웹 기반 원격 제어를 위한 내장형 시스템용 네트워크 모듈 및 웹 서버
(A Network Module and a Web Server for Web-based Remote Control of Embedded Systems)

선 동 국 † 김 성 조 †† 이 재 호 ††† 김 선 자 †††
(Dong Guk Sun) (Sung Jo Kim) (Jae Ho Lee) (Sun Ja Kim)

요 약 정보가전의 원격 제어 및 모니터링을 위해서는 실시간 운영체제와 함께 TCP/IP 프로토콜 같은 네트워크 모듈이 요구된다. 하지만 수백 KByte의 코드 크기와 수십 KByte의 메모리를 요구하는 전통적인 TCP/IP는 8비트 또는 16비트 CPU를 사용하고 상대적으로 메모리 자원이 부족한 소형 정보가전에서 이용하기에는 너무 크고 비효율적이다. 따라서 소형 정보가전에 적합하도록 프로토콜 스택의 크기가 소형화된 마이크로 TCP/IP 프로토콜 스택의 개발이 요구되며, 인터넷을 통하여 이들을 원격으로 제어 및 모니터링하기 위해서는 내장형 웹 서버 및 내장형 CGI기술의 개발 또한 필요하다.

본 연구에서는 웹을 기반으로 정보가전을 원격에서 제어하기 위한 마이크로 TCP/IP프로토콜 및 웹 서버를 구현하였다. 이를 위해서 우선 소형 내장형 시스템과 인터넷의 연동 및 정보가전의 웹 기반 제어에 필요한 요구사항을 조사하였다. 그 다음 구현된 마이크로 TCP/IP 프로토콜 스택 및 소형 웹 서버의 성능 및 목적 코드 크기를 타깃 운영체제인 QPlus의 네트워크 모듈 및 RTIP와 비교하였다. TCP/IP의 목적코드 크기는 RTIP와 QPlus의 네트워크 모듈에 비해 각각 약 2/3와 약 1/4정도 크기로 줄일 수 있었다. 지연확인 응답을 사용하지 않은 경우에 약 2.9Mbps의 속도를 나타내어, 본 연구에서 구현한 모듈의 성능은 RTIP 및 QPlus의 네트워크 모듈과 비슷하였다.

키워드 : 내장형 시스템, 마이크로 TCP/IP

Abstract Remote control and monitoring of information appliances require RTOS and TCP/IP network module to communicate each other. Traditional TCP/IP protocol stacks, however, require relatively large resources to be useful in small 8 or 16-bit systems both in terms of code size and memory usage. It motivates design and implementation of micro TCP/IP that is lightweight for embedded systems. Micro embedded web server is also required to control and monitor information appliances through the Web.

In this paper, we design and implement micro TCP/IP and Web server for information appliances. For this goal, we investigate requirements for the interoperability of embedded systems with the Internet and the Web-based control of embedded systems. Next, we compare our micro TCP/IP protocol stack with that of RTIP and QPlus in terms of object code size and performance. The size of micro TCP/IP protocol stack can be reduced by 3/2 and 1/4, respectively, comparing with that of RTIP and QPlus. We also show that the performance of our micro TCP/IP is similar to that of RTIP and QPlus since it handles 2.9Mbps when delayed ACK is not adapted.

Key words : Embedded System, Micro TCP/IP

· 이 논문은 2003학년도 중앙대학교 학술연구비 지원에 의한 것임
† 비 회 원 : 중앙대학교 컴퓨터공학과
dgsun@konan.cse.cau.ac.kr
†† 종신회원 : 중앙대학교 컴퓨터공학과 교수
sjkim@cau.ac.kr
††† 비 회 원 : 한국전자통신연구원 임베디드S/W 기술센터 연구원
bigleap@etri.re.kr
sunjakim@etri.re.kr
논문접수 : 2003년 10월 16일
심사완료 : 2004년 2월 13일

1. 서 론

최근 들어 독립적으로 사용되고 있는 각종 가전제품을 인터넷과 연결된 홈 서버와 연결함으로써 인터넷, 휴대폰, PDA, IMT-2000 단말기를 이용하여 에어컨과 전자레인지, 홈 시큐리티, 팩스, 디지털TV 등 각종 정보가전 제품을 제어할 수 있는 기술이 활발히 연구되고 있다[1].

웹을 기반으로 하여 이러한 정보가전을 원격으로 제어하기 위해서는 실시간 운영체제와 외부 기기와의 통신을 위한 TCP/IP 프로토콜 같은 네트워크 모듈이 제공되어야 한다. 정보가전과 같은 임베디드 시스템은 일반 데스크 톱 환경과는 달리 프로세서의 성능이 낮으며 적은 메모리를 가지고 있다. 이로 인하여 기존 운영 체제와 네트워크 모듈은 그 크기로 인하여 정보가전과 같은 임베디드 시스템에는 적합하지 않다.

그 동안 해외에서는 실시간 운영체제에 내장되는 TCP/IP 프로토콜과 이를 이용한 응용 프로그램 개발이 활발히 이루어지고 있으며, 특히 TCP/IP 스택을 소형화하는 연구[2-5]도 상당한 진척을 이루고 있다. 국내의 경우, 한국전자통신연구원(ETRI)에 의해 독자적으로 개발된 QPlus[6-7]은 BSD기반의 TCP/IP 스택을 지원하고 있다. 또한, 국내 몇몇 기업들도 독자적인 TCP/IP 스택을 개발하여 시리얼 통신과 접목하여 TCP/IP가 지원되지 않던 기존의 임베디드 시스템에 TCP/IP 프로토콜을 지원[8]하기 위한 노력을 하고 있다.

수백 KByte의 코드 크기와 수십 KByte의 메모리를 요구하는 전통적인 TCP/IP는 8비트 또는 16비트 CPU를 사용하고 상대적으로 메모리 자원이 부족한 소형 정보가전에서 이용하기에는 너무 크고 비효율적이다. 따라서 프로토콜 스택의 크기를 소형화한 소형 정보가전용 마이크로 TCP/IP 프로토콜 스택의 개발이 요구되며, 인터넷을 통하여 이들의 원격 제어 및 모니터링용 내장형 웹 서버 및 내장형 CGI기술의 개발 또한 필요하다.

본 논문의 목적은 인터넷을 통하여 정보가전과 같은 내장형 시스템의 원격 제어 및 모니터링을 하기 위하여 내장형 시스템에 적합한 마이크로 네트워크 모듈 및 내장형 웹 서버를 개발하는 것이다. 이를 위하여 소형 내장형 시스템이 인터넷과 연동되기 위해 필요한 기반 기술을 도출한 다음, 마이크로 TCP/IP의 최신 기술 동향을 조사·분석하고, 이를 기반으로 소형 내장형 시스템에 적합한 마이크로 TCP/IP 프로토콜 스택을 설계 및 구현하였다. 또한 내장형 시스템에서 사용가능한 소형 웹 서버와 CGI 기능을 마이크로 TCP/IP 프로토콜 스택 상에서 구현하였다.

본 논문의 구성은 다음과 같다. 제2절에서는 마이크로 TCP/IP의 기술 동향 및 내장형 웹 서버의 요구사항에 대하여 기술한다. 제3절에서는 본 논문이 구현한 마이크로 TCP/IP 프로토콜과 원격제어를 위한 내장형 웹 서버와 CGI의 구현에 대하여 기술한다. 제4절에서는 본 논문에서 구현한 TCP/IP 프로토콜과 웹 서버의 성능을 분석하고, 마지막으로 제5절에서 결론 및 향후 연구과제에 대하여 기술한다.

2. 기반 연구

2.1 내장형 시스템과 인터넷의 연동

인터넷 정보가전은 유무선 네트워크, 전력선, 위성과 연결된 홈 네트워크를 통해 데이타 송수신이 가능한 디지털 TV, 인터넷 냉장고, DVD, 디지털 비디오 등 지능형 단말을 일컫는다. 인터넷 정보가전 제품들은 홈 네트워크를 통해 서로 연결되어 정보를 전달하고 공유하며, 홈 게이트웨이를 통해 외부 네트워크에 접속된다. 홈 네트워킹 기술은 무선을 이용하는 방법과 기존의 유선 또는 새로운 케이블을 설치하여 사용하는 방법이 있다. 무선을 이용하는 방법으로는 블루투스(Bluetooth), Wireless LAN 802.11b, 802.11a, 802.1g와 UWB가 제안되었으며, 유선을 이용하는 방법으로는 IEEE 1394, 홈 PNA(Phoneline Network Alliance), 전력선 통신(Power Line Communication: PLC)등이 연구되고 있다[9,10]. 이들 정보가전과 같은 내장형 시스템이 인터넷과 연동되기 위해서는 우선적으로 소형 내장형 시스템에 적합한 TCP/IP 프로토콜 스택의 구현이 필수적이다.

내장형 시스템을 인터넷과 연동시키기 위한 방법은 다음과 같이 두 가지로 분류될 수 있다. 첫번째는 내장형 시스템에 운영체제를 이식하지 않고, 응용 프로그램 자체에서 TCP/IP 프로토콜 스택을 처리하는 방법이다. 두번째는 내장형 시스템에 운영체제를 이식하고 운영체제에서 제공하는 TCP/IP 프로토콜 스택을 사용하는 방법이다.

(1) 운영체제가 없는 시스템에서의 TCP/IP

이러한 방법은 주로 장치의 모니터링이나 제어용으로 이용되는 소형 내장형 시스템에서 사용된다. 소형 내장형 시스템은 아주 적은 ROM과 RAM을 가지고 있으므로, 프로토콜 스택의 크기가 제한적일 수밖에 없다. 따라서 이와 같은 시스템에서의 TCP/IP는 시스템에서 실제로 필요로 하는 프로토콜 스택 부분만을 선택적으로 구현하게 된다. 예를 들어, 이러한 소형 내장형 시스템이 인터넷과 연결되어 원격으로 관리되어야 한다면, 장치 드라이버, ARP, IP, ICMP, TCP와 같은 스택이 최소한 구현되어야 한다. 또한, 전송된 데이타를 처리하는 부분이 스택과 함께 하나의 응용 프로그램 형태를 이루어 내장되는 것이 일반적이다. 이와 같은 방식으로 구현된 프로토콜 스택은 고급 기능보다는 패킷 송수신에 중점을 두게 된다. 이러한 방식으로 개발된 대표적 시스템인 ezTCP/Ethernet[8]는 랜 환경에서 내장형 시스템이 직렬 통신(RS-232)을 통해 인터넷과 접속할 수 있는 기능을 지원하고 있다.

(2)실시간 운영체제를 기반으로 하는 TCP/IP

이것은 주로 내장형 리눅스나 상용 RTOS에서 사용

하는 방식이며, 대체로 규모가 큰 내장형 시스템에서 사용된다. 전자의 경우와는 달리 이 프로토콜 스택은 독자적인 기능을 수행하는 여러 개의 태스크들로 이루어져 동작한다. 현재 상용화되어 있는 대표적인 프로토콜 스택으로는 ExpressLogic사의 NETX[2], MicroDigital사의 smxNet[3], CMX Systems사의 CMX-TCPIP[4]과 Blunk Microsystems사의 TargetTCP[5] 등이 있는데, 이들은 대부분 EBSnet사의 RTIP[11]을 기반으로 하여 각 시스템에 최적화되도록 이식하여 사용되고 있다.

2.2 소형 내장형 시스템의 TCP/IP

TCP/IP는 ethernet 기반의 네트워크 상에서 이기종 컴퓨터간의 통신을 위한 표준으로 거의 모든 시스템들이 이용하고 있으며 최근에는 그 적용 분야가 내장형 시스템 환경까지 확대되고 있다. 내장형 시스템이 인터넷과 연동될 수 있도록 많은 종류의 마이크로 TCP/IP들이 그동안 구현되었는데, 이들은 크게 BSD를 기반으로 하는 것과 그렇지 않은 것으로 구분될 수 있다.

BSD TCP/IP는 자원에 대한 제약이 거의 없는 시스템에 적합하도록 설계 및 구현되었다. 따라서, BSD기반의 TCP/IP 프로토콜 스택은 그 크기가 커서 8비트 또는 16비트의 낮은 성능을 가지는 마이크로프로세서를 이용하고, 메모리 상에 제약이 많은 소형 내장형시스템보다는 일반적으로 데스크 톱 PC 이상의 시스템에서 주로 이용된다. 하지만 몇몇 마이크로 TCP/IP들은 BSD TCP/IP를 기반으로 구현되는 경우도 있는데, 그 대표적인 예가 InterNiche사의 NicheStack[12]이다. NicheStack은 BSD TCP/IP의 Mbufs 구조체와 소켓을 이용하면서 프로토콜 스택을 내장형 시스템에 최적화시켜 구현하였는데, 32비트 ARM 프로세서를 사용하는 경우의 코드크기는 대략 50KB 정도이다[12].

BSD를 기반으로 하지 않은 TCP/IP 프로토콜 스택의 경우, 일반적으로 특정 응용 프로그램에 최적화되도록 TCP/IP 기능들 중에서 필요한 것들만을 구현하였다. 예를 들어, 웹 서버에 최적화된 경우, 긴급 데이타(urgent data)와 다른 호스트의 능동적 연결을 지원할 필요가 없으므로 이러한 기능들을 삭제함으로써, 코드 크기를 줄일 수 있다. 특히 iPic TCP/IP 스택[13] 같은 경우에는 TCP 연결 상태 정보를 저장하지 않음으로써 코드 사이즈를 256바이트로 극소화하였다. 웹 서버에 최적화 되어 있지 않은 경우에도 TCP 연결 개수를 제한하거나, 전송 가능한 최대 세그먼트 크기를 고정시키며, IP 패킷의 단편화를 지원하지 않는 경우도 있다[14]. 이러한 방식으로 구현된 TCP/IP 프로토콜 스택들은 표준 TCP/IP의 전 기능이 구현된 원격 호스트와의 통신은 정상적으로 이루어질 수 있으나, 그 기능이 축소된 TCP/IP 프로토콜 간에는 원활한 통신이 어려울 수도 있다.

2.3 마이크로 TCP/IP 및 웹 서버 요구사항

TCP/IP 및 웹 서버를 소형 내장형 시스템에 적용하기 위해서는 다음과 같은 사항들이 고려되어야 한다[15].

- 버퍼 관리 : 성능 향상을 위해서는 동적 버퍼 할당보다는 미리 할당된 버퍼들을 사용하는 것이 효율적이다.
- 타이머 관리 : CPU 대역폭을 소모하거나 동기화 문제를 일으키지 않도록 타임아웃 등에 사용되는 타이머는 운영체제에서 관리하여야 한다.
- 지연시간 최소화 : 프로토콜 처리과정에서 운영체제에 의해 유발되는 지연을 줄이기 위해 인터럽트 핸들링 루틴들의 지연시간을 최소화해야 한다.
- 세마포터 지원 : 버퍼 관리 등을 위한 보호 메커니즘으로서 세마포어가 지원되어야 한다.
- 데이타 복사의 최소화 : TCP/IP의 오버헤드 감소를 위해 데이타 복사가 최소화되어야 한다.
- CPU 대역폭 할당 : 성능 보장을 위해 사용되는 응용 프로그램에 따라 내장형 시스템의 TCP/IP 프로토콜 스택에 적절한 CPU 대역폭이 할당되어야 한다.
- 소켓 API 지원 : 응용 계층의 효율적인 개발을 위해서는 소켓 API가 지원되어야 한다. 다만 기존의 BSD 기반 소켓 API의 전 기능 대신 필수적인 기능만을 구현하여 전체 코드의 크기 및 메모리 사용량을 최소화하여야 한다.
- 웹 서버에서의 CGI 지원 : 내장형 시스템의 원격 제어를 위하여 소형 웹 서버와 사용자가 원하는 웹 페이지를 내장형 시스템에 적재할 수 있는 기능이 지원되어야 한다. 또한 단순히 정적인 웹 페이지에 대한 요청뿐만 아니라 원격 제어를 처리하고 그 결과를 보여줄 수 있는 소형 CGI기술도 지원되어야 한다.

3. 마이크로 TCP/IP 및 내장형 웹 서버

본 절에서는 우선 마이크로 TCP/IP의 설계 및 구현에 대하여 설명한 후, 마이크로 TCP/IP상에서 구현된 소형 내장형 웹 서버에 대하여 기술한다.

3.1 TCP/IP 프로토콜

3.1.1 프로토콜 스택 구성

본 연구에서 설계 및 구현된 TCP/IP 프로토콜 스택의 전체 구조는 그림 1과 같다. 수신된 프레임은 장치 드라이버에 의하여 읽혀지고, 이 프레임은 네트워크 관리자와 버퍼 관리자를 통해 버퍼에 저장이 된다. 이렇게 저장된 패킷은 네트워크 데몬에 의하여 ARP 또는 IP 처리 모듈로 패킷을 넘기게 된다. IP 처리 모듈에서는 상위 프로토콜에 따라 ICMP, UDP, TCP 처리 모듈로 패킷이 전송된다. 상위 레벨로부터 전달된 패킷은 네트워크 데몬을 통하여 버퍼에 저장 되고, 네트워크 관리자를 통하여 네트워크 장치로 전달된다.

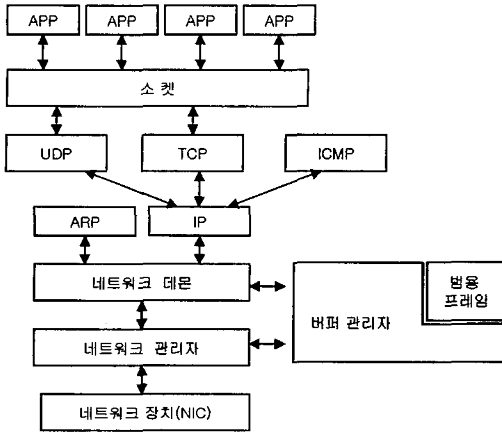


그림 1 프로토콜 스택 구조

3.1.2 버퍼 관리자

버퍼 관리자(Buffer Manager)는 우선 하부 링크 계층에서 전달된 프레임을 상위 프로토콜에 전달하기 전에 링크 계층의 특성을 제거한다. Ethernet의 프레임 구조와 PPP의 프레임 구조를 비교해보면 사용하는 링크의 특성에 따라 헤더와 트레일러만 다를 뿐, 저장하고 있는 데이타가 상위 프로토콜에서는 동일하게 보인다. 따라서 버퍼 관리자에서 링크 계층의 특성을 제거하여 프레임들을 범용화시키면, 상위 모듈에서는 링크 계층과 상관없이 동일한 방식으로 이들을 처리할 수 있다. 즉, 프레임들을 범용화시키면 상위계층과 링크계층간의 독립성이 유지될 수 있다. 본 논문에서 사용한 범용화된 프레임에는 그림 2와 같이 데이타 길이, 프레임이 도착한 장치의 핸들과 단편화를 지원하기 위한 프래그먼트 오프셋이 포함되어 있다.

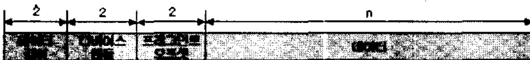


그림 2 범용화된 프레임(GENERAL_FRAME) 구조

버퍼 관리자는 프레임의 범용화뿐만 아니라, 송·수신용 프레임 버퍼를 관리한다. 일반적인 통신에서는 주로 원형 버퍼를 FIFO(First In First Out) 방식으로 이용하는데 비해, 본 논문에서는 소형 내장형 시스템을 목표로 하고 있기 때문에 불필요한 데이타의 복사를 줄이기 위하여 그림 3과 같은 FITO(First In Trial Out)을 기반으로 한 원형 버퍼를 이용하였다.

FITO는 기본적으로 FIFO와 동일하지만 trial 포인터를 하나 더 가지고 있는 것이 특징이다. FITO 구조를 이용한 원형 버퍼에서는 데이타 인출 시 trial 포인터를 우선적으로 증가시킨 후, 데이타의 인출이 올바르게 이루어진 것이 확인될 때 out 포인터를 증가시킨다. 이 방법을 통하면 TCP 계층에서 재전송을 위해 데이타를 다른 공간에 복사할 필요없이 trial과 out 포인터를 이용해 재전송 위치를 손쉽게 결정할 수 있으므로 버퍼 관리가 단순화될 수 있다. 그 외에도 4바이트 크기의 포인터를 통해 TCP 계층에서의 순서 번호를 직접 버퍼 포인터로 사용할 수도 있다.

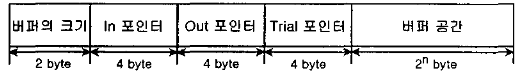


그림 3 FITO를 이용한 버퍼 구조

3.1.3 네트워크 관리자

네트워크 장치와 네트워크 데몬간의 통신을 위해 네트워크 관리자(Network Manager)는 버퍼 관리자를 통해 링크 계층의 특성을 제거함으로써 범용화된 프레임을 이용할 수 있도록 해주는 중개자 역할을 한다. 데이타가 장치로 전송되면, 네트워크 관리자는 이를 범용화된 프레임 형태로 버퍼에서 읽어온 후에 상위 프로토콜 계층으로 전달한다. 마찬가지로 상위계층에서도 전송할 데이타가 있으면 이를 범용화된 프레임 형태로 버퍼에 삽입한다.

3.1.4 인터넷 프로토콜(IP) 및 ICMP

TCP/IP 프로토콜 스택에서 IP는 한 호스트에서 다른 호스트로의 데이타 전송을 책임진다. IP에서는 단편화 처리가 가장 중요한데, IP의 상위 프로토콜 중에서 단편화와 관련이 깊은 프로토콜은 UDP를 사용하는 프로토콜들이다. TCP의 경우에는 연결 설정 시 주고받을 수 있는 데이타 세그먼트의 크기를 협상을 통해 미리 결정할 수 있기 때문에 일반적으로 단편화를 고려하지 않아도 된다. 하지만 UDP의 경우에는 데이타 세그먼트 크기의 협상 과정 없이, 상위계층의 데이타가 IP 데이타그램에 그대로 적재되기 때문에 단편화가 반드시 고려되어야한다.

단편화 패킷의 수신 과정에서는 임시 버퍼가 사용된다. 이때 단편화된 패킷이 연속해서 도착한다고 보장할 수 없을 뿐만 아니라, 여러 송신자가 보낸 단편화 패킷이 동시에 전송될 수 있으므로, 임시 버퍼는 지원되는 단편화 개수보다 많은 양의 메모리를 필요로 한다.

본 논문에서는 최대 6개까지의 단편화를 허용하였는데, 이는 MTU값이 1500바이트인 ethernet 환경에서 8192바이트의 크기를 가지는 NFS 데이타그램에서 6번의 단편화가 일어나기 때문이다. 한 호스트에서 발생하는 단편화 처리를 위한 최대 버퍼의 크기는 6 * 1500 (ethernet MTU) + 200(헤더길이)으로 설정되었고, 동시

에 4개까지의 데이타그램이 처리될 수 있도록 4개의 엔트리를 가지는 해시 테이블이 제공된다. 이 테이블의 각 엔트리에는 이중 연결 리스트를 이용한 큐가 연결된다.

단편화 패킷은 일반 패킷과 동일하지만 임시 버퍼에 저장되어 있다가 다른 단편화 패킷이 도착할 때까지 대기하는 패킷이므로, 타임아웃을·두어 일정 시간이 지나면 버퍼에서 삭제되어야 한다. 본 논문에서는 타이머 대신, 매치 카운트를 이용하여 단편화 패킷이 도착해서 비교될 때마다 매치 카운트를 줄이도록 하였다. 데이타그램의 최초 패킷이 도착하게 되면 매치 카운트는 초기화되고, 이 값이 0이 되거나 데이타그램이 완성되면 임시 버퍼에서 삭제된다.

ICMP에는 총 15개의 메시지 유형이 정의되어 있는데, 이중 본 논문에서는 소형 내장형 시스템에서 꼭 필요하다고 판단되는 목적지 도달 불가 메시지와 에코 요청 및 응답 메시지만을 구현하였다. 주소 마스크 요청과 응답은 시스템 부팅 과정에서 서브넷 마스크를 얻어오기 위해 사용되는 메시지이다. 그런데, 서브넷 마스크는 DHCP나 BOOTP를 이용해서 얻어오는 것이 일반적이므로 본 연구에서는 구현하지 않았다. 다른 시스템과 현재의 시간을 교환하는데 사용되는 타임스탬프 요청 및 응답 메시지는 정보가전의 경우 큰 의미가 없으므로 이 또한 구현하지 않았다. 그 외에도 본 연구에서 구현한 IP 프로토콜은 라우팅 기능을 제공하지 않으므로 라우팅과 관련된 재지정 메시지(5번 유형)와 라우터 광고(9번 유형) 및 라우터 요청(10번 유형) 등의 메시지들도 지원하지 않는다.

3.1.5 전송 제어 프로토콜(TCP)

가. 순서 번호

순서번호의 유지는 버퍼 관리자가 제공하는 FITO 원형 버퍼의 포인터들을 이용하였는데, 송신 버퍼의 trial 포인터를 보낼 TCP 세그먼트의 순서 번호로, out 포인터를 보낸 데이타에 대한 확인 응답 번호로, 수신 버퍼의 in 포인터를 보낼 확인 응답 번호로 대응시켰다(그림 4 참조). 즉, FITO 원형 버퍼를 이용하여 TCP 세그먼트의 순서 번호를 유지하면서 동시에 송·수신 버퍼 관리가 가능하도록 하였다.

그림 4 FITO를 이용한 순서 번호 관리

나. 세그먼트 역다중화

본 논문에서는 시스템의 소켓 수를 사용자가 조정하도록 하여, 사용되지 않는 소켓이 소모하는 자원의 양을 줄였다. 그러나 동시에 여러 세션 연결이 설정될 수 있으므로 수신된 세그먼트가 어느 소켓에 속해있는지 구분하기 위해 역다중화 작업이 필요하다.

역다중화에서는 수신된 세그먼트의 IP 주소와 포트 번호를 기반으로 열려있는 소켓을 검색한다. 수신된 세그먼트와 연결된 소켓이 존재하면 기본 TCP 상태 전이 순서에 따라 작업이 이루어지며, 각 상태에서 수신된 데이타는 상위 응용 프로그램으로 전달되기 위해 콜백(call-back)함수가 호출된다. 콜백 함수는 현재 TCP의 상태를 대기 중인 태스크에 알리고, 태스크가 수신 버퍼에서 데이타를 읽도록 한다.

다. 재전송

TCP의 상태가 전이될 때마다 소켓의 타임스탬프는 갱신되고, 타임아웃 시간 내에 상대방으로부터 응답을 받지 못하면 데이타를 재전송하여야 한다. 재전송을 위해서는 이미 전송된 데이타를 버퍼에 따로 저장하지 않고, FITO 버퍼의 trial 포인터를 이용하여 재전송을 관리한다. 재전송시 타임아웃 시간은 이전 타임아웃의 2배로 증가하며 3번의 시도후 재전송을 멈추게 하였다.

타임아웃의 발생 여부는 각 소켓 마다 가상 타이머(Virtual Timer)를 생성하여 점검하였다. TCP 연결 설정 시에 각 소켓마다 가상 타이머에 대한 핸들러를 등록하고, 타이머에서 타임아웃이 발생되면 해당 핸들러를 호출하여 재전송이 이루어지도록 하였다. 이러한 방식은 전체 소켓에 대한 타이머를 하나 두고 무한 루핑을 하면서 타임아웃 발생 여부를 확인하는 방법에 비하여, 연결 설정이 확립된 이후에 타이머가 동작하게 되므로 사용하지 않는 소켓까지 타임아웃을 검사할 필요가 없어 CPU 대역폭 사용을 크게 줄일 수 있다.

불필요한 재전송이 발생하거나 재전송이 지연되는 현상을 방지하기 위해서는 타임아웃 값이 고정되지 않고 네트워크의 상황에 따라 결정되어야 한다. 본 논문에서는 Jacobson/Karels의 알고리즘[16]를 이용하여 타임아웃 값을 결정하였다.

라. 지연 확인응답 및 혼잡제어

TCP에서는 ACK를 보낼 때 편승(Piggybacking)을 이용하거나 일반적으로 200ms의 지연시간을 두게 된다. 그런데, 원격제어를 위한 소형 내장형 시스템에서는 일반적으로 한번에 여러 개의 TCP 세그먼트가 수신되지는 않고, 대부분의 경우 한 개의 세그먼트만이 수신되므로 지연확인 응답은 전체 네트워크 성능에 큰 영향을 미치게 된다. 따라서 본 논문에서는 상대방으로부터 데이타가 수신된 시점에서 송신할 데이타가 있으면 편승

기능을 이용하지만, 그렇지 않은 경우에는 ACK를 지연시키지 않고 바로 보내도록 하였다. 재전송과 함께 TCP에서 효율적인 데이타 송신을 위해서 혼잡 제어 및 저속 출발, 그리고 빠른 회복(Fast Recovery)을 지원하도록 하였다.

3.1.6 소켓 API

TCP를 이용한 응용 계층의 효율적 개발을 위해서는 소켓 API의 지원이 필수적이다. 본 논문에서는 기존에 개발된 응용 프로그램들과의 호환 및 응용 프로그램 개발이 용이하도록 최대한 BSD 스타일을 유지하면서 새로이 개발한 마이크로 TCP/IP 프로토콜에 적합한 소켓 API를 구현하였다. BSD 기반 TCP/IP에서는 소켓 연결에 필요한 정보들을 PCB(Protocol Control Block) 구조체를 이용하여 별도로 저장 및 관리를 하고 있다. 하지만 본 논문에서는 연결에 필요한 정보들을 소켓 구조체에서 관리되도록 하여 PCB의 생성 및 관리에 필요한 오버헤드를 줄였다.

본 논문에서는 기존의 BSD 스타일의 소켓과 함께 전송 계층과 응용 계층간의 데이타 복사를 제거하기 위하여 Up-call을 지원하는 소켓 API를 추가로 구현하였다. 추가로 구현된 소켓 API에서는 응용 계층에서의 수신 패킷 처리를 위해 일련의 태스크들을 핸들러로 등록하였다. 이렇게 함으로써 네트워크 데몬이 데이타를 수신하면 블로킹과 관련한 일체의 작업을 하지 않고 바로 핸들러에게 데이타를 제공함으로써 네트워크 데몬과 응용 프로그램간의 통신 오버헤드를 줄일 수 있었다.

그림 5는 소켓 API를 이용한 TCP 서버 응용 프로그램의 처리 과정을 도식화한 것이다. 클라이언트와의 연결 설정은 TCP가 Listen 상태로 전환되면, reg_upcall() 메서드를 사용하여 서버 응용 프로그램의 핸들러를 등록하게 된다. 그 다음, accept() 메서드를 통해 수동적 개방(open)을 수행하여, 클라이언트의 연결을 기다리게

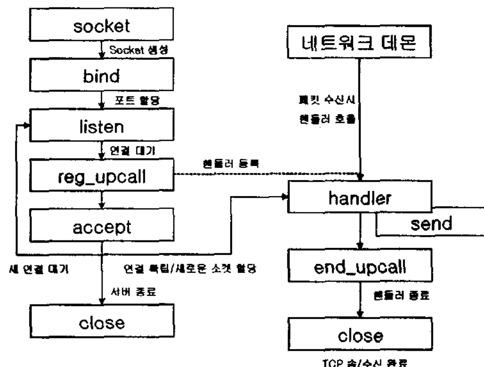


그림 5 핸들러를 이용한 서버 응용 프로그램의 함수 호출도

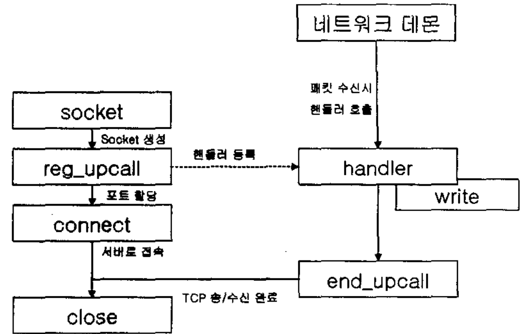


그림 6 핸들러를 이용한 클라이언트 응용 프로그램의 함수 호출도

된다. 클라이언트와의 연결이 완료된 이후에 데이타가 수신되면, 네트워크 데몬은 서버 응용 프로그램의 핸들러를 바로 호출함으로써 서버 응용 프로그램이 처리할 일련의 작업을 바로 수행하도록 해준다.

그림 6은 클라이언트 응용 프로그램의 처리 과정을 보여주고 있는데, 서버 응용 프로그램과는 달리 bind() 과정이 없이 소켓 생성 후 바로 핸들러를 등록하게 된다.

모든 통신이 완료되었을 때, 클라이언트와 서버는 모두 reg_upcall() 메서드를 통해 등록한 핸들러 함수 내에서 반드시 end_upcall() 메서드를 호출하여 핸들러로 등록된 함수의 역할이 모두 끝났음을 알려야 한다. 만약 end_upcall()을 호출하기 전에 응용 프로그램 태스크에서 연결 해제를 위한 close() 메서드를 호출하면, close()함수는 핸들러 내에서 end_upcall()을 호출할 때 까지 대기하게 된다.

UDP를 이용하는 응용 프로그램에서는 TCP 클라이언트와 마찬가지로 소켓 생성 후에 바로 핸들러를 등록하면 Up-call기능을 이용할 수 있다.

3.2 소형 내장형 시스템용 웹 서버 및 CGI

정보가전과 같은 내장형 시스템을 원격 제어하기 위해서는 웹 서버와 같은 응용 계층의 지원이 필요하다. 특히, 인터넷 라우터와 같이 전통적으로 사용자 인터페이스가 존재하지 않는 내장형 시스템에서는 웹을 이용한 원격관리가 표준으로 채택되고 있는 추세이다. 하지만, 기존의 상용 웹 서버와 같은 경우에는 기능이 너무 다양할 뿐만 아니라 그 코드 사이즈가 너무 방대하므로 내장형 시스템에 그대로 적용하기에는 적절하지 않다. 소형 내장형 시스템용 웹 서버는 주로 원격 제어를 위한 목적으로 사용되므로 웹 서버의 모든 기능 중에서 필요한 기능만 구현해도 충분하다. 원격 제어를 위해 필요한 웹 서버의 기능을 정리하면 다음과 같다.

• 파일 시스템 지원 : 내장형 웹 서버가 서비스를 제공하기 위해서는 컨텐츠 저장을 위한 파일 시스템이 필

요하다.

- 동적 컨텐츠 지원 : 많은 응용 프로그램의 경우 정적인 컨텐츠만으로도 충분하나, 웹을 통해 원격으로 모니터링할 경우, 동적으로 생성된 컨텐츠가 처리될 수 있어야 한다.
- 폼(Form) 인터페이스의 지원 : 내장형 웹 서버에서는 폼 인터페이스를 이용하여 원격에서 변수 값을 설정하고, 내장형 시스템의 동작을 제어할 수 있는 기능이 지원되어야 한다.

본 논문에서는 원격 제어 및 모니터링을 위해 내장형 웹 서버에서 필요한 기능들 중에서 파일 시스템은 운영체제에서 제공하는 파일 시스템을 이용하였고, 동적 컨텐츠와 폼 인터페이스 지원은 CGI를 이용하였다.

웹 서버는 본 논문에서 구현한 UP-Call을 기반으로 한 소켓 API를 이용하여 구현하였다. TCP 모듈로부터 웹 서버가 사용하는 포트 번호로 패킷이 수신되면, UP-Call을 이용하여 웹 서버에 등록된 HTTP 핸들러가 바로 호출되고, 수신된 패킷의 데이타들이 HTTP 핸들러에게 전달된다. HTTP 핸들러는 HTTP 헤더를 분석하여 CGI 요청 여부를 판단한다. CGI 요청인 경우에는 CGI 처리를 하게 되고, 아닌 경우에는 요청한 파일이 존재하는지를 확인하여 존재하는 경우에는 해당 파일을 전송하고, 존재하지 않는 경우에는 “HTTP 404 NOT FOUND” 오류 메시지를 전송한다.

HTTP 프로토콜을 사용하는 CGI 요청은 GET 메서드와 POST 메서드 등 두 가지 방법이 있는데, 이 중 GET 메서드가 가장 널리 쓰인다. 뿐만 아니라, POST 메서드를 이용하는 경우도 손쉽게 GET 메서드로의 변경이 가능하므로, 본 논문에서는 GET만을 구현하였다. 다만 GET의 경우 전달 인자가 URI를 통하여 전달되므로 보안 문제가 발생할 가능성이 있다.

CGI 모듈은 해당 실행 파일을 실행하고, 그 결과를 파이프라인을 통하여 전송하도록 되어 있다. 그러나 본 논문에서 타깃 운영체제로 사용한 QPlus는 실행 파일이라는 개념이 존재하지 않으며 파이프라인을 지원하지 못하므로 실행 파일 대신 CGI 태스크를 생성하여 CGI 요청을 처리하도록 하였다. 즉, 파이프라인 대신에 임시 파일을 생성하여 그 결과를 전송하는 방식으로 구현하였다. 이때 임시 파일 생성과 완성된 임시 파일 전송의 동기화를 위하여 세마포어가 사용된다. CGI 요청이 전송된 경우, 웹 서버는 해당 태스크를 실행시키고 세마포어를 기다리게 된다. 태스크는 전달된 파라미터를 분석하여 지정된 모니터링 등의 작업을 수행하고, 그 결과를 임시 파일에 저장한 후 세마포어를 포스트한다. 세마포어를 획득한 웹 서버는 임시 파일을 사용자에게 전송하여 CGI 작업을 마치게 된다.

4. 프로토콜 스택 분석 및 실행 결과

본 절에서는 구현된 TCP/IP 프로토콜 및 웹 서버를 분석한다. 그 다음, 마이크로 TCP/IP의 목적코드 및 성능을 타깃 운영체제인 QPlus에 탑재된 TCP/IP 프로토콜 스택과 많은 내장형 시스템에서 채용하고 있는 RTIP[11]과 비교하고, 마지막으로 그 실행 결과들에 대하여 기술한다.

4.1 프로토콜 스택 분석

4.1.1 IP 프로토콜

IP 프로토콜에서는 라우팅 기능을 삭제하고, 기본적인 패킷 송·수신 기능과 단편화에 중점을 두고 구현하였다. 라우팅은 로컬이나 다른 호스트로부터 전달된 패킷을 다른 네트워크로 전달하는 IP의 중요한 기능 중에 하나이지만, 본 논문에서 타깃으로 하고 있는 정보가전 단말은 일반적으로 라우터로는 사용되지 않으므로 라우팅 기능을 제거하였다.

IP 단편화를 지원하기 위해서는 많은 양의 메모리를 필요로 한다. 특히, 단편화된 모든 패킷이 순서대로 도착된다는 보장이 없을 뿐더러, 하나의 호스트에서만 패킷이 전송된다고 가정할 수도 없는 상황에서 단편화를 지원하려고 하면, 실제 IP 데이타그램 크기보다 훨씬 큰 메모리가 필요하다. 실제적으로 본 논문에서는 ethernet 상에서 NFS를 지원하면서 동시에 4개의 호스트에서 패킷을 전송할 수 있도록 (6 * 1500 + 200) * 4 바이트만큼의 메모리를 미리 할당하여 사용하였다. 그런데 NFS를 사용하지 않는 경우에는 IP 단편화가 거의 일어나지 않으므로, 단편화를 지원하기 위해 할당한 메모리는 대부분이 낭비되게 된다. 물론 단편화된 패킷의 재조립에 사용되는 메모리의 낭비를 방지하기 위해서 단편화된 패킷이 도착했을 경우에만 메모리를 동적으로 할당하여 사용할 수도 있겠지만, 이 방법 또한 메모리를 동적으로 관리해야하는 오버헤드로 인하여 좋은 방법이라 할 수 없다. 따라서 소형 내장형 시스템에서 NFS를 사용하지 않는다면 2개 정도의 단편화만 지원하여도 충분하며, 심지어는 단편화를 지원하지 않아도 대부분의 통신은 가능하다.

4.1.2 TCP 프로토콜

본 논문에서 구현한 TCP 프로토콜 스택은 RFC 표준 규약을 준수하기 때문에 기존의 TCP/IP와 완벽한 호환이 가능하다. 본 논문에서 구현한 마이크로 TCP 프로토콜의 특징을 정리하면 다음과 같다.

- 재전송 처리 시에 폴링 대신 타이머를 이용함으로써 CPU 대역폭 사용을 대폭 줄였으며, 특히 각 소켓별로 별도의 타이머를 두어 사용되지 않는 소켓에 대해서는 불필요한 재전송 검사가 이루어지지 않도록 하

였다.

- 구현된 TCP와 독립적으로 응용 프로그램이 개발될 수 있도록 소켓 API를 구현하였다. 소켓 API은 기존의 BSD 스타일 소켓뿐만 아니라 UP-Call을 지원할 수 있는 소켓도 추가적으로 구현하였다.
- 효율적인 재전송 관리를 위하여 타임아웃 값을 고정식이 아니라 Jacobson/Karels 알고리즘[16]를 이용하여 측정한 RTT를 적용함으로써 현 네트워크 상황에 적절한 타임아웃 값이 유지되도록 하였다.

4.1.3 Zero Copy 지원

본 논문에서는 응용 계층에서 사용하는 수신 패킷 핸들러를 소켓에 등록할 수 있도록 하여 TCP/IP 계층과 응용 계층간의 데이타 복사를 제거하였다. 핸들러 등록은 논문에서 구현된 별도의 소켓 API를 이용하여 이루어진다. 패킷 수신 시에는 TCP나 UDP 계층에서 reg_upcall()을 이용하여 등록된 핸들러를 호출하여 전달할 데이타의 주소를 넘겨줌으로써 별도의 데이타 복사 없이도 응용 계층으로 데이타가 전달되도록 하였다. 하지만, 현재 본 논문에서 구현된 소켓 API는 네트워크 데몬에 데이타가 전송되면 그 크기와 상관없이 바로 등록된 핸들러를 호출하도록 구현되어 있다. 따라서, MTU보다 큰 데이타를 수신해야할 경우, 즉 단편화가 발생하는 경우에는 핸들러가 첫 번째 단편화 패킷을 수신 후에 이를 바로 응용 프로그램으로 전달하므로, 응용 프로그램에서 패킷을 정상적으로 처리하지 못하는 문제가 있다. 그러므로 현재 본 논문에서 구현된 핸들러는 큰 용량의 데이타 수신을 주로 하는 클라이언트보다는 클라이언트 요청에 바로 응답이 필요로 하는 서버측에서 더 유용하게 사용될 수 있다. 향후 패킷이 단편화되어 전달되는 경우에도 정상적인 처리를 할 수 있도록 하기 위해서는 단편화된 패킷을 재조합 후 응용 계층으로 전달되도록 핸들러 호출 과정의 수정이 필요하다.

4.2 웹 서버 및 CGI

본 논문에서는 소형 내장형 시스템의 원격 제어를 위하여 TCP의 상위 응용 프로그램으로서 웹 서버와 간단한 CGI를 구현하였으며, 이들의 특징은 다음과 같다.

- GET 메서드 지원 : 소형 내장형 시스템용 웹 서버이므로 요청 메서드 중 전달 인자 파싱이 가장 간단한 GET 메서드만 지원한다.
- HTTP 헤더의 해석 : 소형화가 가능하도록 GET 메서드만 지원함으로 HTTP 헤더 중 요청 URI만 해석한다.
- 간단한 오류 처리 : 간단한 HTML 요청들을 처리하므로 기존의 웹 서버에서 제공하는 모든 오류 처리가 필요하지 않다. 따라서 본 논문에서 구현한 웹 서버는 "HTTP 404 NOT FOUND" 오류만 처리하도록 구현하였다. 그 이유는 캐시, 리다이렉트, 인증 등과 관련된 대부분의 오류 코드들이 내장형 시스템용 웹 서버에는 거의 사용되지 않으며 "HTTP 404 NOT FOUND"만 가지고도 사용자의 요청을 처리하지 못한다는 의미를 충분히 표현할 수 있기 때문이다.

4.3 코드 크기 및 성능

표 1은 QPlus와 본 논문에서 구현한 프로토콜 스택의 목적 코드 크기를 비교한 결과를 보여주고 있는데, 두 경우 모두 StrongARM CPU상에서 컴파일하였다. 구현된 프로토콜 스택의 목적 코드 크기는 소켓을 포함하여 총 37.9KBytes이어서, QPlus에 탑재되어 있던 프로토콜 스택의 크기 142.0KBytes에 비하여 전체적으로 크기가 약 1/4로 줄어들었음을 확인할 수 있다.

표 2는 x86 CPU 환경에서 구현된 프로토콜 스택을 마이크로소프트의 컴파일러를 이용하여 컴파일한 결과를 RTIP와 비교한 것이다. 표 2를 보면 TCP/IP 프로토콜 스택 중에서 ICMP와 TFTP 클라이언트는 RTIP에 비하여 목적코드 크기가 좀더 큰 것으로 나타나 있다. 하지만 응용 계층을 제외하면 구현된 프로토콜 크기는 RTIP의 약 2/3정도임을 알 수 있다.

본 논문에서 구현된 프로토콜의 성능을 QPlus의 네트워크 모듈과 비교하기 위해서는 에코 서비스를 이용하였다. 데스크 톱 PC에는 에코 서버를 구동시켰고, 타깃 보드에는 에코 클라이언트 태스크를 구동시켰다. 이때 타깃 보드와 데스크 톱 PC의 연결은 ethernet 케이블을

표 1 QPlus와의 목적코드 크기 비교

[illegible]	[illegible]		[illegible] 스택	
	[illegible]	[illegible]	[illegible]	[illegible] 크기
지원 모듈	23.4	16.5%	6.0	15.8%
ARP	5.8	4.1%	1.7	4.5%
IP	19.2	13.5%	5.2	13.7%
ICMP	4.2	3.0%	0.8	2.1%
UDP	3.6	2.5%	3.4	9.0%
TCP	54.2	38.2%	18.3	48.3%
소켓	31.5	22.2%	2.5	6.6%
총합	142.0	100%	37.9	100%

표 2 RTIP와의 목적코드 크기 비교(KBytes)

[illegible]	[illegible]	구현된 프로토콜 스택
IP only(지원모듈포함)	24.1	15
IP + UDP(NIC 포함)	35.0	21.5
IP + TCP(NIC 포함)	49.9	28.3
ARP	2.4	2.2
ICMP	1.0	1.7
TFTP client	5.5	5.7
DHCP client	8.6	4.8

표 3 UDP 에코 서비스 응답 시간

(단위 : msec)

전송 크기 / 횟수	QPlus 네트워크 모듈			구현된 프로토콜 스택			
	100 (bytes)	1,000 (bytes)	1400 (bytes)	100 (bytes)	1,000 (bytes)	1400 (bytes)	5000 (bytes)
1	16	9	13	10	11	11	105
2	7	12	10	10	11	10	105
3	8	10	10	10	10	10	103
4	8	9	10	11	10	10	103
5	10	10	10	10	11	10	103
6	7	9	10	10	11	10	103
7	11	9	10	10	11	10	103
8	7	9	10	10	10	10	103
9	8	12	11	10	11	10	103
10	8	9	10	10	11	11	104
평균	9.0	9.8	10.4	10.1	10.7	10.2	103.5

표 4 TCP 에코 서비스 응답 시간

(단위 : msec)

전송 크기 / 횟수	QPlus 네트워크 모듈			구현된 프로토콜 스택		
	100 (bytes)	1,000 (bytes)	10,000 (bytes)	100 (bytes)	1,000 (bytes)	10,000 (bytes)
1	9	6	221	9	5	173
2	4	9	182	7	5	174
3	3	6	183	3	5	173
4	3	5	181	9	5	169
5	3	5	175	2	5	177
6	3	6	184	3	5	169
7	3	9	191	8	5	172
8	13	9	172	3	5	172
9	3	6	183	3	6	169
10	8	5	181	3	6	174
평균	5.2	6.6	185.3	5.0	5.2	172.2

이용하여 바로 연결시켰고, 클라이언트에서 소켓 생성 바로 직전부터 클라이언트가 응답 데이타를 수신 후 소켓을 종료할 때까지의 시간을 응답시간으로 측정하였다.

표 3은 IP와 UDP를 이용하는 에코 서비스에서 타깃보드를 구동한 후에 반복적으로 에코 서비스를 요청하였을 때, 처음 10회의 시간을 보여주고 있다.

표 3에서 보면 본 논문에서 구현한 프로토콜 스택이 100바이트를 전송하였을 때에는 약 1msec 정도 더 걸리고, 1000바이트와 1400바이트를 전송하였을 경우에는 비슷한 것으로 나타났다. 1500바이트 이상을 전송하는 경우, 즉 IP 단편화가 발생하는 경우에 QPlus는 IP단편화를 지원하지 않아 에코 서비스의 이용이 불가능하였다. 하지만, 본 논문에서 구현된 프로토콜은 데이타의 크기를 1500바이트 이상으로 하여도 정상적으로 에코 서비스를 이용할 수 있었다. 표 3에서 보면 5000바이트를 전송하는 경우, 즉 4개의 단편화가 발생하는 경우에는 평균 약 103msec가 걸려, 1400바이트를 전송했을 때보다 약 90msec의 시간이 더 걸린다. 이는 IP계층에서 단편화된 패킷의 재결합에 소요되는 시간 때문이다.

표 4는 IP와 TCP를 이용하는 에코 서비스에서의 응답 시간을 보이고 있다. 표 4에서 보이고 있는 데이타는 UDP를 이용하는 에코 서비스와 마찬가지로, 타깃보드를 구동한 후에 반복적으로 에코 서비스를 요청하였을 때, 처음 10회의 시간을 보여주고 있다.

표 4를 살펴보면 전체적으로 비슷한 성능을 보이지만, QPlus의 네트워크 모듈이 1,00바이트의 경우에는 1msec, 10,000바이트의 경우에는 10msec 정도의 시간이 더 걸리는 것으로 측정되었다. 특히 10,000바이트를 보내는 경우에는 QPlus의 최초 에코 서비스에서는 다른 경우들 보다 30~40msec 더 걸리는 것으로 측정되었다. 이는 QPlus가 Mbufs구조를 동적으로 관리하기 때문이다. QPlus에서는 메모리 관리의 오버헤드를 줄이기 위해서 시스템 구동 시에 2개의 Mbufs구조체를 미리 할당하여 사용하고 있다. 하지만 10,000바이트의 경우에는

2개의 Mbufs구조체로는 처리가 불가능하여 최초의 에코 서비스에서는 2~3개의 Mbufs구조체를 추가적으로 할당하여 하는데, 이에 따른 오버헤드 때문에 최초의 에코 서비스에서는 좀 더 많은 시간이 걸린 것이다. 그 이후에는 이미 할당되어 있는 Mbufs를 재활용하면 되므로 메모리 할당에 의한 오버헤드는 발생하지 않는다. 본 논문에서는 고정된 버퍼 크기를 사용하여 메모리 관리를 단순화시켰기 때문에 QPlus보다는 빠른 응답 시간을 얻을 수 있었다.

본 논문의 연구 결과와 QPlus 네트워크 모듈 각각에서 1,000바이트와 10,000바이트를 전송하였을 경우를 비교해보면, 두 네트워크 모듈 모두 각각 약 167msec와 179msec의 시간차를 보임을 확인할 수 있다. 이는 메모리 관리 오버헤드에 따른 시간차를 고려하면, 데이타 크기에 따른 전송 오버헤드는 서로 비슷함을 나타낸다.

본 논문에서 구현된 프로토콜과 RTIP와의 성능 비교를 위해서는 지연 확인 응답을 사용하지 않고, TCP 연결을 통하여 20KByte의 데이타를 서버와 클라이언트 간에 전송하였다. 이때 클라이언트는 206MHz의 SA1110 CPU를 사용하였고, 서버로는 1.7GHz의 Intel CPU를 사용하였으며, 10BaseT 네트워크 라인을 이용하여 직접 연결하였다. 표 5는 소켓 연결 시간을 제외하고 실제 데이타 전송 시간을 보여 주고 있다.

표 5에서 다운로드(Download)의 경우에는 실제로 클라이언트가 데이타를 수신하는 데 걸린 시간만을 측정하였고, 업로드(Upload)의 경우에는 서버가 최종 수신까지의 시간을 고려하기 위해서 클라이언트에서 소요된 시간에 RTT/2 값을 합하였다.

표 5를 보면 20KByte를 다운로드 경우에는 평균 6.8msec의 시간이 걸려 약 2.94Mbps의 속도를 나타내었고, 업로드 경우에는 평균 15.4msec의 시간이 걸려 약 1.3Mbps의 속도를 나타냄을 알 수 있다. 상용 운영

표 5 TCP 전송 시간

[illegible]	다운로드(Download)		[illegible]	
	전송 시간 (msec)	전송 속도 (Mbps)	[illegible]	[illegible]
1	7	2.86	14	1.4
2	6	3.33	15	1.3
3	7	2.86	16	1.3
4	7	2.86	17	1.2
5	7	2.86	16	1.3
6	7	2.86	15	1.3
7	7	2.86	15	1.3
8	6	3.33	16	1.3
9	7	2.86	15	1.3
10	6	3.33	15	1.3
[illegible]	6.8	2.94	[illegible]	[illegible]

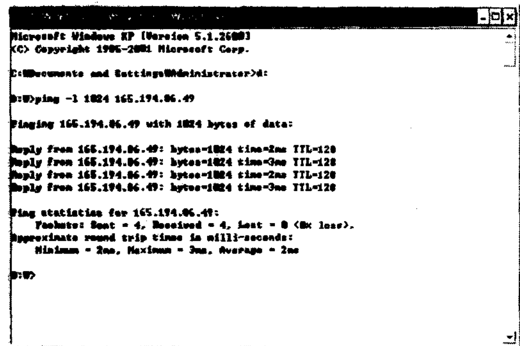
체제에서 채택하고 있는 RTIP의 성능은 클라이언트와 서버의 CPU속도에 따라 다른 성능을 보이는데, 133MHz와 733MHz를 사용하는 시스템 간에 10BaseT 라인을 사용하였을 경우에 1.1Mbps의 속도를 보이고 있다[11]. 따라서 클라이언트와 서버의 성능을 고려할 때, 본 논문에서 구현한 프로토콜 스택과 RTIP가 비슷한 성능을 보이고 있다고 할 수 있다.

4.4 실행 결과

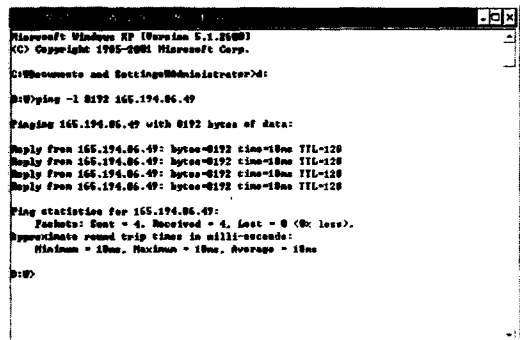
4.4.1 Ping을 이용한 ICMP 및 IP 동작 테스트

Ping은 특정 호스트의 존재를 확인하기 위한 프로토콜로서 ARP, IP, ICMP가 연관되어 동작하는 특성을 활용하여 구현된 프로토콜 스택의 동작 및 성능을 테스트할 수 있다.

ICMP와 IP 단편화 테스트를 위해 Ping을 이용하여 전송되는 데이타의 크기를 1024에서 8192 바이트까지 증가시키면서 응답여부를 확인하였다. 그림 7의 (a)와 (b)는 각각 단편화가 발생하지 않을 1024 바이트의 데이타를 포함시킨 Ping의 결과와 6개의 단편화가 발생할 8192 바이트의 데이타를 포함시킨 Ping의 결과를 보여 주고 있다. 그림 7에서 확인할 수 있듯이 단편화가 발생



(a) 데이타의 크기가 1024 바이트인 경우



(b) 데이타의 크기가 8192 바이트인 경우

그림 7 데이타의 크기에 따른 Ping 요청 결과

하는 경우에도 타깃 보드가 정상적으로 Ping 요청에 대하여 응답함을 확인할 수 있다. 그런데, 그림 7에서 (a)와 (b)를 비교해보면 6개의 단편화가 발생했을 경우 평균 RTT가 18ms로 단편화가 발생하지 않는 경우의 2ms보다 증가한 것을 알 수 있는데, 이것은 타깃 보드에서 단편화된 패킷을 재결합하는데 소요되는 시간 때문이다.

4.4.2 웹 서버 및 TCP 동작 테스트

TCP 프로토콜 및 웹 서버의 동작 테스트를 위해서는 각 브라우저에서 타깃 보드로 접속요청을 보내지 않고 그림 8과 같이 윈도우 시스템 상에서 3개의 웹 요청을 동시에 보낼 수 있는 프로그램을 작성하여 테스트하였다.

그림 8을 보면 동시에 3개의 클라이언트가 접속하였을 경우에도 정상적으로 웹 페이지가 디스플레이 됨을 확인할 수 있다. 동시에 접속할 수 있는 클라이언트의 개수는 본 논문에서 구현된 프로토콜의 소켓의 개수를 나타내는 "socket_num" 값을 이용하여 변경할 수 있다.

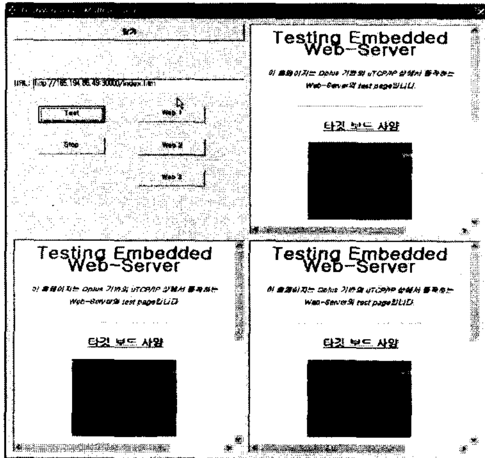
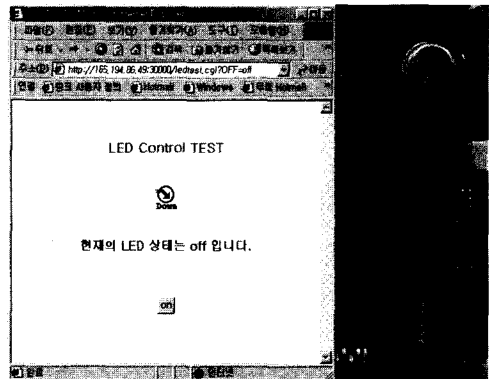


그림 8 다중 클라이언트 동시 접속의 예

CGI 동작 및 원격 제어 테스트를 위해 웹 서버를 통하여 타깃 보드에 부착되어 있는 LED를 온/오프 시킬 수 있는 간단한 CGI를 작성하였다. LED 제어 CGI는 현재 LED의 상태를 읽어 웹 브라우저를 통하여 사용자에게 알려주며 LED의 온/오프 버튼을 제공하고 있다.

본 논문에서는 LED 제어를 위해서 타깃 보드의 CPU인 SA1110에서 제공하는 총 28개의 GPIO (General Purpose I/O)중 0번 포트를 사용하였다. 그림 9는 LED가 "오프"된 상태에서의 CGI 결과 화면과 타깃 보드의 LED를 같이 보여주고 있다. LED가 "오프"된 상태에서 그림 9 하단의 "on" 버튼을 클릭하게 되면 그림 9에서 보는 바와 같이 타깃 보드의 LED가 "온"된다.



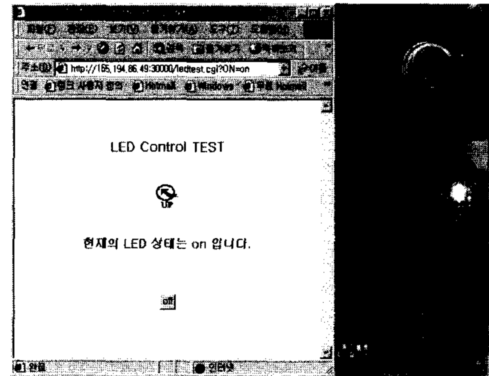


그림 9 LED 제어 CGI의 실행 결과(LED OFF/ON)

5. 결 론

본 논문에서는 웹을 기반으로 정보가전과 같은 소형 내장형 시스템을 제어하기 위한 마이크로 TCP/IP 프로토콜 및 웹 서버를 구현하였다. 데스크 탑 PC와 비교하였을 때 내장형 시스템은 낮은 컴퓨팅 성능과 적은 메모리 등 제한적인 환경을 가지고 있다. 그러나 이러한 내장형 시스템이 인터넷과 연결되면, 인터넷을 통한 원격 관리와 펌웨어 업그레이드 등 현재 내장형 시스템이 가지는 한계를 극복할 수 있다. 이를 위해서는 내장형 시스템에 TCP/IP 프로토콜 및 웹 서버 그리고 CGI 기능이 지원되어야만 한다. 본 논문에서는 내장형 시스템에 적용 가능한 마이크로 TCP/IP 프로토콜 및 웹 서버, 그리고 원격제어를 위한 CGI 모듈을 구현하였다.

본 논문에서 구현한 TCP/IP는 불필요한 데이타 복사 및 무한루프를 제거하여 목적코드 크기를 기존의 RTOS에서 널리 사용하는 RTIP와 QPlus에 탑재되어 있는 네트워크 모듈에 비하여 각각 2/3와 1/4 정도로 줄일 수 있었다. 본 논문에서 구현한 프로토콜 스택을 에코 서비스를 이용하여 QPlus의 네트워크 모듈과 성능을 비교하였을 경우에는 TCP/UDP 모두 비슷한 성능을

보였다. RTIP와 성능 비교를 위해서 전송 속도를 측정하여 본 결과, 다운로드 경우에 2.9Mbps, 업로드 경우에 1.3Mbps의 속도를 나타내어, RTIP와도 비슷한 성능을 보였다. 웹 서버 및 CGI는 필요한 최소한의 기능만을 구현하여 그 코드 사이즈의 최소화하면서도 웹을 통하여 타깃 보드의 IP 주소 변경 등과 같은 제어가 가능하도록 하였다.

본 논문에서 연구 및 구현된 내용을 보다 발전시키기 위해서는 다음과 같은 몇 가지 사항이 추가로 연구되어야 할 것이다. 첫째, 스트리밍 데이타와 같은 대용량의 데이타를 효율적으로 송·수신하기 위하여 버퍼 메커니즘을 개선하여야 한다. 둘째, RTP(Real Time Protocol), RTSP(Real Time Streaming Protocol)와 같이 스트리밍 데이타를 처리할 수 있는 상위 프로토콜 개발이 더 이루어져야 할 것이다. 마지막으로, 본 논문에서 구현된 핸들러를 이용할 경우에 단편화된 패킷을 정상적으로 처리하기 위해서는 단편화된 패킷을 재조합 후 응용 계층으로 전달할 수 있도록 핸들러에 대한 수정이 필요로 하다.

참 고 문 헌

[1] "홈네트워킹 올가이드", 프로그램 세계 2001년 8월호, 신영미디어.

[2] Express Logic Inc, "NetX Technical Features," Web page, URL : http://www.expresslogic.com/nxtech.html

[3] Micro Digital Inc, "TCP /IP stack for smx," Web page, URL : http://www.smxinfo.com/rtos/tcpip/smxnet.htm

[4] CMX Systems, "CMX-MicorNet true TCP/IP Networking," Web page, URL : http://www.cmx.com/micronet.htm

[5] Blunk Microsystems, "Embedded TCP/IP Protocol Stack for Embedded Networking," Web page, URL : http://www.blunkmicro.com/tcp.htm

[6] ETRI and DASAN Co, "확장 가능 조립형 실시간 OS QPlus(Q+)", Web page, URL : http://embenix.com/qplus/

[7] ETRI and DASAN Co, "확장 가능 조립형 실시간 OS QPlus(Q+)", Web page, URL : http://qplus.etri.re.kr

[8] 솔래시스템, "ezTCP," Web page, URL :http://www.eztcp.com.

[9] 박성수, "무선 네트워킹 기술 동향", 전파지. 무선 관리단, 2003.3 URL: http://www.kora.or.kr/kora/radar/200303/sub10.html

[10] 박천교, "홈 네트워크 기술 및 시장 동향", ITFIND 주간기술동향, ETRI IT정보센터, 통권 1086호. pp. 27-28.

[11] EBSnet, "TCP-IP Networking Software for Embedded Systems, Webpage http://www.ebsnetinc.com/

[12] InterNiche Technologies Inc, "NicheStack Portable TCP/IP Protocol Stack," Web page, http://www.iniche.com/products/tcpip.htm

[13] "IPic-A Match Head Sized Web-Server," Web page, URL : http://www-ccs.cs.umass.edu/~shri/iPic.html

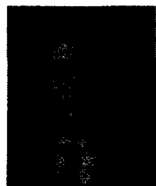
[14] Adam Dunkels, "Full TCP/IP in 8 Bits," April 2002, Web page, URL : http://www.sics.se/~adam/publications.html

[15] Thomas Herbert. "Embedding TCP/ IP," Web page. URL : http://www.embedded.com/internet/0001/0001ia1.htm

[16] Van Jacobson and Michael J. Karels, "Congestion avoidance and control," SIGCOMM'88, ACM Comp. Comm. Rev. Vol. 18, No. 4

선 동 국

1998년 중앙대학교 컴퓨터공학과(공학사). 2000년 중앙대학교 컴퓨터공학과(공학석사). 2000년~현재 중앙대학교 컴퓨터공학과 박사과정. 관심분야는 임베디드 시스템, RTOS



김 성 조

1975년 서울대학교 응용수학과(공학사) 1977년 한국과학기술원 전산과(이학석사) 1977년~1980년 ADD(연구원). 1980년~현재 중앙대학교 컴퓨터공학부(교수) 1987년 Univ. of Texas at Austin(공학박사) 1887년~1988년 Univ. of Texas at Austin(Research Fellow). 1996년~1997년 Univ. of California-Irvine(Visiting Researcher). 관심분야는 이동컴퓨팅, 임베디드 소프트웨어, 유비쿼터스컴퓨팅임

이 재 호

1994년 충남대학교 컴퓨터공학과(공학사). 1999년 충남대학교 컴퓨터공학과(공학석사). 1999년~현재 한국전자통신연구원, 임베디드S/W 기술 센터, 연구원. 관심분야는 임베디드리눅스, RTOS, 무선 인터넷 플랫폼

김 선 자

1985년 숙명여자대학교 수학과(이학사) 2003년 충남대학교 대학원 컴퓨터 공학과(공학박사 수료). 1987년~현재 ETRI 기반기술연구소 임베디드S/W기술센터 무선인터넷플랫폼연구팀장/선임연구원. 관심분야는 임베디드 운영체제, 모바일플랫폼