UDP/IP 메시지 전송의 QoS 성능 향상을 위한 IP Over USB

장 병 철[†] · 박 현 희[††] · 양 승 민[††]

요 약

임베디드 리눅스 기반의 휴대폰, PDA, MP3 플레이어 등의 소형 내장시스템이 널리 사용되고 있다. 이러한 소형 내장시스템에서 컴퓨터 또는 주변장치와의 데이터 통신을 위한 인터페이스 중에는 USB(Universal Serial Bus)가 있다. 특히 지능형 홈 네트워킹 및 멀티미디어 스트리밍을 지원하는 소형 내장시스템에서는 USB를 통한 UDP/IP 메시지를 실시간으로 전송하기 위한 서비스품질(Quality of Service)의 보장을 요구한다. 리눅스에서는 USB Gadget API 기반의 USB 이더넷 드라이버를 지원하고 있지만 TCP/IP Stack에서의 비 예측성으로 인하여 내장시스템에서 요구하는 QoS를 제공하지 못하고 있다. 본 연구에서는 USB와 리눅스를 사용하는 내장시스템에서 UDP/IP 메시지 전송 시의 QoS 성능 향상을 위한 프레임워크인 IP-Over-USB를 제안한다.

키워드 : USB, QoS, TCP/IP Stack, Network, 임베디드 시스템

IP Over USB for Improved QoS of UDP/IP Messages

Jang Byung Chul[†] · Park Hyeon Hui[††] · Yang Seung Min[†††]

ABSTRACT

The Linux-based embedded systems such as mobile telephones, PDAs and MP3 players are widely in use. USB (Universal Serial Bus) is the interface for data communication between the computers and these peripheral devices. Some embedded systems like intelligent home networking and multimedia streaming require guaranteed QoS (Quality of Service), which is needed for real time transmission of UDP/IP messages through USB. Although USB Ethernet driver is supported by USB Gadget API in Linux, it is unable to provide the desirable QoS required by each type of small embedded systems due to the unpredictability of TCP/IP Stack in Linux. This paper proposes IP-Over-USB to improve QoS of UDP/IP message transmission in the embedded systems using USB in Linux system.

Key Words : USB, QoS, TCP/IP Stack, Network, Embedded System

1. 서 론

휴대전화에서 셋톱박스, PDA는 물론 가전기기에 까지 리눅스가 탑재되어 여러 가지 기능을 제공하고 있다. 이러한 소형 내장시스템에서 컴퓨터 또는 주변장치와의 데이터 통신을 위한 인터페이스 중에서 USB가 가장 널리 사용되고 있다. USB (Universal Serial Bus)[1]는 이러한 내장시스템에서 컴퓨터 주변기기간의 빠르고 편리한 연결을 지원한다. USB를 사용하면 주변기기 등을 PC와 연결할 때 소프트웨어나 하드웨어를 별도로 설정할 필요 없이 모든 주변 기기를 동일한 접속기로 접속하기 때문에 사용이 간편하고, 내장시스템의 소형화가 가능하게 되는 장점이 있다.

지능형 홈 네트워킹 및 멀티미디어 스트리밍을 지원하는 소형 내장시스템에서는 USB를 통한 UDP/IP 메시지를 실시간으로 전송하기 위한 서비스품질(Quality of Service)의 보장을 요구한다. 리눅스에서는 USB Gadget API[2] 기반의 USB 이더넷(ethernet) 드라이버[3]를 지원하고 있지만 TCP/IP Stack[4][5]에서의 비 예측성(unpredictability)으로 인하여 각종 내장시스템에서 요구하는 QoS를 제공하지 못하고 있다.

이러한 단점에 대해서 QoS 및 경성 실시간성을 개선하기 위한 운영체제 수준의 RTlinux 와 RTAI 및 이를 통해 실시간 네트워킹을 지원하기 위한 RTNet[6]등의 연구가 제안되었다. 그러나 이러한 경성 실시간 리눅스를 사용하기 위해서는 리눅스 커널을 비롯하여 디바이스 드라이버[7], 응용프로그램에 이르기까지 많은 수정이 필요하여 멀티미디어 스트리밍과 같은 연성 실시간 처리를 요구하는 시스템에는 적합하지 않다.

※ 이 연구는 2006년도 숭실대학교 교내연구비의 지원으로 연구되었음
† 정 회 원 : 바이테크시스템엔지니어링(주) 연구원
†† 준 회 원 : 숭실대학교 컴퓨터학과 박사과정
††† 정 회 원 : 숭실대학교 컴퓨터학부 교수
논문접수 : 2007년 3월 30일, 심사완료 : 2007년 7월 20일

본 논문에서는 UDP/IP 메시지 전송 시에 QoS를 제공할 수 있는 프레임워크인 IP-Over-USB를 제안한다. 이것은 기존의 리눅스 TCP/IP Stack을 사용하지 않고 문자 드라이버[7]를 사용하여 QoS 성능 향상을 위한 독립된 채널을 제공한다. 또한, 네트워크 응용프로그램에서 이를 접근하기 위한 기존의 소켓 API를 대체하는 File I/O Wrapper API와 리눅스 USB Gadget API를 기반으로 하여 USB Common 드라이버와 USB 컨트롤러 드라이버를 설계하고 구현한다. 본 연구에서는 기존의 USB Gadget API에서 리눅스에 의존적인 부분을 최대한 배제하고 하드웨어적인 처리가 필수적인 부분만을 컨트롤러 드라이버에서 담당하도록 하여 각종 USB 컨트롤러와 응용 요구사항에 따른 소프트웨어 구현 비용의 최소화와 플랫폼 간의 이식성을 높일 수 있도록 설계하고 구현한다.

본 논문의 구성은 다음과 같다. 2장에서는 USB의 개요 및 구성요소를 살펴보고 리눅스 커널의 TCP/IP Stack에 대하여 설명한다. 이어 USB Gadget API에 대하여 설명하고 리눅스를 기반으로 한 QoS 및 실시간 네트워킹을 지원하기 위한 연구들을 살펴본다. 3장에서는 UDP/IP 메시지 전송의 QoS를 개선하기 위한 IP-Over-USB를 설계하고 4장에서는 설계를 바탕으로 이를 구현한다. 5장에서는 구현된 IP-Over-USB의 성능 평가를 보인다. 6장에서 결론 및 앞으로의 연구 방향을 제시한다.

2. 관련 연구

2.1 리눅스 TCP/IP Stack

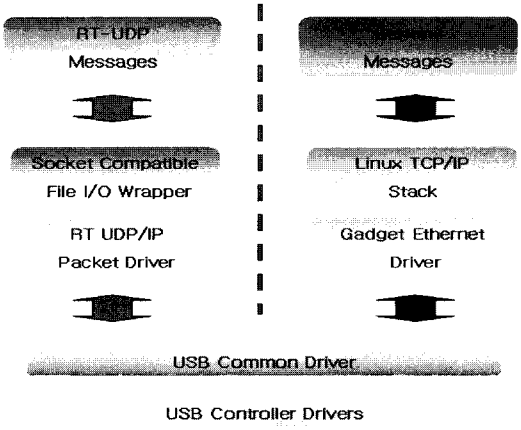
리눅스 커널내의 전체적인 네트워크 구조는 BSD 소켓 인터페이스[5]를 사용자 응용 프로그램에 제공하기 위해서 BSD 소켓 계층을 두고 있으며, 다시 이것에 인터넷 프로토콜[5] 계층을 두어 프로토콜 계층과 연관 짓고 있다. 먼저 BSD 소켓 계층은 시스템의 인터페이스를 지원하기 위해서 존재한다. INET 소켓 계층은 IP에 기반 한 TCP나 UDP 프로토콜의 통신을 관리하는 역할을 한다. TCP와 UDP 계층에서는 사용자로부터 전달 받은 데이터를 실제의 IP Packet 형태로 만들고, 이를 하위의 IP 계층으로 전달하는 역할을 한다. IP 계층은 상위에서 만들어진 데이터를 보내는 것과 하위에서 받은 데이터를 상위로 올려주는 역할을 한다. 이때 보내기 위한 데이터의 재조합이 일어나게 되며, 물론 받은 데이터의 내용에 대해서는 신경 쓰지 않는다. 이 밖에 IP와 동일한 계층에 존재하는 것이 ARP(Address Resolution Protocol) /RARP(Reverse ARP)[5]이다. 이것은 IP 주소를 하드웨어 주소로 주소 바꾸어 주는 역할을 하는 프로토콜이다. 동일 네트워크 내에 있는 상대방에게 패킷을 정확히 전송하기 위해서는 상대의 하드웨어 주소를 알아야 하며, 이는 테이블 형태로 시스템에 저장된다. 시스템이 부팅하게 될 때, 자신이 속한 네트워크의 라우팅(routing) 테이블을 만들고, 실제 패킷전송 시에 해당하는 IP주소에 대해서 ARP를 통해 상대방의 하드웨어 주소를 얻고 이를 사상하게 된다. 나중에 해당하는 IP 주소로 데이터를 보내고자 할 때, 이곳에 등록된 하드웨어 주소를 이용하게 된다. 그러나 이 과정에서 데이터의 전송 시 마다 라우팅 테이블을 참조하게 된다. 더욱이 목적지 주소를 획득하지 못한 최초의 전송이나 ARP timeout에 의해 ARP 정보를 다시 갱신할 필요가 있는 경우와 같이 라우팅 테이블에 해당 하드웨어 주소가 없는 경우에는 예측할 수 없는 전송 지연이 발생할 수 있다.

2.2 리눅스 USB Gadget API

리눅스 USB-Gadget API[2]는 리눅스상의 USB 주변기기를 위한 드라이버 API이다. 구성요소로는 하드웨어, 레지스터(register), FIFO, DMA, 인터럽트 등을 처리하는 USB 컨트롤러 드라이버 계층과 하드웨어 중립적인 USB 기능을 처리하는 Gadget 드라이버 계층, 그리고 CDC (Communication Device Class)[3] 이더넷, 시리얼 그리고 파일 시스템 등의 실제 사용 목적에 따른 드라이버를 제공하는 USB 클래스 드라이버로 이루어져 있다. 그러나 리눅스 USB Gadget API의 이더넷 드라이버는 TCP/IP Stack에 기반하여 구현되어 있기 때문에 실시간 메시지 전송을 지원할 수 없고 드라이버에서 하드웨어적인 부분 이외에도 불필요한 많은 부분도 구현을 해야만 하는 단점이 있다.

2.3 RTNet(Real-Time Network)

RTNet[4]은 RTAI와 Xenomai 기반의 경성 실시간 네트워킹을 지원하기 위한 프레임워크이다. RTNet에서는 리눅스 TCP/IP stack에서의 비 예측성, 비 실시간성을 해결하고자 정적 ARP 관리 메커니즘과 간단한 라우팅 메커니즘을 사용한다. 또한 리눅스에서의 소켓 버퍼[5]의 사용에 따른 오버헤드 및 비 예측성을 해결하고자 Pre-allocated 소켓 버퍼를 사용한다. 그러나 RTNet 을 사용하기 위해서는 리눅스 커널을 비롯하여 디바이스 드라이버[7], 응용프로그램에 이르기까지 많은 수정이 필요하여 멀티미디어 스트리밍과 같은 연성 실시간 처리를 요구하는 시스템에는 적합하지 않은 단점이 있다.



(그림 1) QoS 성능 향상을 위한 IP-Over-USB의 구조

3. QoS 성능 향상을 위한 IP-Over-USB

3.1 QoS 보장을 위한 IP-Over-USB의 구조

(그림 1)은 QoS 성능향상을 위한 IP-Over-USB의 구조이다. 붉은색 점선을 기준으로 오른쪽이 기존의 리눅스 USB Gadget API이며 왼쪽에 있는 부분이 본 연구에서 추가한 부분이다. 기존 리눅스 USB Gadget API를 기반으로 하여 UDP 메시지 전송의 문제점을 해결하고자 리눅스 TCP/IP Stack을 사용하지 않고 QoS를 제공할 수 있는 UDP/IP 패킷 드라이버를 설계하고 구현한다. 또한, 네트워크 응용프로그램에서 이를 접근하기 위한 기존의 소켓 API를 대체하는 File I/O 랩퍼(Wrapper) API와 리눅스 USB Gadget API를 기반으로 하여 이식성 있는 USB Common 드라이버와 USB 컨트롤러 드라이버를 설계하고 구현한다.

3.2 소켓 Compatible File I/O Wrapper

소켓 Compatible File I/O Wrapper는 어플리케이션에서 실시간 메시지를 송수신할 때, UDP/IP 패킷 드라이버를 접근하는 API이다. 각각의 소켓 API를 일반 File I/O로 사상(mapping)하는 랩핑(Wrapping)을 수행한다. 각각의 소켓 API의 함수는 <표 1>과 같이 사상된다.

<표 1> File I/O 랩핑에 의한 소켓 API의 변환

소켓 API	File I/O 시스템 콜
int socket(...)	int open(...)
int bind(...)	int ioctl(...)
int connect(...)	int ioctl(...)
int sendto(...)	int write(...)
int recvfrom(...)	int read(...)
int getsockopt(int...)	int ioctl(...)
int setsockopt(...)	int ioctl(...)

3.3 UDP/IP 패킷 드라이버

UDP/IP 패킷 드라이버는 문자 드라이버를 기반으로 하여 실시간 UDP/IP 데이터를 처리한다. write() 함수에서는 File I/O로부터 수신한 UDP 패킷을 이더넷 패킷으로 변환하여 USB 드라이버로 전달하고 read() 함수에서는 USB 컨트롤러로 부터 수신한 이더넷 패킷을 처리하여 사용자 어플리케이션으로 전달한다. 이 밖에 ioctl() 함수에서는 소켓 API 중, bind() 및 connect()에서 처리하는 주소 및 포트 할당 등의 처리를 하고 setsockopt() 및 getsockopt() 등의 소켓 옵션 처리를 담당한다.

UDP/IP 패킷 드라이버에서는 TCP/IP Stack을 사용하지 않고 QoS 성능 향상을 위한 독립된 채널을 제공한다. 이를 위해 기존의 리눅스 이더넷 드라이버 API를 사용하지 않고 문자 드라이버 API를 사용한다. 드라이버에서 사용하는 어플리케이션과의 데이터 통신을 위한 패킷의 자료구조는 (그림 2)와 같다.

```
typedef struct _qpkt_t {
  struct list_head list;        /* 패킷 list */
  unsigned long len;            /* 패킷 size */
  struct sockaddr_in sock;  /* 패킷의 ip address 및 port 정보 */
  char* data;                   /* UDP 데이터의 Pointer */
}__attribute__((packed)) qpkt_t;
```

(그림 2) UDP/IP 패킷 구조체

```
typedef struct _qsock_udp_t {
  struct list_head list;     /* qsock_udp_t의 list */
  struct sockaddr_in src; /* 송신지 주소 정보 */
  struct sockaddr_in dst; /* 목적지 주소 정보 */
  int sock_type;             /* 소켓 type */
  qpkt_udph_t ph; /* Ethernet Header를 포함한 UDP Header */
  qsock_dev_t* dev; /* device 구조체에 대한 포인터 */
  int event_count;   /* 수신한 event 수 */
  struct list_head rx_queue; /* 수신한 패킷을 저장하는 버퍼 list */
  char* tx_buffer; /* 사용자 영역으로부터 전달받은 송신할 data를
  담고있는 포인터 */
} qsock_udp_t;
```

(그림 3) UDP/IP 패킷 드라이버의 구성요소

UDP/IP 패킷 드라이버는 read() 및 write() 시스템 콜을 통해 사용자 영역으로 송수신한 데이터를 관리하기 위하여 (그림 3)과 같은 내부 자료구조를 사용한다. 이 구조체에서는 UDP/IP 패킷의 송수신을 위한 주소 및 패킷 정보와 이벤트(event) 등을 관리한다. 본 구조체는 open 시스템 콜이 호출될 때 마다 할당이 되도록 하여 파일 디스크립터(File Descriptor)에 따라 독립된 채널을 제공하게 된다.

3.4 USB Common 드라이버

USB Common 드라이버는 이더넷 드라이버와 UDP/IP 드라이버 간의 패킷 라우팅을 중재하고, 열거(enumeration) 및 USB 표준 request를 포함한 제어 전송의 처리와 인터페이스(interface) 등을 관리한다. 열거에는 디바이스의 주소 할당, Endpoint의 기능들을 지정하는 Configuration등이 포함된다[1]. 그리고 IN 과 OUT Endpoint 전송을 처리하고 전역에 걸친 각종 이벤트와 USB suspend, resume, remote wakeup 및 호스트로부터의 연결 해제에 따른 처리를 한다. 본 사항을 위해 리눅스 커널 및 시스템에 의존적인 부분을 별도의 독립 모듈로 설계한다.

```
struct ep_dev {
 struct usb_ep                ep; /* Pointer of USB Endpoint structure */
 struct udc_dev*  dev; /* Pointer of USB controller descriptor */
 const struct usb_endpoint_descriptor *desc;/*Endpoint Descriptor*/
 struct list_head  queue; /* receive queue */
 unsigned long                irqs;   /* Number of IRQ */
 unsigned       wMaxPacketSize;   /* Maximum Packet Size */
 unsigned       fifosize;          /* FIFO Size */
 u8                           bEndpointAddress; /* Endpoint Address */
 u8                       bmAttributes;  /* Attributes of Endpoint */
 unsigned       stopped : 1;  /* Stop Flag */
 u8                           use_dma;     /* DMA flag */
 u8*                          dma_buf;     /* Start Address of DMA */
};
```

(그림 4) USB Common 드라이버의 Endpoint 디바이스 구조체

먼저 USB 디바이스의 Endpoint를 관리하기 위한 자료 구조는 (그림 4)와 같이 구성된다. ep_dev 구조체에서는 Endpoint 별로 다르게 적용되는 FIFO의 크기 및 주소, 인터럽트와 같은 정보를 기술하게 된다.

각종 USB Function 제공을 위한 함수는 ep_dev_ops 구조체를 통해 함수 포인터로 등록된다. 이렇게 등록된 각각의 함수들은 Common 드라이버에 의해 열거와 제어 전송의 처리 시에 사용 된다.

3.5 USB 컨트롤러 드라이버

USB 컨트롤러 드라이버는 하드웨어, 레지스터, FIFO, DMA, 인터럽트 등과 같은 컨트롤러 특성에 따른 처리를 한다. 각기 다른 USB 컨트롤러에서 처리해야만 하는 최소의 일반적인 기능만으로 드라이버를 구현할 수 있도록 설계한다. USB 트랜잭션과 같은 컨트롤러에 의존적이지 않은 부분은 USB Common 드라이버 계층에서 처리하도록 하고 실제 컨트롤러 특성과 관련된 부분만을 처리하도록 한다.

USB 컨트롤러 드라이버 구조체는 컨트롤러에 따른 Endpoint의 정보, 인터럽트, 상태 등을 관리하며, 또한 USB 제어에 필요한 함수들을 함수 포인터로 등록하고, USB Common 드라이버에서 이를 호출하여 일반화된 처리를 할 수 있도록 한다.

4. QoS 성능 향상을 위한 IP-Over-USB 구현

본 연구에서의 IP-Over-USB는 ARP 및 소켓 버퍼의 비사용으로 전송지연 문제를 해결할 수 있고 실시간 메시지 송수신을 위한 독립된 통신 채널을 사용하여 네트워크 트래픽(traffic)의 과부하 시에도 QoS 보장이 가능하다. 본 장에서는 3장에서의 설계를 바탕으로 Freescale i.MX21 USB 1.1 컨트롤러와 리눅스 커널 2.6.10 및 2.4.20을 사용하여 UDP/IP 패킷 드라이버를 구현하고 이식성 있는 USB Common 드라이버와 컨트롤러 드라이버를 구현한다.

4.1 UDP/IP 패킷 드라이버의 구현

3장에서 설명한 바와 같이 UDP/IP 패킷 드라이버는 문자 드라이버를 기반으로 하여 실시간 UDP/IP 패킷을 처리하는 드라이버이다. write() 함수에서는 File I/O로부터 수신한 UDP 패킷을 이더넷 패킷으로 변환하여 USB 컨트롤러로 전달하고 read() 함수에서는 USB 컨트롤러로부터 수신한 이더넷 패킷을 처리하여 상위 어플리케이션으로 전달한다.

먼저 어플리케이션과 USB 컨트롤러 간에 데이터 송수신과 관련된 부분을 설명한다. 어플리케이션에서 write() 시스템 콜을 통해 데이터를 전송하게 되면, 초기화시에 등록한 file_operation을 통해 rtusb_udp_write() 함수가 호출된다.

rtusb_udp_write() 함수에서는 사용자 영역으로부터 커널 영역으로 데이터를 복사하고 sock_addr 헤더로부터 데이터를 전송할 목적지 정보를 얻는다. 그 다음 UDP와 IP 정보를 설정하여 udc_epin_write() 함수를 통해 USB 컨트롤러로 전송한다. 이후 UDP와 IP 정보를 설정한 후에 udc_epin_write() 함수를 통해 USB 컨트롤러로 전송한다. 이 과정에서 기존의 TCP/IP Stack과 비교해서 데이터가 복사는 이루어지지만 동일하지만 불필요한 전송 과정을 줄이고 소켓 버퍼의 할당에 따른 지연 및 ARP 확인 과정을 제거함으로써 성능 향상을 얻을 수 있다.

다음으로 데이터 수신 과정을 설명한다. USB 컨트롤러로부터 데이터 수신에 따른 인터럽트(interrupt)가 발생하면 드라이버 초기화시에 등록한 인터럽트 서비스 루틴인 rtusb_interrupt_handle() 함수가 호출된다. 이 함수에서는 먼저 udc_epout_read() 함수를 호출하여 컨트롤러로 부터 패킷을 수신한다. 이후 해당 패킷이 실시간 처리가 필요한 패킷인지를 확인한다. 일반 패킷일 경우 리눅스 커널에서의 처리와 동일하게 소켓 버퍼를 할당하고 패킷을 복사한 후 TCP/IP Stack으로 처리가 필요한 패킷을 수신했음을 알린다. 실시간 패킷인 경우는 rtusb_packet_handle() 함수를 통해 처리한다. 이 함수에서는 먼저 IP 및 UDP 헤더를 체크하여 대기 큐(wait queue)에서 해당주소와 포트(port)로 대기하고 있는 프로세스(process)가 있는지를 확인하여 대기하고 있는 프로세스가 있다면 수신한 패킷을 수신 큐에 넣고 해당 프로세스를 깨운다(wake up). 이후 프로세스가 깨어나게 되면 read() 함수가 호출되게 된다. rtusb_udp_read() 함수에서는 먼저 현재 파일의 블로킹(blocking) 모드에 따른 처리를 하고 수신 큐에서 데이터를 가져와서 사용자 영역으로 전송할 포맷으로 변환하여 복사하고 수신한 IP 주소 및 포트 정보를 설정하여 사용자 영역으로 전송한다.

4.2 USB Common 및 컨트롤러 드라이버의 구현

USB 컨트롤러 드라이버는 디바이스에 따른 처리를 담당한다. 본 논문에서의 USB 컨트롤러 드라이버는 하드웨어적인 처리가 필수적인 부분만을 컨트롤러 드라이버에서 구현할 수 있도록 한다. 이를 위해 USB Common 드라이버에서는 서로 다른 USB 컨트롤러에서 처리가 필수적인 최소의 기능만으로 USB 장치를 제어할 수 있도록 구현한다. 또한 컨트롤러 및 플랫폼에 의존적이지 않은 부분은 USB Common 드라이버에서 처리하도록 구현한다.

드라이버의 초기화시에는 USB Common 드라이버에서 컨트롤러를 제어하는데 필요한 각 함수를 등록한다. 이때 기술하는 함수로는 컨트롤러 초기화 함수, Endpoint 활성화(enable)/비활성화(disable) 함수, 인터럽트의 요청/해제, USB Common 드라이버와의 데이터 송수신에 필요한 read 및 write 함수, Endpoint stall, 인터럽트 서비스 루틴 등이다. 그리고 해당 USB 컨트롤러의 Endpoint 수, 각 Endpoint별로 전송방식 및 최대로 전송이 가능한 패킷의 크기 및 FIFO의 크기, DMA 사용 여

부 등의 특징을 기술하게 된다. 이렇게 등록된 함수 및 특성들은 USB Common 드라이버에서 USB 트랜잭션을 처리할 때 사용되게 된다.

5. 성능 평가

본 장은 QoS 성능 평가와 플랫폼에 따른 이식성 평가의 두 부분으로 구성된다. 본 실험은 USB 1.1 컨트롤러를 내장한 Freescale i.MX21 평가보드와 리눅스 커널 2.6.10 및 2.4.20을 사용하였다.

5.1 QoS 성능 평가 결과 및 분석

QoS 성능 평가를 위해 네트워크 성능평가 도구인 Iperf[8]를 이용하여 패킷 송수신 대역폭(Bandwidth) 및 네트워크 트래픽에 따른 패킷 손실률(Packet Loss Rate)을 기존의 USB 이더넷 드라이버와 비교 측정한다. 실험 방법은 호스트 PC와 Target에서 각각 Iperf를 송수신 모드에 따라 실행 하여 각각의 성능평가 요소를 시험한다.

먼저 (그림 6)은 패킷 송신에 따른 대역폭 비교 결과이다. 이것은 호스트에서는 Iperf를 (그림 5)와 같이 수신 상태로 실행하고 Target에서 Iperf를 사용하여 USB 1.1 bulk 전송 방식을 기준으로 이론적인 최대 대역폭인 약 10Mbits/ sec으로 데이터를 전송하여 호스트에서 실제 수신한 결과를 측정한다. 실험 결과, 리눅스 커널 2.4.20을 사용한 본 논문에서의 IP-Over-USB가 가장 높은 성능을 보여주고 있다. 다음으로 작은 차이지만 커널 2.6.10을 사용한 IP-Over-USB가 높은 성능을 보여주고 있다. 특히 기존의 커널 2.6.10 에서의 결과와 비교할 때, 높은 성능 향상을 보여주고 있다.

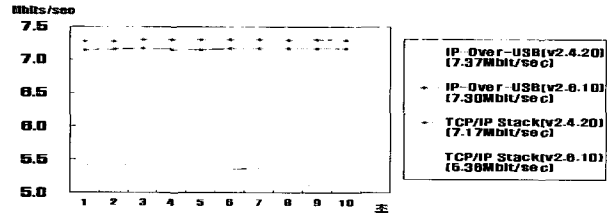
(그림 7)은 (그림 5)와 같이 호스트에서 수신 상태로 실행하고 Target에서 Iperf를 송신 상태로 실행한 후 다시 호스트에서 Iperf를 이용하여 Target측으로 전송한 각각의 부가적인 트래픽에 따른 패킷 손실률을 측정한 결과이다. 실험 결과, 본 논문에서의 IP-Over-USB가 기존의 리눅스 USB 이더넷 드라이버와 비교하여 많은 성능 향상을 보여주고 있다. 위의 실험결과로 메시지 전송 시에 필요한 ARP 및 라우팅 테이블의 비 참조 및 소켓 버퍼를 할당하지 않음으로써 많은 성능 향상이 이루어진 것으로 분석된다.

(그림 8)은 데이터 수신시에도 대역폭 향상이 이루어지는지를 확인하기 위해서 송신의 경우와 반대로 호스트에서 송신 상태로 실행하고 Target에서 Iperf를 수신 상태로 실행한 후 대역폭을 측정한 결과이다. 송신시의 경우와 같이 본 논문의 IP-Over-USB는 기존의 TCP/IP Stack에서의 소켓 버퍼를 할당하지 않음으로써 성능 향상을 기대하였으나 IP-Over-USB와 기존의 리눅스 USB 이더넷 드라이버가 유사한 결과를 보여주고 있다. 이는 송신시의 경우와 비교하여 수신의 경우는 컨트롤러로부터 수신한 데이터를 사용자 어플리케이션으로 전달하기 위해서 커널 영역에서 사용자 영역으로의 데이터의 복사가 불가피하기 때문에 이 과정에서의 성능향상은 이루어 지지 않은 것으로 분석된다.

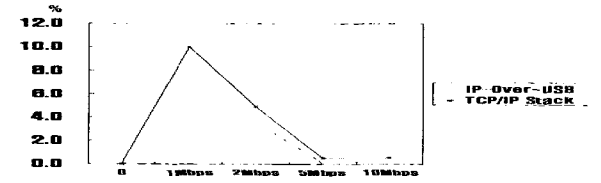
```
송신측$ iperf -c 192.168.1.10 -u -b 10M -i 1

수신측$ iperf -s -u -i 1
```

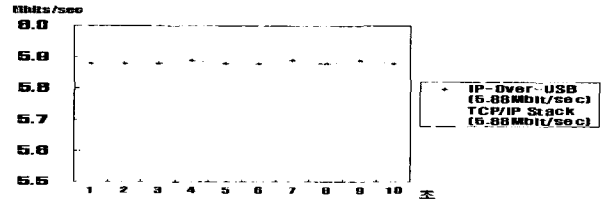
(그림 5) 패킷 송신에 따른 대역폭 실험 방법



(그림 6) 패킷 송신에 따른 대역폭 비교 결과



(그림 7) 트래픽에 따른 패킷 손실률 비교 결과



(그림 8) 패킷 수신에 따른 대역폭 비교 결과

〈표 2〉 플랫폼에 따른 TFTP 다운로드 성능 평가 결과

	Linux System Call	전용 Ethernet Driver API	Interrupt Service 지원	MMU 지원	결과
Linux USB	O	O	O	O	46sec
eCos	X	O	O	O	43sec
eCos	X	O	O	X	100sec
blob-bootloader	X	X	X(Polling)	X	210sec
Linux ethernet (10Mbps)	O	O	O	O	18sec

5.2 플랫폼에 따른 성능 평가 결과 및 분석

(표 2)는 본 연구 과정에서 리눅스에 의존적인 부분을 최대한 배제하고 하드웨어적인 처리가 필수적인 부분만을 컨트롤러 드라이버에서 구현하도록 하여 이를 여러 플랫폼에 적용하여 평가를 한 결과이다. 평가 대상으로는 먼저 앞에서 설명한 리눅스 커널을 사용한 경우와 Embedded RTOS인 eCos[9] 및 Blob Bootloader[10]에 본 논문의 IP-Over-USB 프레임워크를 적용하여 평가한다.

Blob Bootloader의 경우 TCP/IP Stack이 지원되지 않고 인터럽트 서비스 및 캐시(Cache)와 Buffering등의 MMU (Memory Management Unit)를 지원하지 않는다. eCos의

경우에는 FreeBSD Stack을 사용한다. 따라서 이들 플랫폼 간의 상이한 환경에서 평가를 진행하기 위해서 프로토콜 및 구현이 용이한 TFTP 프로토콜을 통해 10Mbytes의 임의 데이터를 Target에서 수신하고, 수신에 걸린 시간 측정을 통해 본 연구에서의 IP-Over-USB가 가지는 플랫폼 간 이식성을 검증하는 방법을 사용하였다.

본 실험을 통해 기존 USB Gadget API가 리눅스만을 지원하는데 비교하여 본 연구의 IP-Over-USB가 리눅스가 아닌 다른 운영체제가 뿐만 아니라 운영체제가 지원되지 않는 플랫폼까지도 지원할 수 있음을 알 수 있다. 이를 통해 드라이버 및 응용 소프트웨어 개발 시에 많은 확장성을 가질 수 있어 본 연구의 IP-Over-USB가 USB를 사용하는 다양한 제품에 응용될 수 있으리라 기대된다.

6. 결 론

본 논문에서는 USB와 리눅스를 사용하는 내장시스템에서 UDP/IP 메시지 전송의 QoS 성능 향상을 위한 프레임워크인 IP-Over-USB를 설계하고 구현하였다.

기존의 리눅스의 USB 이더넷 드라이버는 TCP/IP Stack에서의 비 예측성으로 인하여 각종 내장시스템에서 요구하는 QoS를 제공하지 못하고 있었다. 또한 컨트롤러에 따라서른 다른 소프트웨어를 개발함으로써 어려움이 있었다. 이를 개선하기 위한 운영체제 수준의 경성 실시간을 지원하는 연구가 제안되었으나 스트리밍과 같은 연성 실시간 처리를 요구하는 소형 내장시스템에서는 적합하지 않았다.

본 논문의 IP-Over-USB는 TCP/IP Stack의 ARP 및 소켓 버퍼를 사용하지 않음으로써 TCP/IP Stack에서 발생하는 전송지연을 개선 할 수 있고 실시간 메시지 송수신을 위한 별도의 통신 채널을 사용하여 네트워크 트래픽의 과부하 시에도 QoS의 성능 향상을 보였다. 또한 하드웨어적인 처리가 필수적인 부분만을 컨트롤러 드라이버 에서 구현하도록 하여 각종 USB 컨트롤러와 응용 요구사항에 따른 소프트웨어 구현 비용의 최소화와 플랫폼간의 이식성을 높일 수 있는 효과도 기대된다.

향후 QoS 뿐만 아니라 IPv6 및 Fragmentation의 지원과 같은 다양한 부가기능에 대한 연구가 필요하다. 또한 UDP 뿐만 아니라 TCP에서도 QoS를 보장할 수 있는 통신방법의 연구가 요구되며 IPSec과 같은 데이터의 무결성, 기밀성 등의 보안을 제공할 수 있는 연구가 필요하다.

참 고 문 헌

[1] Universal Serial Bus Specification Revision 2.0, April 27, 2000.

[2] Linux USB Gadget API Framework, http://www.linux-usb.org/gadget

[3] Universal Serial Bus Class Definitions for Communication Devices Version 1.1. January 19, 1999.

[4] Daniel P. Bovet, Marco Cesati, "Understanding the Linux Kernel Third Edition", Oreilly, November 2005.

[5] Gary R. Wright, W. Richard Stevens, "The Implementation TCP/IP Illustrated, Volume 2", January 1995.

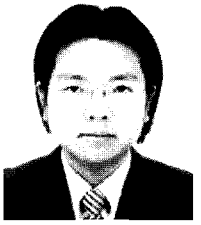
[6] Jan Kiszka, Bernardo Wagner, "RTnet - A Flexible Hard Real-Time Networking Framework", Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference, Sept. 2005.

[7] Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman, "Linux Device Drivers, Third Edition", Oreilly, Third Edition February 2005.

[8] Iperf, http://dast.nlanr.net/Projects/Iperf

[9] eCos, http://ecos.sourceware.org

[10] blob bootloader, http://sourceforge.net/projects/blob



장 병 철

e-mail : jbc8055@gmail.com

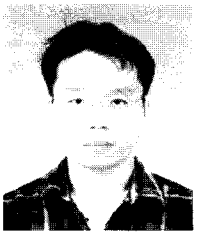
2001년 독학에 위한 전자계산학 학사학위 취득(학사)

2007년 숭실대학교 대학원 컴퓨터학과 (석사)

2002년~2007년 유니데이타커뮤니케이션(주)

2007년~현재 바이테크시스템엔지니어링(주)

관심분야 : 실시간 시스템, 내장 시스템



박 현 희

e-mail : darklight@realtime.ssu.ac.kr

2000년 숭실대학교 컴퓨터학과(학사)

2003년 숭실대학교 대학원 컴퓨터학과(석사)

2007년~현재 숭실대학교 대학원 컴퓨터학과 박사과정

관심분야 : 실시간 시스템, 내장 시스템, 리눅스 커널, 운영체제

양 승 민

e-mail : smyang@ssu.ac.kr

1978년 서울대학교 공과대학 전자공학과 학사

1978년~1981년 삼성전자㈜ 연구원

1983년 미국Univ. of South Florida 전산학 석사

1986년 미국Univ. of South Florida 전산학 박사

1987년 미국Univ. of South Florida 조교수

1988년~1993년 미국 Univ. of Texas at Arlington 조교수

1993년~현재 숭실대학교 컴퓨터학부 교수 겸 (주)엠스톤 대표이사

관심분야 : Real-Time System, Operating System, Fault-Tolerant System