생성하는 signature 생성 에이전트와 생성된 signature 를 정기적으로 업데이트하여 자동으로 트래픽을 분류하는 시스템을 개발하고 있으며 signature 기반의 트래픽 분류시스템의 활용성과 정확성의 더 많은 검증을 위해 signature 데이터베이스를 구축하여 관련 연구 커뮤니티와의 공유를 계획하고 있다.

참고 문헌

[1] S. Sen and J. Wang. "Analyzing peer-to-peer traffic across large networks," Proceedings of ACM SIGCOMM Internet Measurement Workshop, Marseilles, France, November 2002.

[2] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. "Measurement, modeling, and analysis of a peer-to-peer File-sharing workload," Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP-19), Bolton Landing, NY USA ,October 2003.

[3] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and KC Claffy. "Transport layer identification of p2p traffic," Internet Measurement Conference (IMC), Taormina, Sicily, Italy, 2004.

[4] Subhabrata Sen, Oliver Spatscheck, Dongmei Wang. "Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures," New York, USA, WWW 2004 Conference.

[5] H. Kim, B. Karp. "Autograph: Toward automated, distributed worm signature detection," Proceedings of

the 13th Usenix Security Symposium, San Diego, CA, 2004.

[6] S. Singh, C. Estan, G. Varghese, S. Savage. “Automated worm Fingerprinting,” Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation, San Francisco, California ,2004.

[7] Sumeet Singh, Cristian Estan, George Varghese and Stefan Savage. “The EarlyBird System for Real-time Detection of Unknown worms”, UCSD Tech Report CS2003-0761, August 2003.

[8] W. Scheirer, M. Chuah. “Comparison of Three Sliding-Window Based worm Signature Generation Schemes,” , Lehigh University Technical Report LU-CSE-05-025.

[9] Sven Ehlert and Sandrine Petgang, “Analysis and Signature of Skype VoIP Session Traffic,” Franunhofer FOKUS Technical Report NGNI-SKYPE-06b, Berlin, Germany, July, 2006.

[10] Laurent Bernaille and Renata Teixeira, “Early Recognition of Encrypted Applications,” PAM 2007, Louvain-la-neuve, Belgium, April 5-6, 2007, pp. 165-175.

[11] 박병철, 원영준, 김명섭, 홍원기 “자동화된 어플리케이션 레벨의 signature 생성”, KNOM 2007.

[12] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, 2nd Edition, MIT Press, 2001.

[13] Winpcap. http://www.winpcap.org/.

[14] James Newsome and Dawn Song. “Dynamic Taint analysis: Automatic Detection, analysis, and Signature Generation of Exploit Attacks on Commodity Software,” In Network and Distributed Systems Security Symposium, San Diego, California, USA, 2005

[15] The Gnutella protocol specification, http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[16] Young J. Won, Byung-Chul Park, Hong-Taek Ju, Myung-Sup Kim, and James W. Hong. “A Hybrid Approach for Accurate Application Traffic Identification,” IEEE/IFIP E2EMON, Vancouver, April 2006, pp. 1-8.