# 실시간 리눅스 기반의 VoIP 게이트웨이 시스템 구현

류홍석*, 정용진**, 민상원***,정광모****
광운대학교 전자통신공학과, 전자부품연구원
grace73@kwangwoon.ac.kr

# Implementation of VoIP Gateway on Real-Time Linux

Hong-Seok Ryu*, Yong-Jin Jeong**, Sang-Won Min ***, Kwang-Mo Jeong****
Department of Electronic Communication Engineering, Kwangwoon University
Korea Electronics technology Institute

요 약

VoIP(Voice over Internet Protocol)는 일반 전화망이 아닌 인터넷 망을 통해 실시간 데이터인 음성을 전송하는 방식으로 음성 통신 비용의 절감 효과와 다양한 서비스로 인한 부가 가치를 통해 개인, 기업, 기간 통신 사업자들에게 큰 이득을 가져다 줄 수 있다. 이러한 추세에 발맞추어 본 논문에서는 RTLinux 기반에서 VoIP 게이트웨이를 구현 함으로서 실시간 처리를 요하는 시스템의 개발기간 단축과 시스템 성능 향상을 목적으로 시스템을 구현하였다. RTLinux는 기존 리눅스 커널 위에서 동작하며, 또한 하드리얼타임을 제공하여 리눅스가 가지고 있던 실시간 처리 문제를 극복할 수 있다. 구현한 시스템은 end-to-end 간에 코덱 (G.723.1)을 전송한 후 프레임간 Delay와 Jitter, loss를 측정하여 기존 리눅스와 RTLinux간의 시스템 성능을 비교 테스트 하였으며, 그 결과 기존 리눅스에 비교하여 RTLinux 기반의 게이트웨이가 코덱이 제한 하는 시간 안에 음성 전송을 처리 함으로서 실시간 처리를 요하는 시스템 개발에 적절한 솔루션임을 확인하였다.

## 1. 서 론

VoIP(Voice over Internet Protocol)는 인터넷 망 위에서 전화나 음성서비스를 제공하는 기술을 말한다. 기존 인터넷 망을 이용함으로써 비용의 절감효과와 다양한 사용자 어플리케이션을 제공한다. 현재 시장에 출시되고 있는 VoIP 게이트웨이 들이 독자적인 기능에서 탈피하여 기존의 라우터나 ADSL, Cable modem, ATM에 탑재되어 출시되고 있으며, Mobile phone등에도 응용이 되고 있다. 그러나 기업과 기간통신 사업자들의 수익모델에 대한 부분과 통화품질 개선에 대해서는 많은 노력이 필요하다. 본 논문에서는 VoIP 게이트웨이를 개발하면서 저 비용, 고효율, 개발기간의 단축을 위해 RTOS중에 하나인 RT-Linux를 사용하여 구현하였는데, [1]RTLinux는 일반 리눅스 커널 위에서 동작 한다. 리눅스는 이미 안정성과 이식성, 저비용, 개발의 용이성 등에서 검증 받았으나, 실시간을 요하는 시스템에는 적절한 솔루션이 되지 못하기 때문에, RTLinux는 이러한 측면에서 해결 방안을 제시하고 있다.

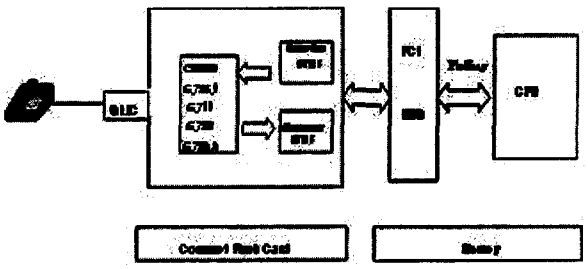

그림 1. 시스템 구성도

그림 1은 구현하고자 하는 VoIP게이트웨이의 시스템 구성도를 보여 주고 있다. 소프트웨어 적인 측면에서 기존 리눅스 (커널버전 2.4.1)위에 RTLinux를 포팅 하였으며, 리눅스가 제공 하는 기본적인 프로토콜들과 게이트웨이 기능을 위한 리눅스용 오픈 H.323 (http://www.openh323.org)을 포팅 하였다. 통화품질 개선을 위해 PCI 버스 방식의 음성 모듈을 사용 하였으며, 이 음성모듈에는 DSP Group사의 CT8022 chip이 임베디드 되어 있고, 4개의 코덱(G.711, G.728, G.729, G.723.1)을 지원하고 있다. 실제 동작은 FXS(Foreign Exchange Station)타입의 일반 전화기가 음성모듈의 슬럿에 연결되어 상대방 IP를 누르면 통화가 가능하게 되어있다.

## 2. RTOS(Real-Time Operating System) 과 RT Linux

RTOS는 기존 OS가 갖고 있는 태스크(TASK)간 Scheduling, Communication, memory management, interrupt service등을 제공하면서도, 주어진 시간 안에 태스크를 수행하는 것을 목적으로 하고 있다. RTOS는 기존 OS들보다 효율성, 속도(엄밀히 시간 제약), 공평성에서 차이가 있다고 할 수 있다. 엄격한 시간 제약을 요하는 의료 장비나 계측장비, 통신 시스템 개발을 위해서 RTOS의 역할과 선택은 대단히 중요하다.

본 논문에서 사용된 RT-Linux는 하드리얼타임을 제공하며 기존 리눅스 커널 위에서 동작하는 RTOS중의 하나이다. 하드리얼타임의 의미는 Predictable, Fast, Low Latency(호출 시간이 길지 않은), Simple Scheduler등의 조건을 충족하는 것을 말한다. RTLinux는 리눅스 커널 위 에서 동작하므로 리눅스에 익숙한 개발자들이 쉽게 접할 수 있으며, 안정성과 확장성이 뛰어나다. 대신 실시간을 요하는 TASK들을 RTLinux 커널 위에서 동작 시켜야 하므로 이에 따르는 프로그래밍이 요구된다. 최근 임베디드 시스템에 올라가는 어플리케이션들이 리소스를 많이 차지하기 때문에 이 들을 태스크를 기능별로 분류하고 태스크간 순위를 정해줘서 처리해 줘야 한다. 또한 각 태스크들을 RTLinux 커널 위에서 동작시키기 위해 스레드 프로그래밍이 요구된다.

* 본 논문은 IDEC의 툴 지원과 전자부품연구원(KETI)의 Electro-0580사업의 지원으로 이루어 졌습니다.

## 3. 구현

### 3.1 구현 절차

아래 그림2를 보면 본 논문에서 구현 시 처리해 줘야 할 부분은 크게 3가지로 나뉘게 된다. [5]CPU는 인텔의 셀러론 366과 DSP Group사의 8022chip을 사용하였으므로 이에 맞게 RTLinux를 포팅하고, 음성 모듈이 PCI버스를 사용하므로 3)PCI 디바이스 드라이버부분과 이를 모듈로 처리해 주어야 한다. 다음으로 이를 다시 RTLinux 커널 위에 TASK로 동작하기 위해 스레드 프로그래밍을 해줘야 한다. 다음으로 기존 리눅스 위에서 TASK로 동작 하고 있는 1)오픈 H.323과 RTLinux위에서 동작하는 음성 모듈 과의 통신을 위해 2)RT-FIFO부분을 같이 처리해야 한다.

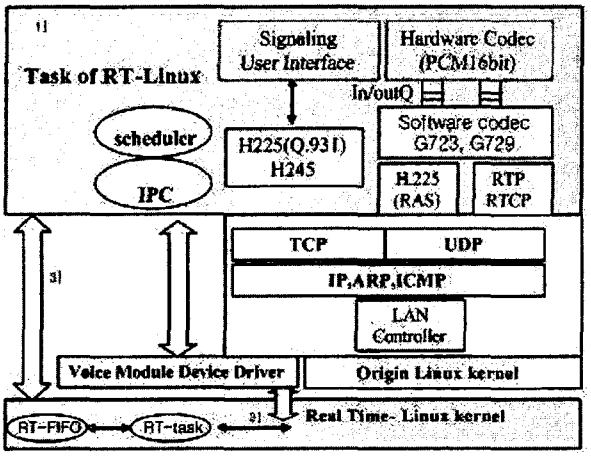


그림 2. 실제 시스템 구현

### 3.2 PCI 디바이스 드라이버(음성모듈)

디바이스 드라이버는 커널에서 리소스의 사용을 줄이기 위해 모듈로서 동작하게 하였고, 음성 모듈은 폴링 방식을 채택하고 있어서 IRQ를 점유하지 않고 일정한 주기로 CPU를 점유하게 된다. 이에 대한 소스 부분은 아래 그림3과 같다.

```
static unsigned int rta_poll(struct file *file_p, poll_table * wait)
 {
   unsigned int mask = 0;

   RTA *j = &rta[NUM (file_p->f_dentry->d_inode->i_rdev)];
         /* registration for  pci device*/

     poll_wait(file_p, &(j->poll_q), wait);

        if (j->read_buffer_ready > 0)
           mask |= POLLIN | POLLRDNORM;  /* readable */
        if (j->write_buffers_empty > 0)
           mask |= POLLOUT | POLLWRNORM;  /* writable */
        if (j->ex.bytes)
           mask |= POLLPRI;

   return mask;
```

그림 3. 음성 모듈 디바이스 드라이버 부분

소스에 대해 구체적으로 설명하면 다음과 같다. 먼저 리눅스 에서 디바이스를 커널에 등록해 주는 과정을 보면, rta_open()은 디바이스를 오픈 하고, *rta_release()*는 디바이스를 닫을 때 사용하게 된다. 이 외에 rta_read(), rta_write() 그리고 rta_ioctl()* 같이 입출력을 제어하는 함수들은 그림 3의 소스에서 file_p라는 자료구조에 등록이 되고 자신의 시작점 주소를 기록 시켜서 디바이스를 동작하게 한다. 이런 과정을 통해서 구현된 시스템 위의 PCI 디바이스는 리눅스 커널 위에서 하나의 태스크로 동작하게 되고, 폴링 방식을 사용하여 CPU가 주기적으로 PCI 디바이스를 점유하게 된다.

### 3.3 RT-Linux 위에서 스레드 프로그래밍

위에서 음성모듈이 구현되면 일반 리눅스에서 태스크로 동작된다. 다시 이를 RTLinux 커널 위에서 태스크로 올려야 한다. 태스크로 동작한다는 의미는 리눅스에서 스레드의 의미와 같다고 보아도 무방하다. 스레드의 의미는 텍스트, 데이터, 스택을 포함하여 실행에 필요한 주소공간과 컨텍스트를 가져서 완전한 하나의 독립적 실행단위로 동작하는 것을 의미한다.

음성 모듈을 RTLinux 커널 위에서 스레드로 동작하기 위해서는 먼저 디바이스 초기 함수인 rta_init() 함수 내에서 pthread_create() 함수를 이용해 스레드로서 초기화 시켜 준다. 스레드를 닫기 위해서는 rta_cleanup()함수 내에서 pthread_exit()를 사용한다. 기존 리눅스 위에서 동작하는 오픈 H.323과 RTLinux위에서 태스크로 동작하는 PCI 디바이스 사이의 통신을 위해 RT-FIFO의 rtf_create()가 사용되는데 , 리눅스의 FIFO와 거의 동일한 의미로 사용되지만, 일종의 device node로 구현되어 리눅스 프로세서에서 단지 device 파일을 오픈해서 read/write만 가능하다. 스레드로 올리기 위해서 자료구조로 rtl_thread_struct 구조체로 정의되며, 스레드와 관련된 모듈로서는 rtl_sched.o를 사용한다.

### 3.4 H.323 동작

그림 1의 구성에서 H.323은 QoS(Quality of Service)를 보장하지 않는 LAN환경에서 멀티미디어 통신 서비스를 가능하게 하며, LAN의 latency를 보정하는 역할을 한다.

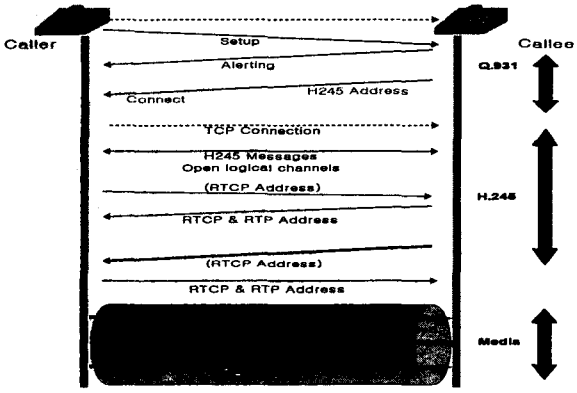


그림 5. H.323 데이터 전송 과정

그림 5는 오픈 H.323의 콜 셋업 과정을 나타내고 있는 데 Q.931/H.225 패킷에 H.245 OpenLogical Channel Request/ Ack 메시지를 전달하게 되며, H.225의 Setup 및 Alerting/ Connect 메시지에 수용된 후, 실제 H.245 *Connection*이 맺어진 후 음성 데이터들(RTP Stream)을 전송하게 된다.

아래 그림 6은 시스템의 성능 테스트를 위해서 end - to - end 간에 음성 트래픽을 가지고 측정한 테스트 베드이다.


그림 6. 테스트 베드

4. 테스트 및 검증

테스트는 그림 6의 [8]환경에서 지원 되는 코덱 중에 G.723.1 6.4Kbps을 선택하였으며 양방향으로 음성 데이터를 전송 하였다. 상대방의 IP를 전화기 버튼을 누르면 신호가 가며 상대방이 수화기를 든 후 call setup이 끝나면 UDP위의 RTP 프로토콜을 이용 음성 데이터를 1분 30초 동안 50회 반복 송수신 하였으며, 동일 시스템에서 구현한 리눅스와 RTLinux상에서 테스트 하였다. 이의 결과를 알기 위해 fluke network사의 네트워크 어날라이저를 이용해 아래 그림 7, 8의 결과를 확인할 수 있었다. G.723.1 코덱은 프레임당 30ms의 전송제한과 패킷 loss는 1%미만으로 규정하고 있다. 네트워크 환경은 평균 일반 유저들이 가지고 있는 대역폭을 512k 바이트라 가정하고 사용된 네트워크 어날라이저에 대역폭을 임의로 설정했다. 즉 프레임과 프레임 사이를 delay라고 보고 30ms가 넘는 지에 대해 측정하였다.

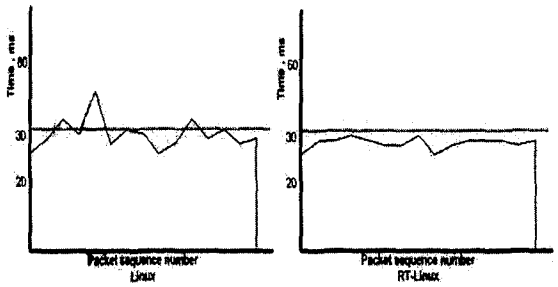


그림 7. G.723.1 Delay 측정

그림 7을 보면 RTLinux와 리눅스에서 G.723.1 코덱을 전송한 결과 RTLinux는 30ms를 초과하지 않았으나, 리눅스는 이를 자주 초과하여 측정되었다.

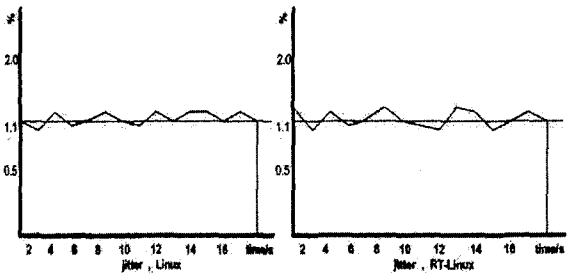


그림 8. G.723.1 Jitter 측정

그림 8에서 Jitter의 의미는 제한 된 시간을 넘어 불규칙적으로 음성신호가 송수신 되는 것을 의미한다. Jitter가 커질수록 상대방의 음성을 듣기 어려워 진다. 이를 해결하기 위해 버퍼에 음성 데이터를 저장해서 보내는 방법을 사용하기도 하는 데 지나친 버퍼의 설정은 latency를 초래하기도 한다.

그림 8에서는 Jitter 부분을 측정 하였는데 RT-Linux부분에서 다소간의 Jitter가 리눅스보다 많이 발생하였다. RTLinux커널 위에서 동작하는 음성 모듈과 오픈 H.323 사이에서의 통신을 위한 RT-FIFO에서 발생하는 문제로 추정된다. 다음으로 Packet loss 부분은 RTLinux나 리눅스 모두 1%미만으로 측정되었다. 측정 결과 RTLinux기반의 시스템은 packet loss, Jitter, delay 모두 허용치를 초과하지 않아 양호한 음성 품질을 제공하고 있다.

5. 결 론

RT-Linux를 사용하여 VoIP 게이트웨이를 구현하면서 RT-Linux는 실시간을 요하는 시스템에 적절한 솔루션임을 알게 되었다. 그러나 개발자가 올리고 싶어하는 태스크에 대해 별도의 스레드와 디바이스 드라이버로 작성해줘야 하며, RTLinux내부적으로도 스케줄링 오버헤드와 과부하 상태에서의 동작 과 시스템 콜 오버헤드, 인터럽트 문제 등을 안고 있지만 계속 개선되어 나가고 있으며, 여러 솔루션들에 적용되어 있는 중이다.

본 논문에서는 VoIP를 구현했지만 이밖에도 라우터나 스위치, 모바일 폰 등 실시간을 요하는 시스템 등에 적용이 가능하며, 오픈 H.323 이나 SIP, MGCP와 같은 현재 VoIP 는 프로토콜들을 RT-Linux 커널 위에 올리는 작업도 가능하리라 본다. RT-Linux는 리눅스에 익숙한 개발자라면 적은 시간의 투자를 가지고서도 원하는 것을 구현할 수 있을 것이다.

참고문헌

[1] 이명근, 이상정, 조성범, 임재용. " 실시간처리 리눅스 기반 VoIP 시스템 설계 및 구현" ., 정보과학회 2001년 봄 학술발표 논문집(A), 제 28권 제 1호, pp. 281-251, 200.
[2] 진성근, 서대화 " 인터넷 폰 설계 및 구현" , 한국 정보과학회 ' 97 가을 학술발표논문집(III), pp. 523-526, 1997.
[3] 신동안 , 신동준 , " C로 배우는 자료구조와 알고리즘" , 기전연구사
[4] 최대수, 임종규, 구용완 , " RT-Linux에서 효율적인 태스크 스케쥴링을 위한 프레임웍 설계" , 2000년 춘계 정보과학회지.
[5] Davidson Peters " Voice over IP Fundamentals," Cisco System co., LTD. pp.192~ 225.
[6] Tankut Akgul, " A Debugger RTOS for Embedded System," IEEE 1089-6503/01 , 2001
[7] Juha Laine, " Real-Time Traffic Measurement in a Differentiated Services Network" IEEE C-7803-7097-1/01 2001
[8] Stefan, " Voice Quality in Converging Telephony and IP Networks" ,white paper, Agilent Tech co., LTD.
[9] Bill Douskalis" Putting VOIP to work" pp.93~ 109.
[10] Victor Yodaiken, " The RTLinux Manifest, " http://www.rtlinux.org."
[11] Victor Yodaiken , " RTLinux approach to hard real-time," http://www.rtlinux.org.
[12] http://www.openh323.org , " faq" , " community" .