지연 ACK 옵션을 사용할 때의 TCP 성능개선

민구봉 김종권
서 울 대 학 교 컴 퓨 터 공 학 부
{kmin, ckim}@popeye.snu.ac.kr

TCP Performance using Delayed ACK option

Ku-Bong Min Chong-kwon Kim
School of Computer Science and Engineering, Seoul National University

요 약

본 논문에서는 TCP 수신자가 지연 ACK 옵션(Delayed ACK Option)을 사용할 경우에 TCP 송신자에게 발생하는 성능 저하요인들을 분석하고 다음과 같이 해결책을 제시하였다. 먼저, 느린 시작 구간(Slow Start phase) 처음에 생기는 ACK 타임아웃은 큰 초기 윈도우(large initial window) 또는 1-bit 마킹 기법을 통해 해결할 수 있다. 그리고, 느린 시작 구간과 혼잡 회피 구간(Congestion Avoidance phase)에서 혼잡 윈도우(cwnd)가 전전히 증가하는 문제는 적절히 바이트 카운팅 기법을 사용함으로써 해결할 수 있다. 마지막으로, 송신자가 버스트(burst)한 데이터를 네트웍에 발생시키는 문제는 트래픽을 평활(pacing) 함으로써 해결할 수 있다. 또한 본 연구에서는 분석적 모델링을 통하여 TCP가 보내는 평균 전송률을 구하였으며 이 결과는 TCP에 친화한 전송률 기반 전송방법(TCP Friendly Rate Based Control)에 응용될 수 있을 것이다. 그리고 시뮬레이션을 통해서 제시한 방법의 성능이 향상됨을 확인하였다.

1. 서론

TCP는 인터넷에서 신뢰성 있는(reliable) 데이터 전송을 위해 가장 많이 사용되는 전송계층(Transport Layer)의 양단간(End-to-End) 프로토콜이다. TCP 데이터 송신자는 수신자가 보내주는 ACK를 바탕으로 네트웍 자원을 공평(fair)하게 분배할 수 있도록 윈도우기반(window based)으로 전송률을 조정한다.

TCP가 사용하는 혼잡제어(congestion control)기법은 수신자가 받은 데이터 패킷에 대해 보내준 ACK에 따라 윈도우 크기를 선형적으로 증가시키고 지수적으로 감소(Additive Increase Multiplicative Decrease)시킴으로써 전송률을 조정하는 방법이다[1].

이 때 TCP 수신자는 지연 ACK(Delayed ACK) 방법을 사용하여 ACK를 전송하는데, 이것은 최소한 매번 2개마다의 최대크기 데이터 패킷을 받았을 때와, 그렇지 않더라도 데이터를 받은 지 500ms 이내에는 반드시 ACK를 보내야하는 것을 말한다. 또한 데이터 패킷의 순서가 바뀌어서 도착하는 경우에는 패킷손실 복구를 빨리 할 수 있도록 매번 도착하는 패킷에 ACK를 보내주어야 한다[1].

TCP 수신자가 지연 ACK 옵션을 썼을 때의 성능 개선에 대한 기존연구로서 M. Allman은 제한 바이트카운팅(LBC:Limited Byte Counting) 기법과 절충 바이트카운팅(ABC:Appropriate Byte Counting) 기법을 제안하였다.

바이트카운팅은 혼잡 윈도우(congestion window)의 크기 cwnd를 받은 ACK의 개수가 아니라 수신자가 새로 받았다고 알려준 데이터의 크기를 기준으로 cwnd를 증가시키는 기법이다. 그러나 이것은 네트웍 혼잡에 공격적인 행동을 할 수 있다[2]. LBC에서는 바이트카운팅을 느린 시작 구간(Slow Start phase)에서만 사용하고 한번에 cwnd를 2까지만 증가시킬 수 있도록 제한하였다. ABC에서는 첫 번째 느린 시작 구간에서만 LBC를 적용하도록 하여 네트웍에 미치는 부정적인 효과를 최소화하였다[3].

본 논문에서는 지연 ACK 옵션을 사용할 때에 생기는 여러 가지 성능 저하 요인을 모델링과 시뮬레이션을 통해서 분석하고 해결 방안을 제시하여 성능이 개선됨을 보였다.

2. TCP 성능 저해요인과 해결방법

수신자가 지연 ACK 방법에 따라 매번 오는 데이터 패킷에 대해 ACK를 보내주지 않는 경우에는 항상 ACK를 보내주는 경우보다 송신자는 다음과 같은 면에서 손해를 보게된다.

첫째, 느린 시작 구간의 시작에서 항상 ACK 타임아웃(Time Out)이 발생하게 된다. 이것은 송신자의 cwnd가 1이어서 송신자는 하나의 데이터 패킷만을 보낼 수 있기 때문이다. 결국

수신자는 2개의 데이터 패킷을 받을 때까지 ACK를 전송하지 않고 기다리기 때문에 타임아웃이 발생하게 되는 것이다[4]. 이 문제는 초기 cwnd 값을 1로 하지 않고 RFC 2414에서 제안한 것과 같이 설정해주면 해결할 수 있다. RFC에서 제안한 초기 cwnd값은 다음과 같다[5].

$$initial\_cwnd = \min(4*MSS, \max(2*MSS, 4380bytes))$$

위의 방법은 첫 번째 느린 시작 구간에서의 ACK 타임아웃이 발생하는 문제만을 해결하고 있다. 모든 느린 시작 구간에서 해결하기 위한 방법으로서 느린 시작 구간의 첫 번째 보내는 데이터 패킷에 대해서는 1-bit 마킹을 할 수 있다. 이것은 수신자가 1-bit 마킹이 되어있는 데이터에 대해서는 항상 다음 데이터를 기다리지 않고 바로 ACK를 전송할 수 있게 함으로써 문제를 해결한다.

둘째, 느린 시작 구간에서 윈도우의 크기가 증가하는 정도가 한 왕복전송시간(RTT:Round Trip Time) 당 2배씩 증가하지 않고 1.5배씩 증가하게 되는 문제가 있다. 이것은 TCP 전송자가 받은 ACK의 개수를 바탕으로 윈도우의 크기를 증가시키기 때문이다[2][3]. 이것은 M. Allman이 제안한 LBC나 ABC를 사용함으로써 네트웍에 버스트한 데이터 패킷들을 보내는 것을 방지하면서도 느린 시작 구간에서 매번 ACK를 받는 것과 어느정도 동등하게 성능을 개선할 수 있다[3].

셋째, 혼잡 회피 구간(Congestion Avoidance phase)에서 1 RTT에 한 패킷 단위만큼 cwnd를 증가시킬 수 없다. 만일 2개의 데이터 패킷마다 1개의 ACK를 보내주는 경우라면 2 RTT가 걸려야 cwnd를 1만큼 증가시킬 수 있다. 이것은 데이터 송신자가 ACK를 한 개 받을 때마다 cwnd를 1/cwnd 만큼씩 증가시키기 때문이다. 이 문제 또한 혼잡 회피 구간에서 Byte Counting 기법을 사용함으로써 해결할 수 있다.

넷째, TCP 송신자의 데이터 전송을 더 버스티(bursty)하게 만들어 엑세스 네트웍 라우터의 버퍼 용량이 작은 경우 버퍼 오버플로우를 발생시킬 수 있다. 이것은 ACK가 매번 보내는 데이터 패킷보다 뜸하게 오는 경우 송신자 측 윈도우에서 슬라이딩(sliding)이 한번에 많이 되기 때문에 네트웍에 버스트(burst)한 트래픽을 보내게 되기 때문이다. 또한 손실된 패킷을 복구하는 빠른 복구(Fast Recovery)과정에서 손실된 패킷보다 늦은 순서번호(sequence number)를 가진 데이터들이 미리 보내질 수 있기 때문에[6], 손실 복구 완료 후에 많은 데이터를 받았다고 알려주는 ACK가 발생하여(Big ACK) 윈도우 슬라이딩이 심하게 이루어지는 경우가 생기게 된다. 이런 현상이 일어나는 것은 시뮬레이션을 통해 4절에서 보이고 있다. 버스트한 TCP 트래픽을 성형(shape)하는 시도로써 ssthresh 값과 cwnd값을 인위적으로 조정하는 방법이 있다[7]. 이것은 Big ACK을 받았을 때, ssthresh 값을 현재의 cwnd값으로 조정하며 cwnd값을 Big ACK가 나타내는 값만큼 줄이고 느린 시작 구간으로 들어가는 것이다. 이렇게 함으로써 3개 이상의 버스티니스(burstiness)가 생기지 않게 조정한다.

또 다른 방법은 전송률 기반 조정(Rate based control) 방법의 요소를 도입하여 TCP 트래픽을 평활(pacing)하는 것이다. 이것은 TCP의 전송률이 cwnd/RTT가 되도록 타이머를 설정하고 ACK와 무관하게 송신자가 보내는 데이터 트래픽을 평활하는 방법이다. 그러나 이 방법은 기존의 방법보다 성능이 좋지 않은 것으로 알려져있다[8]. 따라서 본 논문에서는 최대 버스티니스(maximum burstiness)를 4로 정의하여 한번에 그 이상은 윈도우가 슬라이딩하는 것을 막는 방법을 제안한다.

3. TCP 성능 모델링

TCP 연결이 MTU bytes의 패킷을 연속적으로 보내는 평형상태에 도달했다고 가정할 때에, 처음의 느린 시작 구간을 무시할 수 있다고 가정하면 TCP 연결의 송신자 윈도우 사이즈는 그림 1과 같이 시간에 따라 윈도우 크기가 W/2부터 W까지 주기적으로 변하며 톱니모양처럼 보이게된다. 여기서 우리는 TCP Reno를 사용하고 있다고 생각한다.(TCP Tahoe를 사용하는 경우에는 데이터 손실마다 매번 느린 시작에 들어가므로 매번 걸리는 ACK 타임아웃을 고려해야 한다.)

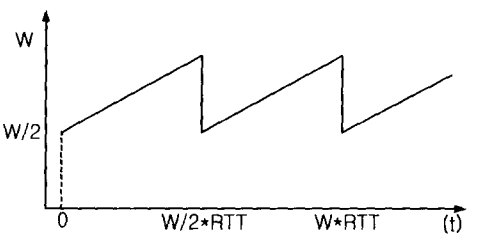

그림 1. TCP 송신자의 시간에 따른 윈도우 크기 변화

패킷이 손실 될 때의 TCP 윈도우 크기가 W라면 그 때의 TCP 송신자의 전송률 S는 다음과 같다.

$$S = \frac{W \times MTU}{RTT} \quad (1)$$

TCP는 손실이 발생했을 때 윈도우 크기를 반으로 줄이고 1RTT당 1씩 윈도우 크기를 늘리므로, 평형상태에서 TCP는 평균적으로 0.75*S의 전송률 BW를 얻게된다.

$$BW = 0.75 \times \frac{W \times MTU}{RTT} \quad (2)$$

TCP 연결의 손실 확률이 P라고 하면, 그림 2에서 보이는 바와 같이 지연 ACK 요소(Delayed ACK factor) d에 따라 한 주기 동안 보내는 총 패킷 중 하나가 손실되는 것이 되므로, 다음과 같은 관계가 성립한다.

$$P = \frac{1}{d\{W/2 + (W/2+1) + \ldots + W\}} \quad (3)$$

식(3)을 W에 대해서 정리하면 다음과 같다.

$$W \simeq \sqrt{\frac{8}{3dP}} \quad (4)$$

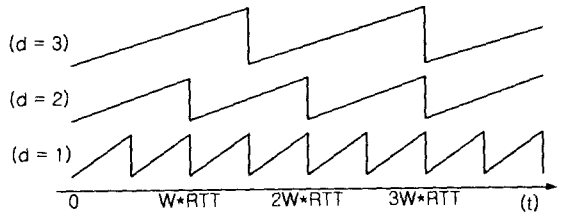

그림 2. 지연 ACK 요소 d와 시간에 따른 윈도우 크기 변화

따라서 식(2), (4)로부터 TCP의 평균 전송률을 구할 수 있다.

$$BW \simeq \sqrt{\frac{3}{2}} \times \frac{MTU}{RTT \times \sqrt{dP}} \qquad (5)$$

4. 시뮬레이션 결과

우리는 ns[9] 네트웍 시뮬레이터(version 2)를 사용해 시뮬레이션을 수행하였다. 가정한 네트웍 구조(topology)는 그림 3과 같다.

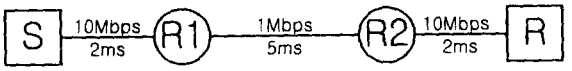

그림 3. 시뮬레이션 네트웍 구조

송신자는 TCP Reno를 사용할 때와 큰 초기 윈도우(large initial window)와 재안한 평활 기법을 사용하는 향상된 TCP Reno를 이용하여 각각 5초 동안 데이터를 보냈다. S-R1간 의 버퍼 사이즈는 5*MSS bytes로 설정하였다.

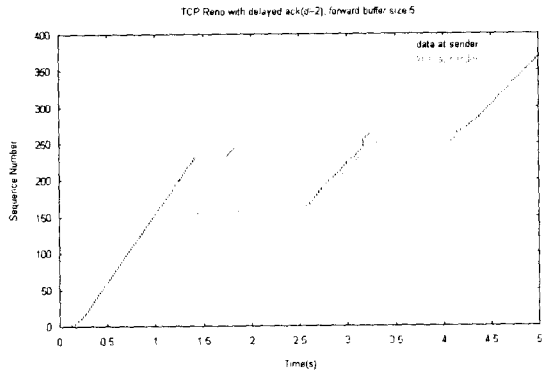

그림 4. 시간에 따라 전송된 데이터 패킷과 ACK의 순서번호

그림 4로부터 TCP Reno 송신자는 1.4초부터 2초 사이의 빠른 복구과정에서 cwnd가 팽창(inflate)되어 데이터를 순서번호 250번까지 보낼 수 있게된다. 여러개의 패킷 손실로 인하여 결국 느린 시작 단계로 넘어가게 되고 3.2초에 Big ACK을 받아서 버스티한 데이터를 네트웍에 전송하게 되어 여러 개의 데이터 손실을 야기함으로 타임아웃이 다시 걸렸다.

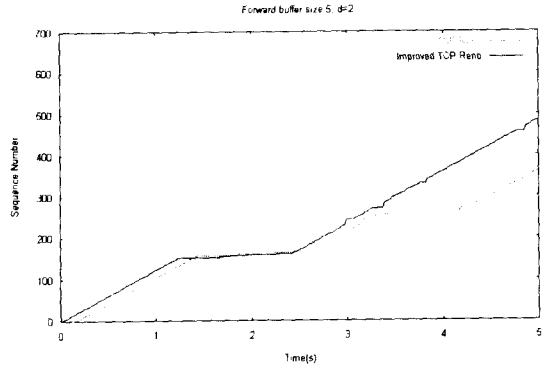

그림 5. 시간에 따라 수신된 데이터 패킷 순서번호

그림 5에서 보이는 바와 같이 수정된 TCP Reno는 초기에 100ms로 설정한 ACK 타임아웃이 걸리지 않는다. 또한 보통의 TCP Reno와는 달리 병목 전송률을 정하는 과정에서만 여러 패킷의 손실이 발생할 뿐 다른 경우에는 발생하지 않음으로써 향상된 성능을 나타내는 것을 볼 수 있다.

5. 결론

본 논문에서는 지연 ACK 옵션을 사용할 때 생기는 여러 가지 성능 저하 요인을 분석하고 해결책을 제시하였다. 분석적 모델링을 통하여 TCP가 보내는 평균 전송률을 구하였으며 이 결과는 TCP에 친화한 전송률 기반 전송방법(TCP Friendly Rate Based Control)에 응용될 수 있다. 또한 시뮬레이션을 통해서 제시한 방법의 성능이 향상됨을 확인하였다.

6. 참고 문헌

[1] M. Allman, V. Paxson, W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.
[2] M. Allman, "On the Generaion and Use of TCP Acknowledgements", CCR 28(5), October. 1998.
[3] M. Allman, "TCP Byte Counting Refinements", ACM CCR29(3), July. 1999.
[4] J. Heidemann, "Performance Interactions Between P-HTTP and TCP Implementation", ACM CCR27(2), April 1997
[5] M. Allman, S. Floyd, C. Partridge, "Increasing TCP' s Initial Window size", RFC 2414, September 1998.
[6] K. Fall, S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", CCR 26(3), July. 1996.
[7] L. S. Brakmo, L. L. Peterson, "Performance Problems in BSD4.4 TCP", CCR 25(5), October. 1995.
[8] A. Aggarwal, S. Savage, T. Anderson, "Understanding the Performance of TCP Pacing", Proceedings of Infocom 2000, March. 2000.
[9] S. McCanc, S. Floyd, "NS(Network Simulator)", 1999 URL http://www-nrg.ee.lbl.gov/.