리눅스 커널 기반의 네트워크 모니터링 방법론

김동수*, 권윤주**, 정태명***
*성균관대학교 컴퓨터공학과
**한국과학기술정보연구원
***성균관대학교 정보통신공학부
e-mail : dskim96@imtl.skku.ac.kr

Network Monitoring Methodology On The Linux Kernel

Dong-Soo Kim*, Tae-Myung Chung*
* Sungkyunkwan University Department of Computer Engineering
** Korea Institute of Science and Technology Information
***Sungkyunkwan University School of Information and Communication Engineering

요 약

일반적인 네트워크 모니터링은 네트워크상의 패킷을 캡쳐하여 분석함으로써 네트워크의 현재 상태정보 및 네트워크 문제 파악 등에 필요한 여러 가지 정보를 수집한다. 그러나 이러한 모니터링 방법은 패킷을 캡쳐하여 분석하는 데에 시스템 자원을 적지 않게 사용함으로써 모니터링이 시스템 성능에 끼치는 영향이 무시할 수 없는 수준까지 커지는 상황이 발생한다. 본 논문에서는 일반적으로 사용되는 패킷 캡쳐 방법이 아닌 Linux Kernel 레벨에서 패킷 분석을 수행함으로써 네트워크 모니터링을 통하여 미치게 되는 시스템 부하를 줄일수 있는 네트워크 모니터링 시스템을 개발 할 수 있도록 환경을 제공하는 Linux Kernel 기반의 네트워크 모니터링 방법론에 대하여 논한다.

1. 서론

범용 운영체제들이 네트워크를 지원을 기본으로 하는 가운데 운영체제가 설치된 각 네트워크 노드의 네트워크 모니터링의 중요성은 다양한 분야의 분석 기반 자료로써 부각되고 있다.

네트워크 모니터링을 통하여 네트워크 노드 자체의 문제 뿐만 아니라, 해당 노드가 사용하는 어플리케이션 및 전체 네트워크에 대한 문제 해결 등의 기반 자료로 매우 중요한 역할을 한다. 아울러 네트워크 모니터링 자료는 향후 네트워크 자원의 증설 등의 기반 자료를 제공한다.

2. 기존 연구 및 Linux Kernel 기반의 네트워크 모니터링의 필요성

일반적인 네트워크 모니터링은 넓은 의미로는 네트워크상의 네트워크 자원들의 상태 등을 파악하는 것을 말한다. 본 논문에서는 넓은 의미로의 네트워크 모니터링 보다 좁은 의미의 개별 네트워크 노드에서의 모니터링 방법론에 대하여 논한다. 개별 네트워크 노드에서의 네트워크 모니터링은 노드 자신의 네트워크 상태에 대한 것을 모니터링 한다. 모니터링할 대상은 네트워크 트래픽, 에러, 충돌 등 다양한 인자들을 모니터링한다. 이러한 모니터링 인자들 중 트래픽 정보는 개별 네트워크 노드에서의 모니터링 정보 중 가장 다양한 용도로 활용 된다.

트래픽 정보는 기본적으로 개별 네트워크 노드로 유입되는 패킷의 양, 나가는 패킷의 양, 전송률 등을 제공한다. 이는 범용 운영체제에서 기본적으로 제공하며, 손쉽게 확인 할 수 있는 정보이다. 트래픽 모니터링 정보가 사용되는 용도는 주로 개별 네트워크 노드에서 발생되고 유입되는 네트워크 패킷에 대한 측량을 통해 해당 네트워크 노드가 얼만큼의 네트워크 부하를 발생 또는 받고 있는지를 점검하기 위해서이다. 아울러 네트워크 트래픽을 발생시키는 네트워크 어플리케이션에 대한 모니터링을 통해 각 네트워크 어플리케이션의 네트워크 사용률에 대한 정보를 축적하여 해당 어플리케이션의 효율적인 네트워크 사용을 위한

디버깅에 사용된다. 이처럼 세세한 트래픽 정보를 수집하기 위해서는 운영체제에서 기본적으로 제공하는 정보만으로는 불가능하다.

운영체제가 기본적으로 제공하는 트래픽 정보의 부족하기 때문에, 네트워크 어플리케이션 단위 등의 자세한 트래픽 모니터링을 위해서 일반적으로 개별 네트워크 노드로 유입되거나 나가는 패킷을 캡쳐하여 패킷의 헤더를 분석함으로써 프로토콜, 포트 등의 단위로 네트워크 트래픽 정보를 수집할 수 있다. 그러나 패킷을 캡쳐하여 분석하는 과정을 통한 트래픽 모니터링은 패킷을 분석하는 과정에서 시스템의 자원을 사용해야 하며, 네트워크 트래픽의 정도에 따라 네트워크 모니터링 작업이 발생시키는 시스템 자원의 부하가 시스템의 사용에 지장을 초래하는 경우도 발생한다.

패킷 캡쳐를 통한 네트워크 모니터링이 발생시키는 부하 외에도 이와 같은 모니터링 방식이 최적의 모니터링 방식이 아닌 데에는 패킷 캡쳐를 통한 방식이 네트워크 정보를 수집하는 데에 있어서 중복된 작업을 수행하는 이유도 있다. 실제로 일반적인 운영체제는 앞서 언급한 것과 같이 기본적인 네트워크 정보를 제공한다. 이는 개별 네트워크 노드로 유입되거나 나간 패킷이 운영체제 내의 단계적으로 나뉘어진 네트워크 레이어를 거치며 수집된 정보를 나타내는 것이다. 즉, 운영체제 내부적으로는 이미 개별 네트워크 노드로 유입되거나 나간 패킷의 정보를 이미 확보하고 있는 상태이다. 단, 포트 단위등의 세부적인 트래픽 정보 형태로 수집되어 있지는 않다.

이러한 트래픽 정보를 운영체제 내부에서 미리 분류하여 수집할 수 있도록 한다면, 패킷을 캡쳐하여 분석하는 과정에서 발생하는 시스템 자원의 사용을 막을 수 있다.

3. Linux Kernel 의 구조

본 논문에서는 네트워크를 지원하는 범용 운영체제 가운데, 소스 코드가 공개되어 분석 및 수정이 가능한 리눅스의 커널을 기준으로 연구하였다.

커널이란 운영체제의 가장 기초적이고 핵심이 되는 부분으로써 주기억장치에 상주한다. 기억장소, 메모리, 파일, 주변 장치 등과 같은 시스템을 구성하는 중요한 자원을 관리하며, 시간 관리, 프로세서 관리, CPU 스케줄링, 입출력 제어, 시스템 자원의 배분 등과 같이 컴퓨터 운영에 필요한 핵심 사항을 처리한다. 리눅스 커널 내에는 네트워크 지원을 위한 네트워크 아키텍쳐가 구현되어있다. 리눅스 커널의 네트워크 아키텍쳐는 [그림 1] 과 같다.

[그림 1]의 각 레이어는 각기 독립적으로 작동한다.

Socket Interface layer 의 BSD Socket Layer 는 시스템의 인터페이스를 지원하기 위해서 존재한다. 사용자 프로그램은 소켓 라이브러리를 통해서 시스템 콜을 하며, 발생된 시스템 콜은 BSD Socket Layer 로 제어가 넘어간다. 시스템 콜들은 소켓 생성부터 소켓 소멸까지의 과정에 필요한 연산을 지원한다.

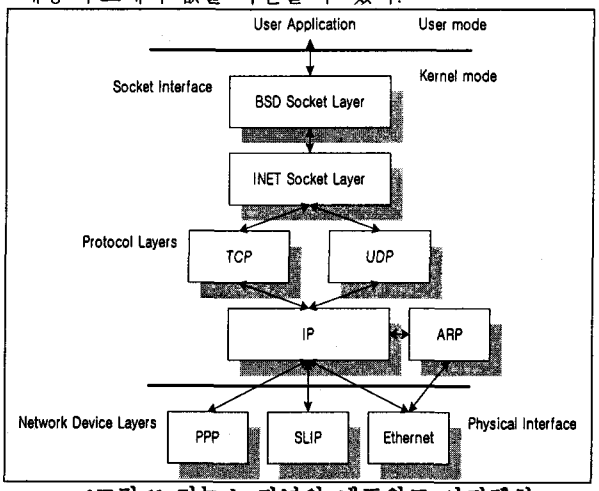
INET Socket Layer 는 IP 에 기반한 TCP 혹은 UDP 프로토콜의 통신을 관리하는 역할을 한다. TCP 와 UDP 에 대해 INET Socket Layer 의 인터페이스가 정의되어 있으며, 이는 BSD Socket Layer 에서 해당하는 연산에 대하여 각각의 소켓 타입에 맞춰서 호출 된다.

Internet protocol layer 의 TCP 와 UDP layer 는 사용자로부터 전달 받은 데이터를 소켓 버퍼(나_buff) 형태로 만들어 하위 IP layer 로 전달하는 역할을 한다.

IP layer 는 상위에서 만들어진 데이터를 보내는 것과 하위 layer 에서 받은 데이터를 상위 layer 로 올려주는 역할을 한다.

Network device layer 는 device driver 와의 인터페이스를 정하여 리눅스의 network device driver 를 일관된 인터페이스로 사용 가능하게 하는 역할을 한다.

이와 같은 각각의 layer 들은 자신을 통과하는 패킷의 헤더 정보를 수집하여 운영체제의 기본 네트워크 정보로 저장한다. 이러한 정보들은 각각의 layer 의 구조체에 들어있으며, 커널 소스코드의 수정을 통하여 해당 구조체의 값을 확인할 수 있다.



[그림 1] 리눅스 커널의 네트워크 아키텍쳐

4. Linux Kernel 기반의 네트워크 모니터링 방법론

리눅스 커널의 수정은 직접 소스 코드를 편집기로 열어 수정하는 방법도 가능하지만, 수정 부분을 쉽게 배포하기 위하여 커널 패치 형식으로 제작이 가능하다. 그러나, 커널의 소스코드 자체에 수정을 가하는 것은 자칫 커널 패닉으로 인한 시스템 가동이 불가해지는 경우가 발생할 수도 있으며, 커널 버전별로 커널 패치를 제작해야 하는 어려움이 따른다.

이러한 어려운 점을 피하기 위하여 본 논문에서는 네트워크 모니터링 기능을 커널 모듈 형식으로 구현함으로써 커널 패치의 위험성을 피하는 방식을 택한다. BSD socket layer 상위에 네트워크 모니터링을 위한 인터페이스를 구현하여 커널 모듈은 해당 인터페이스를 통해 수집한 정보를 user mode 에서 사용할 수 있도록 한다.

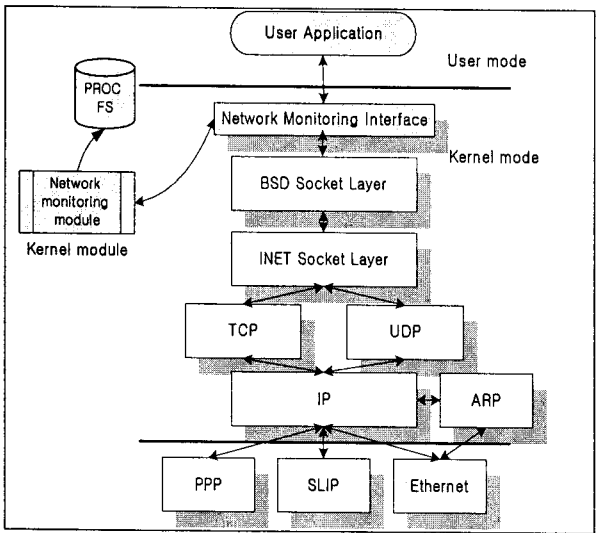
Kernel mode 의 정보를 user mode 에서 사용하기 편하게 제공하기 위하여 리눅스의 proc 파일 시스템을 사용한다.

Proc 파일시스템이란 커널이 가지고 있는 여러가지 데이터 구조체를 시스템 사용자에게 쉽게 전달하기 위해서 사용하는 목적으로 만들어졌다. Proc 파일시스템을 이용하게 됨으로써 보다 쉽게 시스템 정보를 얻어올 수 있으며, 여러 가지 커널 옵션을 특별한 프로그래밍 과정 없이 파일의 정보 변경만을 통해서 쉽게 변경할 수 있도록 해준다.

실제로 proc 파일 시스템을 이용하지 않고 커널 데이터 구조체에서 직접 원하는 시스템 정보를 가져올 수 있기는 하지만, 별도의 프로그래밍 과정을 거쳐야 하고, 복잡한 요건을 만족 시켜야 하는 불편함이 있다. 특별히 성능을 중요시 여기지 않는 경우 proc 파일시스템을 이용하여 정보를 가져오는 것으로도 충분히 기본적인 모니터링을 수행할 수 있으며, 수집된 정보는 시스템 관리와 시스템 최적화를 위한 시스템 성능 분석 등에 유용하게 사용될 수 있다. 리눅스의 경우 다른 운영체제보다 많은 상세 정보들을 proc 파일시스템을 통해서 제공한다.

본 논문에서 제안하는 리눅스 커널 기반의 네트워크 모니터링 방법론에서는 커널에서 수집된 정보를 BSD socket layer 상단에 인터페이스 계층을 두어 이를 통해 커널 모듈이 정보를 수집하고, 사용자가 사용할 수 있도록 데이터를 가공하여 proc 파일 시스템에 저장하는 방식을 사용한다. 이러한 방식을 사용함으로써 네트워크 모니터링 어플리케이션 개발자들에게 복잡한 시스템 콜이나 함수등의 사용을 하지 않아도 되도록 편리한 접근 방식을 제공한다.

리눅스 커널 기반의 네트워크 모니터링은 앞서 언급한 것과 같이 패킷 캡처를 통한 기존의 네트워크 모니터링 방식의 비효율성을 제거함으로써 네트워크 모니터링이 시스템 자원을 사용함으로 인해 다른 프로세스들이 시스템 자원을 사용하는데 영향을 끼치는 것을 방지하는 것이 가장 큰 장점이다.



[그림 2] 리눅스커널 기반의 네트워크 모니터링 아키텍쳐

5. Linux Kernel 기반의 네트워크 모니터링 시스템의 구현

앞서 논한 리눅스 커널 기반의 네트워크 모니터링 방법론에 입각하여 네트워크 모니터링 시스템을 구현하였다. 리눅스 커널 기반의 네트워크 모니터링 시스템은 크게 세 부분으로 나뉜다.

첫째, NAT, firewall 등의 기능을 구현할 수 있는 환경을 제공하는 netfilter 레이어 (네트워크 모니터링 인터페이스 역할)와 이를 후킹하여 정보를 proc 파일시스템에 축적시키는 네트워크 모니터링 모듈이다. 커널 모듈은 커널 버전과는 독립적으로 (단, Linux Kernel 2.4X 에 해당) 작동하는 모듈로써 리눅스 커널은 모듈에 대한 호환성을 자체 지원한다. [그림 3] 은 네트워크 모니터링 커널 모듈의 소스코드 중 일부이다.

네트워크 모니터링 인터페이스 역할을 하는 netfilter 는 Linux kernel 2.4 대에서 커널 기반의 firewall 및 NAT (Network Address Translation) 등을 구현할 수 있도록 해주는 커널 상의 네트워크 레이어이며, BSD Socket Layer 상단에 위치한다. Netfilter 의 사용자 함수 지원을 통해 모니터링할 프로토콜을 임의로 지정할 수 있으며, 각각의 사용자 함수들은 커널의 각 프로토콜 레이어를 후킹 함으로써 . Netfilter 레이어를 이용하여 네트워크 노드 양 종단 간의 트래픽 정보를 모니터링 할 수 있으며, 기본적으로 패킷의 헤더에 대한 정보를 수집할 수 있다. Netfilter 레이어는 커널 configure 시 간단히 활성화 시킬 수 있다. 활성화 후 커널을 재 컴파일 하는 작업을 수행해야 한다.

```
689 #ifdef CONFIG_IP_NF_STATS
690         ct->pkt_count[CTINFO2DIR(ctinfo)]++;
691         ct->byte_count[CTINFO2DIR(ctinfo)] += (*pskb)->len;
692 #endif
693
694         ret = proto->packet(ct, (*pskb)->nh.iph, (*pskb)->len, ctinfo);
695         if (ret == -1) {
696                 /* Invalid */
697                 nf_conntrack_put((*pskb)->nfct);
698                 (*pskb)->nfct = NULL;
699                 return NF_ACCEPT;
700         }
701
702         if (ret != NF_DROP && ct->helper) {
703                 ret = ct->helper->help((*pskb)->nh.iph, (*pskb)->len,
704                                        ct, ctinfo);
705                 if (ret == -1) {
706                         /* Invalid */
707                         nf_conntrack_put((*pskb)->nfct);
708                         (*pskb)->nfct = NULL;
709                         return NF_ACCEPT;
710                 }
711         }
712         if (set_reply)
713                 set_bit(IPS_SEEN_REPLY_BIT, &ct->status);
714
715         return ret;
716 }
717
```

[그림 3] 커널 모듈 소스코드의 예

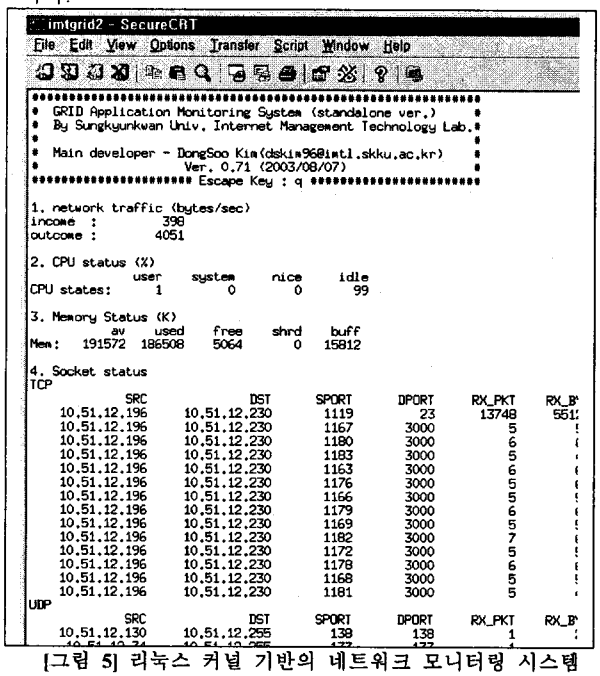
둘째, proc 파일 시스템이다. Proc 파일 시스템에 커널 모듈이 네트워크 모니터링 정보를 저장하게 되는데, 이는 일반 파일 시스템 상의 파일에 쓰기를 하는 것과 같이 정해진 갱신 주기로 파일을 갱신하는 방식이 아닌, 주기억장치 상의 가상 파일 시스템이다. Proc 파일 시스템을 이용함으로써 커널모듈이 kernel mode 에서 수집하는 네트워크 정보를 실시간으로 사용자가 user mode 상에서 쉽게 모니터링 가능하다.

모니터링 값을 확인하는 가장 간단한 방법중 하나가 리눅스의 기본 명령어인 cat (concatenate files and print on the standard output)을 이용하는 방법이다. [그림 4] 는 cat 을 이용하여 proc 파일 시스템 상에 기록된 네트워크 모니터링 정보를 출력한 예이다.

```
udp      17 5 src=10.51.12.130 dst=10.51.12.255 sport=138 dport=138 [UNREPLIED] s
8 use=1 rx=1 229 tx=0 0
udp      17 29 src=10.51.12.137 dst=10.51.12.255 sport=138 dport=138 [UNREPLIED]
38 use=1 rx=1 217 tx=0 0
udp      17 23 src=10.51.12.239 dst=10.51.12.255 sport=138 dport=138 [UNREPLIED]
38 use=1 rx=1 204 tx=0 0
tcp      6 432000 ESTABLISHED src=10.51.12.196 dst=10.51.12.230 sport=4603 dport=
=4603 use=1 rx=139 5647 tx=106 9217
udp      17 29 src=10.51.12.71 dst=10.51.12.255 sport=137 dport=137 [UNREPLIED] s
 use=1 rx=12 936 tx=0 0
udp      17 29 src=10.51.12.78 dst=10.51.12.255 sport=137 dport=137 [UNREPLIED] s
 use=1 rx=8 624 tx=0 0
udp      17 23 src=10.51.12.94 dst=10.51.12.255 sport=137 dport=137 [UNREPLIED] s
 use=1 rx=1 78 tx=0 0
udp      17 24 src=10.51.12.114 dst=10.51.12.255 sport=137 dport=137 [UNREPLIED]
37 use=1 rx=1 78 tx=0 0
udp      17 11 src=10.51.12.183 dst=255.255.255.255 sport=68 dport=67 [UNREPLIED]
t=68 use=1 rx=4 1104 tx=0 0
```

[그림 4] /proc 파일시스템에 기록된 네트워크 모니터링 정보

셋째, 사용자 네트워크 모니터링 어플리케이션이다. 본 논문에서 제시한 리눅스 커널 기반의 네트워크 모니터링 방법론을 이용하는 네트워크 모니터링 어플리케이션은 기존의 패킷 캡쳐를 수행하는 어플리케이션 개발과 같이 복잡한 라이브러리의 사용 등을 하지 않고, 쉽게 /proc 파일 시스템 상에 기록된 네트워크 모니터링 결과 파일을 읽어 들이는 것만으로 네트워크 모니터링 어플리케이션을 구현할 수 있다. [그림 5] 는 리눅스 커널 기반의 네트워크 모니터링 시스템의 예이다.



[그림 5] 리눅스 커널 기반의 네트워크 모니터링 시스템

6. Linux Kernel 기반의 네트워크 모니터링 성능

리눅스 커널 기반의 네트워크 모니터링 시스템의 성능 측정은 네트워크 모니터링 어플리케이션이 사용하는 시스템 자원의 양을 기준으로 측정하였다.

측정 방법은 하나의 개별 네트워크 노드에 커널 기반의 네트워크 모니터링 어플리케이션과 LibPCAP (Packet Capture Library) 를 사용 패킷 캡쳐를 하여 패킷 헤더를 분석하는 네트워크 모니터링 어플리케이션을 동시에 실행하여 top (top CPU processes display application) 을 이용하여 두 프로세스의 시스템 자원 사용량을 비교측정 하였다.

[그림 6] 은 top 을 이용하여 모니터링한 결과이다.

```
  5:38pm  up 13 days,  1:32,  3 users,  load average: 0.78, 0.47, 0.19
55 processes: 51 sleeping, 4 running, 0 zombie, 0 stopped
CPU states:  0.0% user,  0.0% system,  0.0% nice,  0.5% idle
Mem:   191572K av,  184772K used,    6800K free,       0K shrd,   11992K buff
Swap:  393552K av,    1564K used,  391988K free                   65704K cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
22651 root      17   0   552  552   472 R    36.7  0.2   1:10 pcap3
22463 root      15   0   828  828   684 R    22.2  0.4   0:45 in.telnetd
22680 root      12   0  1040 1036   832 R     2.9  0.5   0:00 top
    1 root       8   0   540  488   468 S     0.0  0.2   0:04 init
                        중간 생략
22539 root       9   0  1224 1224   988 S     0.0  0.6   0:00 login
22540 kdsoo      9   0  1296 1296  1004 S     0.0  0.6   0:00 bash
22568 root       9   0  1032 1032   840 S     0.0  0.5   0:00 su
22569 root       9   0  1352 1352  1008 S     0.0  0.7   0:00 bash
22652 root       9   0   540  540   468 S     0.0  0.2   0:00 sysinfo
```

[그림 6] 패킷 캡쳐와 커널 기반 모니터링의 성능 비교

pcap3 프로세스가 LibPCAP 사용 어플리케이션이며, sysinfo 프로세스가 커널 기반의 모니터링 어플리케이션이다. CPU 사용률 측면에서 30 배 이상의 차이가 나는 것을 확인 할 수 있다.

7. 결론

커널 기반의 네트워크 모니터링을 통하여 패킷 캡쳐를 통한 방식보다 시스템 자원 사용량 측면에서 매우 큰 성능 향상을 이룰 수 있다. 반면, 패킷 캡쳐를 통해 수행 할 수 있는 패킷의 payload 부분에 대한 분석은 커널 레벨에서 구현하는 것이 힘들고, 실제 구현하더라도 payload 의 다양한 형식의 분석을 위해 적지 않은 자원을 소비하게 되므로 시스템 성능에 영향을 끼치게 된다.

현재 리눅스 기반의 연구에 이어 향후 다양한 운영체제의 지원을 통하여 커널 기반의 네트워크 모니터링 프레임워크의 구현을 수행할 예정이다.

참고문헌

[1] Linux Kernel 2.4.19 : http://www.kernel.org/pub/linux/kernel/v2.4/linux-2.4.19.tar.gz

[2] BSD Socket layer source code : ~/net/socket.c

[3] INET Socket layer source code : ~/net/ipv4/af_inet.c

[4] TCP source code : ~net/ipv4/tcp_ipv4.c

[5] DANIEL P. BOVET & MARCO CESATI, "Understanding the LINUX KERNEL, 2nd Edition", O'REILLY, December 2002.

[6] Pomerantz, Ori, "Linux Kernel Module Programming Guide", iUniverse.com, 2000.

[7] Jon Crowcroft & Iain Phillips, "TCP/IP and Linux Protocol Implementation", WILEY, 2002.

[8] M Beck, H Bohme, M Dziadzka, U Kunitz, R Magnus, D Verworner, "LINUX Kernel Internals", Addison-wesley, 1996.

[9] Linux Kernel documentation : ~/Documentation

[10] Netfilter project : http://www.netfilter.org

(*주 : "~" 는 Linux kernel 소스 코드 디렉토리임)