# LASER: 자동화된 어플리케이션 레벨의 Signature 생성과 검증 방법

*박병철, *원영준, **김명섭, *홍원기
*포항공과대학교 컴퓨터공학과, **고려대학교 컴퓨터공학과
*{fates,yjwon,jwkhong}@postech.ac.kr, **tmskim@korea.ac.kr

# LASER: Automated Application Signature Generation and Evaluation Methodology

*Byung-chul Park, *Young J. Won, **Myung-Sup Kim, and *James W. Hong
*POSTECH, **Korea Univ.

## 요 약

현재까지 가장 전통적인 트래픽 분류는 미리 정의된 well-known 포트와 비교하는 방법을 통해 이루어져왔지만 이러한 분석 방법은 더 이상 높은 정확도를 보장하지 못하고 있다. 대체 방법인 어플리케이션 레벨의 signature 비교 방법은 더 높은 정확도를 제공하지만 신뢰도가 높은 signature 를 찾기 위해 트래픽에 대해 수작업에 의한 철저한 탐색이 요구된다. 어플리케이션 프로토콜에 대한 정보가 사전에 제공된다면 높은 신뢰도의 signature 를 생성할 수 있지만 많은 어플리케이션들이 독자적인 프로토콜을 사용하게 됨에 따라 많은 시간이 소요되는 수작업에 의한 signature 생성작업을 피할 수 없게 되었다. 본 논문에서는 어플리케이션 프로토콜에 대한 정보가 없이도 어플리케이션 레벨의 signature 를 자동적으로 생성할 수 있는 LASER 알고리즘과 해당 signature 의 정확도를 검증할 수 있는 방법을 제시하고 이를 통해 LASER 가 효율적인 방법으로 높은 정확도의 signature 를 생성함을 보인다.

## I. 서론

오늘날 인터넷 트래픽의 급격한 변화로 포트 기반의 트래픽 분류와 같은 기존의 전통적인 트래픽 분류 기술은 더 이상 높은 정확도를 제공하지 못하게 되었다. 최근 들어 웹(인터넷) storage 와 Peer-to-Peer (P2P) 파일 공유 어플리케이션들의 종류가 급격하게 증가하였고, 이러한 트래픽이 전체 인터넷 트래픽의 많은 부분을 차지하고 있다[1]. 특히 최신의 P2P 어플리케이션들은 동적인 포트 할당과 같은 방법을 통해 detection 과 filtering 을 피하고 있다. 하지만 포트 기반의 분석 방법 backbone 네트워크의 트래픽 양이나 computing 리소스 때문에 현재에도 많이 사용되고 있으며 이러한 방법으로는 정확한 어플리케이션 분석이 이루어질 수 없다.

Signature 기반의 트래픽 분류 방법은 기존의 분류방법들이 갖는 부정확성을 개선하기 위한 대체 방법으로 제안되었다[2][3]. 어플리케이션 signature 는 어플리케이션을 구분 지을 수 있는 패킷 payload 상의 특정한 string 이나 hex 값으로 이루어진다. 어플리케이션 signature 는 수작업을 통해 생성되어 왔으며 이는 어플리케이션 프로토콜에 대한 사전조사나 패킷 payload 에 대한 분석이 요구된다. 사람의 판단이 필요한 이와 같은 작업은 새로운 어플리케이션의 signature 를 생성할 때에 많은 시간을 필요로 한다. 따라서 좀더 효율적인 방법으로 signature 를 생성할 수 있는 체계적인 방법이 요구된다.

본 논문은 프로토콜에 대한 정보 없이 패킷 payload 상의 패턴을 찾아 signature 를 생성할 수 있는 LCS (Longest Common Subsequence)기반의 LASER (LCS-based Application Signature ExtRaction) 알고리즘을 제안한다. 또한 LASER 를 통해 생성한 signature 의 정확도를 검증할 수 있는 방법을 제시하고 이를 통해 LASER 알고리즘의 효율성과 정확성을 검증한다. 본 논문의 구성은 다음과 같다. II 장에서는 관련연구들과 우리의 선행 연구를 소개한다. III 장에서는 자동으로 signature 를 생성할 수 있는 알고리즘에 대해 간략히 설명한다. IV 장에서는 III 장에서 설명한 알고리즘으로 생성한 어플리케이션 signature 의 정확도를 검증하며 마지막으로 V 장에서 결론과 향후 연구에 대하여 기술한다.

## II. 관련 연구

어플리케이션 signature 를 이용한 트래픽 분류에 관한 기존의 연구들은 [2][3] 자체적으로 생성한 signature 를 통해 어플리케이션 트래픽의 분석률을 제시하지만 signature 를 생성하는 방법이나 그 signature 자체의 정확도에 대한 검증을 중요하게 언급하고 있지 않다.

Signature 기반의 트래픽 분류는 웜 탐지를 위한 IDS (Intrusion Detection System) 들에서 주로 사용되어왔다[4]. 이러한 연구들은 웜 signature 를 자동으로 생성할 수 있는 시스템의 개발에 대해 기술하고 있지만 웜 signature 와 일반적인 어플리케이션의 signature 의 각각의 독특한 특징과 차이점 때문에 웜 탐지 시스템에 적용된 방법을 일반적인 어플리케이션 signature 의 생성에 적용하기가 어렵다[5].

우리는 선행 연구[5]에서 IDS 와 일반적인 트래픽 분류시스템의 차이점을 지적하고 웜과 일반 어플리케이션의 signature 가 갖는 차이점을 분석하여 제시하였다. 또한 기존 연구들에서 정의한 다양한 어플리케이션의 signature 포맷을 정리하였다. 이러한 분석을 통해 LCS[5][6] 알고리즘을 활용하여 가장 포괄적인 signature 의 포맷인 sequence of common substring 형태의 signature 를 생성할 수 있는 시스템을 개발하고 알고리즘

본 연구는 2007 년도 두뇌한국 21 사업과 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 지원사업의 연구결과로 수행되었음(IITA-2007-(C1090-0701-0045))

의 효율성에 대한 검증을 제시하였다. 본 논문은 알고리즘의 효율성이 아닌 LASER 알고리즘이 생성한 signature 의 정확성 검증에 초점을 두고 있다.

## III. LASER 알고리즘을 통한 signature 생성

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:    Fi ← select one from the rest of Flow_Pool
15:    result_LCS ← LASER (S, Fi)
16:    S ← select the longest from result_LCS
17:    i++, end while, end while
18:  return S

19: procedure LASER (PacketA[1...m], PacketB[1...n])
20:  PacketA [m...1] ← Reverse byte stream
21:  PacketB [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 PacketA [i] = PacketB [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 PacketA[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 기술한 것이다. Signature 생성을 위한 타깃 어플리케이션의 트래픽은 동일한 5-tuple 정보를 갖는 패킷의 집합인 flow 로 수집된다. Signature 생성의 첫 번째 반복과정의 입력값으로 서로 다른 두 flow $F_1$ 과 $F_2$ 가 사용되며 LASER 알고리즘의 입력값으로는 각각 $F_1$ 과 $F_2$ 에 속해있는 서로 다른 패킷 payload 의 byte stream 이 사용된다 (Line 8).

그림 1 은 하나의 flow 에 해당하는 packet 수에 대한 제약사항 (Line 5, 6)과 패킷 크기에 대한 제약사항 (Line 7)이 적용[]된 후의 과정을 설명하고 있다. LimeWire 에서 발생된 두 개의 패킷을 나타내고 있는데, Gnutella 계열의 클라이언트인 Morpheus 의 사용자가 LimeWire 의 클라이언트에게 보내는 byte stream 의 일부이다. 그림의 candidate signature 는 이 두 byte stream 중 불필요한 strin 들이 LASER 알고리즘에 의해 제거된 결과를 다음과 같이 보여주고 있다: *'HTTP 200 OK Server: * e * Content-type: * e *'*

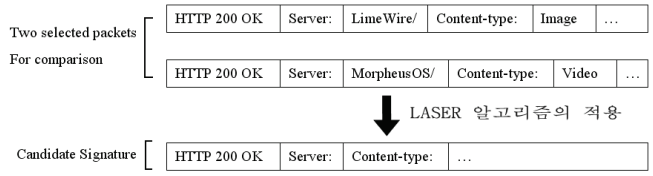


**그림 1. 서로 다른 두 LimeWire 패킷에 LASER 알고리즘의 적용**

하지만 'e'와 같은 substring 은 signature 로써 의미를 갖기에는 길이가 짧기 때문에 substirng 의 길이에 대한 제약사항 (Line42)를 통해 이와 같은 string 은 제거된다. 한번의 반복을 통해 생성된 candidate signature 는 단 2 개의 flow 만을 이용하였기 때문에 (Line 1-12) 불필요한 string 들이 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. 첫번째 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***

## IV. LASER 알고리즘과 Signature 의 검증

우리는 LASER 알고리즘을 통해 생성된 signature 의 정확성을 검증하기 위해 3 가지의 P2P 어플리케이션 (LimeWire, BitTorrent, Fileguri)을 선정하였다. LimeWire 와 BitTorrent 는 전세계적으로 매우 널리 사용되고 있는 파일공유 어플리케이션이며 Fileguri 역시 국내에서 가장 많이 사용되고 있는 P2P 어플리케이션 중의 하나로 기존의 signature 기반의 트래픽 분류 연구[3][7][8]에서 사용되었던 어플리케이션들이다.

### 1. 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)*: OA 는 특정 signature 의 정확도 뿐만 아니라 모든 signature 들을 포괄한 트래픽 분류 결과의 전체적인 정확도를 판단한다. 이 metric 은 FP, FN 뿐만 아니라 각 signature 들 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 모두 잘못된 분석결과의 ratio 를 나타내며 마지막 metric 은 모든 signature 를 적용하여 분석한 트래픽 분류결과의 정확도를 판단하는 것으로 어플리케이션 signature 의 정확도와 신뢰도가 높다고 결론을 내리기 위해서는 낮은 FP 와 FN ratio 와 높은 OA 가 필요하다.

## 2. 트래픽의 수집과 검증 방법

우리는 POSTECH 의 backbone 에서 collect 한 전체 packet trace 를 분석하였다. POSTECH 의 network 는 3000 여명의 구성원이 사용하는 Gigabit Ethernet 으로 어떠한 port 도 block 되지 않으며 다양한 P2P 어플리케이션을 포함한 일반적인 Internet 트래픽으로 이루어져있다. 트래픽은 2007 년 8 월 16 일에 3 시간 동안 수집되었으며 전체 트래픽의 양은 약 450 Gbytes 이다.

수집된 트래픽을 signature 기반의 트래픽 분류 시스템을 통해 분류하여도 분류결과의 정확성을 정확히 판단할 수 없다면 이를 통해 얻은 통계는 신뢰할 수가 없다. 제 2 장의 기존 연구들에서는 이를 계산하기 위하여 수작업을 통해 패킷 내용을 하나하나 분석하거나 port 기반의 트래픽 분류 방법을 이용하여 얻은 결과를 ground truth 로 가정하여 ratio 를 계산하였다. 또 다른 방법으로 isolate 된 test bed 에서 어플리케이션을 실행하여 발생한 트래픽을 수집하여 FN 을 계산하기도 한다. 어떤 연구는 P2P traffic 이 존재하지 않을 것으로 가정되는 VPN (Virtual Private Network)의 traffic 을 사용한다. 하지만 이러한 방법들은 기존에 정확성에 대한 문제를 갖고 있는 port 기반의 분석 방법이나 가정에 의존하기 때문에 정확한 FN/FP 값을 얻기 힘들다.

위와 같은 문제점을 해결하고 좀더 정확한 수치를 확인하기 위해 TMA (Traffic Measurement Agent)를 개발하였다. TMA 는 Windows 기반의 클라이언트에 설치되어 클라이언트에서 발생되는 모든 트래픽에 대하여 로그를 남기는 기능을 수행한다. Windows system call 을 사용하여 발생하는 모든 패킷에 대하여 어떠한 어플리케이션 (프로세스)에서 생성된 패킷인지를 기록하기 때문에 트래픽 분류 결과를 TMA 의 로그와 비교할 경우 정확한 FN/FP 를 얻을 수 있다. 그림 2 는 TMA 의 역할을 간략히 보여준다. 캠퍼스의 edge 클라이언트에 TMA 를 설치하여 각각의 클라이언트들이 발생하는 트래픽에 대한 로그를 남긴다. TMA 로그 서버는 모든 TMA 에서 발생한 로그를 갖고 있으며 이는 앞서 설명한 바와 같이 트래픽 분류 시스템이 분석한 결과의 정확도를 판단할 수 있는 비교 자료로 활용된다. 이러한 TMA 는 본 논문의 연구 주제인 signature 기반의 트래픽 분류뿐만 아니라 다른 모든 트래픽 분류 방법의 의 정확도를 측정하는데 활용되어 정확한 자료를 제공할 수 있다.

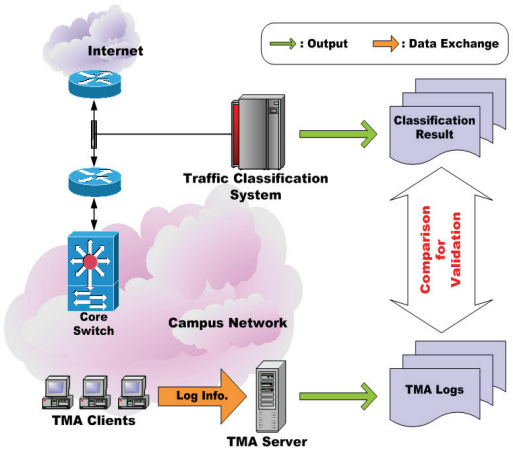


그림 2. TMA 의 활용

## 3. Evaluation Result

2 개의 클라이언트에 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 로그를 남겼다. 그리고 backbone 에서 수집된 트래픽 중 TMA 가 설치된 두 개의 IP 에서 발생한 트래픽만을 따로 분류해낸 후 이 트래픽에 signature 를 적용하여 분류 작업을 수행하였다. TMA 가 설치된 클라이언트에서 직접 트래픽을 수집하지 않고 backbone 에서 수집한 이유는 다음과 같다. 많은 패킷들이 backbone 을 통과할 때 packet 순서의 reordering 이나 비대칭 라우팅등의 영향을 받게 된다. 많은 트래픽 분류 시스템들은 backbone 네트워크의 링크에 설치됨을 고려할 때에 end 호스트에서 수집한 트래픽을 그대로 사용할 경우 앞에서 설명한 traffic 의 변화를 제대로 반영할 수 없기 때문이다. 표 1 은 분석 결과를 보여주고 있다.

표 1. 트래픽 분류와 정확도 분석 결과

| Application | TMA Log (MB) | Classification Result (MB) | FN(%) | FP (%) |
|---|---|---|---|---|
| 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 (%) | | | |

- *FP 분석*: Signature 의 포맷이 substring 들의 sequence[5]를 이루는 엄격한 형태이기 때문에 다른 어플리케이션의 트래픽을 잘못 분류하지 않는 것을 확인할 수 있다. 한가지 주목할 점은 others 에 해당하는 트래픽에는 Internet Explorer 에서 발생한 HTTP 트래픽이 많이 포함되어 있음에도 LimeWire 나 Fileguri 로 분류되지 않았다는 것이다. 물론 signature 에 포함된 'HTTP'라는 substring 이 공통적이긴 하지만 나머지 substring 들과 이들이

이루는 sequence 의 유일함이 FP 를 낮게함을 확인할 수 있는 결과이다.

- *FN 분석*: 세가지 어플리케이션 모두 FN 를 갖고 있는데, 이는 signature 로 탐지하지 못하는 어플리케이션 트래픽이 존재한다는 것을 의미한다. 이러한 misclassification 의 원인은 여러 가지가 있을 수 있으나 이에 해당되는 트래픽을 조사해본 결과 다음과 같은 원인에 의한 것이었다. 패킷에 어플리케이션 레벨의 payload 가 존재하지 않는 경우, 예를 들어 TCP 연결을 위한 SYN, SYN-ACK, ACK 등의 flag 만을 갖는 flow 는 제대로 분석이 되지 않았다. 또한 트래픽을 수집하는 구간에서 비대칭 라우팅으로 인하여 signature 가 존재하는 패킷이 사라진 경우에도 해당 flow 를 제대로 분류하지 못하였다. 그리고 어플리케이션 내부에 웹 브라우저가 내장된 경우 트래픽을 발생한 것은 해당 어플리케이션이지만 실제 트래픽은의 순수한 웹 트래픽과 동일하기 때문에 제대로 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 프로토콜 분석으로 signature 를 생성한 다른 연구[7]의 결과에서 Gnutella 와 BitTorrent 의 FN 이 각각 4.97%, 9.90%임을 감안할 때에는 높은 정확도를 가지고 있다고 할 수 있다. LASER 알고리즘이 생성한 signature 는 수작업으로 어플리케이션의 모든 프로토콜을 분석하는 것과 달리 트래픽 수집을 제외한 모든 과정이 자동으로 이루어지는 것을 생각할 때에 이러한 점은 더욱 확실해 진다. 더욱이 [3]에서 FN 의 기준으로 삼은 port 기반의 트래픽 분류 결과는 TMA 와 달리 어플리케이션에서 발생한 일반적인 웹 트래픽은 구분하지 못하기 때문에 오히려 실제보다 더 낮은 FN 값을 가질 가능성이 있다.

정확성 검증을 통해 제한적이기는 하지만 LASER 알고리즘을 통해 생성된 signature 는 수용 가능한 FN/FP 를 보이는 정확도를 가지고 있음을 확인할 수 있다. 앞에서 간략히 설명한 TMA 를 좀더 많은 클라이언트들에 설치하여 정확도를 측정한다면 좀더 정확한 수치를 확인 할 수 있을 것이다.

표 2 는 LASER 를 통해 생성된 어플리케이션의 signature 를 보여주고 있다. 우리는 VoIP, P2P, 인터넷 TV 등의 카테고리에서 최소 1 개 이상의 어플리케이션을 선정하여 signature 를 생성해 보았다. 앞의 세 어플리케이션을 제외한 signature 에 대한 검증은 비교대상이 될만한 기존 결과의 부재로 인해 제시하지 않는다. Afreeca TV 와 PD-BOX 는 각각 국내에서 많이 사용되는 인터넷 TV 와 웹스토리지 어플리케이션이다. 암호화된 패킷 payload 를 사용하는 어플리케이션에 대한 signature 생성 가능성을 판단하기 위해 Skype 와 KaZaA 의 signature 를 생성해 보았다. 기존에 존재하던 v1.5 와 v2.0 의 Skype signature 는 새로운 버전인 v3.0 에서는 더 이상 사용이 불가능 했으며 패킷의 암호화로 인하여 어떠한 패턴도 나타나지 않았다. 암호화된 프로토콜을 사용하는 어플리케이션을 분류하기 위해서는 signature 가 아닌 새로운 접근 방법이 필요할 것이다.

## V. 결론

본 연구에서 우리는 수작업을 통한 프로토콜 분석이나 패킷 분석 없이 자동으로 어플리케이션 signature 를 생성할 수 있는 새로운 접근 방법을 소개하였다. 또한 트래픽 분류 결과의 정확도를 판단할 수 있는 새로운 검증 방법을 제시하고 이를 통하여 LASER 알고리즘이 사람의 관여 없이 높은 정확도의 signature 를 생성한다는 것을 검증하였다. 향후 연구로 주기적으로 트래픽을 수집하고 이를 이용하여 실시간으로 signature 를 자동으로 생성하는 signature 생성 agent 를 개발하여 배포할 예정이며, signature 기반의 트래픽 분류 시스템의 활용성과 정확성의 더 많은 검증을 위해 signature 데이터베이스를 구축하여 관련 연구 커뮤니티와의 공유를 계획하고 있다.

**표 2. LASER 를 통해 생성한 어플리케이션 Signatrue**

| | 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” |

## 참고 문헌


[1] S. Sen and J. Wang. “Analyzing peer-to-peer traffic across large networks,” 2002 ACM SIGCOMM Internet Measurement Workshop, Marseilles, France, Nov. 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,” 19th ACM Symposium on Operating Systems Principles (SOSP-19), October 2003

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

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

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

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

[7] Subhabrata Sen, Oliver Spatscheck, Dongmei Wang. “Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures,” WWW 2004 Conference.

[8] 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 Workshop, Vancouver, April 2006, pp. 1-8.