혼잡제어를 하지 않는 TCP의 성능

오 홍 균[†] · 김 은 기[††]

요 약

본 논문에서는 RFC 규격을 준수하는 정상적인 TCP와 속도를 제한하는 여러 제약을 무시한 TCP 간의 성능을 비교 하였다. 이를 위하여, TCP의 속도에 영향을 미치는 주요 항목들을 결정 하였으며, 리눅스 커널의 TCP 소스에서 결정된 항목들을 제거하고 정상적인 TCP와의 성능을 비교하였다. 본 연구 결과를 살펴보면, 목적지가 근거리인 경우 패킷 에러가 발생하지 않는다면 정상적인 TCP와 본 연구에서 수정된 TCP 간의 파일 전송 시간의 차이가 그다지 크게 나지 않았다. 하지만, 원거리에 있는 목적지로 웹 페이지와 같은 작은 크기의 파일을 전송할 경우, 혼잡제어 메커니즘 중 저속 출발(slow start)을 적용하지 않았을 때는 전송 시간에서 매우 큰 차이를 나타냈다. 또한, 어느 정도의 패킷 에러가 발생되는 환경에서는 목적지가 근거리이건 원거리이건 혼잡제어를 하지 않도록 수정된 TCP가 RFC 규격을 준수하는 표준 TCP에 비해 빠른 전송 속도를 나타냈다.

Performance of TCP without congestion control

Hong Kyun Oh[†] · Eun Gi Kim[††]

ABSTRACT

In this study, the performance is compared between RFC compatible normal TCP and several speed constraints ignored TCP. To do these, the main algorithms that constraints the transmit rate of TCP are modified. We have modified TCP protocol stack in a Linux kernel to compare the speeds between the standard TCP and our modified TCP. We find that if the destination is short distance away from the source and packet error is scarce then the speed differences between normal and modified TCP may be negligible. However, if the destination is far away from the source and slow start algorithm is not adopted then the transfer time for small file is different greatly. In addition, if packet error occurred frequently, our modified TCP is faster than the standard TCP regardless of distance.

키워드 : TCP, RFC, 혼잡제어(Congestion Control), 저속출발(Slow Start), 혼잡회피(Congestion Avoidance), 빠른 재전송(Fast Retransmission), RTO

1. 서 론

TCP/IP 프로토콜의 상위 계층은 전송 계층과 응용 계층의 두 계층으로 구성된다. 전송 계층은 연결 지향의 안정된 데이터 전송을 지원하는 TCP와 비연결형의 신속한 데이터 전송을 지원하는 UDP로 구성된다[1]. 전송 계층에 해당하는 TCP는 RFC793에 정의되어 있으며, 신뢰성 있는 서비스를 제공하기 위한 사항들이 여러 RFC에 기술되어 있다[9-13]. 이런 요구 사항들은 TCP가 종단 호스트와 데이터를 주고 받을 때, 호스트의 상태나 망의 상태를 파악하고 송수신 속도를 적절히 제어하여 망의 성능을 저하 시키지 않고 원활하게 동작할 수 있도록 한다[2, 3].

하지만, RFC에 정의된 인터넷 프로토콜 규격은 단순한 권고사항 일뿐 강제적인 요구사항이 아니다. 만약, 특정 회사에서 자신의 필요에 따라 RFC 규격을 준수하지 않는 - 예를 들면, 혼잡제어를 하지 않는 등 - TCP를 구현하여 동작시킬 경우 망의 혼잡을 증가시키는 요소가 될 수 있다. 그렇지만, 규격을 무시한 TCP 프로토콜이 망에 미치는 영향이나 속도 향상 등에 관한 연구는 이루어지고 있지 않다.

본 논문에서는 규격을 무시한 TCP 프로토콜의 성능과 그것이 망에 미칠 수 있는 영향을 알아보기 위해 다음과 같은 요구를 지원하는 TCP를 설계하고 구현하여, 그 성능을 분석하였다.

첫째, TCP가 최대의 속도로 동작할 수 있도록 한다. 이를 위하여, 필요한 경우 TCP의 성능을 저하시킬 수 있는 RFC의 요구 사항들을 무시한다.

둘째, 이미 구현되어 있는 TCP와의 상호 운용(interoperability)을 지원한다.

본 논문의 나머지 부분은 다음과 같이 구성되어 있다. 2장 에서는 혼잡제어를 하지 않는 TCP 설계에 필요한 메커니즘들을 정리 하였고, 3장에서는 결정된 각각의 메커니즘

† 준 회 원 : 국립 한밭대학교 대학원 정보통신 · 컴퓨터공학과
†† 정 회 원 : 국립 한밭대학교 정보통신 · 컴퓨터공학부 교수
논문접수 : 2003년 12월 20일, 심사완료 : 2004년 3월 2일

들을 적용한 TCP를 사용하여 실제 인터넷망에서 테스트를 수행하고, 그 결과를 분석하였다. 마지막으로 4장에서는 본 논문의 결론을 기술하였다.

2. 혼잡제어를 하지 않는 TCP의 설계

TCP의 전송 속도에 영향을 미치는 주요 메커니즘들로는 지연된 확인응답(Delayed Acknowledgement), Nagle 알고리즘, 어리석은 윈도우 신드롬 회피(Silly Window Syndrome Avoidance), 슬라이딩 윈도우, 저속 출발(Slow Start), 혼잡 회피 알고리즘, 빠른 재전송과 빠른 복구 알고리즘, 재전송 타임아웃(RTO : Retransmission Timeout) 등이 있다[2, 3].

TCP의 다양한 메커니즘들 중 본 논문에서 관심을 갖는 메커니즘은 대용량 데이터 세그먼트의 전송 속도에 많은 영향을 미치는 것들로써, 다음과 같은 항목들을 수정하였다.

- 저속 출발의 제거 : NoSlowStart(NSS)

저속 출발 메커니즘을 사용하지 않는다. 혼잡 윈도우(cwnd : congestion window)와 광고된 윈도우중 최소값으로 송신 세그먼트의 양을 제한하는 기존의 방식을 오직 광고된 윈도우 만을 사용하여 제한하도록 한다.

- 혼잡회피 알고리즘의 제거 : NoCongAvoid(NCA)

정상적인 TCP의 경우, 저속 출발을 진행하며 송신 세그먼트의 수를 지수적으로 증가 시키다가 cwnd값이 이전 혼잡 발생시 설정된 ssthresh(slow start threshold)값보다 커지게 되면 지수적인 증가에서 가산적인 증가로 변하여 송신 세그먼트의 양을 줄여 미리 혼잡을 회피 하였다. 그러나, 본 연구에서는 cwnd값이 ssthresh값보다 커지더라도 계속해서 저속 출발의 진행을 수행하여 송신 세그먼트의 양을 지수적으로 증가 시키도록 한다. 즉, NCA는 NSS와는 다르게 cwnd값을 계속 사용하지만 ssthresh값과는 무관하게 송신 세그먼트의 양을 지수적으로 증가시킨다.

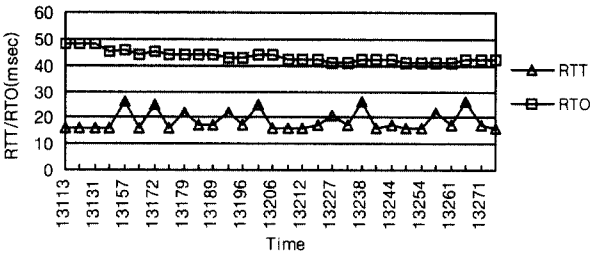
- 빠른 재전송의 변경 : ModifiedFastRetx(MFST)

분실된 세그먼트에 대해 3개의 중복된 확인 응답을 받았을 경우 해당 세그먼트를 재전송하던 기존의 방법을, 1개의 중복된 확인 응답을 받을 경우 즉시 재전송하도록 변경한다.

- 재전송 타임아웃의 변경 : ModifiedRTO(MRTO)

기존 방법으로 계산된 RTO는 (그림 1)에 나타난 바와 같이 세그먼트의 왕복시간인 RTT와 비교했을 때 약 2배 이상 높은 값을 갖는다. 그러므로 분실된 세그먼트에 대해서 타임 아웃이 발생하기까지는 너무 오랜 시간이 소요된다.

따라서, 본 논문에서는 RFC793에서 권고한 재전송 타임아웃의 계산식에서 β를 1로 설정하여 RTO의 값을 RTT와 비슷한 수준으로 맞추었다.



(그림 1) 표준 TCP에서의 RTT와 RTO 예

위에서 선택된 4가지 메커니즘들의 가능한 조합을 만들어, 표준 TCP를 포함한 12가지의 테스트 항목을 설정하였으며, <표 1>에서 이를 보여주고 있다.

〈표 1〉 성능 분석을 위한 항목들

번 호	항 목
항 목 1	ORG(Original)
항 목 2	NSS
항 목 3	MRTO
항 목 4	NCA
항 목 5	MFST
항 목 6	NSS+MRTO
항 목 7	NSS+FST
항 목 8	MRTO+NCA
항 목 9	MRTO+FST
항 목 10	NCA+FST
항 목 11	NSS+MRTO+FST
항 목 12	NCA+MRTO+FST

3. 고속 TCP의 성능 분석

설계된 고속 TCP의 성능 분석을 위하여 리눅스 2.4.19 커널의 TCP 프로토콜을 <표 1>의 각 항목에 맞게 수정하였다[4, 5, 7]. 테스트는 실제 인터넷 망에서 수행 하였으며, 수정된 고속 TCP를 클라이언트로 하였고, RFC 규격을 준수하는 표준 TCP의 discard 서버로 파일을 전송하였다. 이때 각 항목마다 독립적으로 테스트를 수행하였으며, 전송되는 파일의 크기마다 120회씩 테스트하였다. 전송 시간은 연결 설정을 위한 3-way handshake가 끝나고 첫 번째 데이터 세그먼트를 보내는 시점부터 측정하였으며 정확한 전송 시간을 측정하기 위해 tcpdump를 수정하였다[5-7].

테스트 시나리오는 다음과 같이 크게 두 가지 경우로 나누었다.

첫째, 실제의 망에서 <표 1>의 각 항목 별로 파일 전송 시간을 측정하였다. 이때 서버와의 거리는 홉 수(hop count)가 1인 근거리와 홉 수가 22인 원거리(galaxy.CS.Berkeley.EDU)를 구분하여 테스트 하였다.

둘째, 인위적으로 에러를 발생시킨 실제의 망에서, <표 1>의 각 항목 별로 파일 전송 시간을 측정하였으며, 패킷

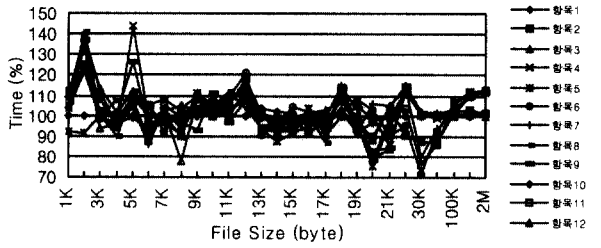
에러 비율(PER : Packet Error Rate)은 0%부터 3%까지 변화 시켜가며 테스트 하였다. 이때도 역시 서버와의 거리는 근거리와 원거리를 구분하여 테스트 하였다.

3.1 실제 인터넷 망에서의 테스트

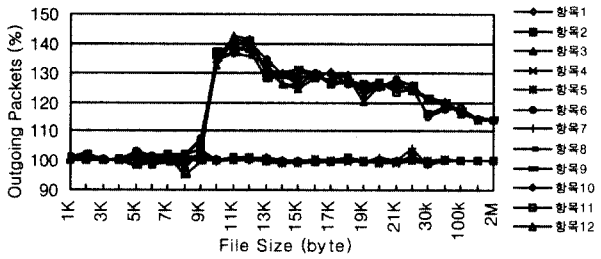
우선 인위적인 에러를 가하지 않은 실제 망에서, 근거리와 원거리에 위치한 discard(잘 알려진 포트번호 9) 서버로 다양한 크기의 파일을 전송하여, 각 항목 마다 전송 시간을 측정하였다. 아래 모든 경우의 그래프는 RFC 규격을 준수하는 TCP인 항목1(ORG)을 100% 기준으로 하고, 나머지 항목을 그에 비례하도록 환산하여 도식하였다.

3.1.1 근거리에 위치한 서버로의 파일 전송

근거리에 위치한 discard 서버로의 파일 전송 시간을 (그림 2)에 나타내었고, 그에 따른 송신 패킷 수를 (그림 3)에 나타내었다. 파일 크기는 1KB부터 2MB까지 변경하였다.



(그림 2) 파일 크기에 따른 전송 시간(근거리)



(그림 3) 파일 크기에 따른 송신 패킷수(근거리)

(그림 2)에 나타난 바와 같이 파일 크기가 100KB 이하의 비교적 작은 파일은 각 항목에 따른 전송 시간의 차이가 뚜렷하게 나타나지 않았다. 이는, 근거리는 RTT가 매우 작기 때문에 cwnd 값에 해당하는 세그먼트를 모두 송신하기 전에 이전 세그먼트에 대한 확인 응답이 도착하여 cwnd 값이 증가하게 되고, 클라이언트는 대기시간 없이 바로 다음 세그먼트를 송신할 수 있기 때문이다.

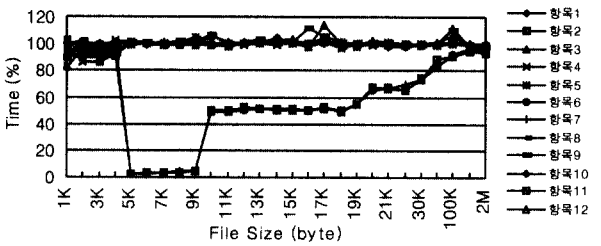
1MB, 2MB처럼 파일 크기가 큰 경우에는 전송 시간의 차이가 뚜렷이 나타나고 있다. ORG에 비해 높은 전송 시간을 보이는 것은 모두 MRTO를 적용한 항목들(항목 3, 6, 8, 9, 11, 12)이다. 이렇게 높은 전송 시간을 갖는 이유는 MRTO를 적용하여 작게 계산된 재전송 타임아웃으로 인해 세그먼트의 재전송 횟수가 많아 지면서 상대적으로 전송 시간이 오래 걸리기 때문이다.

파일 크기가 10KB(10240바이트)일 때부터 갑자기 전송되는 세그먼트의 수가 증가하는 이유는 discard 서버의 수신 버퍼 용량(10136바이트)의 한계에 의한 지연 시간 때문에 확인 응답이 늦어지고, 이때 클라이언트에서는 타임 아웃이 발생하여 세그먼트가 재전송 되기 때문이다.

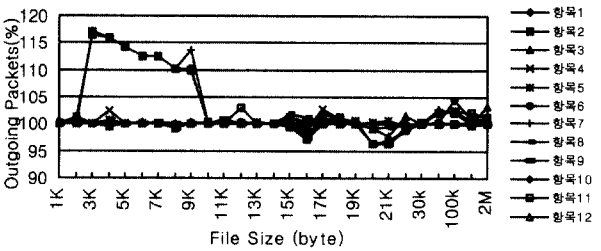
결론적으로, 에러 없는 환경에서 근거리와 같이 RTT가 매우 작을 때, 크기가 작은 파일 전송에서 각 항목에 따른 전송 시간의 차이는 그다지 크게 나지 않는다는 것을 알 수 있다.

3.1.2 원거리에 위치한 서버로의 파일 전송

(그림 4)와 (그림 5)는 원거리에 위치한 discard 서버로 파일을 전송할 때, 전송 시간과 그에 따른 송신 패킷 수를 각각 나타내고 있다.

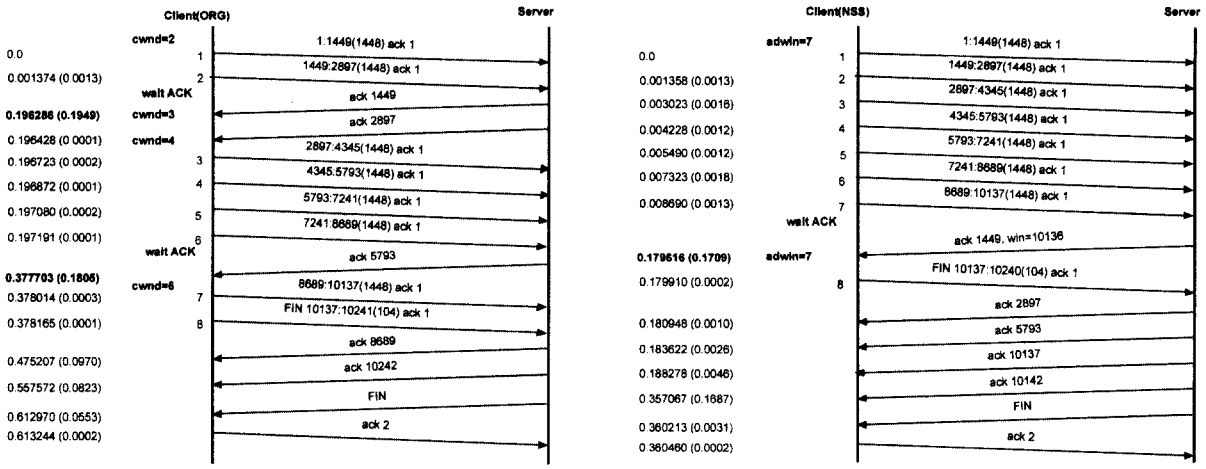


(그림 4) 파일 크기에 따른 전송 시간(원거리)



(그림 5) 파일 크기에 따른 송신 패킷수(원거리)

(그림 4)는 (그림 2)와 다르게 NSS가 적용된 항목들(항목 2, 6, 7, 11)의 전송 시간이 다른 항목들에 비해 월등하게 빠른 것을 볼 수 있다. 5KB에서 9KB사이의 파일 전송에서는 무려 98%정도의 전송 시간 감소율을 보이고 있다. 이렇게 큰 전송 시간의 차이를 보이는 이유는 다음과 같다. 테스트 환경에서의 MSS(maximum segment size)는 1448바이트 이고, 서버의 광고된 윈도우 크기(adwin)는 7(10136/1448)이다. 이때, 5KB에서 9KB의 파일을 전송하기 위해서는 최대7(9KB/1448B)개의 세그먼트가 필요하게 된다. (그림 6)은 표준 TCP와 NSS를 적용한 TCP에서 원거리에 있는 discard 서버로 10KB의 파일을 전송할 때의 타임라인을 보여주고 있다. 리눅스의 초기 cwnd값은 2로 설정되므로, (그림



(a) 표준(ORG) TCP 클라이언트 (b) NSS를 적용한 TCP 클라이언트

(그림 6) 원거리에 위치한 서버로의 파일 전송 예(10K바이트)

6)(a)에서 보는 바와 같이 초기에 클라이언트가 서버의 확인 응답 없이 한번에 전송할 수 있는 세그먼트의 수는 광고된 윈도우 크기와 cwnd값 중 최소값인 2가 된다. 따라서 NSS가 적용되지 않은 항목들은 초기에 2개의 세그먼트를 송신하고, ACK를 받기 위해 대기하여야 하지만, NSS가 적용된 항목들은 (그림 6)(b)와 같이 ACK의 수신을 기다리지 않고 광고된 윈도우 크기만큼의 세그먼트를 연속해서 송신할 수 있기 때문에 전송 시간이 크게 줄어들게 된다.

4K(4096바이트) 이하의 파일은 최대 3개의 세그먼트가 필요하게 되지만, cwnd의 초기값이 증가하지 않은 상태에서 마지막 세그먼트만 남았을 경우, ACK를 받지 않고도 연결의 종료를 알리는 FIN 플래그를 설정하여 바로 송신하게 되어 모든 항목에서 파일 전송 시간의 차이가 뚜렷이 나타나지 않는다.

NSS가 적용된 경우, 10KB에서 전송 시간이 갑자기 높아진 이유는 (그림 6)(b)에 나타난 것처럼 10KB의 파일을 보내기 위해서는 최소 8개의 세그먼트를 송신하여야 하지만 광고된 윈도우 크기(adwin = 7)에 제한을 받아 마지막 8번째 세그먼트를 전송 하기 전에 ACK를 받기 위한 대기 시간 때문이다. 그러므로, 파일의 크기가 커질수록 ACK를 받기 위해 대기하는 횟수가 많아지게 되어, 전송 시간이 ORG에 근접하게 변하는 것을 볼 수 있다.

(그림 5)의 송신 패킷 수를 보면 3KB에서 9KB파일 전송 시, NSS가 적용된 항목의 송신 패킷 수가 다른 항목들에 비해 많은 것을 볼 수 있다. 이유는, NSS를 적용하지 않은 항목에서는 대기시간 동안 합쳐진 마지막 데이터 세그먼트와 FIN 세그먼트가 하나의 세그먼트로 전송 되지만, NSS를 적용한 항목에서는 대기 시간이 없기 때문에 FIN 세그먼트와 데이터 세그먼트를 각각 따로 전송하여 다른 항목들 보다 1개의 세그먼트를 더 전송하게 된다. 그러나, 10KB부터는 모든 항목이 마지막 데이터 세그먼트와 FIN 세그먼트를 함께 설정하여 보내기 때문에 차이가 크게 나타나지 않는다.

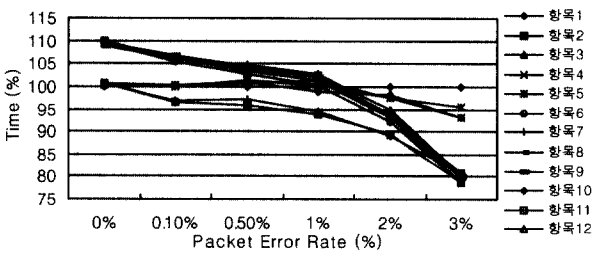
결론적으로, 원거리와 같이 RTT가 큰 환경에서 웹 페이지와 같은 작은 크기의 파일을 전송하는 경우, 저속 출발을 수행하지 않는 TCP가 표준 TCP에 비해 전송 시간에 대해 큰 이득을 얻을 수 있다.

3.2 인위적인 에러가 있는 인터넷 망에서의 테스트

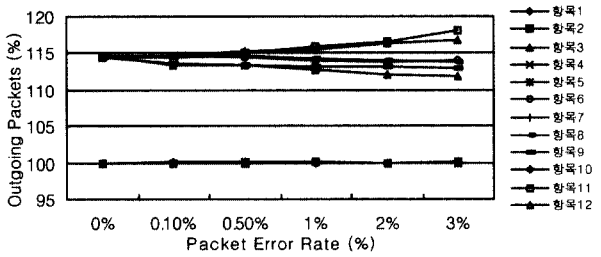
패킷의 에러가 존재하는 망에서 1MB의 파일을 전송할 때, 각 항목에 따른 전송 시간을 테스트 하여 보았다. 일정한 비율의 에러가 발생하는 실제의 망 상황을 만들기 위하여 리눅스 드라이버 단에서 필요한 경우 랜덤하게 패킷을 제거하도록 하였다. PER은 0%부터 3%까지 변화시켜 가며 테스트 하였다. PER 0%는 인위적인 패킷 제거가 없고, 패킷 에러가 거의 발생하지 않고 있는 실제의 인터넷 환경을 의미한다.

3.2.1 근거리에 위치한 서버로의 파일 전송

(그림 7)과 (그림 8)은 근거리에 위치한 discard(잘 알려진 포트번호 9) 서버 데몬으로 파일을 전송할 때 PER에 따른 전송 시간과 전송되는 패킷수의 변화를 나타내고 있다.



(그림 7) PER에 따른 전송 시간(근거리)



(그림 8) PER에 따른 송신 패킷수(근거리)

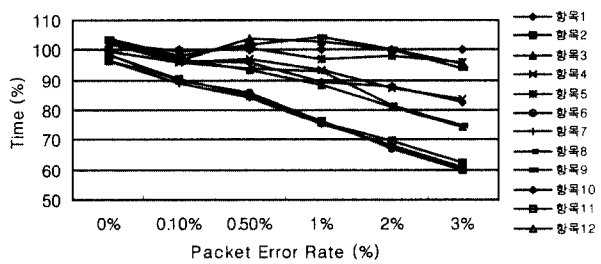
(그림 7)을 보면, PER이 0%일 때는 (그림 2)에서 1MB 파일을 전송할 때와 같은 이유로 MRTO를 적용한 항목들과 그렇지 않은 항목들 사이에 차이를 보이고 있다. 이후, PER이 높아 지면서 모든 항목이 항목 1(ORG)에 비해 점점 빠른 전송 시간을 보이고 있다. 특히, NSS가 적용된 항목 2와 항목 7은 3% PER에서 약 20% 이상의 감소를 보였다. 그럼에도 불구하고 (그림 8)에서 보는 바와 같이 송신 패킷의 수는 ORG와 거의 차이를 보이지 않았다.

MRTO가 적용된 항목들(항목 3, 6, 8, 9, 11, 12)도 PER이 높아지면서 전송 시간의 감소를 보이다가 3% PER에서는 NSS가 적용된 항목들과 거의 같은 수준의 전송 시간을 보였다. 하지만, (그림 8)에서 볼 수 있는 것처럼, 송신된 패킷의 수는 빈번한 타임아웃에 의해 ORG에 비해 평균 15%정도 많은 세그먼트를 전송하였다. 혼잡 회피를 제거한 (NCA) 항목 4와 항목 10은 3% PER 지점에서 약 7%의 전송시간 감소를 가져왔으며, 항목 5(MFST)는 약 4%의 전송시간 감소를 가져왔지만, 이들의 송신 패킷 수는 ORG와 거의 같은 수준을 보였다.

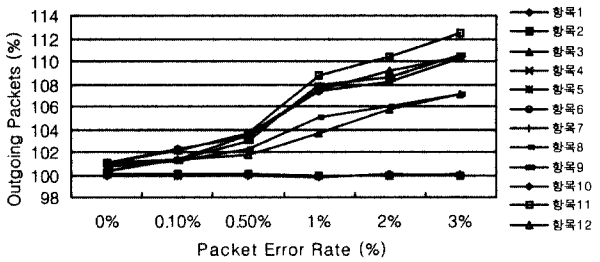
결론적으로, 근거리와 같이 RTT가 매우 작더라도 패킷의 에러가 어느 정도 발생되는 환경에서는 혼잡제어를 무시한 모든 항목이 표준 TCP에 비해 빠른 속도를 갖는 것을 볼 수 있다.

3.2.2 원거리에 위치한 서버로의 파일 전송

원거리에 위치한 discard(잘 알려진 포트번호 9) 서버 데몬으로 패킷 에러가 발생하는 환경에서 테스트를 수행 하였다. (그림 9)는 각 항목에 대한 파일 전송 시간을 나타내고, (그림 10)은 송신되는 패킷의 수를 나타낸다.



(그림 9) PER에 따른 전송 시간(원거리)



(그림 10) PER에 따른 송신 패킷수(원거리)

(그림 9)를 보면 PER이 높아지면서 각 항목에 따라 현저한 전송 시간의 차이를 보이고 있다. NSS가 적용된 항목들(항목 2, 6, 7, 11)은 PER이 높아지면서 전송 시간이 급격히 감소하여, 3% PER에서는 ORG에 비해 약 40% 빠른 전송 시간을 보였다. 혼잡 회피를 하지 않는 NCA와 MRTO가 결합된 항목 8과 항목 12는 약 25%의 감소율을 보였고, NCA가 적용된 항목 4와 항목 10은 약 15%의 전송시간 감소율을 보였다. 나머지 항목 3, 항목 5, 항목 9는 ORG에 비해 약 5%의 감소율을 보였다.

(그림 10)의 송신 패킷 수를 분석해 보면, 3% PER일 때 MRTO가 적용되지 않은 항목들은 ORG와 거의 같은 수준의 패킷 수를 보였지만, MRTO가 적용된 항목들은 ORG에 비해 최소 7%에서 최대 12% 이상 많은 패킷을 송신하였다.

NSS가 적용된 항목들이 ORG에 비해 빠른 전송 시간을 갖는 이유는 패킷의 오류로 인하여 재전송이 발생될 때, ORG의 경우는 재전송이 발생되고 난 후 패킷 송신율을 줄이게 되지만, NSS가 적용된 경우는 광고된 윈도우 크기만큼 최대의 세그먼트를 계속해서 전송할 수 있기 때문이다. 따라서, 패킷의 오류나 분실이 빈번히 발생 할수록 NSS가 적용된 항목은 ORG에 비해 더 빠른 전송 속도를 갖게 된다.

또한, 혼잡 회피를 하지 않는 NCA가 적용된 항목들은 전송 가능한 세그먼트의 수가 ssthresh 이후에도 계속해서 지수 함수적인 증가를 하게 되지만, cwnd에 의한 제한을 받기 때문에 NSS가 적용된 항목보다는 전송 시간이 다소 길어지게 된다.

결론적으로, RTT가 큰 환경에서 패킷 에러 율이 높아질수록 모든 항목에서 파일 전송 시간이 크게 줄어 들었다. 하지만 어느 항목에서는 파일의 전송 시간이 줄어드는 반면 송신되는 패킷의 수가 증가하여 망을 혼잡에 빠뜨리는 문제점이 발생되었다.

4. 결 론

본 논문에서는 리눅스 TCP를 수정하여, RFC에서 권고하고 있는 혼잡제어와 관련된 사항들을 무시하고 최대한 고속

으로 데이터를 전송할 수 있도록 하였다. 이를 위하여 RFC에서 권고된 TCP의 기능들 중에서 저속 출발, 혼잡회피, 빠른 재전송, 재전송 타임아웃 등 4가지 항목을 수정하였고, 이들을 조합하여 총 11개의 테스트 항목을 만들었다.

표준 TCP와 각 항목을 적용한 TCP를 다양한 환경에서 테스트 하였으며 다음과 같은 결론을 얻었다. 우선 목적지가 근거리와 같이 RTT가 매우 작은 환경에서는 패킷의 에러나 분실이 발생하지 않는다면 RFC 규격을 준수하는 표준 TCP나 본 논문에서 수정한 각 항목들 사이에는 파일 전송 시간의 차이가 그다지 크게 나지 않았다. 하지만, RTT가 큰 원거리에 있는 목적지로 웹 페이지와 같은 작은 크기의 파일을 전송할 때, 혼잡제어 메커니즘 중 저속 출발을 적용하지 않았을 때는(NSS) 전송 시간에서 매우 큰 효과를 나타냈다.

또한, 어느 정도의 패킷 에러가 발생되는 환경에서는 목적지가 근거리이건 원거리이건 혼잡제어를 하지 않도록 수정된 고속 TCP가 RFC규격을 준수하는 표준 TCP에 비해 빠른 전송 속도를 나타냈다.

하지만, 전송 속도를 높이기 위해 수정된 고속 TCP는 망의 혼잡을 제어하기 위한 RFC의 권고 사항을 무시하였다. 이렇게 설계된 고속 TCP는 RFC 규격을 준수하는 TCP보다 필요 이상 많은 세그먼트를 송신하게 되어 망의 혼잡을 더욱 가중 시키게 되지만, 혼잡이 발생된 망을 복구하기 위한 노력을 전혀하지 않게 된다.

오늘날 무선 환경의 대두와 같이 패킷의 에러와 손실이 높아지고, TCP의 적용 범위가 다양해지는 환경에서, 각 벤더들이 전송 속도를 높이기 위해 혼잡제어를 하지 않는 TCP를 구현할 가능성이 있으며, 이로 인하여 전체 인터넷 망이 혼잡으로 인하여 붕괴되는 상황을 가져 올 수 있다. 그러므로 RFC 권고안을 준수하지 않는 TCP의 사용으로 발생될 수 있는 망의 붕괴를 막기 위해서는 불법 트래픽의 감시나 제어뿐 아니라 인터넷 프로토콜 품질 인증(Internet Protocol Qualification) 제도와 같은 프로토콜 인증 절차와 인증기관의 도입 등이 필요한 것으로 사료된다.

참 고 문 헌

[1] 안순신, 김은기, "정보통신 네트워크", 이한출판사, pp.168-180, 1998.

[2] W. Richard Stevens, "TCP/IP Illustrated, Volume1 : The Protocols," Addison-Wesley, pp.223-322, 1999

[3] Behrouz A. Forouzan, "TCP/IP Protocol Suite," Second Edition, McGraw-Hill, pp.297-335, 2003.

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

[5] Daniel P. Bovet, Marco Cesati, "Understanding the Linux Kernel," O'Reilly, pp.138-157, pp.233-248, 2001.

[6] Tcpdump/libpcap, http://www.tcpdump.org/.

[7] W. Richard Stevens, "UNIX Network Programming, Volume1," Second Edition, Prentice Hall PTR, pp.85-140, 1998.

[8] Alessandro Rubini, Jonathan Corbet, "Linux Device Drivers," O'Reilly, pp.425-469, 2001.

[9] J. Postel, "Transmission Control Protocol," IETF, RFC 793, September, 1981.

[10] R. Braden, "Requirements for Internet Hosts Communication Layers," IETF, RFC 1122, October, 1989.

[11] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," IETF, RFC 2001, January, 1997.

[12] S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP," IETF, RFC 2883, July, 2000.

[13] V. Paxson, M. Allman, "Computing TCP's Retransmission Timer," IETF, RFC 2988, November, 2000.

오 홍 균

e-mail : netohk@netian.com

2002년 국립 한발대학교 정보통신·컴퓨터공학부(공학사)

2004년 국립 한발대학교 정보통신 전문대학원(공학석사)

관심분야 : 인터넷프로토콜, 임베디드, 라우팅 프로토콜 등

김 은 기

e-mail : egkim@hanbat.ac.kr

1987년 고려대학교 전자공학과(공학사)

1989년 고려대학교 전자공학과(공학석사)

1994년 고려대학교 전자공학과(공학박사)

1995년~현재 국립 한발대학교 정보통신·컴퓨터공학부 부교수

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