임베디드 시스템을 위한 경량의 패킷필터

이 병 권[†] · 전 중 남[††]

요 약

통신 기술과 컴퓨터의 발전으로 임베디드 시스템에 네트워크 통신 인터페이스가 포함되었다. 이로써, 임베디드 시스템에서도 네트워크 기술의 사용으로 보안 이슈가 나타나게 되었다. 이러한 보안문제 해결 방법으로 일반 컴퓨터에서 사용하고 있는 패킷필터를 임베디드 시스템에 적용하는 것이다. 하지만, 호스트용으로 개발된 패킷필터는 기능이 복잡하여 임베디드 시스템에 부적합하다.

본 논문에서는 경량의 임베디드 패킷필터를 제안한다. 경량의 패킷필터는 커널의 코어 수준에서 구현되었다. 그리고 유저가 원격에서 쉽게 보안정책을 세울 수 있도록 Web GUI 인터페이스를 추가하였다. 실험 결과로 제안된 패킷필터는 호스트용으로 설계된 패킷필터보다 패킷 전달 시간이 향상되었고 패킷필터가 포함되지 않은 시스템과 대등한 성능을 보였다.

키워드 : 임베디드 시스템, 패킷필터링, 원격제어, 방화벽

A Lightweight Packet Filter for Embedded System

ByongKwon Lee[†] · Joongnam Jeon[††]

ABSTRACT

The advance of computer and communication technologies enables the embedded systems to be equipped with the network communication interfaces. Their appearance in network leads to security issues on the embedded systems. An easy way to overcome the security problem is to adopt the packet filter that is implemented in the general computer systems. However, general packet filters designed for host computers are not suitable to embedded systems because of their complexity.

In this paper, we propose a lightweight packet filter for embedded systems. The lightweight packet filter is implemented in the kernel code. And we have installed a Web-GUI interface for user to easily set the filtering policies at remote space. The experimental results show that the proposed packet filter decreases the packet delivery time compared to the packet filter designed for host computers and it is comparable to the systems without packet filter.

Key Words : Embedded System, Packet Filtering, Remote Control, Firewall

1. 서 론

인터넷 이용이 확산되면서 홈 네트워크 및 산업 장비의 네트워크 통신사용이 증가하였다. 이로써 언제 어디서나 시스템 감시 및 제어가 쉽게 되었다. 네트워크 통신은 기존의 낮은 속도를 갖는 시리얼 통신의 문제를 해결하는 방법으로 이용되고 있다. 하지만 인터넷의 급속한 발전은 정보보호라는 문제를 야기 시킨다. 각 분야에 네트워크 기능이 임베디드 형태로 포함되면서 컴퓨터 시스템에서만 있었던 보안 문제가 산업 및 가정의 제품까지 영향을 받게 되었다. 홈 네트워크의 발전으로 네트워크 보안은 사생활 보호 입장에서 큰 문제로 제기되고 있다[1, 2].

모바일 및 홈 네트워크 시스템은 임베디드 운영체제를 이용해 시스템을 관리하도록 설계되어 있고 시스템을 구성할 때 운영체제로써 윈도우CE 또는 리눅스를 사용한다[3].

기존의 시스템은 보안 문제를 해결하기 위해 방화벽 기능을 제공한다. 방화벽 기능의 한 예가 패킷필터(packet filter) 기능이다. 하지만 구현된 패킷필터들은 임베디드 시스템의 특성을 고려하지 않고 일반컴퓨터용 패킷필터를 그대로 사용하여 시스템부하를 발생시키고, 필터 모듈의 큰 용량을 차지함은 물론 복잡한 시스템 운영으로 임베디드 시스템에 적당하지 않다. 또한, 호스트용 패킷필터를 그대로 사용함으로써 필요 없는 기능도 필터링 기술에 포함되어 시스템 성능을 저하시키는 요인으로 작용하며 사용자에게 복잡한 인터페이스를 요구해 네트워크 관리자가 아닌 일반사용자가가 방화벽 관리시 문제점을 가지고 있다.

본 연구에서는 임베디드 시스템으로 만들어진 장비 또는 시스템을 외부로부터 보호하기 위한 방화벽 기능을 갖는 경

※ 이 연구에 참여한 연구자(의 일부)는 『2단계 BK21 사업』의 지원비를 받았음.
† 준 회 원 : 충북대학교 전자계산대학원 박사과정
†† 종신회원 : 충북대학교 전기전자컴퓨터공학부 교수
논문접수 : 2006년 6월 16일, 심사완료 : 2006년 8월 25일

량의 패킷필터를 구현하였다. 구현된 경량의 패킷필터는 커널의 TCP/IP 프로토콜 스택 중 네트워크 계층에 직접 구현하고 기존의 호스트 용 패킷필터와 비교하여 성능을 평가하였고, 필터링 기술을 구현하기 위한 운영체제로써 임베디드 리눅스 기반으로 방화벽 기술을 구현하였다. 또한 네트워크 전문 사용자가 아니더라도 일반사용자가 쉽게 방화벽을 만들 수 있도록 필터링 규칙을 적용할 수 있는 GUI 인터페이스를 추가하였다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 구현된 방화벽 기술과 형태에 대하여 기술하고 3장에서는 제안된 경량의 패킷필터 동작방법에 대하여 논한다. 4장에서 구체적인 구현 기술에 대하여 설명하고 5장에서는 기존의 호스트용 패킷필터와 제안된 패킷필터와의 성능을 평가하고, 마지막 6장에서는 결론을 맺는다.

2. 방화벽(Firewall)

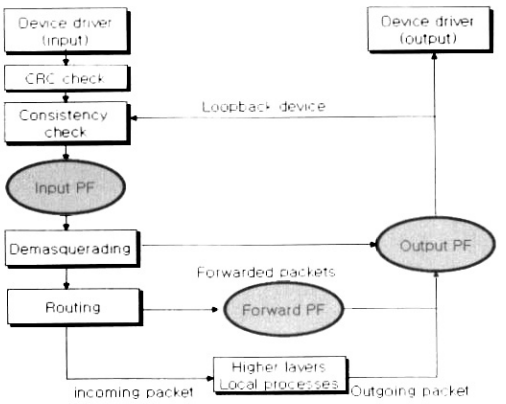
컴퓨터 세계에서의 방화벽은 네트워크의 패킷을 감시하여 잘못된 동작이나 데이터의 이동 경로를 차단하거나 변경하는 방법이다. 이러한 방화벽 기술로 내부 네트워크에 존재하는 시스템이 외부에 공개되지 않도록 할 수 있고 외부 네트워크의 침입을 사전에 막을 수 있다. 이러한 방화벽 형태로 일반적인 것이 패킷필터와 프록시 방화벽이 있다[4, 5].

2.1 패킷필터(Packet Filter)

패킷은 네트워크를 통하여 데이터를 전달하는 가장 기본적인 데이터 전송 단위이다. 패킷필터는 패킷의 헤더를 조사하여 패킷의 운명을 결정하는 기술이다. 패킷필터를 사용하는 목적은 통신되는 데이터에 대한 흐름제어(flow control)와 데이터 보안이다. 흐름제어는 내부 네트워크에서 다른 네트워크로 접속하고자 할 때 데이터 전송 방법이나 데이터 목적지를 결정하는 동작을 수행한다. 데이터 보안은 외부에서 들어오는 허용되지 않은 패킷을 차단하고 내부 네트워크에서 나가는 패킷을 허용하는 기능을 수행한다[6, 7].

리눅스에 포함된 패킷필터는 리눅스 버전 1.1에서 시작되었다. 제 1세대는 BSD의 ipfw를 기본으로 하여 시작되었고, 제2세대는 리눅스 커널 2.0에 필터링 규칙을 제어하는 툴로 'ipfwadm'을 사용하였다. 이어 제 3세대는 리눅스 커널 2.2에 필터링 규칙을 제어하는 툴로 'ipchains'를 사용하였으며, 마지막으로 최근 사용되는 리눅스 커널 2.4에 필터링 규칙을 제어하는 툴로 'iptables'를 사용한다. 리눅스 상의 패킷필터는 모듈로 구현되어 필요시 커널에 포함되거나, 커널 소스 중 네트워크 기능이 구현된 부분에 프로그램 하여 패킷필터 기능을 구현한다. 일반적인 리눅스의 경우 패킷필터는 필터링 동작을 직접 수행하는 커널영역과 커널 영역의 필터링 동작을 관리 감시하기 위한 사용자 영역 어플리케이션으로 구성될 수 있다.

(그림 1)은 리눅스 커널에서 패킷필터의 동작 과정을 보여준다. 패킷의 흐름은 기본적으로 Incoming packet,



(그림 1) 리눅스 커널의 패킷필터

Forwarding packet, Outgoing packet의 3개의 패킷 형태로 구분되어 이동한다. 각각의 패킷의 형태에 따라 패킷필터인 Incoming PF, Forwarding PF, Outgoing PF를 만들어 패킷들의 흐름을 제어한다. 최초에 패킷이 네트워크 디바이스 드라이버에 도착된 이후에 CRC 조사, length 조사와 같은 에러 핸들링을 한다. Incoming PF에 도착한 패킷은 미리 정의된 규칙을 확인하여 내부 네트워크로 들어갈지를 결정하고, 목적지가 내부 네트워크의 특정한 시스템이라면 패킷은 내부로 들어간다. 하지만 패킷의 최종 목적지가 내부에 존재하는 특정 시스템이 아니라면 Forward PF로 전달된다. Forwarding PF에 있는 패킷은 내부 시스템으로 들어가지 않고 다른 네트워크로 전달된다. Outgoing PF는 모든 나가는 패킷들을 조사하는 필터이다. 이곳에 도착하는 패킷들은 내부 시스템에서 들어온 패킷일 수 있고 Forward PF를 통하여 나가는 패킷일 수 있다[8].

또한, 패킷필터의 형태로 정적 패킷필터(static packet filter)와 동적 패킷필터(dynamic packet filter)가 있다. 정적 패킷필터는 패킷의 주소 정보만을 토대로 패킷의 통과 여부를 결정하는 반면 동적 패킷필터는 IP 주소와 포트번호 등과 같은 세션 정보를 기록함으로써, 정적 패킷필터보다 더욱 엄격한 보안 상태를 이행할 수 있다. 동적 패킷필터는 접속 상태를 감시하여 패킷들이 방화벽을 통과하도록 허용할 것인지를 결정하는 방법이다[9]. 동적 패킷필터는 요청과 응답을 추적하여 서로 대조함으로써 요청에 대응되지 않는 응답을 걸러낼 수 있으며 요청이 기록될 때 규칙을 만들어 원하는 패킷의 응답들만이 들어올 수 있도록 허용한다.

2.2 프록시 방화벽(Proxy Firewall)

프록시 방화벽은 패킷이 거쳐 갈 수 있는 프록시 서버를 이용한다. 프록시 서버는 내부 네트워크를 보호하기 위한 필터링 정책을 포함하고 있다. 내부 네트워크에 존재하는 사용자는 직접 외부에 연결하는 것처럼 보이지만 실제적으로 프록시 서버를 통하여 간접으로 통신하는 것이다. 프록시 서버는 웹 접속시 유효 HTTP를 점검하기도하고 버퍼 범람(Buffer Overflow)과 같은 부당 이용 사례를 탐지할 수 있다. 하지만, 모든 프록시들이 똑같이 지능적인 것은 아니

다. 특정 프록시 방화벽은 어플리케이션 프로토콜 계층에서만 작동하며, 프록시가 프로토콜 스택을 점검하고 있을 때 파괴적인 데이터 페이로드가 통과한다. 프록시 서버로 구성된 필터링의 단점은 내부 네트워크에 있는 프록시 서버와 통신하기 위한 특별한 어플리케이션을 설치해 통신해야 한다. 이러한 프록시 서비스를 도와주는 소프트웨어로 SOCKS[10]가 있다.

3. 경량의 패킷필터

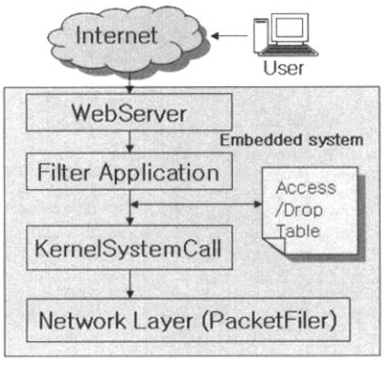
임베디드 시스템에 기존의 호스트용 필터를 그대로 사용하기에는 용량이 크고, 방화벽을 위한 필터링 설정이 힘들다. 또한 호스트용 필터링 시스템은 임베디드 시스템에 불필요한 기능을 포함하고 있으며, 임베디드 시스템을 이용하여 서버로 사용하는 일이 드물기 때문에 서버 시스템에서 사용하듯이 복잡한 규칙을 적용할 필요가 없는 경우가 많다. 대부분의 임베디드 장비들은 특별한 목적으로 사용하므로 단순한 필터링 기능을 이용하는 것이 효과적이다. 또한 최소한의 용량으로 특정 기능을 수행하는 것이 경제적일 것이다.

본 절에서는 경량의 패킷필터 동작 과정과 필터 구현을 위한 TCP/IP 프로토콜 스택에 대하여 알아본다. 또한 필터 동작에 필요한 4가지 동작 모듈에 대한 시스템 콜과 패킷 처리에 대하여 설명한다.

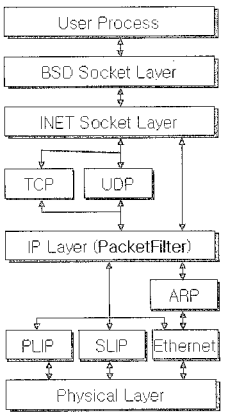
3.1 동작

(그림 2)는 사용자가 인터넷을 통하여 패킷필터가 포함된 임베디드 시스템에 접근하는 과정과 외부 사용자와 시스템과의 연결을 위해 웹서버를 사용하였다. 사용자는 웹서버를 통하여 패킷필터 관련 정보(access IP, drop IP, Filter Start, Filter Stop)를 입력 및 저장할 수 있다. (그림 2)에서 Filter Application은 커널에 시스템 콜 동작을 수행시키는 어플리케이션이다. 허용테이블과 차단테이블을 참조하여 IP 리스트를 커널에 전달한다. 전달 받은 IP 리스트는 네트워크 계층에 구현된 필터에 의해 참조된다.

패킷필터는 TCP/IP 스택이 커널 어디에 구현되어 있는지



(그림 2) 패킷필터 동작



(그림 3) TCP/IP 스택

를 확인해야 커널에 필터링 기능을 추가할 수 있다. TCP/IP 스택은 (그림 3)과 같은 계층 구조를 갖고 있다. 이러한 계층 구조에서 패킷필터 구현 시 어느 계층을 선택하여 프로그램을 하느냐는 결국 방화벽의 성능을 좌우하게 된다. 가장 이상적인 구현위치는 ISO 최하위인 물리 계층에 구현하는 것이겠지만 이더넷 장치마다 서로 다른 사양을 갖고 있으므로 구현은 가능하지만 호환성에서 떨어지는 문제를 가지고 있다. TCP/IP 스택은 리눅스 커널 소스의 "/linux/net/ipv4/"에 구현되어 있다. 네트워크 통신 관련 함수로 ip_rcv(), ip_forward(), ip_queue_ximt(), ip_build_xmit()함수가 있다[11].

본 연구에서는 (그림 3)과 같이 리눅스의 네트워크 TCP/IP 스택 중 네트워크 계층에 제안된 경량의 패킷필터를 구현하였다. ISO 계층 중 3계층에 해당하는 네트워크 계층은 IP주소를 관리 하는 계층으로써 IP계층이라고도 한다. 임베디드 시스템에 이용될 경량의 패킷필터는 TCP/IP 프로토콜 스택중 상위 계층에 해당하는 TCP 나 UDP까지 고려하지 않아도 될 것이다. 만약 구현한다면 포트(Port) 필터를 구현하게 되어 필터에 따른 시스템 부하가 많이 걸린다. 대부분의 임베디드 장비는 특별한 목적으로 사용하므로 단순한 필터링 기능을 이용하는 것이 효과적 일 수 있다.

방화벽을 위한 커널 내의 패킷필터링 작업은 커널과의 동기를 맞추어 동작하도록 구현해야 한다. 동기화가 이루어지지 못하면 커널은 패닉 상태로 들어가 버릴 것이다. 이러한 커널과의 동기화 처리하는 방법이 시스템 프로그램을 통한 커널과의 통신이다. 리눅스 커널의 경우 시스템 프로그램을 위해 커널의 엔트리 테이블과 시스템 콜을 관리하는 함수의 이름 및 번호를 등록해야 한다. 본 연구에서는 패킷필터를 위해 시스템 함수로써 허용테이블(access table)과 차단테이블(drop table)을 리스트로 만들어 관리하도록 하였다. 허용테이블은 외부와 통신이 가능하도록 허가된 IP를 저장하는 테이블이며, 차단테이블은 외부와 연결을 차단하거나 외부에서 들어오는 특정 IP를 차단하는 테이블이다.

3.2 시스템 콜과 패킷 처리

시스템 콜이란 사용자 프로그램에서 운영체제의 기능을

사용할 수 있게 해주는 방법이다. 대부분의 운영체제는 여러 가지 레벨(low level) 연산을 수행하기 위한 루틴들을 가지고 있다. 예를 들어, 하드 디스크에서 파일을 복사하는 루틴이라든가, 랜 카드를 통해 인터넷에서 어떤 데이터를 송수신하는 등의 루틴은 모두 커널에서 제공하는 기능들이다. 만약 사용자 프로그램에서 운영체제에 있는 루틴을 실행시켜 어떠한 결과를 얻기를 원한다면 시스템 콜을 이용해야 한다.

제안된 패킷필터는 커널과 통신하기위해 시스템 콜을 사용하였다. 시스템 콜을 작성하기 위해서는 콜 테이블 및 시스템 함수를 커널에 등록해야 한다. 구현에 관한 구체적인 삽입 과정은 구현부분에서 설명한다. 시스템 콜 처리를 위한 준비가 완료되면 커널 내에서 동작하는 패킷필터 구현이 가능하게 된다. 실제적으로 패킷을 잡아서 처리하는 부분이 네트워크 드라이버 상에 존재한다.

네트워크 드라이버 중 함수 "int netif_rx(struct sk_buff *skb)"이 외부에서 패킷이 들오는 위치에 해당하는 함수로 장치 드라이버와 TCP/IP 스택의 중간에 위치한 함수이다. 모든 외부에서 들어오는 패킷은 이곳을 통과하게 된다. 여기에 패킷을 허가할 것인지 차단할 것인지 구현하면 효과적인 패킷필터가 될 것이다.

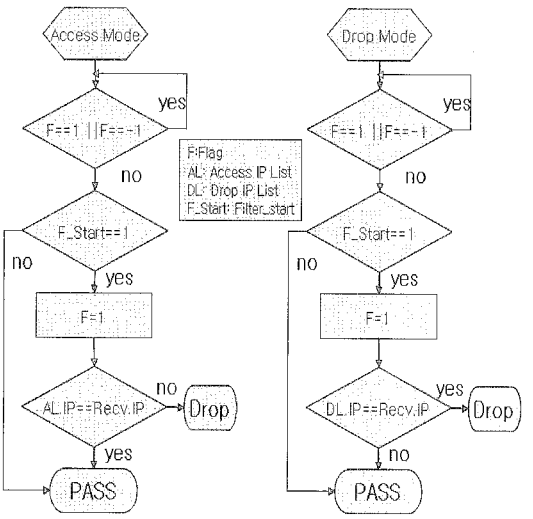
하지만 커널 내의 패킷필터는 커널과 동기화가 이루어져 동작해야 한다. 즉, 허용테이블과 차단테이블에 IP를 추가하거나 삭제시 커널에 NULL 포인터를 전달한다면 커널 패닉 현상이 발생할 수 있기 때문이다. 본 연구에서는 동기화 문제를 해결하는 방법으로 <표 1>과 같은 플래그 비트(0, 1, -1)를 두어 현 상황을 판단해 동기화 부분을 구현하였다.

〈표 1〉 동기화 플래그

Flag = 0 : 삽입, 삭제, 필터링 작업이 완료된 경우
Flag = 1 : 패킷필터링을 수행하고 있는 경우
Flag =-1 : 테이블의 삽입, 삭제, 초기화가 진행 중인 경우이다.

(그림 4)는 커널 내에서 제한된 경량의 패킷필터 동작과정이다. 현재 동작하는 패킷필터는 허가모드(access mode)와 차단모드(drop mode)로 구분하여 동작한다. 허가모드인 경우는 허가테이블에 등록된 패킷에 대하여 접근을 허가하는 방식으로 동작한다. 차단모드인 경우는 차단테이블에 등록된 패킷에 대하여 차단하고 나머지는 접근을 허가한다.

(그림 4)의 액세스 모드인 경우는 플래그 비트를 조사해 삽입, 삭제 동작(F=-1)이거나 필터동작 상태(F=1)라면 동작이 끝날 때까지 루프를 반복한다. 동기화 과정이 지나면 현재 패킷의 동작여부를 알려주는 F_start를 확인하여 필터 수행 여부를 판단하고, 이어 플래그 값을 1(F=1)로 주어 현재 패킷필터중임을 표시한다. 마지막으로 현재 저장된 IP값과 외부에서 들어온 IP를 비교하여 PASS할지 DROP 할지를 결정한다. 마찬가지로 차단모드인 경우는 DROP IP 리스

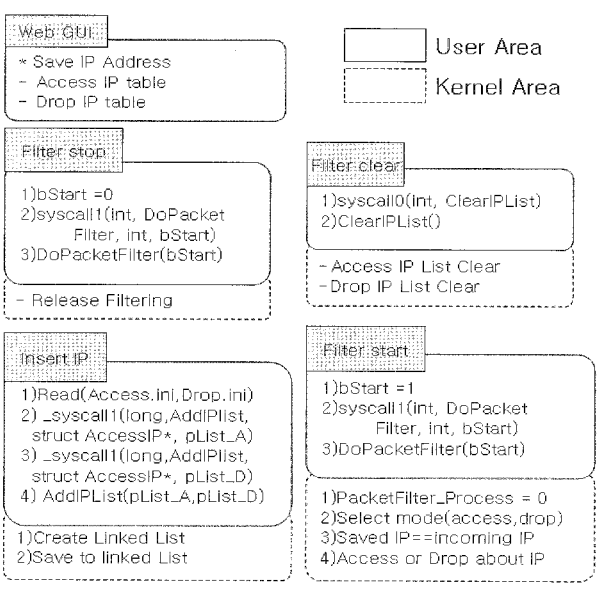


(그림 4) access/drop mode 패킷처리

트를 참조해서 패킷을 PASS할지 DROP 할지를 결정한다.

(그림 5)는 IP입력을 위한 Web GUI, 허용, 차단하려는 IP의 추가, 필터링 기능의 동작, 중지, 패킷리스트의 삭제 동작 방법을 사용자 영역과 커널영역으로 구분하여 도식화 한 것이다.

Web GUI는 웹서버를 이용하여 필터 사용자에게 패킷필터에 사용될 IP를 입력하는 GUI 인터페이스를 제공한다. Insert IP는 웹을 통하여 저장된 IP(access.ini, drop.ini)를 참조하여 시스템 콜을 통한 커널내의 IP리스트에 전달하는 역할을 담당한다. 이렇게 함으로써 패킷필터가 동작시 리스트에 입력된 IP를 참조해 필터기능이 동작된다. Filter start는 필터링 수행을 알리는 플래그를 커널에 전달하여 필터링을 수행한다. Filter stop는 시스템 콜을 통하여 필터링을 멈추는 동작을 한다. Filter clear는 access 리스트와 drop 리스트를 모두 삭제한다.



(그림 5) 패킷필터 동작 모듈

4. 구 현

본 절에서는 제안된 경량의 패킷필터를 구현하기위한 하드웨어 플랫폼 설명과 리눅스 커널 소스에 시스템 콜 등록 과정에 대하여 설명한다. 또한 필터 사용자에게 제공되는 GUI 인터페이스에 대하여 설명한다.

4.1 실험 환경

제안된 필터는 구현을 위한 플랫폼으로 임베디드 리눅스 교육용 보드를 사용하여 구현하였다. 실험에 사용된 코어는 ARM기반의 Intel Xscale로 내부 버스 속도가 400Mhz를 사용한다. 임베디드 리눅스를 포팅하기 위해 x86용 기본 리눅스 커널에 ARM 패치와 XScale 패치 작업과 컴파일 후 임베디드 플랫폼에 포팅 하였다. <표 2>는 패킷필터를 구현 및 실험하기 위한 임베디드 플랫폼의 사양이다[12].

또한 사용자에게 GUI 인터페이스를 제공하기 위해 임베디드 웹서버인 goahead을 설치하여 CGI인터페이스를 사용할 수 있도록 구현하였다.

〈표 2〉 임베디드 플랫폼 사양

	임베디드 플랫폼
CPU	XScale PXA255(ARM) 400Mhz
Kernel version	linux-2.4.19,Arm-patch, Xscale-patch
SDRAM/ROM	16M/16 Bit, 4M/8 Bit
Web server	goahead-2.4.1
	include CGI
Ethernet Port	2 Ports, 10/100M

4.2 시스템 콜 등록

제안된 경량의 패킷필터는 시스템 콜에 등록된 함수로 테이블(access, drop table)을 초기화 하는 DoPacketFilter, 허용 테이블에 IP를 추가하는 함수 AddIPList_A, 테이블에 있는 IP리스트를 모두 초기화하는 함수 ClearIPList, 차단테이블에 IP를 추가하는 함수 AddIPList_D로 구성된다.

▫ 패킷필터를 위한 시스템 콜 이름 및 번호 등록

우선 시스템 프로그램을 위해 첫째로 해야 할 일은 시스템 콜 이름과 번호를 추가하는 것이다. 리눅스 커널이 제공하는 모든 시스템 콜은 각각 고유번호를 갖는다. 따라서 새로운 패킷필터 시스템 콜을 추가하기 위해서는 시스템 콜의 이름과 번호를 등록해야 한다. 일반적으로 x86의 경우 시스템 콜 번호는 "include/asm-i386/unistd.h" 헤더 파일에 구현되어 있다. 하지만 임베디드 플랫폼이 ARM기반이므로 "include/asm-arm/unistd.h"에 패킷필터 이름과 번호를 등록하였다[13]. 표 3은 등록한 시스템 콜 이름과 번호이다.

▫ 패킷필터를 위한 시스템 콜 테이블 등록

시스템 콜이 호출될 때 해당 시스템 콜 함수를 호출하기 위해 시스템 콜 테이블을 수정한다. 시스템 콜 테이블은 x86 기반에서는 "arch/i386/kernel/entry.S"를 수정해야 되지만, 최종적으로 포팅하여 구현될 플랫폼이 ARM이므로 "arch/arm/kernel/calls.S"에 새롭게 추가되는 패킷필터 함수의 엔트리를 <표 4>와 같이 등록하였다. <표 3>과 <표 4>를 거치면 시스템 콜 호출시 해당 시스템 콜 함수로 점프할 수 있다.

〈표 3〉 시스템 콜 이름 및 번호 등록

```
#define __NR_DoPacketFilter   (__NR_SYSCALL_BASE+226)
#define __NR_AddIPList_A      (__NR_SYSCALL_BASE+227)
#define __NR_AddIPList_D      (__NR_SYSCALL_BASE+229)
#define __NR_ClearIPList      (__NR_SYSCALL_BASE+228)
```

〈표 4〉 시스템 콜 테이블 등록

```
.long   SYMBOL_NAME(sys_DoPacketFilter)
.long   SYMBOL_NAME(sys_AddIPList_A)
.long   SYMBOL_NAME(sys_AddIPList_D)
.long   SYMBOL_NAME(sys_ClearIPList)
```

▫ 패킷필터를 위한 함수 구현

리눅스 커널 내에서 패킷필터 기능이 동작해야 하므로 커널 컴파일시 필터 함수를 추가해 컴파일 해야 한다. 이를 위해 <표 5>와 같이 패킷필터의 동작 함수를 "linux/kernel/sys.c"에 등록하여 컴파일 하였다.

〈표 5〉 패킷필터 함수

```
asmlinkage int sys_DoPacketFilter(int bStart){..}
asmlinkage unsigned long sys_AddIPList_D(struct
  PacketFilterList_D *pList_D){..}
asmlinkage unsigned long sys_AddIPList_A(struct
  PacketFilterList_A *pList_A){..}
asmlinkage int sys_ClearIPList(void){..}
```

마지막으로 사용자 영역에서 패킷필터를 동작시키고 IP를 추가 및 삭제하기 위한 사용자 어플리케이션을 <표 6>과 같이 4개의 프로그램을 작성하였다.

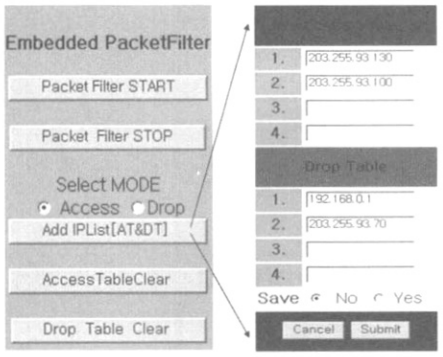
〈표 6〉 사용자 어플리케이션

```
IPADD: 허용 및 차단테이블에 IP를 추가한다.
IPSTART: 패킷필터 동작을 시작 한다.
IPSTOP: 패킷필터 동작을 중지한다.
IPCLEAR: 허용 및 차단테이블에 등록된 내용을 모두 삭제한다.
```

<표 6>의 4개의 어플리케이션은 사용자가 인터넷을 통하여 패킷필터를 제어 가능하도록 Web GUI에 연결하였다.

4.3 사용자 인터페이스

사용자가 원격으로 인터넷을 통하여 제어 가능하도록



(그림 6) Web GUI 인터페이스

web server를 설치하고 (그림 6)과 같이 인터페이스를 구성하였다. 패킷필터의 동작의 여부를 결정하는 START, STOP과 패킷필터의 모드를 선택할 수 있는 인터페이스로 구성 되어있다. 또한, 허가테이블과 차단테이블의 IP를 삭제하는 기능과, 새롭게 차단하거나 허가할 IP를 등록하는 인터페이스로 구성되었다.

구현된 경량의 패킷필터의 성능을 평가하기 위해 기존의 패킷필터 툴인iptables이 포함된 커널과 필터가 포함되지 않은 커널을 작성하여 비교하였다(No Filter kernel, Iptables Filter kernel, Proposed kernel). iptables 툴을 성능비교에 사용된 이유는 현재 iptables 툴이 일반컴퓨터 및 임베디드 시스템 장비에 포팅 되어 있다.

일반 컴퓨터 시스템에 사용되는 iptables툴은 패킷 필터링 기능(netfilter)은 물론 IP공유(MASQUERADE)를 포함하고 있다[14][15]. 그 외에도 IP관리 PORT관리 등의 다양한 기능을 포함하고 있다. 제한된 패킷 필터에 iptables에서 사용하는 많은 기능을 포함하면 좋지만 용량을 줄이고 패킷처리 속도를 높이기 위해서 코어 수준에서 필터 기능만 구현하는데 중점을 두었다. 제안된 필터는 간단한 보안을 요구하는 산업장비나, 가정용 임베디드 기기에 사용할 수 있다.

5. 성능 비교

제안된 패킷필터를 평가하기 위해 호스트용 패킷필터인 iptables를 커널에 포팅하여 커널, 어플리케이션, 라이브러리 크기를 비교하였다. iptables 필터 크기보다 제안된 필터가 약 100k정도 작아 졌다. 호스트 시스템에서는 100k정도는 문제가 되지 않지만 호스트용 필터를 그대로 임베디드 시스템에 포팅하여 사용한다면 100k의 크기가 낭비되어 최적화에 문제가 될 수 있다.

응답속도 실험을 위해 특정 DNS서버(WAN)에 패킷의 크기를 변화시켜 보내고 응답시간을 측정하여 성능을 평가하였다. 실험에 대한 신뢰성을 높이기 위해 같은 플랫폼에 커널만 교체하면서 실험하였고, 데이터를 1000회 보내고 받은 시간을 측정하였다. 평가에 사용된 항목으로 가장 빠른 응답시간(min response time), 평균 응답시간(avg response time)을 평가하였다.

〈표 7〉 MRT(min response time)

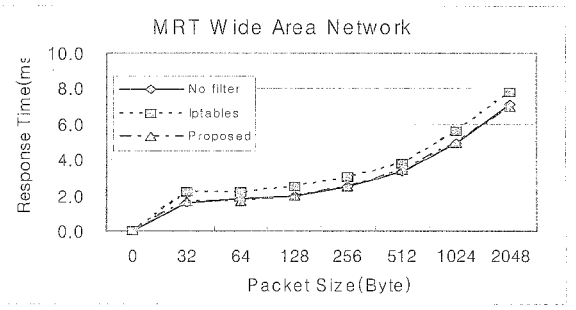
Packet size \ Method	NoFilter	Filter	
		Iptables	Proposed
0	0.0	0.0	0.0
32	1.6	2.2	1.7
64	1.8	2.2	1.7
128	2.0	2.5	2.0
256	2.5	3.0	2.5
512	3.3	3.8	3.4
1024	4.9	5.6	4.9
2048	7.1	7.8	7.0

〈표 8〉 ART(avg response time)

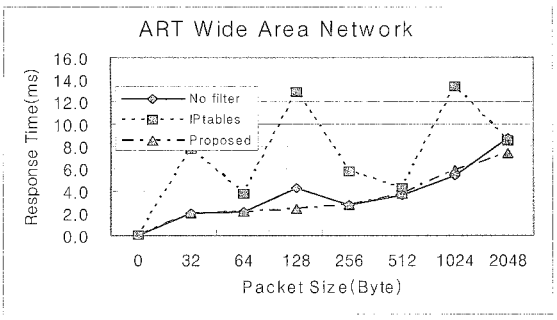
Packet size \ Method	NoFilter	Filter	
		IPtables	Proposed
0	0.0	0.0	0.0
32	2.0	7.8	2.0
64	2.1	3.7	2.1
128	4.2	12.9	2.4
256	2.7	5.8	2.8
512	3.6	4.2	3.8
1024	5.4	13.4	5.9
2048	8.7	8.5	7.4

<표 7>과 <표 8>은 일정한 크기의 패킷을 보내고 최소 응답시간(MRT)과 평균응답시간(ART)을 측정한 값이다. MRT=Min(R(N)), ART=Avg(R(N))으로 계산되었다. R은 응답시간이며, N은 패킷을 보내는 수이다. 평균 응답 시간은 네트워크 상태에 따라 평균값을 높아지게 하거나 낮아지게 할 수 있다. 그러므로 네트워크 상태가 최적의 상태를 갖는 상황에서 실험을 해야 하므로 최소 응답시간과 평균 응답시간을 가지고 성능을 평가하였다.

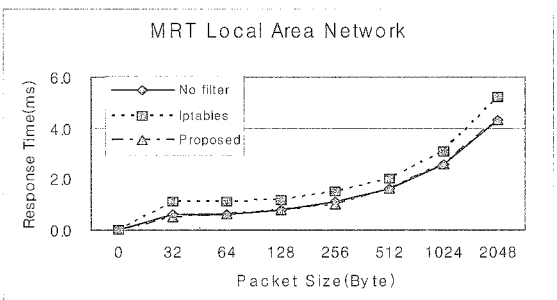
(그림 7~10)은 WAN과 LAN를 구성하여 패킷(32, 64, .. 2048)을 보내어 응답 시간을 도식화 한 것이다. (그림 7)은 WAN를 구성하여 최소 응답시간으로 기존의 호스트용 패킷필터 보다 제안된 필터가 응답 시간에서 빠름을 볼 수 있다. 또한 제안된 필터는 패킷필터가 포함되지 않는 시스템과 비슷한 성능을 보임을 알 수 있다. (그림 8)은 평균 응답 시간을 나타내는 것이다. 평균 응답시간의 경우 급격한 응답시간의 지연은 네트워크 트래픽 지연으로 판단된다. 이러한 네트워크 트래픽 지연을 배제하기위해 LAN를 구성하여 실



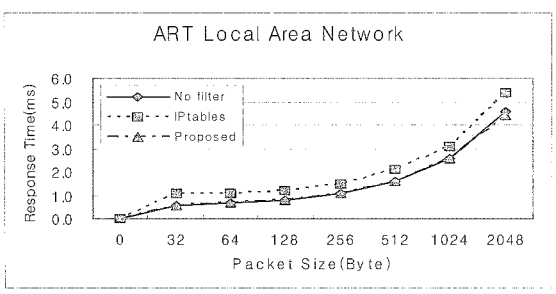
(그림 7) WAN의 최소 응답시간



(그림 8) WAN의 평균 응답시간



(그림 9) LAN의 최소 응답시간



(그림 10) LAN의 평균 응답시간

험하였다. (그림 9, 10)은 LAN을 구성하여 최소 응답시간과 최대 응답시간을 도식화 한 것이다. 제안된 필터는 커널 수준에서 직접 필터를 구현하여 패킷필터가 없는 것과 비슷한 성능으로 동작함을 알 수 있다. 일반 컴퓨터에 사용되는 iptables 패킷필터는 복잡한 필터기능으로 필터하는 동안에 지연으로 응답시간에 영향을 받는다는 것을 확인하였다.

결론적으로 경량의 패킷필터는 기존 일반컴퓨터용 필터보다 용량과 응답시간에서 안정성을 보임은 물론 패킷필터가 없는 것과 거의 같은 성능을 나타냄을 실험으로써 확인하였다.

6. 결 론

임베디드 시스템의 특성은 특정한 목적으로 동작해야 하므로 복잡하거나 많은 부가적인 기능이 필요하지 않다. 간단하면서 적은 용량으로 최대한 효과적으로 동작하도록 구현되어져야 한다. 많은 임베디드 시스템 장비나 기기들은 기존의 일반컴퓨터에서 사용하는 기능을 그대로 답습하여 적용하는 경우가 많다. 이로 인해 필요 없는 기능과 불필요한 용량을 차지하여 효율성과 경제적인 면에서 문제로 나타나고 있다. 또한 점점 더 네트워크 기능이 임베디드 시스템에 포함되면서 보안 문제를 간과하고 넘어 갈 수 없다.

본 연구에서는 기존의 일반컴퓨터용 패킷필터를 임베디드 시스템에 그대로 사용함으로써 발생하는 문제를 해결하기 위해 새로운 경량의 임베디드 패킷필터를 구현하였다. 구현된 패킷필터는 필터링 기능을 포함하면서 필터 기능이 없는 시스템과 성능이 비슷하게 동작함을 확인하였다. 또한 복잡한 필터링 작업을 하는 기존 방식을 사용하지 않고, 웹 서버를 설치하여 Web GUI 사용함으로써 기본적인 네트워크 지식을 가진 누구라도 쉽게 방화벽 정책을 구현할 수 있도록 사용자 인터페이스를 제공하였다. 제한된 시스템에 대한 응용 분야로써 제한 메모리 용량을 갖는 네트워크 장비 또는 임베디드 시스템에 적용 가능하고, 네트워크 통신 기능을 포함하는 산업용 장비에 간단한 방화벽 서비스를 제공할 수 있다. 향후 연구과제로 산업용 장비나 인증된 네트워크 장비 간에 패킷을 인증하여 통신할 수 있는 동적 패킷필터 구현이 필요하다.

참 고 문 헌

[1] William R. Cheswick, Steven M Bellowin, Firewalls and Internet Security, ISBN 0-201-63357.

[2] Home Network Security, http://www.cert.org/tech_tips/home_networks.html.

[3] MobileOS, http://www.tdgresearch.com/pdfs2006/TDG_PR_020706_AdvancedMobileOS.pdf.

[4] Ioannidis, S. Anagnostakis, K.G. Ioannidis, J. Keromytis, A.D., "xPF: packet filtering for low-cost network monitoring," High Performance Switching and Routing, 2002. Merging Optical and IP Technologies. Workshop on, May, 2002.

[5] S.M. Bellovin, W.R. Cheswick, "Network firewalls," *Communications Magazine IEEE*, Vol.32, pp.50-57, Sept., 1994.

[6] J. Reumann, Hani Jamjoom, Kang Shin, "Adaptive packet filters," *Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE, Vol.4*, pp.2331-2335 Nov., 2001.

[7] T. Verwoerd, R. Hunt, "Policy and implementation of an adaptive firewall," *Networks, 2002. ICON 2002. 10th IEEE International Conference*, pp.434-439, Aug., 2002.

[8] Klaus Wehrle and Frank pahlke, The Linux Networking Architecture, *Prentice Hall*, 2005.

[9] H. Julkunen, C.E. Chow, "Enhance network security with dynamic packet filter," *Computer Communications and Networks Proceedings*, pp.268-275, Oct., 1998.

[10] SOCKS[LGLK+96], http://www.ufasoft.com/socks/

[11] Bovet and Cesati, Understanding the Linux Kernel-third edition, O'Relly.

[12] HANBACK ELECTRONICS CO., LTD, http://www.hanback.co.kr/htm/sub2_2.htm.

[13] Karim Yaghmour, Buliding Embedded Linux system, O'Relly, 2003.

[14] R. Zalenski, "Firewall technologies," *Potentials IEEE*, Vol.21, pp.24-29, Mar., 2002.

[15] D.L. Herbert, S.S. Devgan, C. Beane, "Application of network address translation in a local area network," Southeastern Symposium on System Theory, 2001. Proceedings of the 33rd, pp.315-318, Mar., 2001.

이 병 권

e-mail : sonic747@hanmail.net

1999년 한밭대학교 전자계산(학사)

2002년 한남대학교 컴퓨터공학(석사)

2003년 충북대학교 전자계산대학원 박사과정

관심분야 : 임베디드 시스템, 컴퓨터구조, 퍼지이론

전 중 남

e-mail : joongnam@cbu.ac.kr

1981년 연세대학교 전자공학과(학사)

1985년 연세대학교 전자공학과(석사)

1990년 연세대학교 전자공학과(박사)

1990년~현재 충북대학교 전기전자 컴퓨터공학부 교수

관심분야 : 임베디드 시스템, 컴퓨터구조, 병렬처리