TCP 과잉밀집 제어 및 회피에서 빠른 재전송 알고리즘 개선방안

조형재, 양대헌, 송주석
연세대학교 컴퓨터과학과

Improving the Fast Retransmit Algorithm for TCP's Congestion control and Avoidance

H.J. Cho, D.H. Nyang, J.S. Song
Dept. of Computer Science, Yonsei University

요 약

전송 프로토콜은 응용 프로그램과 네트워크의 인터페이스로서, 응용 프로그램에서 요구한 QoS(Quality of Service)를 제공하는 역할을 한다. 이 중 TCP는 인터넷의 전송 흐름 제어를 위해서 사용되는 프로토콜이다. TCP의 흐름 제어를 위해서 수신된 데이터의 ACK(acknowledgement)에 따라 허가된 윈도크기만큼의 데이터를 보내는 크레디트 할당 기법을 사용한다. 따라서 네트워크의 과잉밀집 제어를 위해서 과잉밀집 윈도(congestion window)를 사용한 slow-start 알고리즘을 사용하며, 손실된 데이터를 재전송하기 위한 방법으로 빠른 재전송 및 회복 알고리즘을 사용한다. 본 논문에서는 빠른 재전송 알고리즘에서 나타나는 문제점을 알아보고, 이 알고리즘이 빠른 시간에 데이터 손실을 회복하고 데이터를 보낼 수 있도록 수정한 알고리즘을 소개한다. 또한 수정된 알고리즘을 확장하여 네트워크의 상태에 따라 더 많은 데이터를 보낼 수 있도록 개선한 알고리즘을 제안한다.

1. 서 론

인터넷의 폭발적인 증가로 말미암아 전세계 컴퓨터 네트워크에서 IP(Internet Protocol)와 더불어 가장 많이 쓰이는 전송 프로토콜로 TCP(Transmission Control Protocol)가 있다. TCP는 IP와 달리 연결형 위주의 메시지 전달로 사용자에게 안전하고 신뢰성 있는 서비스를 제공해 주고 있다. 최근 통신 장비 및 네트워킹 기술의 발달은 응용 프로그램이 많은 대역폭과 빠른 전송 속도의 이점을 활용할 수 있게 하였고, 이러한 사실들이 TCP 프로토콜의 성능에 많은 관심을 불러일으키고 그 중요성을 인식시키고 있다. 예를 들어 TCP 상에서 HTTP 프로토콜이 구동되는 경우 대역폭과 지연시간이 작은 환경에서는 TCP가 잘 작동하지만, 대역폭과 지연시간이 큰 환경에서는 TCP의 오버헤드가 커지는 것으로 나타났다[1]

네트워크의 트래픽을 제어하기 위해 TCP는 과잉밀집 윈도를 사용하는데, 송신측의 윈도를 잘 관리하여 TCP가 과잉밀집 없이 잘 작동할 수 있도록 다음과 같은 세 가지 기법이 사용된다 (1) slow-start (2) 빠른 재전송(Fast Retransmit) (3) 빠른 회복(Fast Recovery)

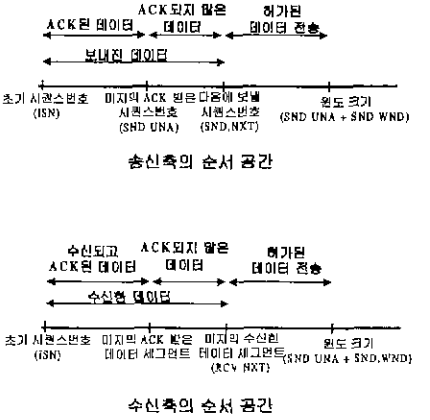
특히 연결 설정 시기와 과잉밀집 상태에서 회복하는 시기의 적절한 초기값과 설정 및 시작 방법은 빠른 속도를 지원하는 네트워크에서 중요한 역할을 한다.

본 논문에서는 TCP 트래픽의 과잉밀집 제어 기법들을 살펴보고 빠른 재전송 및 회복 알고리즘을 개선하기 위한 알고리즘을 제시한다.

2. TCP 흐름/과잉밀집 제어

2.1 TCP 흐름제어

TCP에서 사용되는 흐름제어는 크레디트 할당 기법이다. 이 기법에서는 전송되는 세그먼트가 시퀀스번호(sequence number)를 가지고 있다고 가정한다. TCP 개체(entity)는 들어오는 세그먼트를 받았음을 알리는 메시지 형태를 (A=i, W=j)와 같이 나타낸다. 여기서 시퀀스번호 i-1을 받고 나서 시퀀스번호 i의 세그먼트를 다음에 받기를 원하며, j개의 추가적인 윈도크기의 데이터가 보내지도록 허가된다는 뜻이다. [그림1]은 송신측과 수신측 관점에서의 이러한 메커니즘을 보여준다.



[그림 1] 송신측과 수신측의 흐름제어

2.2 Slow-Start

Slow-start는 과잉밀집 윈도(congestion window)를 사용한다. TCP는

항상 다음과 같은 관계를 유지하며 전송한다. 즉 크레디트 윈도 값과 과잉밀집 윈도 값 중 작은 것을 선택한다

awnd = MIN [credit, cwnd]

◆ awnd (allowed window) : ACK가 없이 TCP가 보낼 수 있는 세그먼트의 개수를 유지하는 윈도

◆ cwnd (congestion window) : TCP 연결에서 시작할 때와 과잉밀집으로 트래픽을 줄여갈 때 쓰이는 윈도

◆ credit 가장 최근 받아들인 ACK를 제외하고 남은 크레디트의 양

새로운 연결이 설정되었을 때, TCP 개체는 cwnd를 1로 초기화 시킨다. 즉 TCP는 1개의 세그먼트만을 전송하고 두 번째 세그먼트를 보내기 전에 ACK를 기다려야 한다 ACK가 수신될 때마다 cwnd의 값을 최대값까지1씩 증가시킨다. 그 결과 slow-start 기법은 이미 과잉밀집된 상황에서 너무 많은 세그먼트를 보내지 않도록 네트워크를 제어한다. 실제로 cwnd는 지수적으로 증가한다.

다음과 같이 slow-start로 시작해서 다음과 같이 cwnd가 선형적으로 증가할 수 있는 방법을 제안되었다[2].

시간초과가 발생하면,

1) slow-start의 한계치(threshold)를 현재 과잉밀집 윈도의 절반으로 설정한다 ; ssthresh = cwnd / 2
2) cwnd = 1로 설정하고 slow-start 과정을 cwnd = ssthresh일 때까지 진행시킨다. 이 과정에서는 수신된 하나의 ACK마다 cwnd를 1씩 증가시킨다.
3) cwnd ssthresh일 때 각 왕복시간에 대해 cwnd를 한 개씩 증가시킨다

2.3 빠른 재전송

TCP가 중복된 ACK을 받았다는 것은 다음과 같은 것을 의미한다 세그먼트가 지연되었거나 잘못된 순서로 도착한 경우와 세그먼트를 분실한 경우가 있다. 첫 번째 경우에 초점을 맞추기 위해 같은 세그먼트에 대해 3개의 중복된 ACK(총 4개의 ACK)를 받을 때까지 TCP 송신측이 기다리는 방안이 제안되었다[3]. 즉, 재전송 타이머가 끝나기를 기다리지 않고 바로 빠른 재전송 알고리즘을 수행할 수 있다.

만약 TCP 수신측이 순서가 잘못된 세그먼트를 받았을 때 수신측은 받은 세그먼트 중 마지막으로 잘못된 것에 대해 즉시 ACK을 보낸다. TCP는 버퍼에서 빠진 세그먼트를 전송받을 때까지 마지막 세그먼트에 대한 ACK을 계속해서 보낸다. 그리고 빠진 자리를 채우게 되면 여태껏 재대로 수신한 세그먼트에 대해 축적된 ACK를 보낸다

2.4 빠른 회복

빠른 재전송 알고리즘을 수행한 후 과잉밀집 회피 모드로 들어가게 되는데, 오류가 발생한 데이터의 재전송을 위해 앞서의 slow-start 방법이 지나치게 비관적이라고 가정하고, 대안으로 잃어버린 세그먼트를 재전송한 후, cwnd를 반으로 설정하고 cwnd를 선형적으로 증가시키는 빠른 회복 방안을 제시하였다[3] 이러한 기법은 초기에 지수적 slow-start 작업을 피하기 위해 만들어졌다

좀 더 자세히 빠른 회복 방법을 살펴보면 다음과 같다

1) 3개의 중복된 ACK가 도착하였을 경우,
   a. ssthresh = cwnd/2 로 설정한다.
   b 분실된 세그먼트를 재전송한다
   c cwnd = ssthresh + 3으로 설정한다
2) 추가적으로 중복된 ACK이 도착할 때마다 cwnd를 1씩 증가시키고 세그먼트를 전송한다
3) 받은 ACK가 새로운 데이터를 위한 것이라는 것을 확인하는 다음 순서의 ACK가 도착했을 때 cwnd = ssthresh로 설정한다.

3. 수정된 빠른 재전송 및 회복 알고리즘

3.1 기존 빠른 재전송 알고리즘의 문제점

앞 절에서 살펴본 빠른 재전송 알고리즘은 연결 초기동안의 데이터 손실의 회복에 불필요하게 많은 시간을 소모하는 경우가 있다. 만약 같은 윈도 내의 하나의 패킷만을 손실했다면, 빠른 재전송 알고리즘으로 회복 가능하지만, 같은 윈도 내(한번의 round-trip)의 여러 개의 패킷을 손실했다면, 빠른 재전송 알고리즘으로 회복할 수 없다[5]. 이런 현상이 발생하는 예는 다음과 같다

세그먼트 2개에 해당하는 하나의 ACK가 들어오고, 하나의 세그먼트와 한 개 건너 세그먼트가 손실되었다고 가정하자. 첫 번째 세그먼트가 손실되었을 때 3개의 중복된 ACK가 송신측으로 수신되고 빠른 재전송 알고리즘이 시작된다. 이어서 손실된 세그먼트에 대한 재전송이 실행되고 그에 대한 ACK를 받았다. 그 다음 손실된 다른 하나의 세그먼트에 대해 재전송이 시작된다 이 때 또 하나의 빠른 재전송 알고리즘이 실행되든지 재전송 타임아웃 후에 재전송이 이루어지야 한다. 하지만 손실된 첫 번째 세그먼트에 대한 3개의 중복된 ACK를 받았을 때까지 이미 송신측은 자신이 보낼 수 있는 윈도 크기의 데이터를 전부 전송한 상태이다. 중복된 ACK를 받았을 경우도 과잉밀집 윈도 크기를 겨우 한 개의 세그먼트만큼 늘일 수 있다 그래서 송신측은 더 이상의 데이터를 전송할 수 없고 그에 따라 그에 대한 ACK를 받을 수 없는 상태이다. 즉 여태껏 보내지 않았던 데이터에 대한 중복되지 않는 ACK가 도착하기 전까지 새로운 데이터를 전송할 수 없게 된다 이러한 경우에 재전송을 위해서는 재전송 타임아웃(약 1초)까지 대기하고 있어야 한다. 이렇게 대기하는 시간은 TCP 성능을 심각하게 떨어뜨릴 수 있다.

또한 이전에 여러 개의 세그먼트가 손실되어 나중에 손실되지 않고 수신측의 캐쉬에 남아있는 세그먼트에 대한 거짓 빠른 재전송(false Fast Retransmit) 상태가 발생되는 경우도 있다 이러한 문제들을 SACK(Selective ACK)방식[6]으로 해결 할 수도 있지만 이러한 방식은 수신측의 메커니즘까지 수정해야 하는 번거로움이 있다.

3.2 수정된 재전송 및 회복 알고리즘

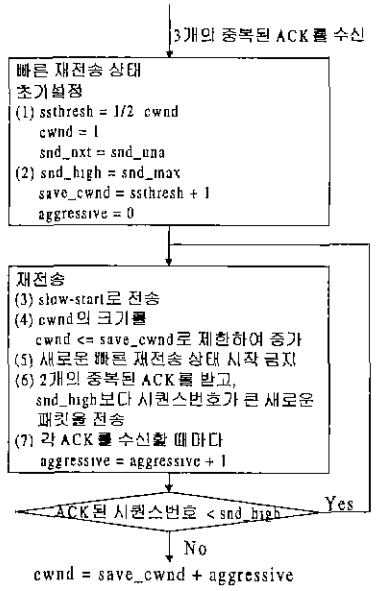
Hoe는 빠른 재전송 알고리즘에서 재전송 타임아웃까지 기다릴 필요 없이 여러 개의 패킷 손실로부터 빠르게 회복할 수 있는 알고리즘을 제시하였다[4] 이 알고리즘 세그먼트의 손실을 회복하기 위해 더 많은 데이터를 전송할 수 있도록 빠른 재전송 알고리즘을 수정하는 것이다. 원래의 빠른 재전송 알고리즘에서 크게 두 부분이 수정되었다. 첫 번째는 두 개의 중복된 ACK을 받았을 때, 세그먼트 번호 snd_nxt와 함께 새로운 세그먼트를 전송시키는 것이다. 두 번째는 송신측에서 빠른 재전송 상태에 들어가기 전에 가장 최후에 보냈던 세그먼트의 시퀀스번호를 highest_seq라고 정의한다. 나중에 이 값을 빠른 재전송 상태의 종료를 판단하는데 사용한다. 빠른 재전송 상태는 3개의 중복된 ACK을 받은 시점으로부터 시퀀스번호 highest_seq가 ACK되었을 때까지의 시점을 말한다

하나의 세그먼트만을 잃어버렸다고 가정하면, 송신측은 highest_seq까지의 모든 데이터에 대한 ACK를 받기를 기대할 것이다. 하지만, 만약 송신측이 highest_seq보다 작은 값의 m인 시퀀스번호를 받았다면 시퀀스번호 m인 세그먼트가 손실되었다는 것을 알 수 있을 것이다. 그래서 손실된 세그먼트에 대해서는 3개의 중복된 ACK을 받지 않고도 즉시 세그먼트 m을 전송할 것이고, 시퀀스번호 highest_seq을 받기 전까지 빠른 재전송 상태에서 이러한 동작을 반복한다. 이러한 방법으로 개선된 빠른 재전송 알고리즘이 여러 개의 손실된 패킷을 전송할 수 있다는 것을 알 수 있다

이 알고리즘은 원래의 빠른 재전송 알고리즘보다 더 많은 데이터를 전송할 수 있는 방법을 취한다. 여러 개의 패킷 손실 후에 네트워크의 큐를 비우기 위한 재전송 타임아웃을 지나치게 크게 둔다는 것에 착안한 방법이다. 재전송 타임아웃은 빠른 재전송 알고리즘이 실패했을 경우 차선책으로 존재한다

4. 개선된 빠른 재전송 알고리즘 제안

이 절에서는 앞 절의 수정된 빠른 재전송 알고리즘에서 과잉밀집 회피 모드에서의 네트워크 상태를 반영할 수 있는 방안을 제시한다. 같은 윈도에서 여러 개의 세그먼트 손실이 발생할 경우 손실된 데이터를 회복하는 동안 수신되는 ACK로 네트워크의 상태를 파악할 수 있다. ACK가 계속 수신된다는 것은 네트워크에서 계속 데이터의 전송이 진행될 수 있다는 것을 의미한다. 따라서 매번 들어오는 ACK에 대응하여 과잉밀집 상태를 회복하는 시점에서의 과잉밀집 윈도(cwnd) 크기를 늘려 줄 수 있다. 빠른 재전송 상태를 회복한 후 사용할 수 있는 과잉밀집 윈도 크기가 이전 상태의 과잉밀집 윈도 크기의 반이 되는데, 더 적극적인 방법을 취하여 손실된 데이터를 수신하는 동안의 네트워크 상태에 따라 더 큰 과잉밀집 윈도를 설정할 수 있다. 이러한 방법은 [그림 2]에서 알고리즘이 제시된다.



[그림 2] 개선된 빠른 재전송 알고리즘

단계 (1)에서는 원래의 빠른 재전송 기법과 같은 방법으로 각 파라미터에 값을 할당한다. 단계 (2)에서는 시퀀스번호 snd_high값에 네트워크에서 현재까지 전송한 시퀀스번호를 저장한다. 이 값에 송신측에 들어오는 각 ACK의 시퀀스번호를 비교하여 빠른 재전송 상태가 끝난 것인지를 판단할 수 있다. 수신된 ACK가 snd_high보다 크거나 같다면 과잉밀집 회피 상태가 종료한다는 것을 알 수 있다. save_cwnd는 과잉밀집 회복 후의 과잉밀집 윈도 크기를 점해주기 위해서 사용되며, 현재 과잉밀집 윈도 크기의 절반이 된다. aggressive는 빠른 재전송 상태에서 네트워크로부터 들어오는 ACK의 개수를 기장하여 과잉밀집 회복 후 cwnd값을 크게 해 주기 위해 사용된다. 단계 (3)에서는 송신측이 빠른 재전송 상태에 있는 동안 손실된 모든 패킷이 복구될 때까지 slow-start를 사용하여 데이터를 재전송한다. 단계 (4)에서는 여러 개의 패킷 손실을 유발한 원래의 데이터 전송률의 반을 넘지 않은 범위에서 패킷을 재전송할 수 있다. 단계 (5)에서는 송신측이 빠른 재전송 상태에 있는 동안 모든 중복된 ACK를 무시하여 거짓 빠른 재전송 문제를 해결하였다. 단계 (6)에서는 2개의 중복된 ACK를 수신할 때마다 전에 전송하지 않았던 새로운 데이터(시퀀스번호 snd_high 이상)를 전송시킨다. 즉 손실된 데이터를 회복하기 위한 파이프의 새로운 데이터 전송을 위한 파이프 2개를 유지한다. 단계 (7)은 (6)에서 받는 모든 ACK를 너중에 복귀할 과잉밀집 윈도 크기에 반영하기 위해 각각의 ACK에 대해 aggressive 값을 더해 준다. 빠른 재전송 상태가 종료하면 과잉밀집이 일어난 직 후의 과잉밀집 윈도 크기의 절반 값과 재전송 상태에서 받은 ACK의 값을 합한 수치로 데이터를 전송한다.

5. 결 론

인터넷의 발달과 더불어 사용자에게 신뢰성 있는 서비스를 제공하기 위한 TCP는 널리 사용되고 있는 전송제어 프로토콜이다. TCP에서는 연결형 위주의 흐름 제어와 과잉밀집 상태에서의 제어 기법을 수행한다. 특히 빠른 속도를 제공하는 네트워크 환경에서 TCP의 트래픽 제어 방법들은 네트워크 상태와 제공되는 서비스에 상당한 영향을 미칠 수 있다

현재 TCP는 과잉밀집에서의 회복을 위해 빠른 재전송 및 빠른 회복 기법을 제공한다. 하지만 빠른 재전송 알고리즘이 여러 개의 데이터 손실이 있을 경우에 제대로 동작하지 않는 문제가 있다. 이러한 상황을 극복하기 위해 제안된 기존 알고리즘을 소개하였다. 그리고 본 논문에서 빠른 재전송 상태에서 벗어났을 때 재전송 기간의 네트워크 상태에 따라 더 많은 데이터 전송률을 송신측에게 할당할 수 있는 기법을 제안하였다. 이런 방법은 과잉밀집에서 벗어나 더 적극적으로 많은 데이터를 전송할 수 있는 효과를 가져올 것이다. 향후 과제로는 제시한 방안에 대해 시뮬레이션을 통한 정확한 분석이 필요하다.

[참고문헌]

[1]John Heidmann. Katia Obraczka, and Joe Touch. Modeling the performance of HTTP over several transport protocols. To appear, IEEE/ACM Transations on Networking, November 1996

[2] V. Jacobson. Congestion avoidance and control. In Proceedings of the ACM SIGCOMM '88, pages 314-329, August 1998.

[3] V. Jacobson. Modified TCP congestion avoidance algorithm. end2end-interest mailing list (Apr.), 1990.

[4]Janey C. Hoe. Improving the start-up behavior of a congestion control scheme for TCP. In Proceedings of the ACM SIGCOMM '96, pages 270-280, Stanford, CA, August 1996. ACM.

[5] J. C. Hoe Start-up dynamics of TCP's congestion control and avoidance schemes, 1995

[6] M. Mathis, J Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgement options. ftp://ftp ietf cnri.reston.va.us/internet-drafts/draft-ietf-tcplw-sack-00.txt, April 1996.