흐름제어를 하지 않는 고속 TCP의 구현 및 성능

오홍균*, 김은기
국립 한밭대학교 정보통신전문대학원
e-mail : netohk@netin.com*, egkim@hanbat.ac.kr

Implementation and performance of flow uncontrolled fast TCP

Hong-Kyun Oh*, Eun-gi Kim
Graduate School of Information & Communications, HANBAT National University

요 약

인터넷 프로토콜들은 RFC(Request For Comments)에 정의되어 있다. 하지만 RFC 는 단순한 권고사항 일뿐 강제적인 요구사항은 아니다. 그렇기 때문에 RFC 에 정의되어 있는 흐름제어 메커니즘 같은 권고사항을 무시하고 인터넷 프로토콜들을 구현할 수도 있다. 그러나 이렇게 구현된 프로토콜들이 망에 미치는 영향이나, 그것을 제한하기 위한 연구는 아직까지 이루어 지지 않고 있다. 본 논문에서는 RFC 규격을 따르지 않는 TCP 와 RFC 규격을 따르는 표준 TCP 와의 성능 차이를 비교하였다. 그것을 위하여 리눅스 커널의 TCP 프로토콜을 수정하여 흐름제어를 하지 않는 서로 다른 6개의 항목을 만들고, 표준 TCP 와의 성능을 비교하였다. 그 결과, 목적지가 근거리인 경우 정상적인 TCP 와 본 연구에서 수정된 TCP 간의 파일 전송 시간의 차이는 크게 나지 않았다. 하지만, 원거리에 있는 목적지로 웹 페이지 정도의 작은 파일을 전송할 경우, 흐름제어 메커니즘 중 저속 출발(slow start)을 적용하지 않았을 때는 전송 시간에서 매우 큰 차이를 나타냈다.

1. 서론

현재 동작하고 있는 인터넷 프로토콜들은 RFC에 정의되어 있다. TCP/IP의 전송 계층에 해당하는 TCP는 RFC793에 정의되어 있으며, 신뢰성 있는 서비스를 제공하기 위한 사항들이 여러 RFC에 기술되어 있다[7][8][9][10]. 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[1][2].

하지만, RFC에 정의된 인터넷 프로토콜 규격은 단순한 권고사항 일뿐 강제적이지 않다. 만약, 특정 회사에서 자신의 필요에 따라 RFC 규격을 따르지 않는 - 예를 들면, 흐름 제어를 하지 않는 등- TCP를 구현하여 동작시킬 경우 망의 혼잡을 증가시키는 요소가 될 수 있다. 그렇지만, 규격을 무시한 인터넷 프로토콜이 망에 미치는 영향이나 속도 향상 등에 관한 연구는 이루어지지 않고 있다.

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

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 논문에서는 위의 요구에 맞는 고속 TCP를 구현하여 실제 망에서 테스트 하였으며, RFC 규격을 따르는 표준 TCP와의 성능을 비교 하였다. 또한 RFC를 따르지 않는 고속 TCP가 망에 미칠 수 있는 영향에 대해서도 알아보았다.

본 논문의 나머지 부분은 다음과 같이 구성되어 있다. 2장 에서는 고속 TCP 설계에 필요한 요소들을 정리 하였고, 3장에서는 결정된 각각의 요소를 적용한 TCP를 사용하여 테스트를 수행하고, 그 결과를 분석 하였다. 마지막으로 4장에서는 결론을 기술 하였다.

2. 고속 TCP의 설계

2.1 TCP의 성능에 영향을 미치는 알고리즘들

TCP 트래픽을 살펴 볼 때, 세그먼트수의 관점에서

본다면 대용량 데이터(FTP, 전자우편, 유즈넷 뉴스)가 반이고, 나머지 반은 대화식 데이터(Telnet, Rlogin)로 볼 수 있다. 바이트 수의 관점에서 본다면 약 90%가 대용량 데이터이고, 약 10%가 대화식 데이터이다. 그 이유는 일반적으로 대용량 데이터는 세그먼트의 최대 길이를 모두 사용하는 반면, 대화식 데이터는 그보다 훨씬 작기 때문이다[1]. 따라서 TCP는 각 트래픽의 형태에 맞는 다양한 알고리즘을 사용하여 효율적으로 처리하게 된다.

다음은 TCP의 데이터 전송속도에 영향을 미치는 주요 알고리즘들 이다.

- **효율적인 망의 사용을 위한 알고리즘들**
  - 지연된 확인응답(Delayed Acknowledgement)
  - Nagle 알고리즘(Nagle Algorithm)
  - 어리석은 윈도우 신드롬 회피(Silly Window Syndrome Avoidance)
- **흐름 제어에 목적을 둔 알고리즘들**
  - 슬라이딩 윈도우(Sliding Window)
  - 저속 출발 (Slow Start Algorithm)
  - 혼잡회피(Congestion Avoidance Algorithm)
  - 빠른 재전송과 빠른 복구(Fast Retransmit and Fast Recovery Algorithms)
  - 재전송 타임아웃(RTO:Retransmission Timeout)

2.2 본 논문에서 선택한 알고리즘들

2.1절에서 살펴본 여러 알고리즘들 중 본 논문에서 관심을 갖는 알고리즘은 대용량 데이터 세그먼트의 전송속도에 많은 영향을 미치는 것들로 선택하였다. 따라서 대화식 데이터 전송에서 효율적으로 사용되는 알고리즘인 "지연된 확인응답"이나 "Nagle 알고리즘", 그리고 "어리석은 윈도우 신드롬 회피"는 제외되었다.

다음은 고속 TCP의 설계를 위해 선택된 알고리즘들과 그것의 수정 내용이다.

- **저속 출발의 제거: NoSlowStart (NSS)**
  저속 출발 알고리즘을 사용하지 않는다. 혼잡 윈도우(cwnd: congestion window)와 광고된 윈도우(advertised window)중 최소값으로 확인 응답을 받지 않고도 한번에 전송할 수 있는 송신 세그먼트의 양을 제한하던 기존의 방식을 오직 광고된 윈도우 만을 사용하여 제한 하도록 한다.
- **혼잡회피 알고리즘의 제거: NoCongAvoid (NCA)**
  저속 출발 진행 중 혼잡이 발생하면, cwnd의 반값을 ssthresh변수에 넣고, cwnd를 1로 설정한 후 다시 저속 출발을 수행하고, cwnd가 ssthresh보다 커지게 되면 가산적인 증가를 하여 미리 혼잡을 회피하던 기존의 방법을 변경하여, cwnd가 ssthresh보다 커지더라도 계속해서 저속 출발의 진행을 수행하여 송신 세그먼트의 양을 지수적으로 증가시킨다. 즉, NCA는 NSS와는 다르게 cwnd를 계속 사용하지만 ssthresh와는 상관없이 지수적인 증가를 보이게 된다.
- **빠른 재전송의 변경: Fast1 (FST1)**
  3개의 중복된 확인응답을 받았을 경우 해당 세그먼트를 재전송하던 기존의 방법을 1개의 중복된 확인응답을 받을 경우 빠르게 재전송하도록 변경한다.
- **재전송 타임아웃의 변경: ModifiedRTO (MRTO)**
  기존 방법으로 계산된 RTO는 세그먼트의 왕복시간인 RTT와 비교했을 때 약 2배 이상 높은 값을 갖는다. 그러므로 분실된 세그먼트에 대해서 타임 아웃이 발생하기 까지는 너무 오랜 시간이 소요된다. 오히려 타임아웃이 발생되기 보다는 중복된 확인응답에 의한 재전송이 발생하게 된다. 본 논문에서는 RFC793에서 권고한 재전송 타임아웃의 계산식에서 다음과 같이 β를 1로 설정하여 RTO의 값을 RTT와 비슷한 수준으로 맞추었다.

$$R \leftarrow \alpha R + (1-\alpha)M$$

$$RTO = R\beta$$

단, $\beta = 1$

위에서 선택된 4가지 알고리즘들의 가능한 조합을 만들어, 표준 TCP를 포함한 7가지의 테스트 항목을 설정하였으며, 표 1에서 이를 보여주고 있다.

표 1. 테스트를 위한 항목들

No.	경우
경우 1	ORG(Original)
경우 2	NSS
경우 3	MRTO
경우 4	NCA
경우 5	FST1
경우 6	NSS+MRTO
경우 7	NCA+MRTO

3. 고속 TCP의 성능 분석

설계된 고속 TCP의 성능 분석을 위하여 리눅스 2.4.19 커널의 TCP 프로토콜을 표1 의 각 항목에 맞게 수정하였다[3][4][6]. 테스트는 실제 인터넷 망에서 수행 하였으며, 수정된 고속 TCP를 클라이언트로 하고, RFC 규격을 따르는 표준 TCP의 discard서버를 목적지 서버로 하여 각 경우에 대한 파일 전송 시간을 비교 하였다.

테스트는 각 경우를 독립적으로 수행 하였으며, 전송되는 파일 크기 마다 120회씩 테스트 하였다. 전송 시간은 연결 설정을 위한 3-way handshake가 끝나고 첫 번째 데이터 세그먼트를 보내는 시점부터 측정 하였으며 정확한 전송 시간을 측정하기 위해 tcpdump를 수정 하여 측정하였다[4][5][6].

또한, 테스트를 하기 위한 서버와의 거리는 홉 수(hop count)가 1 인 근거리와 홉 수가 20인 원거리로 구분하여 테스트 하였다.

아래 모든 경우의 그래프는 RFC 규격을 따르는

TCP 인 경우1(ORG)을 100% 기준으로 하고, 나머지 경우를 그에 비례하도록 환산하여 도식하였다.

3.1 근거리 에서의 테스트

근거리에 위치한 discard 서버로의 파일 전송 시간을 그림1에 나타내었고, 그에 따른 송신 패킷 수를 그림2 에 나타내었다. 파일 크기는 1K부터 2M까지 변경 하였다.

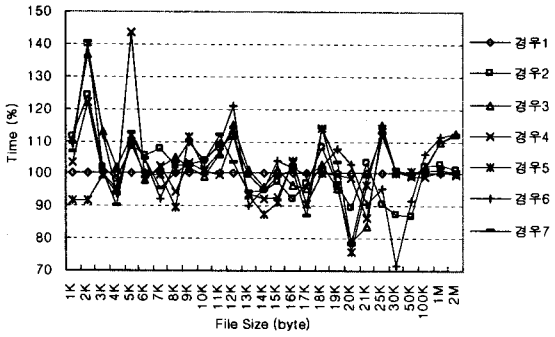

그림 1. 파일크기에 따른 전송 시간 비교(근거리)

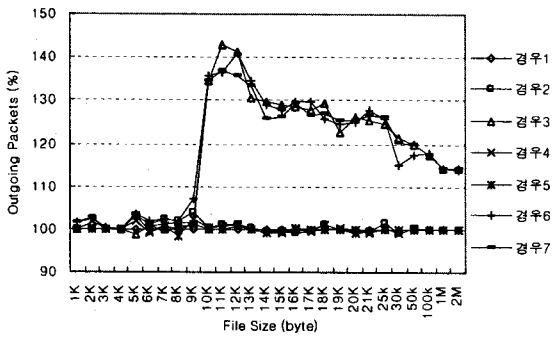

그림 2. 파일크기에 따른 송신 패킷수 비교(근거리)

그림1에 나타난 바와 같이 파일 크기가 1K에서 100K 이하의 비교적 작은 파일은 각 경우에 따른 전송 시간의 차이가 뚜렷하게 나타나지 않았다. 이유는, 근거리는 RTT가 매우 작기 때문에 세그먼트를 수신한 서버가 클라이언트에게 보내는 확인응답(ACK) 메시지가 빠르게 도착한다. 그러므로 클라이언트에서 cwnd값에 해당하는 세그먼트를 모두 송신하기 전에 이전 세그먼트에 대한 확인 응답이 도착하여 cwnd값이 증가하게 되고, 클라이언트는 다음 세그먼트를 보내기 위한 대기시간 없이 바로 세그먼트를 송신할 수 있기 때문이다.

1M, 2M처럼 파일의 크기가 클 때는 전송 시간의 차이가 뚜렷이 나타나고 있다. ORG에 비해 높은 전송 시간을 보이는 것은 모두 MRTO를 적용한 경우들(경우3, 6, 7)이다. 이렇게 높은 전송 시간을 갖는 이유는 MRTO를 적용하여 작게 계산된 재전송 타임아웃 값으로 인해 타임아웃이 자주 발생하여 세그먼트의 재전송 횟수가 많아 지면서 상대적으로 전송 시간이 오래 걸리기 때문이다.

그림 2를 보면 MRTO가 적용된 모든 경우들은 MRTO가 적용되지 않은 경우들에 비해서 파일 크기가 10K(10240바이트)일 때부터 갑자기 전송되는 세그먼트의 수가 증가하는 것을 볼 수 있다. 이유는 discard서버의 수신 버퍼 용량의(10136바이트) 한계로 인한 지연시간 때문에 확인 응답이 늦어지는 상태에서 클라이언트에서는 MRTO가 적용되어 작게 계산된 타임아웃 값에 의해 타임 아웃이 발생하고 세그먼트가 재전송 되기 때문이다.

결론적으로 근거리와 같이 RTT가 매우 작을 때, 크기가 작은 파일 전송에서 각 경우에 따른 전송 시간의 차이는 그다지 크게 나지 않는다는 것을 알 수 있다. 또한 파일 크기가 클 때, 어느 경우는 빈번한 타임아웃으로 인해 송신되는 세그먼트의 수가 증가하게 된다면 전송시간이 오래 걸리게 된다.

3.2 원거리 에서의 테스트

그림3과 그림4는 원거리에 위치한 discard 서버로 파일을 전송할 때, 전송 시간과 그에 따른 송신 패킷수를 각각 나타내고 있다.

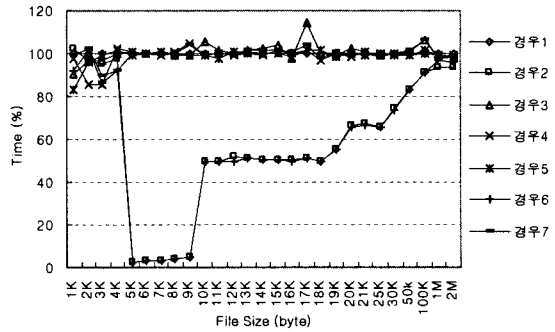

그림 3. 파일크기에 따른 전송 시간 비교(원거리)

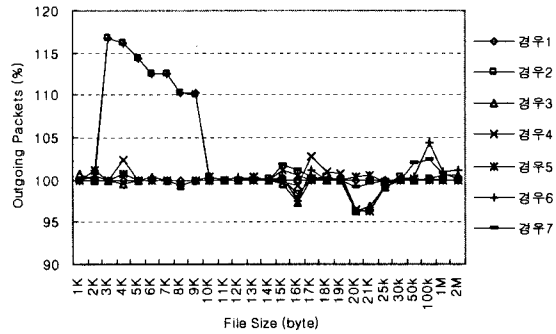

그림 4. 파일크기에 따른 송신 패킷수 비교(원거리)

그림3은 그림1과 다르게 NSS가 적용된 경우들(경우2, 6)의 전송 시간이 월등하게 빠른 것을 볼 수 있다. 이유는 테스트 환경의 연결에서는 MSS가 1448이고, 서버의 광고된 윈도우 크기가 7로 설정 되어있다. 때문에 5K(5120바이트)의 파일을 전송하기 위해

서는 4개의 세그먼트가 필요하게 된다. 리눅스의 초기 cwnd값은 2로 설정 되므로 초기에 클라이언트가 서버의 확인응답 없이 한번에 전송할 수 있는 세그먼트의 수는 광고된 윈도우 크기와 cwnd중 최소 값인 2가 된다. 따라서 NSS가 적용되지 않은 경우들과 ORG는 초기에 2개의 세그먼트만 송신할 수 있고, ACK를 수신하기 전에는 더 이상 세그먼트를 송신하지 못하고 대기하게 된다. 원거리는 RTT가 매우 크기 때문에 discard서버 에서 송신한 ACK를 수신 하기 위해서는 오랜 지연 시간이 걸리게 되고, 그만큼 파일 전송 시간이 증가 하게 된다.

하지만, NSS가 적용된 경우는 오로지 광고된 윈도우 크기에만 제한을 받기 때문에 ACK를 받지 않고 한번에 전송할 수 있는 세그먼트의 수는 7이 된다. 따라서, 5K에서 9K사이의 파일을 전송할 때는 ACK를 기다릴 필요 없이 모든 세그먼트를 전송 하므로 파일 전송 시간이 크게 줄어 들게 된다.

4K(4096바이트)이하의 파일은 최대 3개의 세그먼트가 필요하게 된다. 하지만 리눅스에서는 cwnd의 초기값이 증가하지 않은 상태에서는 모든 경우들에서 마지막 세그먼트는 연결의 종료를 알리는 FIN 플래그를 설정하여 함께 송신하기 때문에 ACK를 받기 위한 대기 시간이 필요하지 않게 되어, 파일 전송 시간의 차이가 뚜렷이 나타나지 않는다.

NSS가 적용된 경우, 10K에서 전송 시간이 갑자기 높아진 이유는 다음과 같다. 10K(10240바이트)전송 시는 총 8개의 세그먼트를 송신하여야 한다. 하지만 광고된 윈도우 크기에 제한을 받아 한번에 전송할 수 있는 세그먼트의 수는 7이 된다. 그러므로, 마지막 세그먼트를 전송 하기 전에 ACK를 받기 위해 대기 하여야 하고, 그만큼 전송 지연시간이 걸리게 된다. 그러므로 파일의 크기가 커질수록 ACK를 받기 위해 대기하는 횟수가 많아지게 되어, 지연 시간이 증가 하게 되고, 점점 전송 시간이 ORG에 근접하게 변하는 것을 볼 수 있다.

그림4의 송신 패킷수를 보면 3K에서 9K사이, NSS가 적용된 경우의 송신 패킷이 다른 경우에 비해 많은 이유는, NSS가 적용되지 않은 경우들은 마지막 데이터 세그먼트에 연결을 종료하기 위한 FIN 플래그를 설정하여 동시에 보내는 반면, NSS가 적용된 경우는 마지막 데이터 세그먼트와 FIN 플래그만 설정된 세그먼트를 따로 전송 하게 되어 다른 경우들 보다 1개의 세그먼트를 더 전송하기 때문이다. 10K부터는 모든 경우가 마지막 데이터 세그먼트에 FIN 플래그를 함께 설정하여 보내기 때문에 파일 전송 시간의 차이가 크게 나타나지 않는다.

결론적으로 RTT가 큰 환경에서는 저속 출발을 수행하지 않을 때는 웹 페이지와 같은 작은 크기의 파일전송 시 전송 시간에 대해 큰 이득을 얻을 수 있다. 하지만 어느 경우는 빠른 전송 시간을 얻기 위해 필요이상으로 많은 세그먼트를 전송하게 되고, 이로 인해 망의 혼잡을 더욱 가중시키는 문제를 갖게 된다.

4. 결론

본 논문에서는 리눅스 TCP를 수정하여, RFC에서 권고하고 있는 흐름제어와 관련된 사항들을 무시하고 최대한 고속으로 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 6개의 테스트 항목을 만들었다. 각 경우를 다양한 환경에서 테스트 하여 표준 TCP와 비교 하였으며 다음과 같은 결론을 얻었다.

목적지가 근거리와 같은 RTT가 매우 작은 환경에서는 RFC규격을 따르는 표준 TCP나 본 논문에서 수정한 각 경우들 사이에는 파일 전송 시간의 차이가 그다지 크게 나지 않았다. 하지만, RTT가 큰 원거리에 있는 목적지로 웹 페이지와 같은 작은 크기의 파일을 전송할 때, 흐름제어 메커니즘 중 저속 출발을 적용하지 않았을 때(NSS)는 전송 시간에서 매우 큰 효과를 나타냈다.

그러나, 전송 속도를 높이기 위해 수정된 고속 TCP는 망의 혼잡을 제어하기 위한 RFC의 권고 사항을 무시하였다. 이렇게 설계된 고속 TCP는 RFC 규격을 따르는 TCP보다 필요 이상 많은 세그먼트를 송신하게 되어 망의 혼잡을 더욱 가중 시키게 되지만, 혼잡이 발생된 망을 복구하기 위한 노력을 전혀 하지 않는다.

오늘날 무선 환경의 대두와 같이 패킷의 분실이나 에러율이 높아지고, TCP의 동작 환경이 다양해지는 환경에서 각 벤더들이 전송 속도를 높이기 위해 흐름 제어를 하지 않는 TCP를 구현할 가능성이 있으며, 이로 인하여 전체 인터넷 망이 혼잡으로 인하여 붕괴되는 상황을 가져 올 수 있다.

그러므로 RFC 권고안을 따르지 않는 TCP의 사용으로 발생될 수 있는 망의 붕괴를 막기 위해서는 불법 트래픽의 감시와 제어뿐 아니라 인터넷 프로토콜 품질 인증 (Internet Protocol Qualification) 제도와 같은 프로토콜 인증 절차와 인증기관의 도입 등이 필요한 것으로 사료된다.

참고문헌

[1] W. Richard Stevens, "TCP/IP Illustrated: The Protocols", Addison-Wesley, 1999.

[2] Behrouz A. Forouzan, "TCP/IP Protocol Suite", McGraw-Hill, 2003

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

[4] Daniel P. Bovet, Marco Cesati, "Understanding the Linux Kernel", O'Reilly, 2001

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

[6] W. Richard Stevens, "UNIX Network Programming", Prentice Hall PTR, 1998

[7] RFC 793, "Transmission Control Protocol", http://www.ietf.org

[8] RFC 1122, "Requirements for Internet Hosts – Communication Layers", http://www.ietf.org

[9] RFC 2988, "Computing TCP's Retransmission Timer", http://www.ietf.org

[10] RFC 2883, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", http://www.ietf.org