TCAM을 이용한 고성능 패턴 매치 알고리즘

성 정 식[†] · 강 석 민[††] · 이 영 석[†††] · 권 택 근[††††] · 김 봉 태[†††††]

요 약

수 기가비트 급의 네트워크 성능 저하 없이 모든 패킷의 페이로드를 검사하여 유해 패킷을 검출해 내기 위해서 일반 메모리보다 빠른 검색을 지원하는 TCAM을 이용한 고성능 패턴 매치 알고리즘을 제안한다. 본 논문에서 제안하는 고성능 패턴 매치 알고리즘은 페이로드내에서 m 바이트의 문자열당 한 번의 TCAM 룩업을 수행하는 m-바이트 점핑 윈도우 기법을 이용하여 패킷의 페이로드당 TCAM 룩업 횟수를 줄여 페이로드 스캐닝 속도를 증가시킨다. 본 논문에서 제안한 방법과 TCAM 기반 슬라이딩 윈도우 패턴 매치 방법을 이용하여 페이로드 스캐닝 성능을 비교해 보고, 제안한 방법의 우수성을 시뮬레이션을 통해 보인다. 또한 m-바이트 점핑 윈도우 패턴 매치 알고리즘이 약 2천개의 패턴을 가지는 Snort 규칙을 이용한 시뮬레이션을 통해 9Mbit TCAM에서 10Gbps 이상의 페이로드 스캐닝 성능을 낼 수 있음을 보인다.

키워드 : 패턴 매치, TCAM, 통합 보안, 침입 탐지, 네트워크 프로세서, Snort

High-Speed Pattern Matching Algorithm using TCAM

Jungsik Sung[†] · Kang, Seok-Min[††] · Youngseok Lee[†††] · Kwon, Taeck-Geun[††††] · Bongtae Kim[†††††]

ABSTRACT

With the increasing importance of network protection from cyber threats, it is requested to develop a multi-gigabit rate pattern matching method for protecting against malicious attacks in high-speed network. This paper devises a high-speed pattern matching algorithm with TCAM by using an m-byte jumping window pattern matching scheme. The proposed algorithm significantly reduces the number of TCAM lookups per payload by m times with the marginally enlarged TCAM size which can be implemented by cascading multiple TCAMs. Due to the reduced number of TCAM lookups, we can easily achieve multi-gigabit rate for scanning the packet payload. It is shown by simulation that for the Snort rule with 2,247 patterns, our proposed algorithm supports more than 10 Gbps rate with a 9 Mbit TCAM.

Key Words : TCAM, Pattern Matching, DPI, Intrusion Detection, NP, Snort

1. 서 론

최근의 보안 기술은 산재되어 있던 보안을 하나의 네트워크안에서 연동하려는 통합네트워크 보안과 더불어 한 장비안에 방화벽, 침입탐지(IDS), 침입방지(IPS), QoS 등 다기능을 구현한 통합 보안 솔루션이 추세이다[1, 2]. 통합네트워크 보안은 네트워크 장비에 보안 가능을 추가하므로써 비용 절감에 효과적이고, 사용자들이나 기업은 보안을 위해 보안 제품을 사는 것이 아니라 통신 서비스와 같이 보안 서비스를 받음으로써 보안 기능을 지원받을 수 있다. 그러나 네트워크 장비에서 보안 기능을 지원할 때 가장 큰 문제점은 네트워크에 부하를 주지 않고, 완벽한 보안 기능을 지원할 수 있는가 하는 점이다. 수 기가비트 급의 네트워크 성능 저하 없이 모든 네트워크 패킷 검사를 위해서는 네트워크 장비는 고성능 프로세서를 사용해야 한다[3]. 네트워크 프로세서(NPU)는 고속으로 패킷을 처리할 뿐만 아니라, 새로운 형태의 공격 패턴이 발견되면 네트워크 장비에 새 공격 패턴을 프로그램해 넣을 수 있는 장점을 가지고 있다. 네트워크 프로세서가 가지는 네트워크 인터페이스의 전송속도가 기가비트 급을 상회하게 되면서 네트워크 프로세서의 패킷 처리 엔진을 다중 프로세서로 구성하고 있다. 예를 들면 인텔의 IXP28xx 네트워크 프로세서는 16개의 패킷 처리 엔진을 지원한다[4]. 패킷 처리 엔진은 모든 네트워크 트래픽에 대해 패킷을 분석하여 정상 트래픽과 유해 트래픽을 정확히 구분할 수 있어야 하는데 특히 모든 페이로드에 대한 Deep Packet Inspection(DPI)을 지원해야 한다. 이를 지원하기 위해서 패킷 처리 엔진은 모든 페이로드에 대해 메모리에 저

† 정 회 원 : 한국전자통신연구원 광통신연구센터 선임연구원
†† 준 회 원 : 충남대학교 컴퓨터공학과 박사과정
††† 정 회 원 : 충남대학교 컴퓨터공학과 교수
†††† 정 회 원 : 충남대학교 컴퓨터공학과 교수
††††† 정 회 원 : 한국전자통신연구원 광통신연구센터장
논문접수 : 2005년 4월 22일, 심사완료 : 2005년 6월 16일

장되어 있는 공격 패턴을 검색해야 한다. 네트워크 프로세서는 공격 패턴을 Dynamic Random Access Memory (DRAM) 또는 Static RAM(SRAM) 등의 메모리에 저장할 수 있는데, 이와 같은 메모리는 액세스 속도가 느릴 뿐만 아니라 공격 패턴을 검색할 때에도 검색 속도가 느리기 때문에 이를 이용해서는 수 기가비트 급의 라인 속도를 지원할 수 없다. Tenary Contents Addressable Memory(TCAM)은 하드웨어적으로 병렬 검색을 지원하므로 일반 메모리에 비해 빠른 검색을 수행할 수 있으므로 일반적으로 네트워크 프로세서는 TCAM을 사용하여 IP 룩업 및 포워딩 기능을 지원한다. 본 논문에서는 네트워크 프로세서에서 수기가비트 급의 네트워크 성능을 보장하면서 모든 페이로드에 대해 DPI를 지원할 수 있는 TCAM을 이용한 고성능 패턴 매치 알고리즘을 제안한다. 모든 페이로드를 조사하면서 고성능을 보장하기 위해서는 TCAM 룩업 횟수를 최대한 줄이는 것이 필요하다. 본 논문에서 제안하는 알고리즘은 한 번의 TCAM 룩업으로 페이로드의·다수 바이트를 처리하는 점핑 윈도우 패턴 매치(jumping window pattern matching) 기법을 이용하여 페이로드당 TCAM 룩업 횟수를 줄인다.

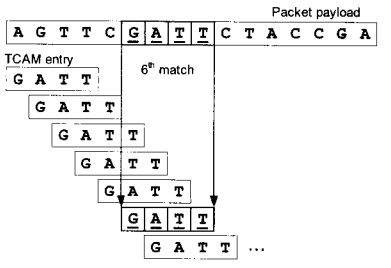
본 논문의 2장에서는 DPI에 사용될 수 있는 패턴 매치를 지원하는 알고리즘들에 대해 살펴보고 수 기가비트급의 성능을 지원하는 패턴 매치시 그 문제점에 대해 논의한다, 3장에서는 본 논문에서 제안하는 TCAM 기반 m-바이트 점핑 윈도우 패턴 매치를 이용한 수기가비트급 고성능 패턴 매치 알고리즘에 대해 상세히 논한다. 패턴 매치를 위한 시그너처로 사용하는 Snort 시그너처에 대한 분석 및 알고리즘에 대한 성능 분석를 4장에서 다루고, 5장에서는 네트워크에서 캡쳐링한 패킷을 이용하여 본 논문의 패턴 매치 알고리즘을 시뮬레이션한 결과를 보여준다. 6장에서는 결론 및 앞으로의 연구 과제에 대해 언급한다.

2. 관련 연구

네트워크 트래픽의 페이로드를 검색하여 유해 패킷을 탐지하기 위해서 시그너처 기반 탐지 기법을 많이 사용하고 있다. 이러한 시그너처는 검색테이블에서 검색 패턴으로 저장된다. 다양한 시그너처로부터 검출된 패턴은 검색테이블에 다중 패턴 형태로 존재하므로 입력되는 패킷의 페이로드를 검색테이블에서 검색할 때 다중 패턴 매치 수행이 이루어져야 한다.

[5, 6, 7, 8]은 고성능 다중 패턴 매치를 지원하는 알고리즘을 제안한 논문들이다. [5]는 Boyer-Moore(BM) 알고리즘[9]의 변형인 BM Horspool (BMH) 알고리즘[10]의 단순성을 그대로 살리면서 다중 패턴을 매치시킬 수 있는 Set-wise BMH(SBMH) 알고리즘을 제안하고 있다. [6] 또한 BM 알고리즘의 아이디어를 이용하는 것으로 BM 알고리즘에서 사용하는 것과 유사한 SHIFT 테이블을 사용한다. 그러나 shift 값이 0일 때 HASH 테이블과 PREFIX 테이블을 이용하여 다중 패턴 매치를 수행한다. [5, 6] 이외에도 [11, 12] 등도 컨텐츠 필터링을 위해 다중 패턴 매치를 지원한다. 그러나 위와 같은 다중 매턴 매치 알고리즘들은 소프트웨어를 기반으로 한 접근 방식으로 네트워크의 성능 저하를 초래한다.

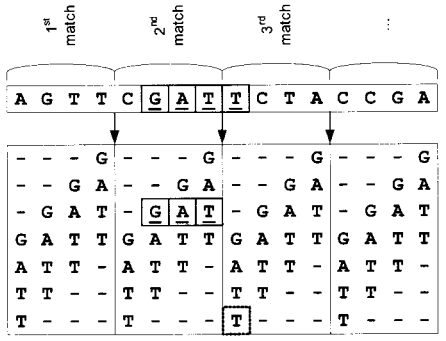
[7, 8]은 본 논문과 같이 TCAM을 이용하여 다중 패턴 매치를 지원하는 것으로, [7]은 캐시와 비슷한 역할을 담당하는 self-study 테이블을 두어 성능 향상을 꾀하고 있는데 이는 네트워크 트래픽의 burst 모드 특성 때문에 가능하다. 검색테이블에 저장되는 패턴의 길이가 다양하므로 [7]은 슬라이드 윈도우 개념을 사용하며, 슬라이드 윈도우는 패턴의 길이에 따라 가변적이다. 모든 페이로드에 대해 패턴이 매치되는지 조사해야 하므로 페이로드에 대해 슬라이드 윈도우를 한 바이트씩 포워딩시켜야 한다. 따라서 [7]은 최악의 경우 페이로드의 길이만큼 패턴 매치를 수행해야 한다. [8]은 검색테이블에 저장되는 패턴의 길이가 TCAM 길이보다 큰 경우, 패턴을 prefix, suffix 패턴으로 분리하고, SRAM에 long 패턴을 위해 패턴테이블, PHL(Parital Hit List) 테이블, 매칭테이블 등을 생성하여 일치되는 패턴을 검색한다. [8] 또한 [7]과 마찬가지로 페이로드에 대해 한 바이트씩 시프트하여 TCAM 룩업을 수행하여 공격 패턴을 검색해 내므로 최악의 경우 페이로드의 길이만큼 TCAM 룩업을 수행해야 한다. 본 논문에서는 [7, 8]과 같이 페이로드 한 바이트당 TCAM 룩업을 수행하는 패턴 매치의 방법을 슬라이딩 윈도우 패턴 매치라 명한다. (그림 1)의 예와 같이 슬라이딩 윈도우 방식의 패턴 매치를 이용하여 패턴 "GATT"을 검색할 경우 입력 패킷에서 패턴을 검출해 낼 때까지 6번의 TCAM 룩업이 이루어진다. 슬라이딩 윈도우 패턴 매치 방식의 페이로드 스캐닝 최대 성능은 한 번의TCAM 룩업으로 8비트를 처리하므로 TCAM 룩업 시간을 4ns라 가정할 때, 8bits/4ns = 2Gbps를 지원한다. 이것은 패턴의 길이가 TCAM 길이보다 짧은 경우이고, 긴 패턴인 경우에는 2Gpbs보다 낮은 성능을 보인다[9]. 이는 네트워크 프로세서에서 전송 속도가 기가비트 급을 상회하는 네트워크 인터페이스를 2개 이하일 때만 모든 페이로드에 대해 패턴 매치를 지원하면서 네트워크 성능 저하를 방지할 수 있음을 의미한다. 따라서 수 기가비트 급의 네트워크 인터페이스를 지원하기 위해서는 TCAM을 이용한 보다 빠른 페이로드 검색이 지원되어야 한다. 본 논문에서는 점핑 윈도우 패턴 매치 기법을 사용하여 한 번의 TCAM 룩업으로 페이로드의 다수 바이트를 처리하는 고성능 패턴 매치 알고리즘을 제안한다.



(그림 1) 슬라이딩 윈도우 방식의 패턴 매치

3. 고성능 패턴 매치 알고리즘

본 논문에서 제안하는 m-바이트 점핑 윈도우 패턴 매치 방식은 (그림 2)와 같이 페이로드의 모든 m 바이트마다 TCAM 룩업을 수행하여 패턴를 검색해 내는 것이다. 이 때, TCAM 엔트리의 -는 don't care를 나타낸다. (그림 2)에서 살펴보면 4-바이트 점핑 윈도우를 사용하는 경우, 3번의 TCAM 룩업으로 패턴을 검색해 낼 수 있으므로 (그림 1)의 슬라이딩 윈도우 방식의 패턴 매치보다 스캐닝 성능이 뛰어남을 알 수 있다.



(그림 2) 점핑 윈도우 방식의 패턴 매치 (m=4)

(그림 2)와 같이 하나의 패턴이 페이로드의 연속되는 점핑 윈도우에 걸쳐 있는 경우, 각각의 점핑 윈도우로 해당되는 패턴의 부분 패턴들을 검색해 낼 수 있어야 하나의 패턴이 매치된다. 즉, 패턴이 페이로드내에 존재할 수 있는 모든 위치의 패턴을 TCAM 엔트리로 생성할 경우, 패턴이 페이로드의 임의의 위치에 존재하더라도 점핑 윈도우 방식을 이용하여 패턴 매치를 수행할 수 있다. 페이로드의 길이가 n 바이트인 패킷의 페이로드 $T = t_0,t_1,\cdots,t_{n-1}$ 이라 가정하고, 패턴 $P = p_0,p_1,\cdots, p_{m-1}$ 라고 가정하자. 이 때 P는 다음과 같은 T의 서브스트링 위치에 존재할 수 있다.

$$t_s \cdots t_{s+m-1} = p_0 \cdots p_{m-1},\ 0 \le s \le n-m$$

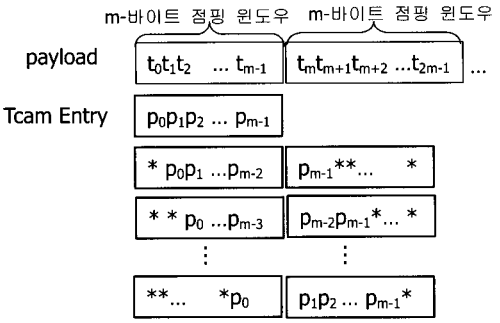
m-바이트의 점핑 윈도우를 이용하면 페이로드는 다음과 같은 점핑 윈도우 서브스트링으로 구분된다.

$$t_{sm} \cdots t_{(s+1)m-1}, 0 \le s \le \left\lceil \frac{n-m}{m} \right\rceil$$

s번째 점핑 윈도우 서브스트링에서 P는 다음과 같이 서브스트링의 i번째 위치에 $m-i$ 바이트가 일치할 수 있다.

$$t_{sm+i} \cdots t_{(s+1)m-1} = p_0 \cdots p_{m-1-i},\ i=0,1,2,\cdots,m-1$$

이 때 나머지 i 바이트는 (s+1)번째 점핑 윈도우 서브스트링에 이어서 나타나야 P가 T 내에 존재함을 의미한다. 따라서 P가 T에서 검색되기 위해서는 (그림 3)과 같이 P를 임의의 m-바이트 점핑 윈도우 서브스트링내에서 0~$(m-1)$만큼 오른쪽으로 시프트한 곳에 위치할 수 있도록 P로부터 TCAM 엔트리를 생성하면 된다.

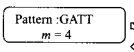


(그림 3) (0~m-1) shift-right TCAM 엔트리 생성

4-바이트 점핑 윈도우를 사용할 때 "GATT" 패턴이 페이로드 상에 나타날 수 있는 위치는, 패턴을 0~3만큼 오른쪽으로 시프트한 "GATT", "-GATT", "- -GATT", "- - -GATT"중 하나일 수 있다. 이렇게 파생된 패턴을 TCAM 엔트리로 생성하면 페이로드에서 m 바이트당 한 번의 TCAM 룩업을 수행할 수 있다. (그림 4)에서처럼 1이상 시프트한 패턴은 그 길이가 m 바이트를 초과하므로 m 바이트의 서브패턴으로 잘라서 TCAM 엔트리를 생성한다. (그림 4)의 "GATT" 패턴은 m=4이므로 시프트한 패턴을 각각 4바이트로 쪼갠 서브 패턴으로 만들어 TCAM 엔트리를 생성한다. 이 때 m 바이트씩 쪼개어진 서브 패턴들은 한 패턴에서 파생한 서브 패턴이므로 패턴이 매치될려면 모든 서브 패턴들이 연속적으로 일치되어야 한다. 따라서 모든 서브 패턴들을 연속적으로 검색하기 위해서는 이전의 서브 패턴에 대한 정보를 연속되는 TCAM 룩업에 참조할 수 있어야 한다. 이를 위해 이전의 서브 패턴 룩업시 서브 패턴에 대한 정보를 해시값(associated hash)으로 만들어 TCAM 룩업 리턴 결과 데이타에 저장하고, 연속되는 TCAM 룩업 시에 이전 TCAM 룩업 결과 데이터인 해시값을 TCAM 룩업 키로 포함시킨다. TCAM 룩업 키는 (그림 4)에서 보이는 것과 같이 (해시값, 서브 패턴)으로 구성된다.

(그림 4)는 m=4일 때 "GATT" 패턴에서 생성된 TCAM 엔트리들이 서브 패턴들의 연속성을 지원하기 위하여 해시를 이용해 구축된 TCAM 엔트리 및 결과 데이터 테이블을 보여준다. 첫 엔트리인 "<hash0>GATT"의 해시값인 <hash0>은 이전 서브 패턴이 없으므로 0으로 설정된다. "<hash0>GATT"가 매치될 때 리턴되는 결과데이타(Result data)의 해시값(associated hash)은 서브 패턴의 끝임을 의미하는 0이다. 이는 "GATT" 서브 패턴이 패턴의 처음이자 마지막 서브 패턴이므로 더 이상 연속되는 서브 패턴이 존재하지 않기 때문이다. "<hash0>-GAT"가 매치될 때 리턴되는 해시값은 hash(<hash0>, "-GAT")를 계산하여 <hash1>로 설정한다. <hash1>은 이어지는 서브 패턴인

"T- - -"를 검색할 때 "-GAT" 서브 패턴의 연속되는 서브 패턴임을 나타내기 위해 "T- - -"의 검색키로 사용되어, 서브 패턴 "T- - -"를 검색하기 위한 TCAM 엔트리는 "<hash1>T- - -"가 된다. 이 서브 패턴은 패턴의 마지막 서브 패턴이므로 더 이상 연속되는 서브 패턴이 존재하지 않는다. 따라서 "<hash1>T- - -"가 매치될 때 리턴되는 해시값을 0으로 설정한다. 즉, TCAM 룩업시 일치되는 엔트리가 존재하여 리턴되는 해시값이 0이면 하나의 패턴으로부터 분리된 모든 서브 패턴이 매치되어 하나의 패턴이 매치되었음을 의미한다.



Key & Mask		Result data
Hash	Sub_pattern	Associated hash
<hash0>	GATT	0
<hash0>	- GAT	<hash1>=hash(<hash0>, "GAT")
<hash1>	T - - -	0
<hash0>	- - GA	<hash2>=hash(<hash0>, "GA")
<hash2>	TT- -	0
<hash0>	- - - G	<hash3>=hash(<hash0>, "G")
<hash3>	ATT-	0
...	...	...

(그림 4) 해시를 이용한 TCAM 엔트리 생성

본 논문에서 제안하는 고성능 패턴 매치 알고리즘은 전처리 단계와 스캐닝 단계로 구성된다. 전처리 단계에서는 일련의 시그너처에서 패턴들을 추출하여 TCAM 엔트리 및 규칙 테이블을 생성한다. 스캐닝 단계에서는 페이로드를 TCAM에 매치시키고, 일치하는 패턴에 대해 매칭테이블을 생성한다.

3.1 전처리 단계

전처리 단계는 패턴으로부터 TCAM 엔트리를 생성하는 것과 규칙테이블을 생성하는 역할을 담당한다. 전처리 단계의 알고리즘을 설명하기 위해 <표 1>과 같은 문자 표기를 사용한다. 전처리 단계에서는 일련의 규칙에서 패턴을 추출하여 TCAM 엔트리를 생성한다. 알고리즘 1은 패턴 P에 대해 TCAM 엔트리를 생성해 내는 알고리즘이다.

〈표 1〉 전처리 단계의 문자 표기

m	Window size
Pi	i-right shifted pattern ($0 \le i \le m-1$)
Sj	Sub-patterns which are derived from P_i, the size of S_j is equal to m ($j \ge 1$)
Hj	Hashing with concatenation H_{j-1} of S_j hash($H_{j-1}S_j$), H_0= 0
Kj	key which is derived from S_j (= $H_{j-1}S_j$)

하나의 시그너처는 여러 개의 패턴으로 구성될 수 있으므로 시그너처를 구성하는 모든 패턴들이 일치될 때만 해당 시그너처가 일치되는 것으로 볼 수 있다. 이를 위해 결과 데이터 테이블에 규칙식별자 필드와 패턴식별자 필드를 추가하였다. 이 필드들은 결과 데이터의 associated hash가 0인 경우, 즉 패턴이 일치하는 경우에만 추가된다. 전처리 단계에서는 TCAM 엔트리를 생성할 때 규칙테이블도 함께 생성한다. 규칙테이블은 각 규칙식별자별로 패턴식별자 리스트를 가진다.

Algorithm 1. 패턴 P의 TCAM 엔트리 생성 알고리즘

```
for each pattern
  for each i-shifted pattern P_i
    begin
      if(length (P_i ) <= m) then
        begin
          S_1 ← P_i(m - length (P_i ))
          create key H_0S_1 and associated hash 0
        end
      else
        begin
          left_length ← length (P_j ),   j ← 1
          while(left_length > m) do
              begin
                fetch m-byte S_j
                calculate hash H_j ← hash(H_j-1, S_k)
                create key H_j-1S_j and associated hash H_j
                left_length ← left_length – m
                j ← j+1
              end
          create key H_j-1S_j and associated hash 0
        end
    end
```

3.2 스캐닝 단계

전처리 단계에서 TCAM 엔트리들이 생성되면 스캐닝 단계에서 패킷의 페이로드 T에 일치하는 패턴이 있는지 알아내는 작업을 수행한다. 스캐닝 단계의 알고리즘을 설명하기 위해 <표 2>와 같은 문자 표기를 사용하고, 알고리즘 2는 스캐닝 알고리즘을 나타낸다.

m-바이트 점핑 윈도우를 사용할 때, 스캐닝 단계에서는 입력 패킷의 페이로드를 m 바이트씩 읽어 해시값과 결합시켜 TCAM 룩업을 위한 검색키를 생성한다. 이때 해시값은 해시리스트인 H[]에 저장된 값을 이용한다. 입력 패킷의 페이로드 m 바이트에 대해 해시리스트 H[]에 존재하는 해시값 수만큼의 검색키를 생성한 후, 각 검색키에 대해 TCAM 룩업을 수행한다. 입력 패킷의 페이로드의 첫번째 점핑 윈도우의 m 바이트는 해시리스트 H[0]를 가지게 되며, 이 값은 초기 해시값이므로 0으로 설정된다. TCAM 룩업의 수행 결과 매치되는 서브 패턴이 존재하는 경우, 해당 서브 패턴이 마지막 서브 패턴이 아닐 때에는 (즉, associated hash가 0이 아닌 경우) 계속적으로 서브 패턴을 더 검색할 필요가 있으므로 결과 데이터의 해시값을 넥스트 해시리스트인

〈표 2〉 스캐닝 단계의 문자 표기

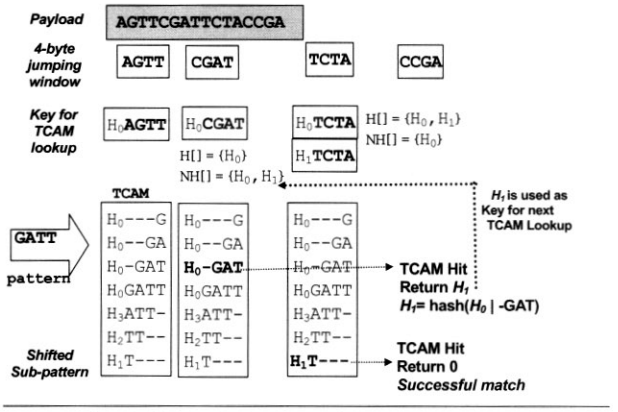
T	The total payload of a packet
$t_i \cdots t_j$	A continuous byte segments belong to a single packet payload T
n	Length of T
$H[]$	Hash list
$NH[]$	Next hash list
associ_hash	Hash from result data when TCAM lookup is successful

Algorithm 2. Scanning algorithm

```
left_length ← n, i ← 1
H[0] ← 0, NH[0] ← 0
while(left_length > 0) do
  begin
    fetch m-byte t_i...t_i+m-1
    j ← 0, k←1
    while(H[j] is not empty) do
      begin
        key ← make_key(H[j]t_i...t_i+m-1)
        lookup_tcam(key)
        if(match) then
          begin
            if(end-of-subpattern) then
              return SUCCESS
            else        { need continuous match }
              NH[k++] ← associ_hash
          end
        j← j+1
      end
    move(H, NH)
    NH[0] ← 0
    i ← i+m, left_length ← left_length – m
  end
```

NH[]에 추가한다. 즉, 입력 패킷의 페이로드의 i번째 점핑 윈도우에 대해 TCAM 룩업을 수행하여 패턴의 마지막 서브 패턴이 아닌 서브 패턴이 매치되는 경우 결과 데이터의 해시값은 넥스트 해시리스트 NH[]에 추가된다. 넥스트 해시리스트 NH[]는 입력 패킷의 페이로드의 $(i+1)$번째 점핑 윈도우의 TCAM 룩업 수행시 해시리스트 H[]로 사용된다. 반면에 TCAM 룩업의 수행 결과 매치되는 서브 패턴이 마지막 서브 패턴일 때 (즉, associated hash가 0인 경우) 패턴 매치는 성공적으로 끝나게 되고, 이 때 결과 데이터에서 규칙식별자와 패턴식별자를 가져와 매칭테이블을 생성한다. 매칭테이블은 페이로드에서 매치되는 다중 패턴 정보를 유지하며, 패턴이 속하는 규칙식별자 정보를 함께 가지고 있으므로 입력 패킷의 페이로드의 패턴 매치 작업이 종료한 후, 규칙테이블과 매칭테이블을 이용하여 다중 패턴을 만족시키는 규칙을 찾아낼 수 있다. 매칭테이블은 각 패턴식별자별로 규칙식별자 리스트를 가진다.

점핑 윈도우 패턴 매치 방식을 이용하여 스캐닝 과정을 수행하는 것을 예를 들어 설명한다. 패턴 "GATT"로부터 (그림 5)와 같이 TCAM 엔트리들을 생성할 수 있다. (그림 5)의 예제에서는 4-바이트 점핑 윈도우를 사용하므로 입력 패킷의 페이로드로부터 4바이트의 문자를 읽어와 초기 해시값 H_0과 함께 TCAM 룩업을 위한 검색키를 생성한다. (그림 5)에서 페이로드에서 "CGAT"를 읽어와 H_0를 연결시켜 "H_0CGAT" 검색키를 생성하여 TCAM 룩업을 수행했을 때, 시프트 서브 패턴 "H_0*GAT"에 매치되어 associated hash H_1가 리턴된다. 이것은 연속적인 서브 패턴이 있다는 의미이므로 페이로드에서 다음 4-바이트 점핑 윈도우를 이용하여 "TCTA"를 읽어와 TCAM엔트리에 "*GATTCTA" 라는 패턴이 있는지 검색하기 위해, "TCTA"와 H_1으로 검색키 "H_1TCTA"를 생성한다. 이는 시프트 서브 패턴 "H_1T***"에 매치되며 마지막 서브 패턴이므로 associated hash는 0이 리턴되어 패턴 매치가 성공적으로 이루어진다.



(그림 5) 점핑 윈도우를 이용한 패턴 매치 예

4. 알고리즘 성능 분석

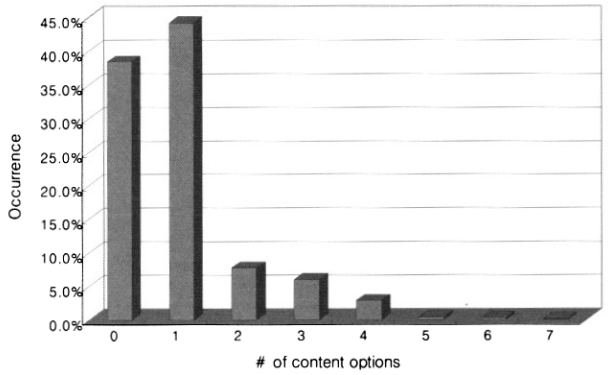
4.1 시그너처 분석

Snort[13]는 가장 대표적인 공개 침입탐지시스템이다. 국외에서도 Snort를 이용한 침입탐지 결과를 SANS 등에 알려 공격동향 정보를 공유하는 경우가 많다. Snort는 패킷 수집 라이브러리인 libpcap[14]에 기반한 네트워크 스니퍼인데, 쉽게 정의할 수 있는 침입탐지 규칙들에 일치되는 네트워크 트래픽을 감시하고 기록하고 경고할 수 있는 도구이다. (그림 6)은 Snort의 규칙을 표시한 예이다. Snort의 규칙은 헤더 부분과 옵션 부분으로 구별되는데 헤더 부분은 액션과 프로토콜, 소스 IP, 소스 포트, 목적지 IP, 목적지 포트로 구성된다. 헤더 부분 다음에 오는 ( ) 부분이 옵션 부분이다. 옵션 부분 중 'content:'라는 옵션이 패킷의 페이로드에서 검색해야 할 패턴에 해당된다.

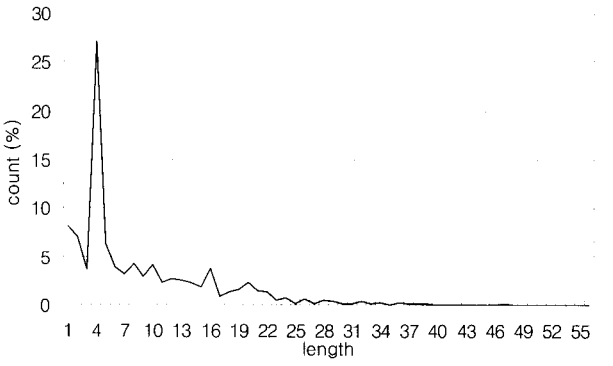
```
alert tcp $HOME_NET any -> $EXTERNAL_NET any
(msg:"ATTACK-RESPONSES directory listing"; flow:
 from_server,established; content:"Volume Serial Number";
 classtype:bad-unknown; sid:1292; rev:8;)
```

(그림 6) Snort 규칙 예

(그림 7)은 Snort 2.1.0[15]에 존재하는 2,394개의 규칙에서 'content' 옵션에 대한 분포도를 보여 준다. (그림 7)를 살펴보면 Snort 규칙에서 하나 이상의 'content'옵션을 가지는 규칙은 전체 규칙의 61.6%에 달한다. 이것은 'uricontent' 옵션을 제외한 부분으로 이를 포함한다면 페이로드를 검색해야 할 규칙은 93%에 달한다. 'content' 옵션이 존재하는 규칙은 하나 이상의 다중 패턴으로 구성된다. 'content' 옵션이 존재하는 규칙 중 싱글 패턴의 비율은 71.6%이고 둘 이상의 패턴을 가지는 다중 패턴의 비율은 28.4%를 차지한다. 다중 패턴을 가지는 규칙은 패킷의 페이로드에서 모든 패턴이 일치가 될 때 해당 규칙에 부합된다. (그림 8)은 Snort 규칙의 'content' 옵션에 대한 패턴 길이별 분포도를 나타낸다. 패턴 길이의 평균 길이는 약 9바이트이다.



(그림 7) Snort 규칙의 'content' 옵션 분포도



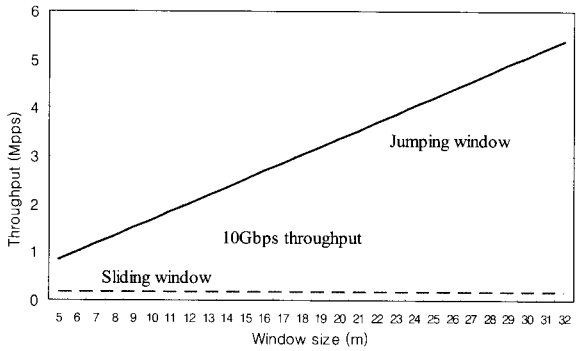
(그림 8) Snort 규칙의 'content' 옵션 길이별 분포도

4.2 성능 분석

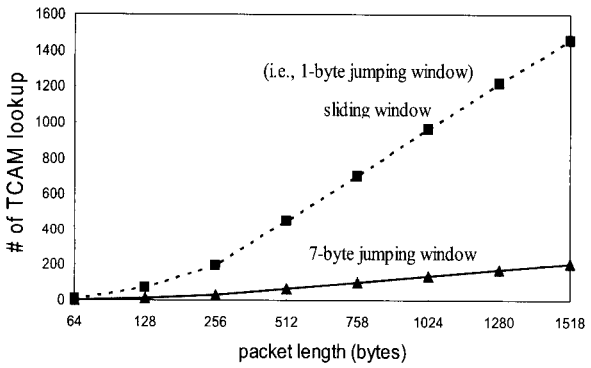
TCAM 룩업 시간을 t, 패킷의 평균 페이로드 길이를 n, 한 패턴의 평균 길이를 l 이라고 가정하자. 슬라이딩 윈도우 패턴 매치 방식의 페이로드 스캐닝 속도는 nt이다. 반면 본 논문에서 제안하는 m-바이트 점핑 윈도우 패턴 매치 방식은 TCAM 룩업 당 m 바이트의 문자열을 페이로드에서 처리하므로 페이로드당 '패킷의 페이로드 길이/m' 만큼의 TCAM 룩업이 수행된다. 따라서 페이로드 스캐닝 속도는 $(n / m)t$이다. n은 패킷의 평균 페이로드이므로 네트워크 트래픽에 따라 가변적이고, t는 TCAM 룩업 시간이므로 윈도우 크기인 m이 페이로드 스캐닝 속도에 영향을 줄 수 있다. m이 커지면 그만큼 패턴 검색 횟수가 줄어들어 페이로드의 스캐닝 속도는 빨라진다. 따라서 네트워크 대역폭 성능을 지원하기 위해서는 검색 시간이 빠른 TCAM 지원과 더불어 점핑 윈도우 사이즈를 크게 설정하면 된다. 그러나 m이 커지면 TCAM의 엔트리 수가 증가한다는 단점이 있다. 아래 수식은 패턴 당 생성되는 TCAM의 엔트리 수(e)를 계산하는 식이다.

$$e = m(l/m+1) + (l\%m - 1)$$

윈도우 크기인 m 변화에 따른 점핑 윈도우와 슬라이딩 윈도우의 성능 비교를 (그림 9)에서 볼 수 있다. (그림 9)는 TCAM 룩업을 가장 많이 요구하는 최장 길이인 1,518바이트 패킷(최대 1,472바이트 페이로드)에 대해 윈도우 크기별 성능을 분석한 것이다. 10Gbps 성능을 내기 위해서는 초당 약 810,000개의 1,518 바이트 패킷을 처리해야 한다(0.81Mpps). 슬라이딩 윈도우 방식은 m의 크기에 상관없이 초당 약 170,000개의 1,518바이트 패킷을 처리한다(0.17Mpps). 반면에 본 논문의 점핑 윈도우 방식은 10Gbps 성능을 지원할 수 있으며, m이 증가할수록 스캐닝 속도가 줄어들게 되므로 점점 성능이 증가함을 볼 수 있다. (그림 10)은 슬라이딩 윈도우 방식과 m=7일 때의 점핑 윈도우 방식에 대해 패킷 길이별로 TCAM 룩업 횟수를 비교한 것이다. 슬라이딩 윈도우 방식에서는 패킷 길이가 커질수록 TCAM 룩업 횟수가 급격하게 증가함을 볼 수 있고, 반면에 점핑 윈도우 방식은 패킷 길이가 커져도 TCAM 룩업 횟수가 서서히 증가함을 볼 수 있다.



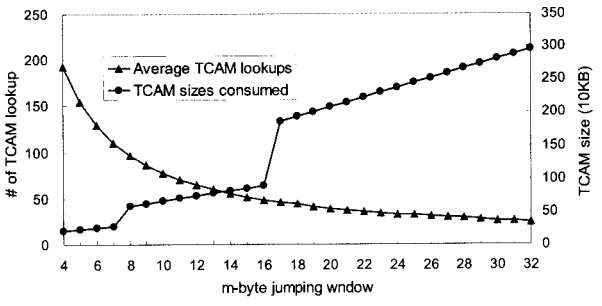
(그림 9) 윈도우 크기별 패킷 처리 성능



(그림 10) 점핑 윈도우와 슬라이딩 윈도우의 TCAM 액세스 비교

5. 시뮬레이션 결과

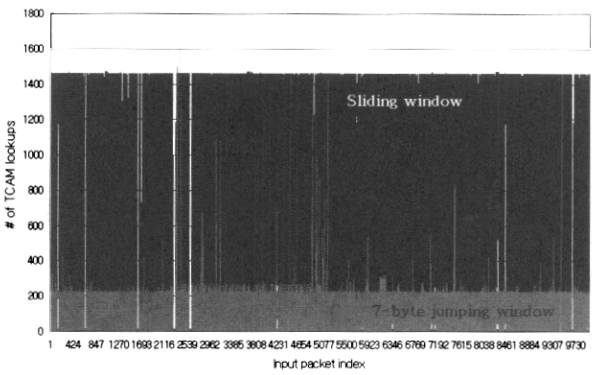
본 논문에서 제안하는 알고리즘이 패킷의 페이로드를 올바르게 필터링하는 지 검증하고, Snort 시그너처를 모두 처리했을 때 점핑 윈도우 크기에 따른 TCAM 엔트리수를 측정하기 위해 다음과 같은 방식으로 시뮬레이션을 해보았다. Snort 2.1.0에 존재하는 모든 규칙들을 파싱하여 이 중 'content'라는 옵션으로부터 2,247개의 패턴을 캐치하여 TCAM 엔트리 테이블을 생성하였다. 입력 패킷은 libpcap를 이용하여 수집한 평균 페이로드 길이가 690바이트인 10,000개의 패킷을 대상으로 하였다.



(그림 11) m-바이트 점핑 윈도우의 TCAM 룩업 횟수 및 TCAM 크기

(그림 11)은 10,000개의 패킷을 대상으로 하여 패킷의 페이로드에 대해 점핑 윈도우 패턴 매치를 수행했을 때 윈도우 크기인 m에 따라 평균 TCAM 검색 횟수를 보여준다. 패킷의 평균 페이로드 길이가 690바이트이므로 4.2절에서 설명한 바와 같이 한 패킷당 $690/m$ 정도의 TCAM 룩업이 수행됨을 시험 결과에서 볼 수 있다. (그림 11)의 시험 결과에서 m의 크기가 커질수록 페이로드당 평균 TCAM 검색 횟수는 현저히 줄어들어 성능이 증가함을 알 수 있다. 반면에 한 패턴당 0~(m-1)개의 시프트 패턴이 생성되므로 m의 크기가 커질수록 TCAM의 엔트리 수는 증가함을 볼 수 있다.

TCAM은 1Mb, 2Mb, 4.5Mb, 9Mb, 18Mb 등의 크기를 지원하는데, 만약 9Mbit 크기를 가지는 TCAM의 경우, 128K x 72bit 개의 키와 매스크 셀로 구성된다. 셀의 수는 TCAM 검색 길이에 따라 달라질 수 있으며 36, 72, 144, 288-bit으로 검색할 수 있다. 즉, 9Mbit TCAM인 경우, 72bit 로 검색할 때 128K개의 TCAM 엔트리를 둘 수 있고, 144bit로 검색할 때에는 64K개의 TCAM 엔트리를 둘 수 있다. 따라서 점핑 윈도우 크기 m이 4~7일 때에는 TCAM 길이가 72bit, m이 8~16일 때에는 TCAM 길이가 144bit, m이 17~34일 때에는 TCAM 길이가 288bit로 동작하므로 (그림 11)에서 점핑 윈도우 크기가 8, 17일 때 TCAM의 크기가 급격히 증가한다. (그림 11)에서 살펴보면, 본 논문에서 제안하는 점핑 윈도우 패턴 매치 알고리즘은 9Mbit TCAM에서 16-바이트 점핑 윈도우를 사용하면 TCAM 엔트리도 모두 수용 가능하면서 최대 성능을 낼 수 있다.



(그림 12) 실제 패킷에 대한 TCAM 룩업 결과 시뮬레이션

10,000개의 입력 패킷에 대해 슬라이딩 윈도우 패턴 매치와 7-바이트 점핑 윈도우 패턴 매치 방식에 대한 TCAM 룩업 횟수를 (그림 12)에 나타내었다. 슬라이딩 윈도우 방식의 각 패킷에 대한 TCAM 룩업 결과는 (그림 10)에서의 분석 결과와 동일함을 알 수 있다. 그러나 7-바이트 점핑 윈도우 방식에서는 다소의 패킷들에 대한 TCAM 룩업 횟수가 (그림 10)에서 분석한 것보다 높게 나타남을 볼 수 있다. 패킷 내에서 임의의 점핑 윈도우의 문자열이 TCAM 엔트리에 일치되면 다음 점핑 윈도우의 문자열에 대한 TCAM 검색은 이전 점핑 윈도우의 문자열으로부터 연속되는 문자열을 대상으로 이루어진다. 이 또한 TCAM 엔트리와 일치될 때 다음 점핑 윈도우의 문자열에 대한 TCAM 검색은 제일 처음 매치된 점핑 윈도우의 문자열으로부터 현재 점핑 윈도우까지의 문자열에 대한 TCAM 검색이 수행된다. 그러나 하나의 패턴으로부터 생성된 모든 서브 패턴이 일치되지 않는 경우에는, 제일 처음 일치된 점핑 윈도우를 제외한 나머지 점핑 윈도우들은 해당 문자열을 처음으로 하여 일치되는 패턴이 TCAM에 존재하는 지를 검색해야 하므로 다소의 패킷에 대해 TCAM 룩업 횟수가 4.2절에서 분석한 것보다 약간 높게 나왔다.

6. 결 론

본 논문에서는 네트워크 침입 탐지, 계층 7 스위치, 바이러스 탐지 등의 기술에 이용될 수 있는 네트워크 트래픽의 페이로드를 검색하여 패턴 매치를 지원하는 수기가급의 스캐닝 성능의 고성능 패턴 매치 알고리즘에 관하여 논하였다. 본 논문에서 제안하는 고성능 패턴 매치 알고리즘은 TCAM을 기반으로 하고, 윈도우 크기가 다양하게 변할 수 있는 점핑 윈도우 패턴 매치 기법을 이용하여 패킷의 페이로드당 TCAM 룩업 횟수를 줄여 페이로드 스캐닝 성능을 높였다. 본 논문에서 제안한 점핑 윈도우 패턴 매치 기법은 패킷의 페이로드에서 일정 길이의 문자열을 한번의 TCAM 룩업 수행으로 처리하는 것으로 이는 시프트 패턴들의 생성과 해시 함수를 통하여 지원된다. 본 논문의 알고리즘은 다중 패턴 매치를 지원할 뿐만 아니라 규칙테이블과 매칭테이블을 생성하여 하나의 규칙에 포함되어 있는 모든 패턴을 검색해 낼 수 있었다.

5장의 시뮬레이션에서는 본 논문에서 제안하는 알고리즘이 TCAM에 저장된 시그너처를 입력 패킷에서 제대로 검출해 내는지의 여부와 점핑 윈도우 크기에 따른 TCAM 크기와 각 패킷의 TCAM 룩업 횟수에 대해 알아보았다. 본 논문의 알고리즘을 10Gbps를 지원하는 보안 라우터 등에 이용하기 위해서는 TCAM을 이용하는 라우터 등에 직접 구현하여 그 성능을 평가하는 작업이 필요하다. 당 연구실에서는 10Gbps 보안 라우터를 개발하기 위하여 현재 인텔의 IXDP2850[4] 플랫폼을 이용하여 제안한 알고리즘을 구현하고 있으며, 이를 통해 10Gbps 보안 라우터에서 본 논문에서 제안한 점핑 윈도우 패턴 매치 알고리즘의 실제 스캐닝 성능을 측정할 수 있다. 현재 IXDP2850 플랫폼을 이용하여

Snort 규칙으로부터 일부 패턴을 캐치하여 TCAM 엔트리 테이블을 생성하였고, 패킷 처리 엔진에 제안한 알고리즘을 적용한 후 스캐닝 성능을 측정한 결과 1Gbps 정도의 라인 속도를 보이고 있다. 이는 단일 패킷 처리 엔진에서 싱글 쓰래드를 적용하였기 때문이며 또한 구현된 마이크로 코드의 최적화가 이루어지지 않은 상태에서의 성능이기 때문에, 멀티 패킷 처리 엔진 사용 및 멀티 쓰래드 방식 적용, 코드 최적화 등을 적용하면 몇 배의 성능 향상이 예측되는 바, 본 논문에서 제안한 알고리즘이 수기가급 보안 라우터에 적합할 것으로 보인다.

참 고 문 헌

[1] 최현희, 정태명, "통합 보안 관리 시스템을 위한 보안 정책 일반화에 관한 연구," 정보처리학회논문지C, 제9-C권 제6호, pp.823-830, 2002.

[2] 장윤정, "차세대 네트워크 통합보안 시장 현황," 네트워크 타임즈, pp.151-165, June, 2004.

[3] P. Jungck and S. S.Y. Shim, "Issues in high-speed internet security," IEEE Computer Magazine, pp.22-28, July, 2004.

[4] M. Adiletta, et. al, "The Next Generation of Intel IXP Network Processors," Interl Technology Journal, Vol. 6, Issue 3, pp.6-18, Aug., 2002.

[5] M. Fisk and G. Varghese, "Fast content-based packet handling for intrusion detection," UCSD Technical Report CS2001-0670, May, 2001.

[6] S. Wu and U. Manber, "A fast algorithm for multi-pattern searching," Tech. Report TR94-17, University of Arizona, May, 1994.

[7] J. Bo and L. Bin, "High-speed discrete content sensitive pattern match algorithm for deep packet filtering," Int'l Conference on Computer Networks and Mobile Computing, 2003.

[8] F. Yu, R. H. Katz and T. V. Lakshman, "Gigabit rate packet pattern-matching using TCAM," IEEE Int'l Conference on Network Protocols, pp.174-183, Oct., 2004.

[9] R. S. Boyer and J. S. Moore, "A fast string searching algorithm," Communications of the ACM, Vol. 20, No. 10, pp.762-772, Oct., 1977.

[10] R. N. Horspool, "Practical fast searching in stgrings," Software Practice and Experience, Vol.10, No.6, pp.501-506, 1980.

[11] Y. Huang, P. Zhang, S. Li, Y. Chen, and D. Zhang, "Research on distributed real time network information auditing system," Int'l Conference on Information, Communications & Signal Processing, 2001.

[12] eSafe Gateway, URL:http://www.eAladdin.com/eSafe

[13] M. Roesch, "Snort - lightweight intrusion detection for networks," Systems Administration Conference, USENIX, 1999.

[14] libpcap, URL: http://ee.lbl.gov/.

[15] Snort.org, URL: http://www.snort.org.\

성 정 식

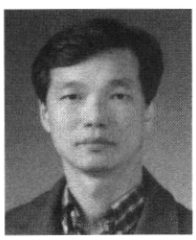
e-mail : jssung@etri.re.kr
1992년 부산대학교 컴퓨터공학과(학사)
1994년 부산대학교 대학원 컴퓨터공학과 (공학석사)
1994년~현재 한국전자통신연구원 선임연구원
관심분야 : 광대역 네트워크 서비스, 통신시스템, 인터넷 보안, 통방융합 등

강 석 민

e-mail : esemkang@gmail.com
1999년 충남대학교 컴퓨터공학과(학사)
2002년 충남대학교 대학원 컴퓨터공학과 (공학석사)
2002년~2003년 한국전자통신연구원, 계약직연구원
2004년~현재 충남대학교 컴퓨터공학과 대학원 박사과정
관심분야 : 초고속 인터넷, 통신 시스템 등

이 영 석

e-mail : lee@cnu.ac.kr
1995년 서울대학교 컴퓨터공학과(학사)
1997년 서울대학교 대학원 컴퓨터공학과 (공학석사)
2002년 서울대학교 대학원 컴퓨터공학부 (공학박사)
2002년~2003년 University of California, Davis 방문연구원
2003년~현재 충남대학교 전기정보통신공학부 전임강사
관심분야 : 차세대 인터넷, 트래픽 엔지니어링, 인터넷 트래픽 측정 및 분석 등



권 택 근

e-mail : tgkwon@cnu.ac.kr
1988년 서울대학교 컴퓨터공학과(학사)
1990년 서울대학교 대학원 컴퓨터공학과 (공학석사)
1996년 서울대학교 대학원 컴퓨터공학과 (공학박사)
1992년~1998년 LG전자 정보통신연구소 연구원
1993년~1994년 미국 Washington University, 방문연구원
1998년~현재 충남대학교, 부교수
2002년~2003년 미국 Erlang Technology, 초빙연구원
관심분야 : 초고속 인터넷, 통신 시스템, 인터넷 보안 등

김 봉 태

e-mail : bkim@etri.re.kr
1983년 서울대학교 전자공학과(학사)
1991년 NCSU 컴퓨터공학과(공학석사)
1995년 NCSU 컴퓨터공학과(공학박사)
1983년~현재 한국전자통신연구원 광통신연구센터장
관심분야 : 컴퓨터 네트워크, 광통신, 광대역 네트워크 서비스