실시간 운영체제에서 타이머를 이용한 TCP 오류 제어*

류현수[0], 성영락[†], 이철훈
충남대학교 컴퓨터공학과, [†] 국민대학교 전자정보통신공학부
{hsryu, hkyang, chlee}@ce.cnu.ac.kr, [†] yeong@mail.kookmin.ac.kr

Error Control in TCP Using Timers on Real-Time Operating Systems

Hyeon-Soo Ryu[0], Yeong Rak Seong[†], Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National Univ.
[†] School of Electrical Engineering, Kookmin Univ.

요 약

TCP(Transmission Control Protocol)는 신뢰성 있는 전송계층 프로토콜이다. 이것은 데이터 스트림을 TCP 로 전달하는 응용프로그램이 전체 스트림을 순서에 맞고 오류 없이 전달하는 것을 의미한다. TCP 는 오류 제어를 이용하여 신뢰성을 제공하는데, 오류제어는 손상 세그먼트, 손실 세그먼트, 순서가 어긋난 세그먼트, 그리고 중복 세그먼트를 감지하는 메커니즘이 포함되며 특히 타이머(timer)를 이용한 오류제어를 본 내용에서 설명하고 있다.

1. 서론

TCP 세그먼트의 오류제어는 검사합(checksum), 확인(acknowledgement), 그리고 시간-초과 등 세가지 방법을 통해 수행된다. 각 세그먼트의 검사합 필드는 세그먼트가 손상되었는지를 검사하기 위해 사용된다. 만일 세그먼트가 손상되면, 수신측은 손상 세그먼트로 판명하며 그 세그먼트를 버린다. TCP 에서는 수신측 TCP 가 손상되지 않은 세그먼트를 수신하였다는 것을 송신측에 알려주는 확인-응답 방식을 이용한다. TCP 에서는 부정 확인(Negative acknowledgement)을 사용하지 않으며, 만일 세그먼트가 시간-초과 전까지 확인되지 않으면, 송신 TCP 는 그 세그먼트를 손상되었거나 손실되었다고 간주한다.

TCP 에 의해서 사용되는 오류 제어 방법은 간단하다. 발신지 TCP 는 전송되는 각 세그먼트마다 하나의 시간-초과 카운터를 실행한 후, 각 카운터를 주기적으로 확인한다. 만일 카운터가 설정된 값에 도달하면 해당 세그먼트는 손상되었거나 손실되었다고 간주하고 재전송된다.

본 논문에서는 타이머를 이용한 오류제어를 실시간 운영체제의 TCP/IP 스택에 좀더 효율적으로 적용하였다. 본 논문의 구성은 2 장에서 관련 연구를, 3 장에서는 타이머를 이용한 오류제어, 4 장에서는 테스트 환경 및 결과를, 5 장에서는 결론 및 향후 연구 과제에 대해 기술한다.

2. 관련 연구

TCP 세그먼트 오류제어를 위해 재전송(Retransmission), 영속(Persistence), 킵얼라이브(Keep-alive), 시간-대기(Time-waited) 타이머가 사용된다.

2.1 재전송 타이머

TCP 는 손실 또는 손상된 세그먼트를 처리하기 위해 TCP 는 재전송 타이머(retransmission timer)를 이용한다. 재전송 타이머는 재전송 시간을 두어 처리하는데, 재전송 시간은 한 세그먼트를 전송하고 확인응답을 기다리는 시간을 말한다. TCP 는 세그먼트를 전송한 후, 해당 세그먼트를 위한 재전송 타이머를 구동한다. 다음과 같은 두 가지의 상황이 일어날 수 있다.

- 타이머가 만료되기 전에 세그먼트에 대한 확인응답이 도착하면 해당 타이머는 소멸된다.
- 확인응답이 수신되기 전에 타이머가 만료되면 해당 세그먼트는 재전송되고 타이머는 초기화 된다.

2.2 영속 타이머

TCP 는 윈도우의 크기가 0 인 경우 사용하기 위해 영속 타이머를 두고 있다. 만약 수신 윈도우 크기가 0 이라면 송신 TCP 는 수신 TCP 로부터 0 이 아닌 윈도우 크기를 알리는 확인응답을 수신하기 전까지 세그먼트 전송을 보류하게 된다. 이때 이러한 확인응답이 손실되었다면 송수신 TCP 가 서로를 기다리는 교착상태(deadlock)가 발생하게 되는데, 이를 해결하기 위하여 TCP 는 각 연결마다 하나의 영속 타이머를 사용한다. 송신 TCP 가 0 인 윈도우 크기를 가진 확인응답을 수신하면, 송신 TCP 는 영속 타이머를 구동하고, 영속타이머가 종료되면, 송신 TCP 는 프로브(probe)라고 하는 특수한 세그먼트를 전송함으로써 수신 TCP 에게 확인응답이 손실되었음을 알리고 확인응답을 재전송하도록 한다.

* 본 논문은 산업자원부 중기거점과제 연구비 지원에 의한 것임

2.3 킵얼라이브 타이머

킵얼라이브 타이머는 두 TCP 사이에 설정된 연결이 오랜 기간동안 정지(idle)상태에 있는 것을 방지하기 위하여 사용된다. 한 클라이언트가 서버로 TCP 연결을 설정하고, 데이터를 전송한 후, 더 이상의 데이터 전송이 없이 클라이언트에 고장이 발생한 경우 TCP 연결은 연결된 상태로 계속 유지하게 된다.

이와 같은 문제를 해결하기 위해 서버에는 킵얼라이브 타이머가 구현되어 있으며, 서버는 세그먼트를 받을 때마다 타이머를 초기화한다. 타이머가 만료되면 서버는 프로브 세그먼트를 전송하고 이에 대한 응답이 없으면 연결을 종료한다.

2.4 시간-대기 타이머

TCP 는 연결을 종료할 때 시간-대기 타이머를 사용하여 그 기간동안 연결을 유지한다. 이 기간동안 목적지에 중복 FIN 세그먼트가 도착하면, 이 세그먼트를 버린다.

3. 타이머를 이용한 오류 제어

3.1 재전송 시간의 계산

TCP 프로토콜은 전송 계층이다. 각 연결은 두 개의 TCP 를 연결하는데 각기 상이한 네트워크에 따라 경로의 길이가 판이하게 다를 수 있다. 이것은 TCP 가 동일한 재전송 시간을 모든 연결에 일률적으로 적용할 수 없다는 것을 의미한다. 만일 충분한 재전송 시간을 설정하지 않으면, 전달중인 세그먼트가 재전송되는 결과를 초래하게 되며, 반대로 재전송 시간이 너무 길게 설정되면 응용프로그램의 지연을 가져올 수 있다.

단일 연결의 경우에도 연결망의 상태에 따라서 전송시간이 가변적이기 때문에 재전송 시간을 동적으로 할당해 주어야 한다. 보통 재전송 시간은 왕복 시간을 기반으로 하여 동적으로 계산되며, 가장 보편적인 방식은 재전송 시간을 RTT (round trip time)의 두 배로 한다.

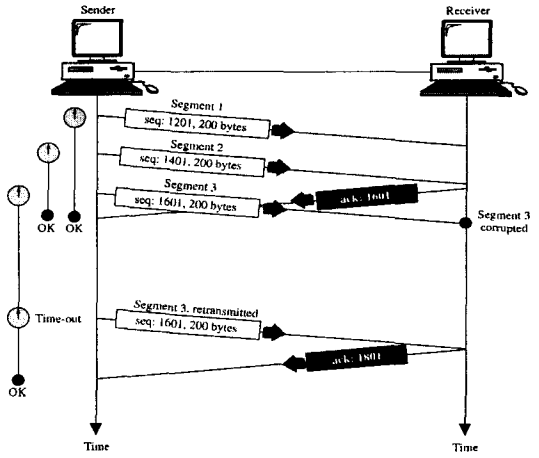
$$\text{재전송 시간} = 2 \times \text{RTT}$$

3.2 RTT 계산

RTT 를 계산하는 방법은 두 가지가 있다. 첫째는 타임스탬프 옵션을 사용하는 방법이고, 둘째는 TCP 가 세그먼트를 전송하고 타이머를 구동하여 확인응답을 기다리는 방법이다. 다음 세그먼트의 재전송 시간을 계산하는데 사용되는 RTT 는 다음과 같다.

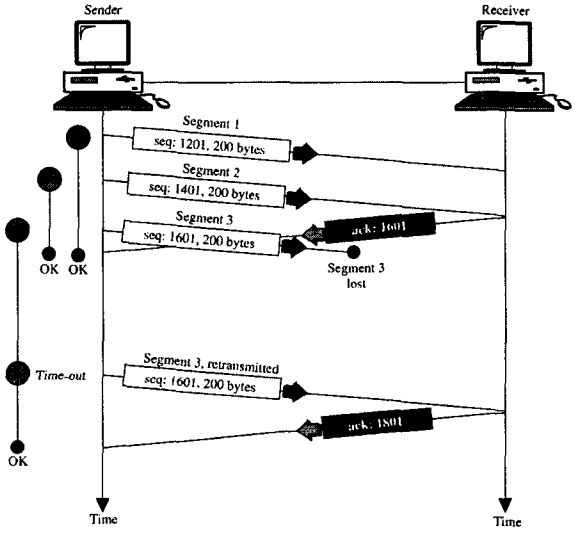
$$\text{RTT} = \alpha \times \text{이전 RTT} + (1-\alpha)\ \text{현재 RTT}$$

α 의 값은 보통 90 퍼센트이다. 이것은 새로운 RTT 는 이전 RTT 값의 90 퍼센트와 현재 RTT 값의 10 퍼센트를 합한 값이 된다.



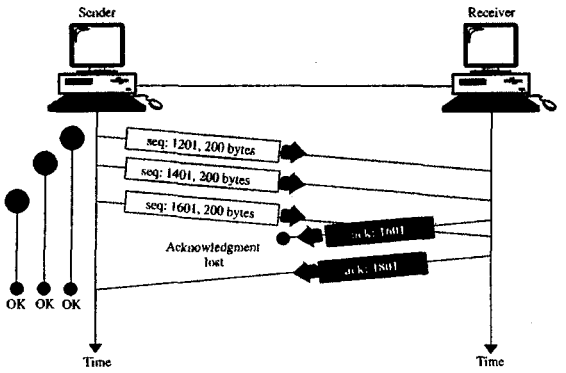
[그림 1] 훼손 세그먼트에 대한 오류제어

[그림 1]는 훼손 세그먼트에 대한 오류정정 메커니즘을 보이고 있다. 세그먼트 3 의 경우 수신측에 훼손된 정보가 도착을 하고 검사합을 검사한 결과 오류로 판정, 확인 응답 메시지 전송을 중단한다. 처음 세그먼트에 대한 타이머가 만료되고 세그먼트를 재전송한다.



[그림 2] 손실 세그먼트에 대한 오류제어

[그림 2]은 손실 세그먼트에 대한 오류정정 메커니즘을 보이고 있다. 세그먼트 3 이 전송 중에 분실된 경우, 타이머가 만료된 후에 해당 세그먼트를 재전송한다.



[그림 3]손실 ACK에 대한 오류제어

[그림 3]는 손실 ACK 에 대한 오류정정 메커니즘을 보이고 있다. ACK 세그먼트가 타이머 만료시간 안에 도착하지 않을 경우, 처음 세그먼트가 재전송되며, 수신측에서는 세그먼트의 중복 수신을 확인하여 처리한다.

[그림 4]는 타이머에 대한 소스 코드이다.

```
PROCESS tcptimer()
{
   lastrun = Ticks_10ms; /* initialize to "now" */
   tqmutex = MK_SemaphoreCreate(MK_SEM_PRIO,1);
   /* mutual exclusion semaphore */
   tqpid = MK_GetCurrentPID(); /* record timer process id */
   MK_SemaphorePost(Net.sema); /* start other network processes*/
   while(TRUE){
     MK_SleepTicks(TIMERGRAN);     /* real-time delay*/
     if (tqhead == 0)      /* block timer process if delta */
           MK_Suspend(tqpid);
     MK_SemaphorePend(tqmutex, MK_WAIT_FOREVER);
     now = Ticks_10ms;
     delta = now – lastrun;              /* compute elapased time*/
     if (delta<0 || delta>TIMERGRAN*100)
       delta = TIMERGRAN*10;
     lastrun = now;
   while(tqhead !=0 && tqhead->tq_timeleft <= delta){
     delta -= tqhead->tq_timeleft;
     if(MK_MessageQueueCount(tqhead->tq_msgq)<=
           tqhead->tq_msgqlen)
        MK_MessageQueuePost(tqhead->
           tq_msgq, tqhead->tq_msg,4,MK_WAIT_FOREVER);
        tq = tqhead;
        tqhead = tqhead->tq_next;
        MK_FreeMemory(tq,sizeof(struct tqent));
     }
     if(tqhead)   tqhead->tq_timeleft +=delta;
     MK_SemaphorePost(tqmutex);
   }
}
```

[그림 4] Source Code : 타이머

4. 테스트 환경 및 결과

위에 기술한 네 개의 타이머는 본 연구팀에서 개발한 실시간 운영체제인 iRTOS™ 의 TCP/IP 스택에 구현하였으며, ARM920T 32-bit RICS CPU가 탑재되어 있는 SMDK2400 보드에서 SDT2.51 통합 개발 환경을 사용하여 테스트하였다. 테스트 결과 안정적으로 패킷이 송수신됨을 확인할 수 있었다.

5. 결론 및 향후 연구 과제

본 논문에서는 실시간 운영체제의 TCP/IP 스택에서 TCP 계층의 효율적 오류제어를 위한 4 개의 타이머를 구현하였다. 차후에 좀더 많은 오류상황에 대한 테스트를 거쳐 이에 대한 보완이 연구되어야 하겠다. 그리고 실시간 운영체제에서의 TCP/IP 를 위한 효율적 오류 제어 및 흐름 제어에 대한 연구가 진행되어야 하겠다.

6. 참고문헌

[1] http://www.inestech.com

[2] David Stepner, *et. al.*, , " Embedded Application Design Using a Real-Time OS" , IEEE, 1999.

[3] Jean, J. Labrosse, *μ C/OS The Real-Time Kernel*, R&D Publications, 1995.

[4] 정충희 " 실시간 운영체제 상에서의 TCP/IP 구현" 충남대학교 석사학위논문, 2002.

[5] Behrouz A. Forouzan., TCP/IP Protocol Suite, MRC 미래컴, 2000

[6] Douglas E. Comer David L. Stevens, Internetworking with TCP/IP, Prentice Hall, 1995