GPGPU 를 이용한 네트워크 트래픽에서의 HTTP 패킷 추출 성능 향상

한상운*, 김효곤**
*고려대학교 컴퓨터정보통신대학원
**고려대학교 정보통신대학 컴퓨터학과
e-mail : system_maker@hanmail.com, hyogon@korea.ac.kr

Performance Improvement in HTTP Packet Extraction from Network Traffic using GPGPU

SangWoon Han*, Hyogon Kim**
*Graduate School of Computer Information & Communication, Korea University
**Dept. of Computer Science and Engineering, Korea University

요 약

웹 서비스를 대상으로 하는 DDoS(Distributed Denial-of-Service) 공격 또는 유해 트래픽 유입을 탐지 또는 차단하기 위한 목적으로 HTTP(Hypertext Transfer Protocol) 트래픽을 실시간으로 분석하는 기능은 거의 모든 네트워크 트래픽 보안 솔루션들이 탑재하고 있는 필수적인 요소이다. 하지만, HTTP 트래픽의 실시간 데이터 측정 양이 시간이 지날수록 기하급수적으로 증가함에 따라, HTTP 트래픽을 실시간 패킷 단위로 분석한다는 것에 대한 성능 부담감은 날로 커지고 있는 실정이다. 이제는 응용 어플리케이션 차원에서는 성능에 대한 부담감을 해소할 수 없기 때문에 고비용의 소프트웨어 가속기나 하드웨어에 의존적인 전용 장비를 탑재하여 해결하려는 시도가 대부분이다. 본 논문에서는 현재 대부분의 PC 에 탑재되어 있는 그래픽 카드의 GPU(Graphics Processing Units)를 범용적으로 활용하고자 하는 GPGPU(General-Purpose computation on Graphics Processing Units)의 연구에 힘입어, NVIDIA 사의 CUDA(Compute Unified Device Architecture)를 사용하여 네트워크 트래픽에서 HTTP 패킷 추출 성능을 응용 어플리케이션 차원에서 향상시켜 보고자 하였다. HTTP 패킷 추출 연산만을 기준으로 GPU 의 연산속도는 CPU 에 비해 10 배 이상의 높은 성능을 얻을 수 있었다.

1. 서론

인터넷에서의 웹 서비스는 매우 개방적이며 쉬운 접근 방식을 지향하는 서비스 형태이다. 반대로 이러한 웹 서비스의 특징은 불순한 의도를 가진 사람들이 웹 서비스에 장애를 일으키기 위한 행위를 손쉽게 할 수 있는 구조적인 취약성을 지니고 있다. 가장 대표적인 시도가 DDoS(Distributed Denial-of-Service) 공격이라고 할 수 있는데, 이러한 공격성이나 유해성 인터넷 트래픽을 탐지 또는 차단하기 위한 방편으로 많은 기관들이 네트워크 트래픽 보안 시스템을 도입하고 있다. 특히, 대부분의 웹 서비스 형태가 HTTP(Hypertext Transfer Protocol)을 사용하기 때문에 HTTP 트래픽을 실시간 패킷 단위로 분석하는 능력은 시스템의 도입을 결정짓는 중요한 요소이다[1].

하지만, 인터넷에서의 HTTP 트래픽 양은 시간이 지남에 따라 기하급수적으로 증가하고 있기 때문에 HTTP 트래픽을 실시간 패킷 단위로 분석한다는 부담감은 그대로 성능 문제로 연결될 수 밖에 없다. 이러한 성능 문제를 해결하고자 대부분의 수많은 솔루션 제작 업체들은 응용 어플리케이션 차원이 아닌 고가의 소프트웨어 가속기나 하드웨어에 의존적인 전용 장비를 탑재하기에 이른다.

본 논문에서는 이러한 현상에 근거하여 대부분의 PC 사용자들에게 친숙한 그래픽 카드의 GPU(Graphics Processing Units)를 범용적으로 사용하고자 하는 GPGPU(General-Purpose computation on Graphics Processing Units)의 목적에 따라, GPU 를 이용하여 네트워크 트래픽에서 HTTP 패킷을 추출하는 프로그램을 구현함으로써 응용 어플리케이션 차원에서 성능을 향상시켜 보고자 하였다.

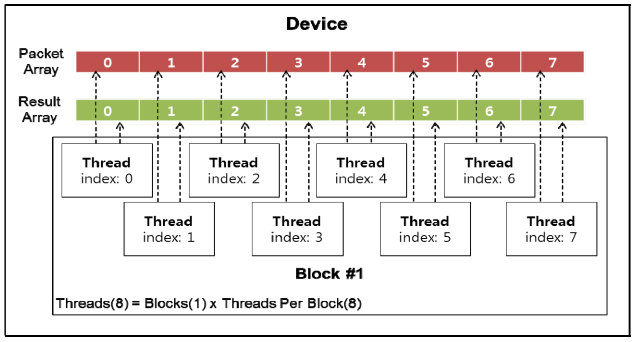
프로그램 개발은 NVIDIA 사의 CUDA(Compute Unified Device Architecture)를 사용하여 C 언어로 구현하였으며, GPU 를 사용한 것과 사용하지 않은 것 두 개의 프로그램을 구현하여 성능을 비교하였다.

2. GPGPU 를 이용한 HTTP 패킷 추출 기능 구현

2.1 프로그램의 기본 구성

우선, 네트워크 트래픽 데이터를 패킷 단위로 나누어 호스트 PC 에 정해진 크기만큼 패킷들을 배열화하여 연속된 공간에 적재할 수 있도록 메모리를 할당한다. GPU 를 이용하는 시점에는 호스트 메모리에 저장된 패킷 데이터들을 GPU 의 전역 메모리로 복사하게

되는데, 이렇게 패킷 데이터들을 연속된 공간에 적재하게 되면 GPU 에 의해 할당된 다량의 쓰레드는 각각의 고유한 인덱스를 부여 받게 되어 해당 인덱스에 부합하는 메모리 영역에 접근할 수 있다. 만약, 쓰레드의 수를 메모리에 저장된 패킷 데이터의 개수만큼 할당하게 되면 GPU 하나의 쓰레드는 단지 하나의 패킷 데이터만 독립적으로 분석하면 된다.



(그림 1) GPU 에서의 프로그램 구조

(그림 1)을 살펴보면, 패킷 데이터를 GPU 의 전역 메모리에 연속적으로 8 개를 적재하고 GPU 에서 쓰레드를 패킷 데이터 개수만큼 생성하게 되면 각각의 쓰레드는 0 에서 7 까지의 인덱스를 부여 받게 되는데, GPU 의 각 쓰레드는 이 인덱스를 사용하여 전역 메모리에 배열 형태로 저장된 패킷 데이터 하나를 분석하게 되므로 각 쓰레드의 독립성을 확보할 수 있게 된다.

패킷 데이터를 분석한 결과 데이터는 GPU 의 전역 메모리 공간에 배열 형태로 미리 할당된 결과 테이블의 각 쓰레드 고유 인덱스에 부합하는 영역에 저장하게 된다.

2.2 네트워크 트래픽 데이터 제공 방법

프로그램에서 GPU 의 성능을 제대로 살펴보기 위해서는 대용량의 네트워크 트래픽 데이터가 필요하다. 따라서, 패킷 캡쳐 프로그램인 WinPcap[2]을 이용하여 대용량의 트래픽 데이터를 인터넷 네트워크 상에서 캡쳐하여 2GB 정도의 트래픽 데이터를 파일로 저장하였다. 본 논문에서 구현한 프로그램은 실행 초기에 WinPcap 에 의해 저장된 대용량 트래픽 데이터 파일을 읽어 사전 정의된 패킷 개수만큼 트래픽 데이터를 패킷 단위로 호스트 메모리에 적재한다.

2.3 HTTP 패킷 추출 선별 기준 및 데이터 구조

HTTP 패킷 분석은 네트워크 트래픽 보안 분야에서 DDoS 공격에 대한 탐지 목적으로 많이 활용되고 있으며, 특히, HTTP get flooding 공격은 DDoS 공격 형태 중 많은 점유율을 차지하고 있다[1]. 본 논문에서 구현한 프로그램에서는 HTTP get flooding 공격 탐지를 가정하여 선별 기준을 마련하였다.

HTTP 패킷을 분석하여 추출하기 위한 선별 기준들을 편리하게 하나의 묶음으로 설정할 수 있도록 패킷 필터를 (그림 2)와 같이 정의하였다.

```
typedef struct _packetFilter {
    u_char      tcpFlags;
    ip_address  dstIpAddress;
    ip_address  dstIpMask;
    u_short     dstPort;
    char        httpMethod[MAX_HTTP_METHOD_LEN + 1];
    u_int       httpMethodLen;
} packetFilter;
```

(그림 2) 패킷 필터 구조체 정의

(그림 2)의 패킷 필터 구조체는 TCP 프로토콜을 기준으로 정의하였으며, 목적지 IP 주소(dstIpAddress)는 IP 주소의 마스크(dstIpMask)를 사용하여 하나의 목적지 IP 뿐만 아니라 목적지 IP 대역으로 정의할 수 있다. 예를 들면, dstIpMask 필드를 255.255.255.0 으로 지정하여 dstIpAddress 필드에 저장된 목적지 IP 주소를 바탕으로 xxx.xxx.xxx.1 부터 xxx.xxx.xxx.255 까지 범위를 설정할 수 있다. 다음의 (그림 3)은 프로그램에서 설정한 패킷 필터의 내용이다.

```
packetFilter  filter;
filter.tcpFlags = 0x10;
filter.dstIpAddress = xxx.xxx.xxx.xxx;
filter.dstIpMask = 255.255.255.0;
filter.dstPort = 80;
memset(httpMethod, 0, MAX_HTTP_METHOD_LEN + 1);
memcpy(httpMethod, "GET ", 4);
filter.httpMethodLen = 4;
```

(그림 3) 패킷 필터 설정

아래의 (그림 4)는 호스트 메모리 및 GPU 의 전역 메모리에 적재될 패킷의 구조체를 정의한 것이다.

```
typedef struct _packet {
    time_t      readTime;
    u_short     dataLen;
    u_char      data[MAX_PKT_LEN];
} packet;
```

(그림 4) 패킷 구조체 정의

(그림 4)의 패킷 구조체에서 readTime 필드는 패킷이 수집된 시각을 나타내며, 실제로 패킷 데이터는 최대 1500bytes 를 넘지 않도록 data 필드에 저장된다.

사전 정의된 패킷 필터에 의해 실제 패킷이 걸러질 경우, 추출된 HTTP 패킷에서 소스 IP 주소 및 HTTP 의 URI 정보를 가져와 아래의 (그림 5)와 같이 정의된 결과 구조체의 할당 영역에 결과 데이터를 저장한다.

```
typedef struct _result {
    ip_address  srcIpAddress;
    u_char      pathStartIdx;
    u_short     pathLen;
} result;
```

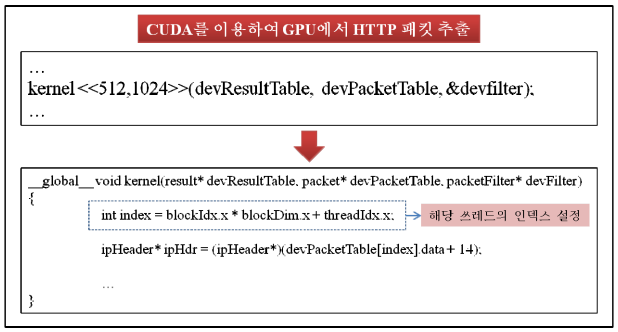
(그림 5) 결과 구조체 정의

(그림 5)의 결과 구조체에서 pathStartIdx 필드는 URI 정보가 포함된 패킷 데이터 내에서 URI 문자열의 시작 인덱스를 나타내며, pathLen 필드는 URI 문자열의 총 길이를 나타낸다. URI 정보 필드를 이러한 형태로 정의한 이유는 이미 패킷 데이터에 URI 문자

열이 포함되어 있기 때문에 URI 문자열 인덱스와 길이를 가지고 URI 정보를 알아낼 수 있기 때문이다.

2.4 CUDA 를 이용한 HTTP 패킷 추출

CUDA 를 이용하여 프로그램을 실행한다라는 의미는 GPU 에 할당된 다수의 쓰레드가 동시에 공통된 연산들의 집합, 즉, 단위의 함수 내에 정의된 연산들을 처리한다라는 의미이다. 프로그램에서 함수 호출 시점에 해당 함수를 사용할 GPU 의 블록 수 및 블록당 쓰레드 수를 지정하게 되면, 지정된 수만큼 GPU 의 쓰레드가 할당되어 함수 내에 정의된 연산들을 동시에 처리하게 되는 것이다.



(그림 6) CUDA 를 이용한 패킷 추출의 구현

(그림 6)은 CUDA 를 이용하여 HTTP 패킷 추출 함수를 호출하는 모습을 보여주는 것으로, GPU 블록의 개수를 512 개, 블록당 쓰레드의 개수를 1024 개로 설정하여 kernel 함수를 호출하고 있다. 각각의 쓰레드는 함수 내에서 저마다의 고유 인덱스 값을 가지게 되며, 이 인덱스 값을 사용하여 devPacketTable 에 적재된 패킷 데이터를 분석한 결과 데이터를 devResultTable 에 저장하게 된다. 즉, 적재된 GPU 에 할당되는 총 쓰레드의 수는 524,288 개에 이르기 때문에 동시에 524,288 개의 패킷 데이터 분석이 가능한 것이다.

(그림 6)에서 kernel 이라는 함수를 호출할 때 넘기는 인자들은 함수 호출 전에 GPU 의 전역 메모리에 적재된 데이터들의 메모리 주소 값이다. 즉, kernel 함수가 GPU 내에서 처리하기 위해 필요한 데이터들은 사전에 GPU 메모리에 적재되어 있어야 한다. 다음의 (그림 7)은 CUDA 를 이용하여 GPU 의 전역 메모리에 데이터를 적재하는 모습이다.

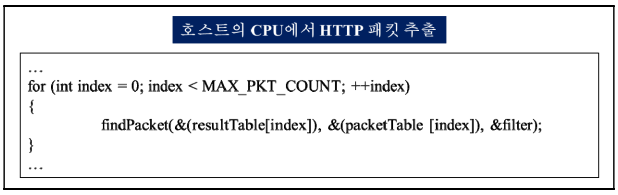
```
/* 결과 데이터 테이블 */
result* devResultTable;
int resultTableSize = sizeof(result) * MAX_PKT_COUNT;
cudaMemalloc(& devResultTable, resultTableSize);
cudaMemset(devResultTable, 0, resultTableSize);

/* 패킷 데이터 테이블 */
packet* devPacketTable;
int packetTableSize = sizeof(packet) * MAX_PKT_COUNT;
cudaMemalloc(&devPacketTable, packetTableSize);
cudaMemcpy(devPacketTable, &hostPacketTable, packetTableSize, cudeMemcpyHostToDevice);

/* 패킷 필터 */
packetFilter* devFilter;
cudaMemalloc(&devFilter, sizeof(packetFilter));
cudaMemcpy(devFilter, &filter, sizeof(packetFilter), cudeMemcpyHostToDevice);
```

(그림 7) CUDA 를 이용한 데이터 적재

CUDA 를 이용한 HTTP 패킷 추출의 성능 향상을 비교 측정하기 위하여, 아래의 (그림 8)과 같이 CUDA 를 사용하지 않은 호스트 프로그램을 작성하였다.



(그림 8) CPU 를 이용한 패킷 추출의 구현

(그림 8)은 호스트에서 하나의 메인 쓰레드가 반복문을 돌면서 호스트 메모리에 적재된 패킷 데이터 배열을 순차적으로 접근하여 패킷을 조사하는 방식으로 구현되었다.

2.5 구현 결과

프로그램은 Microsoft Windows XP(SP3) 운영체제에서 Visual C++ 2008 Express Edition[3]과 NVIDIA CUDA Toolkit 4.0[4]을 설치하여 개발하였으며, 프로그램 성능 측정은 Intel(R) Core(TM)2 6400 @ 2.13GHz 의 CPU 및 NVIDIA 의 Geforce GTX 550 Ti 그래픽 카드가 탑재된 호스트 PC 에서 진행하였다.

다음의 <표 1>은 CPU 와 GPU 의 HTTP 패킷 추출 기능의 속도를 측정하여 비교한 것으로 GPU 에서의 실행시간이 약 10 배 이상 빠른 것으로 나타났다. 이 실행시간 측정은 분석 대상이 되는 패킷 데이터들이 CPU 를 사용할 경우에는 호스트 메모리에, GPU 를 사용할 경우에는 GPU 의 전역 메모리에 모두 적재되어 있는 상태에서 이루어진 것이다.

<표 1> CPU 와 GPU 의 HTTP 패킷 추출 기능 속도 비교

패킷의 총 개수	패킷의 총 데이터량	실행시간	
		CPU	GPU
524,288 개	756MB	63ms	5.72ms
655,360 개	945MB	94ms	6.64ms

3. 관련 연구

3.1 HTTP 헤더 구조

본 논문에서 구현된 HTTP 패킷 추출 프로그램은 아래의 (그림 9)와 같이 HTTP 요청 헤더 정의의 첫 번째 라인을 참고하였다[5].

```
Request-Line    = Method SP Request-URI SP HTTP-Version CRLF
```

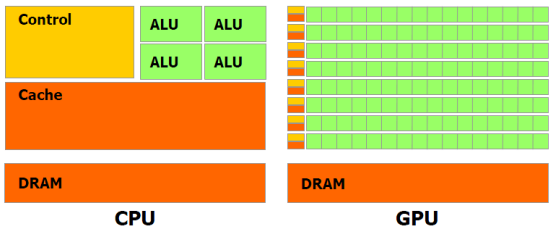
(그림 9) HTTP Request-Line 정의

HTTP 요청 헤더의 가장 첫 번째 요소로 HTTP 메소드가 등장하며 하나의 스페이스 문자 다음에 URI 정보가 나타난다.

본 논문에서 구현한 프로그램에서는 HTTP 요청 패킷 중 GET 메소드를 사용하는 HTTP 패킷을 추출하도록 패킷 필터를 설정하였다.

3.2 CPU 와 GPU 의 구조

CPU 와 GPU 의 구조를 자세히 살펴보면, 각각의 설계를 바탕으로 그 사상의 차이를 엿볼 수 있다. CPU 는 하나의 쓰레드에서 복잡한 제어 논리들을 빠르게 처리하기 위해 설계되었으며, 커다란 캐쉬 메모리를 사용하여 명령어 및 데이터 접근 지연을 줄이고자 하였다. 이와 다르게 GPU 는 대량의 그래픽 데이터를 빠르게 계산하여 처리하는 데에 초점을 두고 있기 때문에, 수치 연산에 최적화된 대량의 코어들을 동시에 사용하여 특정 수치 연산들의 나열을 빠르게 처리할 수 있도록 설계되어 있다. 다음의 (그림 9)는 CPU 와 GPU 의 설계상의 차이를 확실하게 보여주고 있다[6].



(그림 9) CPU 와 GPU 의 구조

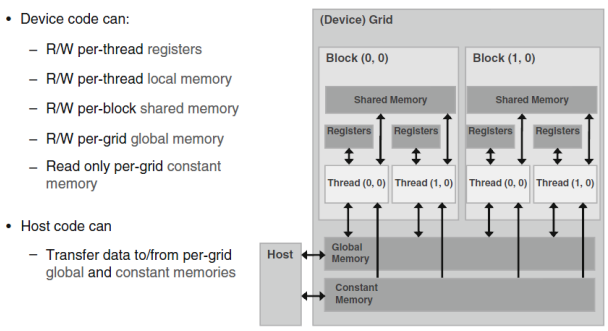
CPU 의 이러한 설계는 순차적으로 복잡한 명령어들을 처리하는 범용적인 용도로는 적합하지만, 단순한 수치 연산들의 나열을 빠르게 처리하는 데에 있어서는 GPU 와 비교하면 성능이 매우 떨어진다.

이러한 GPU 의 특징에 착안하여 빠른 그래픽 처리에 근간을 두고 있는 GPU 를 GPGPU 라는 연구를 통하여 그래픽 분야뿐만 아니라 의학, 과학 등 빠른 수치 연산을 필요로 하는 다양한 분야로 확장하여 활용하고자 하는 움직임이 활발하게 이루어지고 있다.

3.3 CUDA 의 구조

CUDA 는 개발자들이 범용적인 목적으로 GPU 를 사용하여 소프트웨어를 개발하고자 할 때, 그래픽 처리의 내부적인 상황들을 자세히 알지 못해도 어플리케이션 개발 차원에서 매우 친숙한 개발 인터페이스를 제공함으로써 GPU 를 보다 손쉽게 사용할 수 있도록 NVIDIA 사에서 제안한 모델이다[7].

아래의 (그림 10)은 CUDA 가 사용 가능한 GPU 의 메모리 모델을 보여주는 그림으로, CUDA 를 통하여 사용 가능한 메모리 구조를 확인할 수 있다[8].



(그림 10) CUDA 디바이스의 메모리 모델

(그림 10)을 살펴보면, 하나의 그리드는 여러 블록들의 집합으로, 하나의 블록은 여러 쓰레드의 집합으로 구성되어 있다. 각각의 블록 내부에는 같은 블록의 쓰레드들만이 접근할 수 있는 Shared 메모리가 존재하는 반면에, 블록 외부에는 모든 블록의 모든 쓰레드가 접근할 수 있는 읽기/쓰기가 가능한 Global 메모리와 읽기만 가능한 Constant 메모리가 존재한다.

본 논문에서 구현한 프로그램은 실행 초기에 패킷 데이터 전체를 GPU 의 전역 메모리에 적재하게 되는데, 바로 GPU 의 Global 메모리에 적재하는 것이다.

4. 결론

본 논문에서는 네트워크 트래픽 데이터에서 HTTP 패킷 추출의 성능 향상을 목적으로, NVIDIA 사의 CUDA 를 사용하여 기존의 개발 환경과 동일한 개발 환경에서 쉽게 GPU 를 이용한 프로그램을 구현할 수 있었다. 또한, GPU 를 이용한 프로그램의 성능을 비교해 보고자, 호스트 PC 의 CPU 를 이용하는 같은 기능의 프로그램을 구현하여 두 프로그램의 성능을 비교 측정해 본 결과, GPU 를 이용한 프로그램의 실행 속도가 약 10 배 이상 빠른 것으로 나타났다.

이러한 성과를 바탕으로 실제 상용 네트워크 트래픽 보안 솔루션에 GPGPU 를 이용한 HTTP 패킷 추출 기능을 탑재하기 위해서는, (1) 실시간 네트워크 트래픽 데이터를 패킷 단위로 GPU 의 메모리에 빠르게 적재시키는 방법, (2) 패킷 필터링 기능을 보완하여 추출하려는 HTTP 패킷 종류를 다양하게 설정할 수 있는 방법, (3) 솔루션의 목적에 맞게 HTTP 패킷 추출 결과 데이터를 어떤 용도로 활용할지 등의 문제들이 같이 고려되어 적용되어야 할 것이다.

참고문헌

[1] 유승엽, 박동규, 장종수, “URI 및 브라우저 행동 패턴의 특성을 이용한 HTTP get flooding 공격 탐지 알고리즘”, 한국정보기술학회논문지 제 9 권 제 1 호, pp. 159-170, 2011,

[2] WinPcap 4.1.2, http://www.winpcap.org,

[3] Visual C++ 2008 Express Edition with SP1, http://www.microsoft.com/visualstudio/en-us/products/2008-editions/express,

[4] CUDA Toolkit 4.0, http://developer.nvidia.com/cuda-toolkit-40,

[5] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.1”, RFC 2616, June 1999,

[6] NVIDIA Corporation, “NVIDIA CUDA C Programming Guide”, Version 4.0, May 2011,

[7] Jason Sanders, Edward Kandrot, “CUDA by Example: An Introduction to General-Purpose GPU Programming”, 1st Edition, Addison-Wesley Professional, July 2010,

[8] David B. Kirk, Wen-mei W. Hwu, “Programming Massively Parallel Processors: A Hands-on Approach”, 1st Edition, Morgan Kaufmann, February 2010.