한국항행학회 논문지 제 9 권 제 1 호 2005년 6월

실시간 전송기능을 지원하는 TCP의 설계 및 구현

Design and implementation of real-time TCP

우정만*, 조성언**, 김은기***, 권영도***

Jung-Man Woo*, Sung-Eon Cho**, Eun-Gi Kim***, Yong-Do Kwon***

요 약

인터넷 전송계층 프로토콜에는 TCP와 UDP가 있다. TCP는 연결형 프로토콜로서 에러 제어 및 흐름 제어를 제공하여 신뢰성있는 데이터 전송은 보장하지만 그에 따른 전송 지연이 발생한다. 반면에 UDP는 비연결형 프로토콜로서 에러 제어 및 흐름 제어를 수행하지 않으므로 데이터 전송에 있어 지연이 발생되지 않아 실시간 전송은 보장하나 신뢰성있는 데이터 전송을 보장할 수 없다. 따라서 현재의 인터넷 전송계층 프로토콜에서는 실시간 전송기능을 제공하는 동시에 높은 수준의 신뢰성을 보장하여야 하는 데이터의 전송에 적당한 프로토콜이 없다. 본 논문에서는 리눅스(Linux) 커널(Kernel)의 TCP의 에러제어 및 흐름제어를 수정하여 실시간 전송기능을 제공하는 동시에 높은 수준의 신뢰성을 보장하는 새로운 프로토콜을 설계, 구현하였다. 본 연구에서 설계된 실시간 전송기능을 지원하는 TCP 프로토콜은 기존의 TCP에 새로운 옵션을 추가함으로써, 기존 TCP와의 호환성을 유지할 수 있도록 하였다.

Abstract

TCP and UDP is a transport layer protocol of Internet. TCP is a connection oriented protocol which supports a reliable data transfer by offering error and flow control, but it bring a transmission delay. On the other hand, the UDP is a connectionless protocol which does not carry out error and flow control, but it guarantees a realtime transmission. There are hardly any protocols which supports not only realtime functions but also data reliability. In this paper, we have designed and implemented a new TCP mode option which supports reliable realtime transmission. Our designed TCP performs an error recovery process during a fixed amount of time. This time is negotiated during the connection establishment phase. Our designed TCP is tested in real environments, and we find that it is relatively faster than the standard TCP and more reliable than the UDP. It can be used for the reliable transfer of realtime multimedia data.

Key words : TCP, real-time, error control, flow control

현재 동작하고 있는 인터넷 전송 계층 프로토콜은 화상회의 및 동영상 강의 등의 데이터 전송에 요구 되어지는 실시간 전송기능을 지원하는 동시에 높은 수준의 신뢰성을 보장할 수 없다. 즉, TCP는 연결형 프로토콜로서 흐름제어 및 에러제어를 제공하여 높은 수준의 신뢰성을 보장하지만 데이터 전송 지연이 발생하여 실시간 전송기능은 제공하지 못한다[1],[2]. 반면에 UDP는 비연결형 프로토콜로서 흐름제어 및 에러제어 기능을 수행 하지 않아 실시간 전송은 제공하지만 신뢰성 있는 데이터 전송을 보장할 수 없다[1],[2]. 따라서 현재의 인터넷 전송계층 프로토콜에서는 화상회의 및 동영상 데이터 전송에

* 주)휴메이트(Fumate. Co. Ltd.)
** 순천대학교 정보통신공학과(Dept. of information & Communications engineering, Sunchon National University)
*** 한밭대학교 정보통신공학과(Dept. of information & Communications engineering, Hanbat National University)
· 접수일자 : 2005년 5월 21일

적합한 프로토콜이 없다. 본 논문에서는 실시간 전송기능과 높은 수준의 신뢰성을 제공하기 위해 여러 RFC에 기술되어 있는 TCP의 에러제어 및 흐름제어 알고리즘들을 수정하였다[3-5].

Ⅱ. 실시간 전송 기능을 지원하는 TCP의 설계 및 구현

본 논문에서는 기존의 TCP를 기반으로 TCP 헤더에 새로운 옵션을 추가하고 기존 TCP의 여러 흐름제어 와 에러제어 알고리즘을 수정하여 실시간 전송을 지원하는 동시에 제한된 시간 동안 에러제어를 수행하여 데이터 전송에 있어 신뢰성을 보장할 수 있는 새로운 TCP를 설계, 구현하였다. 본 연구에서 사용된 리눅스 커널의 버전은 2.6.6 이다[6].

2-1 새로운 TCP 옵션 추가

기존 TCP와의 호환성을 유지하고 응용으로부터 에러제한시간을 전달 받기 위해 기존의 TCP에 새로운 옵션을 추가하였다. 응용계층에서 setsockopt() 시스템 호출을 사용하여 실시간 전송기능을 지원하는 모드를 요청하고 에러 제한 시간인 ELT(Error control Limitation Time)값을 설정한다.

1 바이트	1 바이트	4 바이트
kind=6	len=6	ELT값

그림 1. 추가된 TCP 옵션 형식
Fig. 1 Added TCP option format

그림 1은 실시간 전송기능을 지원하는 TCP 를 구현하기 위해 새로 추가한 옵션의 형식을 보여주고 있다.

실시간 전송기능을 지원하는 TCP모드를 요청하기 위해 응용에서 다음과 같이 호출하여 옵션을 설정한다[7-9].

setsockopt(listenfd, SOL_SOCKET, SO_ELT, &optval, sizeof(optval))

- listenfd : 소켓 디스크립터 번호
- SOL_SOCKET: 옵션 해석을 socket으로 지정
- SO_ELT: 새로 추가한 옵션의 이름
- &optval: 설정할 ELT값
- sizeof(optval): 옵션값의 길이

응용에서 setsockopt()를 위와 같이 설정하여 호출하면 소켓 계층에서는 sys_setsockopt()함수를 호출 하게 되고, 여기서 옵션 해석 코드를 SOL_SOCKET으로 지정하였으므로 결국 sock_setsockopt()함수를 호출하여 sock 구조체에 실시간 전송기능을 지원하는 TCP 모드를 위해 새로 추가한 sk_elt필드에 전달받은 optval 값을 설정한다. 새로 추가한 ELT 옵션은 TCP의 연결 설정 과정인 3단계-핸드쉐이크(three-way handshake) 시 TCP 헤더에 추가하여 보내지게 된다. 수신측은 이 옵션을 수신시 실시간 전송기능을 지원하는 TCP 모드로 들어가며 송신측으로부터 전달받은 ELT 값 동안 제한된 에러제어를 수행한다. 다음 코드는 실시간 전송기능을 지원하기 위해 수정된 TCP 연산 구조체를 보여주고 있다.

```
struct tcp_opt {
                int     tcp_header_len;
              __u32     rv_next, snd_nxt;
                .....................
    struct timer_list     retransmit_timer;
    //실시간 전송기능을 지원하는 TCP모드 설정
              __u32     elt_ok;
    //에러제어를 수행할 에러제어제한 시간
              __u32     tcp_elt;
}
```

2-2 실시간 전송기능을 지원하는 TCP의 연결 설정

실시간 전송기능을 지원하는 TCP를 사용하기 위해 다음과 같이 수정된 연결 설정을 한다.

- 단계 1 : 일반적인 TCP의 연결 설정과정인 3단계-핸드쉐이크 시 클라이언트는 SYN 세그먼트에 2-1절에서 살펴본 옵션을 추가하여 보낸다. TCP 헤더를 만들고 TCP 헤더에 옵션을 추가하는 작업은 tcp_transmit_skb()함수에서 수행한다. 다음은 수정된 tcp_transmit_skb()함수의 내용이다.

```
int tcp_transmit_skb(struct sock *sk, struct sk_buff *skb) {
```

```
.....
if(tp->elt_ok)
tcp_header_size += TCPOLEN_ELT_ALIGNED;
/*실시간 전송기능을 지원하는 TCP모드를 요청
하기 위해 TCP 헤더의 크기를 ELT옵션의 길이
만큼 증가한다.*/
.....
if (tcb->flags & TCPCB_FLAG_SYN) {
        .....
        tp->elt_ok, /*실시간 전송기능을 지원하
                는 TCP모드 요청 플래그*/
        tp->tcp_elt; //에러제한시간 플래그
        .....
}
```

- 단계 2 : 서버측은 SYN 세그먼트을 수신하고 SYN 세그먼트에 실시간 전송기능을 지원하는 새로 추가된 TCP 옵션이 설정되어 있을 시에 클라이언트에 보낼 SYN 세그먼트에 같은 ELT 값을 설정한 옵션을 추가하여 클라이언트에게 응답한다. 서버측은 SYN 세그먼트를 수신시 tcp_parse_options()에서 클라이언트측의 옵션을 분석한다. 다음은 수정된 tcp_ parse_options()이다.

```
void tcp_parse_options(struct sk_buff *skb, struct
tcp_opt *tp, int estab) {
   .....
   case TCPOPT_ELT:
   //SYN세그먼트의 ELT옵션 확인
   if(opsize==TCPOLEN_ELT && th->syn)
   {
        tp->elt_ok=1; /* ELT옵션을 수신했을시
        서버측도 실시간 전송기능을 지원하는
        TCP 모드 설정*/
        __u32 in_elt=ntohl(*(__u32 *)ptr);
        tp->tcp_elt=in_elt;
        //TCP 연산 구조체에 ELT 값 저장
   .....
}
```

- 단계 3 : 클라이언트는 이에 대한 응답으로 ACK 세그먼트를 보내게 되고 이로써 실시간 전송기능을 지원하는 TCP의 연결 설정과정을 마치게 된다. 이후의 데이터 전송과정은 일반적인 TCP가 아닌 본 논문에서 제시한 실시간 전송기능을 지원하는 TCP를 사용하게 된다. 만약 서버측에서 실시간 전송기능을 지원하지 않아 서버에서 보낸 SYN 세그먼트에 실시간 전송기능을 지원하는 TCP 옵션이 설정되어 있지 않으면 클라이언트는 표준 TCP를 사용하여 데이터를 전송하게 된다.

그림 2는 실시간 전송기능을 지원하는 TCP의 연결 설정 과정을 보여주고 있다.

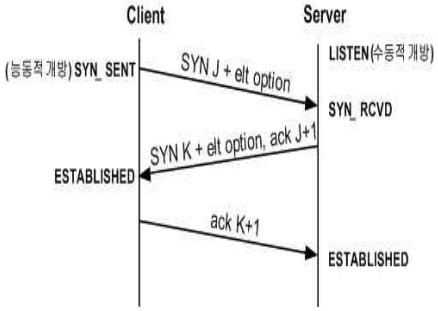

그림 2. 연결 설정 과정
Fig. 2 The process of Connection establishment

2-3 실시간 전송기능을 지원하는 TCP의 동작

2-3-1 송신측의 수신된 동작

송신측은 흐름제어기능을 사용하지 않으며 3단계-핸드쉐이크 시 설정한 ELT 시간 동안만 수정된 에러제어 동작을 수행하게 된다. 데이터 세그먼트를 전송하고 에러 제한 시간인 ELT 시간 만료 시 까지 해당 세그먼트에 대한 ACK를 수신하지 못하였다면, 수신측에 해당 세그먼트를 재전송하고 ACK를 받은 것처럼 동작한다. TCP는 재전송 타이머 만료 시 tcp_write_timer()함수를 호출하고 결국 tcp_retransmit_timer()함수를 호출하여 해당 세그먼트를 재전송하고 다시 재전송 타이머를 설정 한다. 다음은 수정된 tcp_write_timer() 함수이다.

```
static void tcp_write_timer(unsigned long data) {
        .....
        tcp_retransmit_timer(sk);
        //해당 세그먼트를 재전송
        if(tp->elt_ok){
        struct sk_buff *outbuff;
        struct sk_buff *inbuff;
       outbuff=skb_peek(&sk->sk_write_queue);
```

```
        do {
        inbuff = alloc_skb(MAX_TCP_HEADER,
        GFP_ATOMIC);
        } while ( inbuff != NULL);
        TCP_SKB_CB(inbuff)->flags = 0x10;
        // ACK 플래그 설정
        TCP_SKB_CB(inbuff)->sacked = 0;
        TCP_SKB_CB(inbuff)->seq          =
        TCP_SKB_CB(inbuff)->end_seq      =
        tp->rcv_nxt;
        TCP_SKB_CB(inbuff)->ack_seq      =
        TCP_SKB_CB(outbuff)->end_seq+1;
        tcp_ack(sk, inbuff, 0);
        ..........
}
```

그림 3은 첫 번째 데이터 세그먼트는 수신측에 잘 전송됐고 ELT 시간 만료전까지 그에 대한 확인 응답을 받았으나, 두 번째 데이터 세그먼트가 송신 중에 유실되었거나 또는 수신측에서 두 번째 데이터 세그먼트를 수신 하였지만 데이터 세그먼트에 대한 ACK 세그먼트가 유실되었을 때 ELT 시간이 만료되어 송신측에서 해당 세그먼트를 재전송하는 동작을 보여주고 있다.

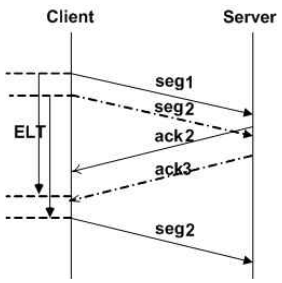

그림 3. ELT 시간이 경과한 경우의 동작
Fig. 3 The process after ELT time expired

2-3-2 수신측의 수신된 동작

수신측은 흐름제어기능을 사용하지 않으며 송신측과 마찬가지로 ELT 시간 동안 일반적인 TCP와 동일하게 에러제어를 수행한다.

송신측으로부터 데이터를 수신 시 각각의 데이터 세그먼트마다 ELT 타이머를 설정하고 ELT 타이머가 만료되었을 때 응용에게 해당 세그먼트를 넘겨준다. 즉, 해당 세그먼트의 ELT 타이머가 만료되기 전까지는 일반적인 TCP와 동일하게 에러제어를 수행할 수 있다. 본 논문에서는 해당 세그먼트마다 ELT 타이머를 설정하기 위해 TCP 연산 구조체에 wait_timer라는 새로운 타이머를 추가 하였다. 다음은 wait_ timer를 초기화 하는 수정된 tcp_init_xmit_timer() 함수이다.

```
void tcp_init_xmit_timers(struct sock *sk) {
   .....
   init_timer(&tp->wait_timer);
   tp->wait_timer.function=&tcp_wait_timer;
   tp->wait_timer.data = (unsigned long) sk;
}
```

순서 번호가 바뀐 데이터 세그먼트를 수신했을 시에는 해당 세그먼트를 저장하고 재전송 ACK를 송신측에 보낸다. 수신측의 ELT 시간 만료 시 까지 해당 세그먼트를 수신하지 못하였을 때는 수신측은 ELT 시간 만료시간까지 받은 모든 패킷에 대한 ACK를 송신측에 보내게 되고 모든 패킷을 정상적으로 수신한 것처럼 동작하게 된다.

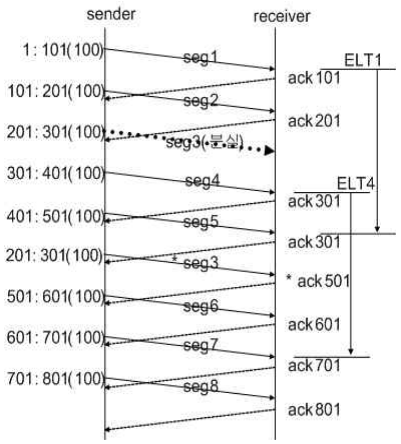

그림 4. 데이터 전송중 세그먼트 유실예(예 1)
Fig. 4 Segment loss example while data is sent (example 1)

그림 4는 첫 번째, 두 번째 데이터 세그먼트는 잘 수신하였고 그에 대한 확인응답도 정상적으로 전송되었으나 세 번째 데이터 세그먼트가 전송중 유실된 것을 보여주고 있다. 네 번째 데이터 세그먼트를 수

신했을 시 수신측은 순서번호가 다른 데이터 세그먼트가 도착한 것을 즉시 알아차리고 재전송 ACK를 보내게 된다. 즉, 네 번째 데이터 세그먼트의 ELT 시간 만료전에 세 번째 데이터 세그먼트가 송신측에서 재전송하여 수신측에서 정상적으로 수신한 것을 보여준다.

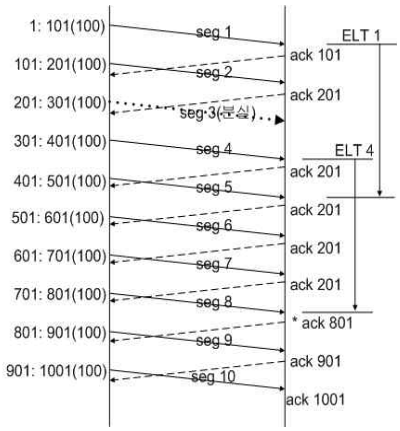

그림 5. 데이터 전송중 세그먼트 유실예(예 2)
Fig. 5 Segment loss example while data is sent (example 2)

그림 5에서는 네 번째 데이터 세그먼트의 ELT 시간 만료전에 세 번째 데이터 세그먼트가 재전송되지 않았다면 ELT 시간 만료시점까지 받은 모든 세그먼트에 대한 확인 응답을 보내게 되고 수신측은 유실된 세 번째 데이터 세그먼트도 정상적으로 수신한 것처럼 동작한다.

2-4 혼잡제어 알고리즘의 수정

실시간 전송을 보장하기 위해 리눅스 2.6.6 커널의 TCP 스택을 사용하여 다음과 같이 혼잡제어 알고리즘을 수정하였다.

- 저속 출발의 제거
- 혼잡회피 알고리즘 제거
- 빠른 재전송 알고리즘의 변경
- 재전송 타임아웃 수정

2-4-1 저속 출발과 혼잡 회피 알고리즘의 제거

저속 출발 알고리즘의 제거를 위해서는 혼잡 윈도우(cwnd)와 광고된 윈도우중 최소값으로 송신 세그먼트의 양을 제한하는 기존의 방식을 오직 광고된 윈도우만을 사용하도록 수정 하여 광고된 윈도우 크기만큼 최대로 세그먼트를 전송할 수 있도록 한다. 혼잡 회피 알고리즘의 제거는 cwnd 값이 ssthresh 값보다 커지더라도 계속해서 저속 출발의 진행을 수행하여 송신 세그먼트의 양을 지수적으로 증가시키도록 한다. 리눅스는 tcp_cong_avoid() 함수에서 reno_cong_avoid() 함수를 호출하여 저속 출발과 혼잡 회피 기능을 제공하게 된다. 다음은 수정된 reno_cong_ avoid() 함수이다.

```
static __inline__ void reno_cong_avoid(struct tcp_opt *tp) {
        .....
        if(tp->elt_ok) {
                //cwnd 값을 무한대로 설정
                tp->snd_cwnd=0x7fffffff
        .....
}
```

2-4-2 빠른 재전송 알고리즘의 변경

분실된 세그먼트에 대해 3개의 중복된 확인 응답을 받았을 경우 해당 세그먼트를 재전송 하던 기존의 방법을, 1개의 중복된 확인 응답을 받을 경우 즉시 재전송 하도록 변경한다. 리눅스는 tcp_v4_init_sock() 함수에서 상수 값을 사용하여 빠른 재전송의 경계 값을 설정한다. 다음은 수정된 tcp_v4_init_sock() 함수이다.

```
static int tcp_v4_init_sock(struct sock *sk) {
        .....
        tp->reordering = 1;
        .....
}
```

2-4-3 재전송 타임아웃의 변경

기존 TCP의 재전송 타임아웃값 설정 방법을 사용하지 않고 송신측에서 SYN세그먼트에 옵션으로 전달해준 ELT 값으로 재전송 타임아웃값을 설정한다. 리눅스는 tcp_reset_xmit_timer() 함수를 호출하여 갱신된 RTO를 현재 전송하는 세그먼트의 RTO로 설정한다. 다음은 수정된 tcp_reset_xmit_timer() 함수이다.

```
static inline void tcp_reset_xmit_timer(struct sock
*sk, int what, unsigned long when) {
   .....
   if(tp->elt_ok) {
        when = tp->tcp_elt; //RTO를 ELT로 설정
   .....
}
```

Ⅲ. 실시간 전송 기능을 지원하는 TCP의 성능 분석

3-1 실험 환경

수정된 리눅스 TCP 스택을 사용하여 인위적으로 에러를 발생시킨 실제 인터넷 망에서 수행 하였고, 테스트를 수행할 서버와 클라이언트 모두 수정된 TCP 스택을 사용하였다.

실험 내용은 다음과 같이 세 가지 경우로 나누었다.

- 첫째, 인위적으로 에러를 발생시킨 실제 인터넷 망에서 서버에서 클라이언트로 고정된 크기의 데이터를 전송하여 실제 패킷 수신율을 측정 하였다. 이때 패킷 에러 비율(PER: packet Error Rate)을 0%부터 3%까지 변화시켜가며 실험 하였다. 수정된 TCP에 대해서는 ELT 시간을 변화시켜가며 테스트 하여 각각에 대한 패킷 수신율을 측정 하였다.
- 둘째, 인위적으로 에러를 발생시킨 실제 인터넷 망에서 서버에서 클라이언트로 고정된 크기의 데이터를 전송하여 전송 속도를 측정 하였다. 이때 PER을 0%부터 3%까지 변화시켜가며 실험 하였다. 수정된 TCP에 대해서는 ELT시간을 변화시켜가며 테스트 하여 각각에 대한 전송 속도를 측정하였다.
- 셋째, 인위적으로 에러를 발생시킨 실제 인터넷 망에서 서버에서 클라이언트로 고정된 크기의 데이터를 전송하여 기존 TCP, UDP, 수정된 TCP의 패킷 수신율을 비교 측정 하였다. 이때 PER을 0%부터 3%까지 변화시켜가며 실험 하였다.

3-2 망의 인위적인 혼잡 발생

다양한 PER에 맞도록 패킷 분실을 발생시키기 위해서 모든 항목의 TCP 스택을 다음과 같이 수정하였다.

```
extern int per; /* 0, 1, 5, 10, 20, 30  */
int tcp_transmit_skb(struct sock *sk, struct
sk_buff *skb) {
        .....
        drop_rate = net_random() % 1000;
        if(per != 0){
                if (drop_rate <= per){
                __kfree_skb(skb);
                /* 패킷을 폐기 한다 */
                err = 0;
  /* 정상적으로 패킷을 전송한 것으로 설정 */
                }else
        err = tp->af_specific->queue_xmit(skb);
        }else
        err = tp->af_specific->queue_xmit(skb);
        if (err <= 0)
                return err;
         .....
}
```

패킷의 분실을 구현하기 위하여 응용프로그램 시작 시 특정한 시스템 콜을 사용하여 설정된 PER을 사용하여 패킷의 분실을 수행할 것인지 결정하게 된다. 이때 패킷의 분실을 수행하더라도 err을 0으로 설정하여 패킷의 송신이 잘 이루어진 것처럼 가정하게 된다. 결국 기존의 TCP는 타임아웃이나 중복된 확인응답으로 패킷의 분실을 알게 되고, 다양한 혼잡제어 알고리즘을 수행하게 되고, 수정된 TCP는 타임아웃이나 중복된 확인응답으로 패킷의 분실을 알게 되지만, 응용으로부터 전달 받은 ELT 시간 동안만 에러제어를 수행하고 흐름제어는 수행하지 않는다.

3-3 인위적인 에러를 발생시킨 실제 인터넷 망에서의 실험

3-3-1 패킷 수신율

인위적으로 에러를 발생시킨 실제 인터넷 망에서 서버에서 클라이언트로 고정된 크기의 데이터를 전송하여 패킷 수신율(수신/송신 비)을 측정 하였다

[11]. 패킷 에러 비율(PER: packet Error Rate)을 0%부터 3%까지 변화시켜가며 실험 하였다. 수정된 TCP에 대해서는 ELT 시간을 변화시켜가며 테스트 하여 각각에 대한 패킷 수신율을 측정하였다. 아래 모든 경우의 그래프는 표준 TCP와의 비교를 쉽게 하기 위해 표준 TCP를 100% 기준으로 하고, 나머지 항목을 그에 비례하도록 환산하여 도식 하였다.

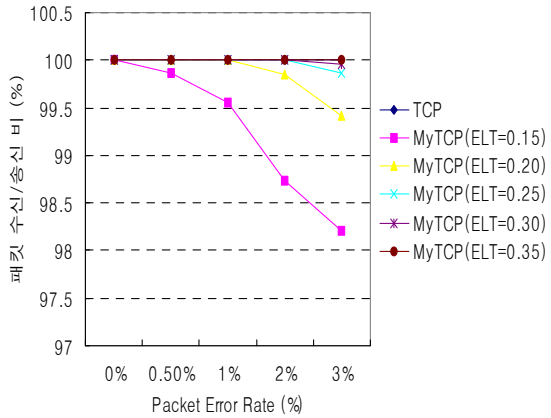

그림 6. 패킷 에러율에 따른 패킷 수신율 비교(실험 1)
Fig. 6 The comparison with packet error rate and receiving rate(test 1)

그림 6을 살펴보면 ELT 값이 0.15 일때 패킷 수신율이 급격히 떨어지고 있는 것을 볼 수 있다. 이것은 ELT 값이 너무 작아 송신측에서 잃어버린 패킷에 대한 에러제한시간이 충분하지 않기 때문이다. ELT 값이 0.20 일 때 패킷 수신율은 ELT 값이 0.15 일 때 보다는 높아 졌지만 표준 TCP에 비해 에러제한시간이 충분하지 않아 표준 TCP보다는 비교적 패킷 수신율이 떨어진 것을 볼 수 있다. ELT 값이 0.25 이상 부터는 충분한 에러제한시간을 가지기 때문에 패킷 수신율이 기존의 TCP와 비슷한 것을 볼 수 있다.

3-3-2 전송 속도

인위적으로 에러를 발생시킨 실제 인터넷 망에서 서버에서 클라이언트로 고정된 크기(10MB)의 데이터를 전송하여 전송 속도를 측정 하였다. 수정된 TCP에 대해서는 ELT 시간을 변화시켜가며 테스트 하여 각각에 대한 전송 속도를 측정하였다. 표준 TCP와의 비교를 쉽게 하기 위해 표준 TCP를 100% 기준으로 하고, 나머지 항목을 그에 비례하도록 환산하여 도식 하였다.

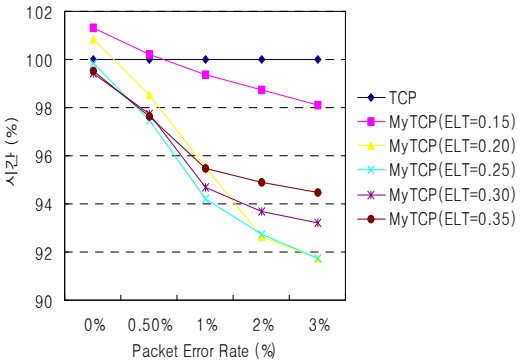

그림 7. 패킷 에러율에 따른 전송 시간 비교(실험 2)
Fig. 7 The comparison with packet error rate and transmission time(test 2)

수정된 TCP는 ELT 값이 0.15일 때 PER이 0%와 0.50% 일때는 표준 TCP에 비해 비교적 작은 재전송타임아웃시간으로 인해 세그먼트의 재전송 횟수가 많아지면서 상대적으로 전송시간이 오래 걸리게 되었고, PER이 1% 부터는 기존의 TCP보다 향상된 전송시간을 나타냈다. ELT 값이 0.20이고 PER이 0%일 때도 기존의 TCP에 비해 비교적 작은 재전송타임아웃시간으로 인해 세그먼트의 재전송 횟수가 많아지면서 상대적으로 전송시간이 오래 걸리게 되었고, PER이 0.50% 이상 부터 표준 TCP보다는 향상된 전송시간을 나타냈다. ELT값이 0.25 이상 부터는 표준 TCP보다 향상된 전송속도를 나타냈다. ELT 값이 0.35일 때 0.25 일 때 보다는 낮은 전송속도를 나타내고 있는데 이는 PER이 증가함에 따라 재전송 횟수가 많아지고 ELT 값이 증가하면서 재전송 타임아웃값도 커지기 때문이다.

3-3-3 패킷 수신율 비교

인위적으로 에러를 발생시킨 실제 인터넷 망에서 서버에서 클라이언트로 고정된 크기의 데이터를 전송하여 표준 TCP, UDP, 수정된 TCP의 패킷 수신율을 비교 측정 하였다. 이때 PER을 0%부터 3%까지 변화시켜가며 실험 하였다. 수정된 TCP에 대해서는 실험 1과 실험 2의 결과로 얻어낸 가장 좋은 전송시간을 갖고, 비교적 높은 패킷 수신율을 갖는 ELT 값으로써 0.2로 고정하여 실험하였다. 표준 TCP와의 비교를 쉽게 하기 위해 기존의 TCP를

100% 기준으로 하고, 나머지 항목을 그에 비례하도록 환산하여 도식 하였다.

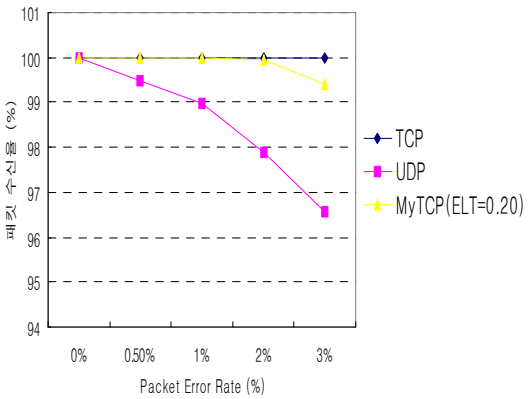

그림 8. 패킷 에러율에 따른 패킷 수신율 비교(실험 3)

Fig. 8 The comparison with packet error rate and receiving rate(test 3)

그림 8을 살펴보면 수정된 TCP는 실제 패킷 수신율이 기존 TCP보다는 낮지만 UDP에 비해 월등하게 향상된 것을 볼 수 있다. 이런 결과는 UDP에 비해 향상된 신뢰성을 보장 할 수 있다는 것을 보여준다.

Ⅳ. 결 론

TCP는 연결형 프로토콜로서 에러 제어 및 흐름 제어를 제공하여 신뢰성 있는 데이터 전송은 보장하지만 그에 따른 전송 지연이 발생한다. 반면에 UDP는 비연결형 프로토콜로서 에러 제어 및 흐름 제어를 수행하지 않으므로 데이터 전송에 있어 지연이 발생되지 않아 실시간 전송은 보장하나 신뢰성 있는 데이터 전송을 보장할 수 없다. 따라서 인터넷 전송계층 프로토콜에는 실시간 전송 기능을 지원하는 동시에 높은 수준의 신뢰성을 보장하여야 하는 데이터 전송에 적합한 프로토콜이 없다. 본 논문에서는 리눅스 커널 2.6.6 버전의 표준 TCP 스택의 흐름제어 및 에러제어를 수정, 제거하여 실시간 전송 기능을 지원하는 TCP를 설계하고 구현하였다.

표준 TCP와 UDP, 수정된 TCP를 실제 인터넷 망에서 에러를 인위적으로 가하여 전송시간 및 패킷 수신율을 비교 측정하여 다음과 같은 결론을 얻었다. 적절한 ELT값을 설정함으로써 표준 TCP에 비해 전송시간이 비교적 뚜렷이 향상 되었으며 UDP에 비해 높은 패킷 수신율을 나타냈다. 이런 결과는 수정된 TCP는 기존의 TCP보다 빠른 전송속도를 보이고 기존의 UDP에 비해 신뢰성이 향상된 것을 보여준다. 따라서 수정된 TCP를 사용하여 화상회의 및 동영상 데이터 전송에 있어 지연 없이 실시간으로 데이터를 전송할 수 있으며 높은 수준의 신뢰성을 보장할 수 있을 것으로 본다.

본 연구에서 설계된 실시간 전송기능을 지원하는 TCP 프로토콜은 기존의 TCP에 새로운 옵션을 추가함으로써, 일반 TCP와의 상호 운용을 지원하므로 실시간 전송 기능을 지원하지 않는 단말기에서는 기존의 TCP를 사용하면 되므로 실제 적용에 있어 무리가 없을 것으로 판단된다.

참 고 문 헌

[1] W. Richard Stevens, “*TCP/IP Illustrated, Volume1 : The Protocols*”, *Addison-Wesley*, pp. 223-322, 1999.

[2] Behrouz A. Forouzan, “*TCP/IP Protocol Suite, Second Edition*”, *McGraw-Hill*, pp. 297-335, 2003.

[3] J. Postel, “Transmission Control Protocol”, *IETF, RFC 793*, September 1981.

[4] W. Stevens, “TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms”, *IETF, RFC 2001*, January 1997.

[5] S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky, “An Extension to the Selective Acknowledgement (SACK) Option for TCP”, *IETF, RFC 2988*, November 2000.

[6] The Linux Kernel Archives 2.4.19, http://www.kernel.org/

[7] Daniel P. Bovet, Marco Cesati, “*Understanding the Linux Kernel*”, O'Reilly, pp. 138-157, pp. 233-248, 2001.

[8] W. Richard Stevens, “*UNIX Network Programming, Volume1, Second Edition*”, Prentice Hall PTR, pp. 85-140, 1998.

[9] Alessandro Rubini, Jonathan Corbet, “*Linux Device Drivers*”, O'Reilly, pp. 425-469, 2001.

우 정 만(禹貞萬)

2002년 2월 : 한밭대학교 공업화학과(공학사)
2005년 2월 : 한밭대학교 정보통신전문대학원 정보통신공학과(공학석사)
2005년 1월 ~ 현재 : (주)휴메이트

관심분야 : 인터넷프로토콜, 무선랜, 라우팅 프로토콜, 임베디드 등

조 성 언(趙誠彦)

1989년 2월 : 항공대학교 항공정보통신공학과(공학사)
1991년 2월 : 항공대학교 항공정보통신공학과(공학석사)
1997년 2월 : 항공대학교 항공전자공학과(공학박사)
1997년 3월 ~ 현재: 순천대학교 공과대학 정보통신공학과 부교수

관심분야 : 무선 통신 시스템, RFID, RTLS 등

김 은 기(金殷箕)

1987년 2월 : 고려대학교 전자공학과(공학사)
1989년 2월 : 고려대학교 전자공학과(공학석사)
1994년 2월 : 고려대학교 전자공학과(공학박사)
1995년 ~ 현재: 국립 한밭대학교 정보통신공학과 부교수

관심분야 : 인터넷프로토콜, 무선랜, 보안 프로토콜, 라우팅 프로토콜 등

권 영 도(權寧島)

1966년 2월 : 항공대학교 항공정보통신공학과(공학사)
1982년 2월 : 건국대학교 전자공학과(공학석사)
1977년 ~ 현재: 국립 한밭대학교 정보통신공학과 교수

관심분야 : 전자, 정보통신, 초고주파 등