韓國 OA學會 論文誌
第5卷 第3號, 2000. 9.

2000-5-3-3-10

TimeStamp를 이용한 TCP 혼잡제어 알고리즘

김 노 환*

TCP Congestion Control Algorithm using TimeStamp

No-Whan Kim*

요 약

TCP는 인터넷에서 이미 많은 사용자들이 사용하고 있고 그 성능도 입증되어 왔지만, 망 구조의 급격한 변화에 따라서 TCP의 성능을 이에 맞게 개선하려는 노력들도 많이 제안되고 있다. 특히, 망이 고속화되고 위성 링크 등에서 지연이 증가하면서 Bandwidth-Delay Product도 상당히 커지게 되었다. 이에 대응시키기 위한 방법으로 TCP 옵션에 Window Scale Option과 TimeStamp option, 및 PAWS(Protection Against Wrap Sequence Numbers) 등을 추가하는 방법이 제안되었다. Bandwidth-Delay Product가 큰 망에서는 TCP의 윈도우 크기도 증가하게 됨으로, 현재 사용중인 TCP의 곱 감소 선형적인 증가 방식으로는 망 내의 버퍼 언더플로우 및 망의 불안정을 야기하여, 결국은 TCP 처리율의 감소를 초래하게 된다. 그러므로, 본 논문에서는 위와 같이 윈도우 크기가 큰 망에서의 TCP 혼잡제어 알고리즘을 개선하기 위한 방법으로 TCP의 TimeStamp 옵션을 이용하여 윈도우 크기를 조절하는 혼잡제어 방안을 제시하였다.

Abstract

Through many users employ TCP of which the performance has been proved in Internet, but many papers proposed to improve TCP performance according to varying network architecture. In particular, BWDP(bandwidth-delay product) grew larger because of the increasing delay in satellite link and the network's speed-up. To consider these increased bandwidth-delay product, it is suggested that TCP options include Window Scale option, Timestamp option, and PAWS. Because TCP window size should be commonly high in the network with these increased bandwidth-delay product, the multiple decrease and linear increase scheme of current TCP would cause underflow and instability within network. Then TCP performance is reduced as a result. Thus, to improve TCP congestion control algorithm in the network which has large sized window, this paper proposes the congestion control scheme that controls window size by using Timestamp option.

* 동우대학 사무자동화과 조교수

Ⅰ. 서 론

혼잡 제어 알고리즘(CCA, Congestion Cont Algorithm)은 인터넷상에서 많은 트래픽을 제어하기 위한 방법으로, 그 중요성은 많은 논문들에서 언급된 바 있다[1, 2, 3, 4, 5]. 초기의 TCP 혼잡제어는 패킷 손을 근거로 망의 상황을 판단한 후 윈도우 크기를 조절하거나 재전송 타이머가 타임아웃 되면 망으로 전송되는 패킷의 양을 조절하는 방법을 주로 사용하고 있다.

그러나, 망의 전송속도가 고속화되고 위성 링크 등을 통한 지연이 증가하면 BWDP(Bandwidth-Del Product)가 크게 증가하므로 TCP의 처리율을 고려하여 윈도우 크기도 이에 맞게 증가하여야 한다. BWDP가 작은 망에서 패킷 손실이 발생하면, TCP는 윈도우 크기를 이전 윈도우 크기의 반으로 줄인 후 RTT(Round Tri Time)당 1 패킷씩 선형 증가하므로 원래 윈도우 크기까지 도달하는데 몇 번의 RTT가 필요하다. 한편, BWDP가 큰 망에서 윈도우 크기가 상당히 증가한 상태에서 패킷 손실이 발생하면, 급격히 반으로 감소한 윈도우 크기로 인해 망내에 전송되는 패킷 수가 갑자기 감소하므로 라우터의 버퍼내에 존재하는 패킷 수가 크게 변화하여 망이 불안정해지면서 원래 윈도우 크기까지 도달하는데 긴 시간이 소요되므로 처리율(throughput)이 감소되는 문점이 발생한다.

따라서, BWDP가 큰 망에서는 윈도우 크기를 반으로 줄이는 것보다는 조금만 감소시키는 것이 바람직하며, 작은 윈도우 감소는 망에 부담이 가중되므로 작은 윈도우 감소에 걸맞는 작은 윈도우 증가 방식도 필요하다. 즉, 작은 윈도우 감소와 작은 윈도우 증가 방식이 BWDP가 큰 망내의 패킷 수의 변화를 일정하게 유지함으로 망의 버스트한 특성을 최소한으로 줄일 수 있어서 망 안정화에 큰 도움이 된다.

인터넷은 사용자 급증으로 언제 어떤 트래픽이 발생할지 예상하기 어려우므로 위의 방안만으로 망 안정화를 이루는 데 한계가 있다. 안정화되어 있는 망에 추가적인 트래픽이 입력되면 망 안정화에 커다란 손실이 되고 안정 상태로 복구하는데 많은 시간이 소요되므로, 위의 방안은 더욱 망의 혼잡을 가중시키는 결과를 초래한다. 그러므로, 망 상황을 판단하여 망의 혼잡이 증가하지 않은 경우에만 위의 방안을 실행하는 것이 바람직하다.

망으로 전송하는 트래픽 양을 조절하는 가장 적합한 방식은 망이 요구하는 트래픽 만큼만 사용자가 전송하는 것인데, 현재 망과 사용자간 대화방법이 없으므로 사용자가 망 상황을 추정 판단하는 방법 이외에는 해결책이 없다. 물론, 라우터에서 ECN(Explicit Congest Notification)을 사용하여 망으로 전송되는 트래픽 양 조절할 수는 있지만, ECN은 단지 망의 혼잡 여부만을 사용자에게 알려 혼잡제어를 수행하게 함으로 기존의 문제점인 망의 불안정은 그대로 존재한다.[6, 7].

사용자가 망 상황 판단시 좋은 방법은 RTT 정보를 활용하는 것으로, 현재 TCP는 tick을 이용하므로 좋은 RTT 정보를 얻기가 어렵지만, 그림 1과 같은 TCP header 내의 option 필드에 있는 TimeStamp를 이용면 한 윈도우 내에서 수신되는 패킷마다 정확한 RTT 측정이 가능하므로 망 상황에 대한 정확한 판단 기준을 설정할 수 있다.

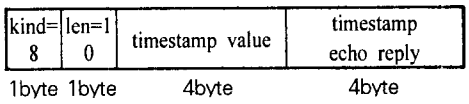

그림 1. TCP option - timestamp

Vegas는 혼잡제어시 RTT를 이용하여 사용중인 대역폭을 판단하는 방법을 사용하고 있는데, RTT가 작으면 망에 혼잡이 없음으로 많은 패킷을 전송하고 RTT가 크면 패킷 전송의 양을 적게 유지하여, 망 안정화 및 TCP 처리율을 일정수준 유지하고 일시적인 망 불안에도 빠른 복구가 가능하다.

Ⅱ. 기존 혼잡제어 알고리즘

2.1 고전적인 TCP의 혼잡제어 알고리즘

현재 인터넷에서 TCP는 Tahoe와 Reno가 가장 많이 사용되고 있다. Tahoe TCP[8]는 3개의 DACK(Duplicate AC

패킷의 손실로 간주하여 손실된 패킷을 빠른 재전송하며, 재전송된 패킷이 손실된 경우에는 타임아웃을 통해 패킷 손실을 감지하고 망 내의 혼잡으로 인한 것으로 간주하고 slow start를 실행한다.

본 논문에서는 Reno를 기준으로 하였다. Reno는 그림 2와 같이 slow start를 ssthresh(Slow S Threshold)까지 계속하다가 ssthresh를 초과하면 RTT당 한 개의 패킷으로 윈도우 크기가 선형 증가하는 혼잡 회피 과정에 들어간다. 혼잡 회피 중 패킷 손실 발생시, 윈도우 크기는 패킷 손실을 감지하는 방법에 따라 다른 동작을 수행한다. 첫째, 3개의 DACK가 수신되면 패킷 손실이 발생된 것으로 간주하여 ssthresh를 현재 윈도우 크기의 반으로 설정하고 Fast Retransmit, F Recovery 후에 ssthresh와 같은 값으로 설정하고 혼 회피 과정을 수행한다. 둘째, 타임아웃으로 패킷 손실이 감지되면 ssthresh를 현재의 윈도우 크기의 반으로 줄이고(곱감소) 윈도우 크기를 한 패킷으로 설정한 후 slow start를 수행한다.

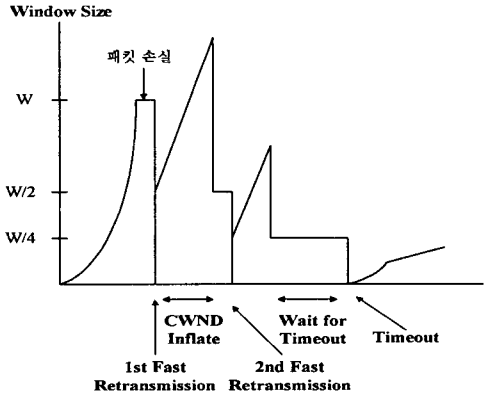

그림 2. 다수 패킷 손실에 대한 Reno의 윈도우 크기 변화

이들을 수식으로 표현하면 다음과 같다.

● 패킷손실 시 ssthresh,

ssthresh = max {min(cwnd/2, adwnd), 2× MSS}

● 패킷손실 시 cwnd

타임 아웃인 경우 : cwnd = 1 × MSS

DACK인 경우 : cwnd = ssthresh

여기서, MSS : Maximum Segment Size

adwnd : receiver advertised window

그러나, Reno TCP는 BWDP가 작은 망에서는 좋은 성능을 보이지만 BWDP가 큰 망의 경우 패킷 손실로 인한 곱 감소, 선형적인 증가로 원래 윈도우 크기까지 증가하는데 많은 시간이 소요되며 곱 감소로 인해 망 내의 버퍼가 급격히 감소하여 심하면 언더플로우가 발생되기도 하므로 망의 불안정과 TCP 처리율의 감소를 초래하는 문제점이 있다.

2.2 Vegas

Vegas는 라우터에 저장되는 패킷 수를 최소한으로 유지하기 위해서 Reno의 slow start, 재전송, 혼잡 회피 알고리즘 수정한 것이다. 특히, 라우터에 저장되는 여분의 데이터 양이 증가하게 되면 망의 혼잡을 가중시키므로 Vegas의 혼잡 회피 알고리즘은 실제 측정한 전송율과 계산에 의한 기대치 전송율의 차이의 값에 따라서 윈도우 크기를 증가, 감소 또는 변화를 주기 때문에 라우터에 저장되는 패킷 수가 일정하게 유지되며 동작 원리는 다음과 같다.

$$\text{Expected rate} = \frac{WindowSize}{BaseRTT}$$

$$Diff = Expected\ rate - Actual\ rate$$

$Diff < \alpha$: 윈도우 증가

$Diff > \beta$: 윈도우 감소

$\alpha < Diff < \beta$: 윈도우 변화 없음

여기서, BaseRTT : 측정한 RTT중에 가장 작은 값.

Actual : 실제 측정한 전송율

α 〈 β의 두 개의 문턱치 값은 망내의 여분의 데이터 양을 제어하는 계수 값이다(α = 1, β = 3).

Ⅲ. 제안된 타임스탬프를 이용한 TCP 혼잡제어 알고리즘

TCP의 혼잡제어 알고리즘은 망의 혼잡 발생 여부를 패킷 손실로 판단하는 방법으로, TCP 송신측이 망상황을 알 수 없는 경우에 적합한 방법이다. TCP Vegas는 라우터의 버퍼 공간을 최소한으로 사용하여 망의 부담

을 최소한으로 유지하는 알고리즘이지만 현재 사용되는 TCP에 비해 보존적인 성향이 강하다[8, 9].

제안된 방법은 slow start, 혼잡 회피 단계는 이전 알고리즘을 그대로 사용하였고 DACK로 인한 Fast retransmit 단계만 수정하였다. 이미 기술한 바와 같이, BWDP가 큰 망에서 기존 TCP의 혼잡회피 단계의 윈도우 크기 변화의 폭이 너무 심하여 망의 불안정하고 처리율이 감소되는 문제점이 발생함으로, 이를 해결하기 위해, 윈도우 크기의 변화를 곱 감소 및 선형 증가를 수행하지 않고, "20% decrease - 0.4 packet increase"를 수행한다. 그러나, 이 방법도 망내의 트래픽이 갑자기 증가되었을 때, 망 혼잡을 가중시키는 문제점이 발생할 수 있다. 망의 상황을 파악하는 방법에는 여러 가지가 있지만, RTT에 의한 방법이 송신측에서 얻을 수 있는 가장 손쉬운 방법 중의 하나이다. TCP 자체의 RTT을 이용하여 망의 혼잡정도를 판단할 수도 있지만 현재 사용 중인 TCP tick이 300~500ms 정도로 너무 커서 정확한 RTT 측정에 어려움이 있다. 그러므로, TCP 옵션 중에 하나인 Timestamp를 활용하면 정확한 RTT를 측정하는 것이 가능하다. 이 Timestamp의 필요성은 윈도우의 크기가 큰 경우에 TCP의 RTT측정은 한 윈도우 당 한번의 측정으로 인한 불확실한 재전송 타이머 값이 설정되는 문제점을 개선하여, 정확하고 많은 RTT 측정을 수행하기 위해서 제안되었다.

새로운 알고리즘은 수신되는 패킷에 대해 최소 및 최대 RTT를 측정하고 새로이 측정되는 RTT가 최대 RTT와 최소 RTT의 특정 범위 내에 있으면 망에 큰 변화가 없는 것으로 판단하고 패킷 손실이 발생된 경우에 현재 윈도우 크기의 20%만 감소시킨 후 패킷 복구 후에 RTT당 0.4 패킷으로 윈도우 크기를 증가한다. 반대로, 수신된 ACK의 RTT가 특정 범위를 초과하면 망에 혼잡이 발생한 것으로 판단하여 현재의 곱 감소와 선형적인 증가를 그대로 수행한다. 망 변화의 기준을 정한 범위는 다음과 같다.

$$Threshold = Max\ RTT - \frac{Max\ RTT - Min\ RTT}{8}$$

수신된 RTT가 이 임계치를 초과하지 않으면 망이 안정한 것으로, 임계치를 초과하면 망이 혼잡한 것으로 판단하여 이전의 곱 감소 선형 증가 방식을 사용한다. 그림 5는 현재 사용중인 곱 감소 선형 증가를 수행하는 혼잡제어 방식과 "20% decrease-0.4 packet increase"를 행하는 TimeStamp를 이용한 혼잡제어 방식간의 윈도우 크기 변화의 차이를 보여준다.

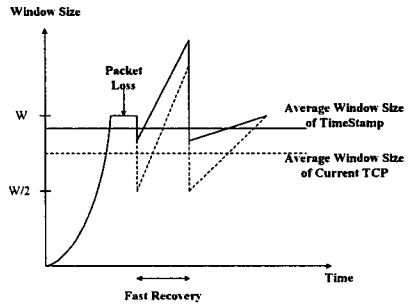

그림 5. 윈도우 크기의 비교

Ⅳ. 모의실험

모의실험은 60초동안 ns-1을 사용하여 수행하였다 송신측의 윈도우 조절이 망에 미치는 영향과 처리율을 비교 분석하기 위해서 그림 6과 같이 망을 구성하였다.

망의 전송 속도를 10Mbps, RTT는 200ms, 1패킷 1Kbyte로 설정하면 BWDP는 250패킷에 해당된다. 송측은 항상 전송할 데이터가 존재하며 윈도우 크기에 의해서만 전송이 지연되고 최대 윈도우 크기는 250패킷으로 설정하였다. 라우터의 혼잡제어 알고리즘은 RED를 적용하였고 버퍼 크기는 100패킷이며, THmin = 25 킷, THmax = 50 패킷, 최대 폐기 확률은 0.02로, q weight는 0.002로 설정하였다.

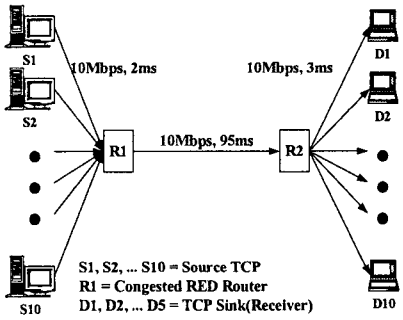

그림 6. 망 배치도

첫번째 모의실험 결과인 그림 7과 8은 10개의 TCP 연결만을 동작시켰을 때 라우터의 버퍼 크기를 나타내었다.

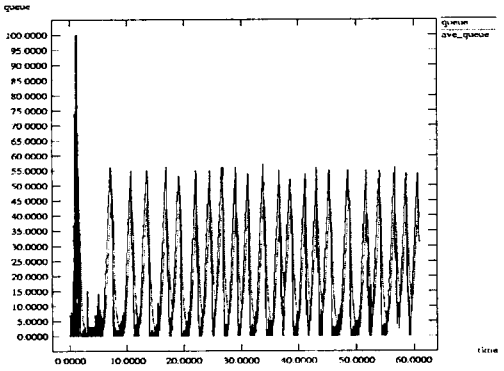

그림 7. 기존 Reno를 사용한 경우에 버퍼의 변화

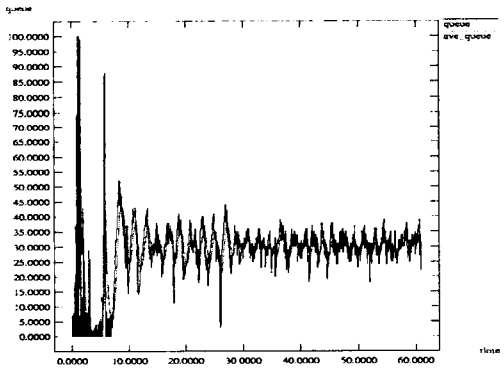

그림 8. TimeStamp를 사용한 경우에 버퍼의 변화

그림 7은 TCP의 곱 감소, 선형 증가로 인한 버퍼의 언더플로가 발생되어 큐에 패킷이 저장되지 않는 반면, 그림 8의 TimeStamp를 이용한 윈도우 변화는 일정한 버퍼의 크기를 유지함으로 TCP의 RTT도 일정하게 유지된다.

두 번째 모의실험 결과인 그림 9와 10은 망을 일시적으로 불안하게 하려는 의도에서 25 - 35초 사이에 10초 동안 1Mbps의 UDP 데이터를 전송한 경우의 버퍼 크기 변화로서 그림 9의 현재의 Reno는 UDP 트래픽의 존재와 상관없이 불안한 상태인 반면에, 그림 10의 TimeStamp를 이용한 윈도우 변화는 UDP 트래픽이 망에 입력될 때와 트래픽이 중지되었을 때에만 일시적인 망의 불안을 보이고 있다.

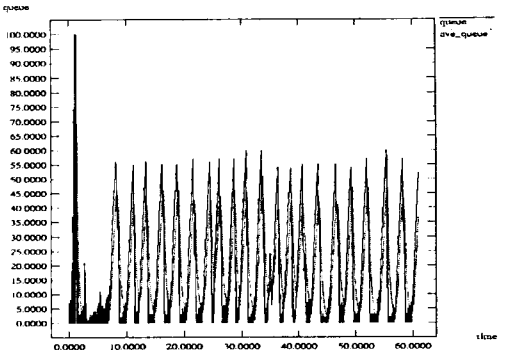

그림 9. 기존 Reno를 사용한 경우에 버퍼의 변화(UDP)

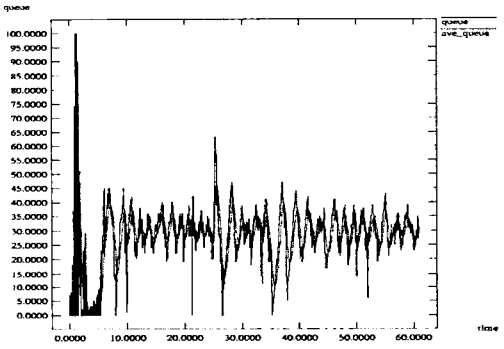

그림 10. TimeStamp를 사용한 경우에 버퍼의 변화(UDP)

그림 11과 12는 두 모의실험을 10번 반복한 결과를 표준편차로 나타낸 것으로, 모두 TimeStamp를 이용한 윈도우 크기 조절 방식이 기존의 혼잡제어 방식보다 좋은 Goodput을 보이고 있다. 이처럼 측정된 RTT를 통하여 망 상황을 판단하여 윈도우를 조절하면 망이 안정화되고 TCP의 처리율이 개선되며 일정한 RTT를 얻을 수 있는 이점이 있다.

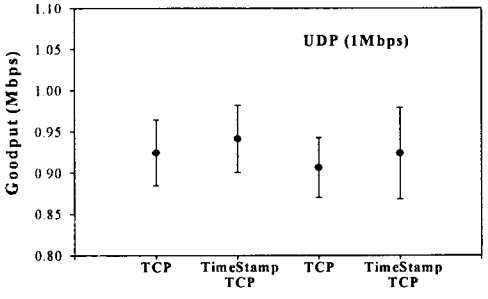

그림 11. Reno 방식의 Goodput의 비교

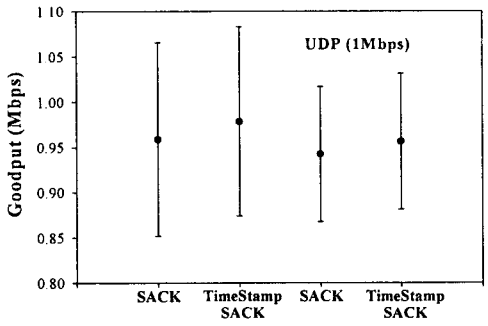

그림 12. SACK 방식의 Goodput의 비교

V. 결론

망의 고속화로 Bandwidth-Delay Product가 크게 가하여 기존 TCP의 혼잡제어 방식인 곱 감소, 선형 증가는 망의 불안정 및 처리율이 감소하는 문제점이 발생한다. 이러한 문제점을 해결하기 위한 방안으로써 TimeStamp를 이용하여 망 상황을 파악하고 이 정보를 토대로 윈도우의 크기를 조절함으로써 망 안정화를 확보하여 TCP의 RTT를 일정하게 유지할 수 있는 혼잡제어 알고리즘을 제안하였다.

참고문헌

[1] V. Jacobson, "Congestion Avoidance and Control", Proceedings of the SIGCOMM' 88 Symposium, pp.314-332, August 1988.

[2] R. Jain. "Congestion control in computer networks : issues and trends, IEEE Network Mag., May 1990, pp24-30.

[3] S. Floyd, Connections with multiple congested gateways in packet-switched networks part 1 : One-way traffic, ACM CCR, Vol No.5, Oct. 1991, pp.30-47

[4] W. Stevens, TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms, RFC 2001, Jan. 1997.

[5] K. Fall and S. Floyd, Simulation-based comparison of Tahoe, Reno, and SACK TCP, ACM CCR, vol.26 no.3, pp.5-21 July 1996.

[6] S. Floyd and V. Jacobson, Random Early detection gateway for congestion avoidance, IEEE/ACM Trans. Networking, vol.1 no.4, pp.397-413, Aug. 1993.

[7] S. Floyd, TCP and congestion Notification, ACM CCR, vol.24 no.5, pp.10-23, Oct. 1994.

[8] M. Mathis and J. Mahdavi, Forward acknowledgement : refining TCP congestion control, Proc. ACM SIGCOM'96 California, Aug. 1996.

[9]. Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance, Proceedings of ACM SIGCOMM'94, Aug. 1994.

저 자 소 개

김 노환
숭실대학교 전자공학과 (공학사)
연세대학교 대학원 (공학석사)
강원대학교 대학원 박사과정 수료.
현재 : 동우대학 사무자동화과 조교수