RTOS를 위한 TCP/IP 프로토콜 스택의 구현

심 형 용[1] · 김 지 환 · 선 동 국 · 김 성 조
중앙대학교 컴퓨터공학과

The Implementation of TCP/IP Protocol Stack for RTOS.

{hyshim[1], jhkim, adonis, sjkim}@konan.cse.cau.ac.kr
Dept. of Computer Science & Engineering, Chung-Ang University

요 약

내장형 시스템 및 RTOS에 대한 관심이 늘어나면서 낮은 성능의 하드웨어상에서의 네트워킹 기능이 중요한 이슈로 떠오르고 있다. 그러나 기존의 BSD기반의 TCP/IP는 많은 메모리를 필요로하고 실제로 RTOS에서 자주 사용되지 않는 기능들도 많이 있기 때문에 기존의 TCP/IP 프로토콜 스택의 수정이 불가피하다. 본 논문에서는 낮은 성능의 하드웨어에 적합하게 TCP/IP프로토콜 스택을 경량화 하고 메모리 사용에 대한 오버헤드를 줄일 수 있는 프로토콜 스택을 구현하고자 한다.

1. 서론

최근들어 디지털 TV 및 IMT-2000 서비스의 개시로 유·무선 홈 네트워크와 기존에 존재하는 인터넷을 서로 연동하여 홈 시어터, 원격 가전기기 및 정보 기기를 제어·활용이 가능하도록 하는 인터넷 정보가전에 대한 수요가 빠르게 증가하고 있다[1]. 정보가전이 인터넷과 연결되기 위해서는 TCP/IP 프로토콜 스택이 필수적이다. 하지만 기존의 전형적인 TCP/IP 프로토콜 스택은 구조가 복잡하고 많은 양의 메모리를 요구하기 때문에 8bit나 16bit처럼 성능이 낮은 CPU를 사용하고 메모리 양이 제한적인 정보가전용 RTOS에는 적합하지않다. 때문에 저사양의 하드웨어에서도 원활하게 동작할 수 있는 소형 TCP/IP 프로토콜스택이 요구되고 있다[2].

따라서 본 연구의 목적은 기존의 BSD기반의 TCP/IP 프로토콜 스택에서 벗어나 정보가전에 사용되는 비교적 낮은 성능의 하드웨어와 이에 탑재되는 RTOS에 최적화된 소형 TCP/IP 프로토콜을 구현하는데 있다.

본 논문의 구성은 다음과 같다. 2장에서는 RTOS를 위한 소형 TCP/IP 프로토콜 스택들에 대하여 기술하고 실시간성을 지원하기 위한 소형 TCP/IP의 요구사항에 대해서 알아보겠고, 3장에서는 2장에서 제시한 요구사항을 만족시키기 위한 TCP/IP 프로토콜을 설계하고 4장에서는 이에 대한 구현사항에 대해서 기술한다. 그리고 5장에서 결론을 맺는다.

2. 기반 연구

2-1. RTOS를 위한 TCP/IP 프로토콜 스택의 동향

인터넷 정보가전에 대한 수요가 증가함에 따라 많은 국내외 연구기관 및 기업에서 RTOS용 TCP/IP 프로토콜을 지원하기 위해 많은 노력을 기울이고 있다. 국내외적으로 개발된 RTOS용 TCP/IP들에 대해 간략히 기술하면 다음과 같다.

▶ **NexGenIP** : NEXGEN이 개발한 NexGenIP는 소형 TCP/IP 프로토콜스택으로 대부분의 RTOS를 지원하나 실시간 운영체제가 없어도 동작이 가능하다. 지원하는 프로토콜은 TCP, UDP, IP, ICMP, IGMP, ARP, 그리고 이더넷 등이 있으며, 코드의 크기는 50KB 이하이고 17~40KB로 코드가 최적화 되어 있다[6].

▶ **NETX** : ExpressLogic에 의해서 개발된 TCP/IP 스택으로 ThreadX라는 RTOS에서 동작한다. 기본적으로 IP, ICMP, ARP, RARP, UDP, TCP 프로토콜을 포함하고 있다[7].

▶ **RTIP-32** : OnTime이 개발한 프로토콜 스택으로 보편적인 유닉스 소켓 API를 이용하여 시리얼 통신을 가능하게 해주는 RTOS-32 시스템에서 TCP/IP를 지원한다[8].

▶ **lwIP** : SICS에서 개발된 프로토콜 스택으로 8비트의 x86과 6502 CPU등을 지원하고 있으며, FreeBSD와 리눅스 상에서 동작한다. 코드의 크기가 작아서 4KB 램보다도 작은 환경에서도 동작 가능하다[9].

▶ **smxNet** : MicroDigital이 개발한 프로토콜 스택으로 대형 호스트들에서 뿐 아니라 ROM에서 동작하는 작은 호스트들에서도 작동하며, 디스크 서비스를 요구하지 않는다. 이로 인해 호스트는 BOOTP를 이용하여 호스트의 네트워크 정보를 자동 설정할 수 있는 기능이 지원된다[10].

▶ **솔내시스템** : 솔내시스템에서는 RTOS 커널과 TCP/IP 스택 기술을 하드웨어 형태로 제공하기 위해 ezTCP를 개발하였다. CDMA 무선망에서 사용될 수 있는 PPP용 제품과 유선망에서 사용될 수 있는 이더넷용 제품이 있다[11].

▶ **세나테크놀로지** : 세나테크놀로지는 RS232 프로토콜을 TCP/IP 프로토콜로 변환해주는 접속장치로 기존 기기를 시리얼 포트와 연결하여, 이더넷 상에서 TCP/IP를 이용할 수 있도록 하였다[12].

2-2. TCP/IP 프로토콜 스택의 요구사항

정보가전에서 실시간 처리기능이 제공되기 위해, RTOS용 TCP/IP 프로토콜 스택은 다음과 같은 요구사항들을 만족시켜야 한다.

▶ **버퍼 관리** : 효율적인 버퍼관리를 위해서는 실행 시에 힙(Heap)으로부터 버퍼들을 할당하는 것보다는 미리 할당된 버퍼들을 사용하는 것이 성능향상을 위해 필요하다[3].

▶ **타이머 관리** : 프로토콜들이 연결 관리, 타임아웃, 재시도 시에 사용되는 타이머가 CPU 대역폭을 소모하거나 동기화 문제들을 일으키는 것을 방지하기 위해서 운영체제에서 타이머를 관리하여야 한다[3].

▶ **지연시간** : 프레임의 물리적인 송/수신 시에 필요한 인터럽트들을 처리하는 과정에서 운영체제에 의한 지연시간이 발생하면 안 된다[3].

▶ **동시성 제어** : 세마포어를 이용한 보호 메커니즘을 사용하여 RTOS의 성능을 향상시켜야 한다. 세마포어를 이용한 보호 메커니즘은 타이머에서의 동시성 문제 해결에도 이용될 수 있다[3].

▶ **데이터 복사의 최소화** : 효율적인 TCP/IP를 위해서는 데이

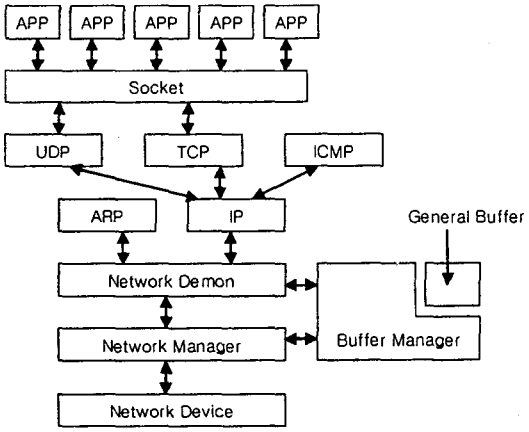
터 복사를 최소화해야 한다. 각 프레임 내부의 데이터는 같은 버퍼 안에 유지될 수 있기 때문에 각 프로토콜 단계에서 CPU에 의한 복사 및 재 복사 횟수를 줄일 수 있다[3].

▶ 링크 계층의 멀티플랙싱(Link Layer Multiplexing) : PPP와 같은 시리얼 점대점(Serial Point-to-Point) 프로토콜이 IP 터널링이나 가상 사설망(Virtual Private Networks: VPN)을 지원하도록 확장되어야 하기 때문에 프로토콜은 유연한 장치 드라이버 인터페이스와 멀티플랙싱 작업을 지원해야한다[3].

▶ CPU 대역폭 : CPU대역폭의 요구량은 해당 어플리케이션마다 서로 다르다. 대부분 인터넷 장비의 경우 TCP/IP 스택은 실시간적인 요소가 고려되지 않기 때문에 많은 대역폭이 요구되지 않는다. 반면, 스트리밍 비디오와 같은 실시간 애플리케이션의 경우, 패킷의 빠른 실시간 처리를 위해 많은 CPU 대역폭을 필요로 하게 된다[3].

3. TCP/IP 프로토콜 스택의 설계

본 연구는 TCP/IP를 경량화하면서도 실시간성에 충실한 프로토콜 스택을 구현하는 것을 목표로 하고 있다. 적은양의 메모리를 사용하기위해 되도록 데이터 카피를 줄이기 위한 방법으로 범용 프레임(General Frame)을 이용하여 TCP나 어플리케이션계층에서 직접 데이터에 접근할 수 있도록 하고 있으며 이를 관리하기 위해서 네트워크 매니저(Network Manager)와 버퍼 매니저(Buffer Manager)라는 모듈을 만들어서 사용한다. 전반적으로 낮은 성능의 하드웨어에서 TCP/IP의 성능을 개선하기 위해 기존의 계층구조를 대폭 축소하여 계층간의 데이터 교환에 대한 비용을 최소화 하였다. 성능을 개선한 TCP/IP 프로토콜 스택의 구조를 살펴보면 (그림 1)과 같다.

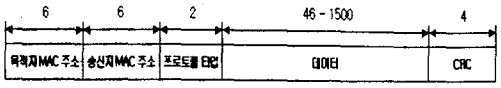


(그림 1) 프로토콜 스택의 구조

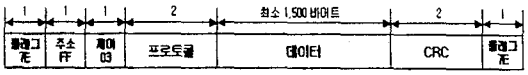
3-1. 범용 프레임

링크 계층과 상관없는 상위 계층에서 프레임을 동일한 형태로 처리하면 패킷처리에 대한 오버헤드를 줄일 수 있다. 때문에 우리는 어떤 동일한 형태가 필요하다 이 형태가 바로 범용 프레임이다.

이더넷의 프레임 구조(그림 2)와 PPP의 프레임 구조(그림 3)를 비교해보면 사용하는 링크의 특성에 따라 헤더와 트레일러만 다를 뿐, 저장하고 있는 데이터는 상위 프로토콜에서 동일하게 보인다.

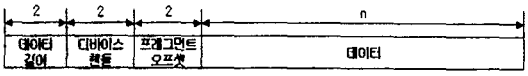


(그림 2) 이더넷 프레임의 구조



(그림 3) PPP 프레임의 구조

범용 프레임에는 데이터 길이, 이 프레임이 도착한 디바이스의 핸들과 단편화를 지원하기 위한 프래그먼트 오프셋을 포함하고 있다(그림 4). 이렇게 각 링크 계층의 프레임을 범용화시킴으로써, 상위 모듈에서는 링크 계층과 상관없이 동일한 프레임으로 처리하게 되어 링크계층과 독립성을 유지할 수 있다.



(그림 4) 범용 프레임(GENERAL_FRAME) 구조

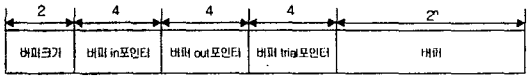
3-2. 네트워크 매니저

네트워크 매니저는 네트워크 디바이스와 네트워크 데몬 사이에서 패킷을 중개한다. 정확하게 말하자면 상위 계층과 하위 계층의 디바이스 계층간의 통신을 위해 버퍼 매니저를 통하여 범용화된 프레임을 만들 수 있는 기능을 제공하고 있다. 네트워크 데몬에서는 송수신되는 패킷이 어떠한 계층으로 전달될 패킷인지를 판단하여 해당 계층에서 패킷에 접근할 수 있도록 한다.

이렇게 네트워크 매니저는 각 링크 계층의 프레임을 상위 모듈에서 링크 계층과 상관없이 동일한 프레임으로 처리하도록 하여 링크계층과의 독립성을 유지하도록 하였다.

3-3. 버퍼 매니저

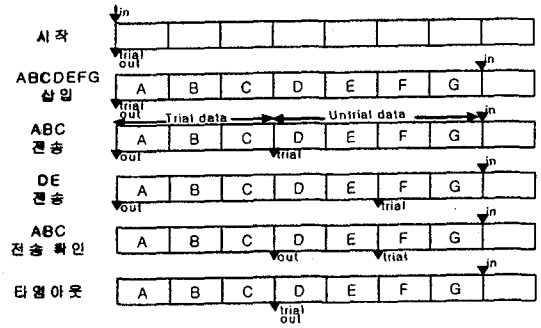
버퍼 매니저는 송/수신하기 위한 프레임 버퍼를 관리하고 하부 링크 계층에서 전달된 프레임을 상위 프로토콜에 전달하기 전에, 링크 계층의 특성을 제거하고 범용화시키는 작업을 한다. 일반적인 버퍼관리는 FIFO(First In First Out)을 이용한 원형 버퍼가 이용된다. 하지만 본 연구에서는 불필요한 데이터의 복사를 줄이기 위하여 일반적인 FIFO대신 FITO(Fisrt In Trial Out)을 기반으로 한 원형 버퍼를 이용하였다(그림 5).



(그림 5) FITO를 이용한 버퍼의 구조

FIFO기반 원형 버퍼에서는 버퍼에서 데이터를 인출할 때 인출한 값만큼 out 포인터가 증가된다. 그러나 FITO구조를 이용한 원형 버퍼에서는 데이터 인출 시 trial 포인터를 우선적으로 증가 시킨 후, 데이터의 인출이 올바르게 이루어진 것이 확인될 때 out 포인터를 증가 시킨다. 이 방법을 통하여 재전송을 위해 데이터를 다른 공간에 복사할 필요 없어지고 대신 trial 포인터를 이용해 재전송이 필요한 값의 위치를 결정할 수 있어 재전송을 고려한 버퍼관리를 간단히 할 수 있다.

예를 들어 'ABCDEF'가 버퍼에 존재하고 이를 전송한다고 가정하고 우선 'ABC'를 전송하면서 trial포인터는 'D'를 가리킨다. 그리고 나서 'DE'를 다시 전송하고 다시 trial 포인터를 'F'로 옮겨진다. 상대방으로부터 ACK가 수신되었다면 out 포인터를 'D'로 옮겨진다. 이후에 'DE'의 전송에 대한 ACK가 도착하지 않은채 타임아웃(Timeout)이 발생하면 'F'를 가리키던 trial 포인터는 다시 현재의 out 포인터로 이동되어 데이터 'DE'를 재전송하게 된다(그림 6).



(그림 6) FITO를 이용한 버퍼의 동작과정

4. TCP/IP의 구현

4-1. 네트워크 매니저의 구현

네트워크 매니저는 디바이스 계층과 범용화된 프레임으로 네트워크 데몬을 연결시켜 주기 때문에 우선 데이터가 디바이스로 전송되면 범용 프레임형태로 버퍼에서 얻어온다. 이후에 상위 프로토콜 계층과 연결된다. 그리고 전송할 데이터가 있으면 이를 네트워크 데몬이 시그널(signal)을 이용하여 범용화된 프레임을 버퍼에 넣는다.

앞에서 살펴보았던 것처럼 범용화된 프레임의 구조<표 1>는 프레임의 길이, 장치핸들 및 단편화 오프셋으로 구성된 헤더와 데이터로 이루어져 있다.

```
// General Header
typedef struct
{
    INT16U  wLen;            /* General Frame Length */
    INT16U  wDeviceHandle;   /* Device Handle */
    INT16U  wFragOffset;     /* Fragment Offset */
}General_Header

// General Frame
typedef struct
{
  GENERAL_HEADER  sGenHeader;/* General Frame Header */
  INT8U aData[MAX_GENERAL_DATA];/* General Frame Data
*/
}General_Frame
```

<표 1> 범용 프레임(General Frame)의 자료구조

4-2. 버퍼 매니저의 구현

버퍼 매니저는 네트워크 매니저의 요청이 있을때 각각의 링크 계층의 프레임을 범용화된 프레임으로 전환시켜 네트워크 매니저로 전달시키는 메소드들이 구현되어 있다. 또한 버퍼를 통하여 교환되는 프레임을 범용 프레임으로 전환시킬 수 있는 메소드, 그리고 재전송시에 trial 포인터를 되돌리는 메소드도 구현되어 있다.

4-3 TCP/IP에서의 기능 구현

IP프로토콜에서는 단편화 기능을 수정하여 코드를 간략화 하였다. 다양한 네트워크 토폴로지를 지원하기 위해서는 하나의 데이터그램이 다양한 프로토콜의 데이터그램 크기에 맞게 단편화 되어야한다. 하지만 정보가전에서는 스트리밍과 같은 매우 큰 데이터그램은 자주 사용하지 않으므로 이를 고려하여 IP 계층에서의 단편화는 2개까지만 허용하도록 구현하였다. TCP 프로토콜에서는 일반적으로 대용량 전송에서 사용되는 혼잡제어 기능들을 축소/삭제하여 RTOS에 적합하도록 구현하였다.

5. 결론

본 프로토콜 스택은 주로 버퍼 및 계층간 데이터 교환에 대한 성능을 개선하는 것에 주안점을 두고 설계되었다. 이렇게 하여 낮은 성능의 하드웨어에서도 동작할 수 있도록 특히 실시간성을 지원 할 수 있는 TCP/IP 프로토콜 스택을 구현하였다. 프로토콜 스택의 크기를 20KB정도 줄여서 경량화 시켰고 계층간의 데이터 복사를 최소화하여 NIC를 제외하고는 메모리 복사 횟수를 줄일 수 있었다. 하지만 아직 TCP 혹은 UDP와 같은 각각의 프로토콜들도 RTOS에 적용하기 위해서는 개선사항들이 많이 있다. 특히 실시간성을 지원하기 위해서는 TCP의 혼잡제어기능이 정보가전에 사용되는 RTOS에 알맞게 설계되어야 할 것이다. 이러한 사항들이 개선된다면 RTOS에서 더욱 더 향상된 실시간성을 지원할 수 있을 것이다.

6. 참고문헌

[1] "홈네트워킹 올가이드", 프로그램 세계 2001년 8월호, 신영미디어
[2] 김성조 외, 정보가전용 운영체제의 네트워크 관리에 관한 연구 최종 보고서, 한국전자통신연구원
[3] 김태준, "소형 임베디드 시스템용 마이크로 TCP/IP 구현", 중앙대학교
[4] Jean J. Labrosse, Embedded Systems Building Blocks, 2nd ed.: Complete and Ready-to-Use Modules in C, R&D Books, 2000.
[5] Jeremy Bentham, TCP/IP Lean: Web Servers for Embedded Systems, CMP Books, 2000.
[6] "NexGenIP" http://www.nexgen-software.com/
[7] "NETX" http://www.expresslogic.com/nxtech.html
[8] "smxNet" http://www.smxinfo.com/rtos/tcpip/smxnet.htm
[9] "RTIP-32" http://www.on-time.com/
[10] "lwIP" http://www.sics.se/~adam/lwip/
[11] "솔내시스템" http://www.eztcp.com/
[12] "세나테크놀로지" http://www.sena.com