TCP 전송 효율 개선을 위한 TCP AD-Vegas 알고리즘

박창용*, 황수진*, 박민우**, 이준호**, 정태명*
*성균관대학교 정보통신공학부
**성균관대학교 전자전기통신공학과
e-mail: pcyincs01@gmail.com, sj1228@skku.edu,
{mwpark, jhlee83}@imtl.skku.ac.kr, tmchung@ece.skku.ac.kr

TCP AD-Vegas algorithm for Improving Efficiency of Transmission in TCP

Chang-Yong Park*, Su-Jin Hwang*, Min-Woo Park**, Jun-Ho Lee**, and Tai-Myoung Chung*
*School of Information Communication Engineering, Sungkyunkwan Univ
**Dept of Electrical and Computer Engineering, Sungkyunkwan Univ

요 약

본 논문에서는 TCP Vegas의 혼잡 제어 알고리즘을 개선하여 전송 효율을 높이는 TCP AD-Vegas를 제안한다. 현재 널리 사용되는 Reno 방식은 Additive Increase Multiplicative Decrease(AIMD) 기반의 혼잡제어 방식으로 혼잡이 생길 경우 세션의 전송 크게 줄이는 기법이다. 최근 네트워크 인프라 증가로 큰 대역폭을 사용할 수 있게 되었으나, TCP Reno의 소극적인 혼잡 윈도우 조절로 충분한 전송 효율을 얻지 못하고 있다. 본 논문에서는 RTT를 이용하여 혼잡 정도를 판단하고 혼잡 윈도우를 조절하는 TCP Vegas를 개선하여 TCP의 전송 효율을 높일 수 있는 새로운 알고리즘을 제안한다.

1. 서론

TCP는 현재 인터넷에서 가장 널리 사용하는 프로토콜로, 패킷 수신에 대한 응답을 제공함으로써 신뢰성을 보장한다[5]. TCP는 스트림 기반 통신 프로토콜로 통신 선로 내에서 다른 TCP 세션과 공평하게 대역폭을 나누어 사용한다. 현재 주로 사용하는 혼잡 제어 알고리즘은 TCP Reno 방식을 기반으로 하는 알고리즘이다.

TCP Reno는 패킷이 손실되었거나 순서가 뒤바뀌었을 경우 발생하는 중복 응답을 이용하여 혼잡 여부를 진단하는 방법이다. 이는 AIMD 방식을 이용하여 혼잡한 상황에서 대역폭을 크게 줄여 혼잡을 피한다. TCP Reno는 먼저 자신의 대역폭을 알기 위해 최소한 한 번의 혼잡을 의도적으로 발생시킨다. 또한, 혼잡을 사전에 예방하기보다 혼잡이 발생한 이후 혼잡 윈도우의 크기를 조절하기 때문에 전체 네트워크의 다른 TCP의 성능을 떨어트릴 수 있다. 특히, 한번 혼잡을 경험한 이후 크게 축소된 대역폭에서 소극적으로 혼잡 윈도우를 조절하기 때문에 혼잡을 경험한 세션은 충분한 전송 효율을 얻을 수 없는 문제가 있다[2][6].

일찍이 이런 단점들을 보완하기 위한 방법으로 TCP Vegas 방식이 제안됐다. Vegas 방식은 중복 응답을 이용하여 혼잡을 판단하는 것이 아닌 RTT의 변화 정도를 이용해 혼잡을 감지한다. 패킷 손실을 유도하지 않고 혼잡을 방지하기 때문에 전송 효율을 높인다[3]. 또한 전송률의 변화가 크지 않기 때문에 네트워크의 다른 TCP 플로우에 미치는 영향이 작다. 하지만 Vegas 방식은 처음에 대역을 충분히 확보하기가 어려워 Reno와의 경쟁에서 불리하다. 또한 나중에 들어온 TCP 플로우가 기존에 존재하던 TCP 플로우보다 유리한 대역을 점유하여 불공정성이 생긴다[3][7].

본 논문에서 제안하는 TCP Reno와 Vegas의 혼합 방식은 이러한 단점들을 극복하여 TCP 내의 대역을 최대한 이용하면서도 TCP 간의 불공정성을 유발하지 않도록 TCP AD-Vegas를 제안한다.

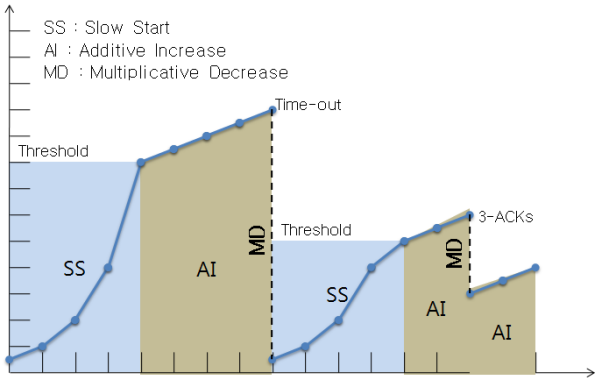
본 논문의 구성은 다음과 같다. 제 2장에서는 TCP Reno 방식과 Vegas 방식의 구체적인 문제점을 지적하고 3장에서는 개선된 TCP 혼잡 제어 알고리즘을 제안한다. 4장에서는 제안한 알고리즘의 성능을 평가하고 5장에서 결론을 맺는다.

2. 관련 연구

가. TCP Reno

TCP Reno 방식은 Slow-start 단계와 Congestion avoidance 단계로 구성된다. Slow-start 단계에서는 혼잡 윈도우의 크기를 1로 초기화하고 지수적으로 그 크기를 증가시킨다. 혼잡 윈도우의 크기가 임계값에 도달하면 Congestion avoidance 단계가 시작된다. Congestion avoidance 단계에서는 혼잡 윈도우의 크기를 1씩 증가시

킨다[1][5].



(그림 1) TCP Reno의 혼잡 윈도우 크기 변화

TCP는 혼잡의 정도에 따라 두 가지 혼잡 제어 방법이 있다. 먼저 혼잡의 정도가 가벼운 경우, 혼잡으로 인해 발생되는 중복 응답을 바탕으로 손실된 것으로 판단되는 패킷을 재전송하고 전송 효율을 반으로 줄여준다. 전송 효율을 줄이기 위해 슬로우 스타트 임계값은 혼잡 윈도우의 반으로 설정하며, 혼잡 윈도우의 크기도 임계값과 동일하게 축소시킨다. 혼잡의 정도가 심각한 경우, 임계값을 현재 혼잡 윈도우의 절반 값으로 설정하고 혼잡 윈도우의 크기를 1로 초기화하여 Slow-start 단계부터 다시 시작한다[1].

TCP Reno 방식에서 혼잡 윈도우 크기 변화는 (그림 1)과 같다. 혼잡을 감지하기 전까지 혼잡 윈도우의 크기는 지속적으로 증가한다. 혼잡이 감지된 후에는 혼잡 윈도우의 크기를 급격히 줄인 뒤 다시 혼잡 윈도우를 조금씩 늘려나간다. 이러한 전송률의 변화는 네트워크에 혼잡을 유도하기 때문에 네트워크에 공존하는 다른 TCP 플로우에도 영향을 미친다. 또한 Congestion Avoidance 단계에서 전송률의 증가율이 낮기 때문에 네트워크의 대역을 효율적으로 사용할 수 없다.

나. TCP Vegas

TCP Vegas에서는 Round Trip Time(RTT)을 이용하여 패킷 손실 전에 혼잡을 제어한다. 데이터를 보내고 응답을 받기까지의 RTT를 계산하고 그것을 미리 결정한 기준 값과 비교한다. 만약 현재의 RTT가 이 기준보다 일정치 이상 클 경우 혼잡 윈도우의 크기를 줄인다. 반대로 현재의 RTT가 기준보다 일정치 이상 작을 경우 혼잡 윈도우의 크기를 늘린다. 만약 RTT와 기준과의 차이가 일정치 이상 나지 않을 경우 혼잡 윈도우의 크기를 늘리거나 줄이지 않는다. RTT와의 비교를 위한 기준은 혼잡이 최소일 경우의 RTT가 되는데 일반적으로 라우터의 큐가 길어지기 전인 첫 번째 TCP 세그먼트에 대한 RTT이다[7].

현재의 RTT를 CurrentRTT, 기준 값을 BaseRTT, 그리고 혼잡윈도의 크기를 W라고 한다면 이들 사이에 다음과 같은 식을 사용한다[3][7].

$$Diff = (W/BaseRTT - W/CurrentRTT) \times BaseRTT \quad (1)$$

이제 식 (1)에서 구한 Diff로 혼잡 윈도우의 크기가 결정된다. 만약 Diff가 어느 특정한 값 α보다 크다면 혼잡 윈도우 크기는 W-1이 된다. 만약 Diff가 어느 특정한 값 β보다 작으면 혼잡 윈도우의 크기는 W+1이 된다. 만약 Diff가 β와 α 사이가 된다면 혼잡 윈도우의 크기는 변화가 없다[2][3][4][7].

이러한 TCP Vegas 방식에는 다음과 같은 한계가 있다. 먼저 동일한 대역폭을 TCP Reno와 함께 사용할 경우 비교적 적은 대역폭을 사용하게 된다. TCP Reno의 Slow-start 단계에서는 혼잡 윈도우의 크기가 지수 적으로 증가하고, Congestion avoidance 단계에서는 패킷 손실이 일어나기 전까지 꾸준히 증가한다. 반면 TCP Vegas 방식에서는 현재의 RTT를 어느 기준 값에 비교하여 혼잡 윈도우의 크기를 조절한다. 이 때문에 Vegas 방식의 TCP와 Reno 방식의 TCP가 같이 쓰일 경우 높은 성능을 보일 수 없다[3][7].

또 다른 한계점으로 먼저 대역폭을 차지하고 있는 Vegas 방식의 세션이 있을 때, 새로 연결되는 Vegas 방식의 TCP가 기존의 세션보다 대역폭을 할당받기 어렵다. 그 이유는 기존의 TCP는 BaseRTT의 값이 새롭게 연결된 TCP의 BaseRTT의 값보다 크기 때문이다. 새로운 TCP가 연결될 당시는 기존 TCP들이 대역을 차지하고 있어 BaseRTT 값이 커질 수밖에 없다. BaseRTT의 값이 커지면 식 (1)에 의한 Diff도 작아지므로 새롭게 연결된 TCP가 유리한 상황이 된다[7].

다. 개선된 TCP Vegas 알고리즘

TCP Vegas의 한계를 개선하기 위해 개선된 알고리즘이 존재한다. 이 방식은 TCP Reno와 TCP Vegas를 혼용하여 사용하는 방법으로, 현재의 RTT인 CurrentRTT를 혼잡이 일어날 때의 RTT인 CongestionRTT와 비교하여 CurrentRTT가 CongestionRTT보다 작을 경우 기존의 Reno 방식에 의해 혼잡 윈도우를 조절한다. 하지만 CurrentRTT가 CongestionRTT보다 크거나 같을 경우 혼잡 윈도우의 크기를 일정하게 유지시킨다. 이 알고리즘에서의 TCP의 흐름 상황에 따른 혼잡 윈도우 설정 프로시저는 <표 1>과 같다[8].

이 알고리즘에는 다음과 같은 문제점이 있다. 먼저 트래픽 증가로 인해 네트워크가 혼잡해져도 패킷 손실이 일어나지 않는 이상 CWND의 값을 변화시키지 않는다. 따라서 패킷 손실 전에 혼잡을 줄이기는 힘들다. 다만 더 이상 혼잡한 상황이 악화되지 않도록 유지시킬 뿐이다. 또 다른 문제는 CongestionRTT 업데이트 방법에 있다. 만약 혼잡하지 않은 네트워크 상황, 즉 MaxRTT의 값이 낮은 상태에서 TCP의 추가로 급격한 트래픽 증가가 생길 경우 네트워크 혼잡에 의해 Three duplicate ACK를 수신할 수 있다. 이렇게 되면 낮은 MaxRTT의 값이

CongestionRTT의 값이 되는데 혼잡이 완화된 이후에도 CongestionRTT의 값이 낮은 상태를 유지하여 CWND를 늘릴 수 없다.

<표 1> 동작 과정

1. 패킷을 정상 수신했을 경우
   CurrentRTT = PacketRTT
   MaxRTT = max(CurrentRTT, MaxRTT)

2. Slow-start 상태일 경우
   if (CurrentRTT >= CongestionRTT)
      keep CWND
   else increase CWND Wth normally

3. Congestion avoidance 상태일 경우
   if (CurrentRTT >= CongestionRTT)
      keep CWND
   else increase CWND Wth normally

4. Retransmit Time이 만료될 경우
   set Wth = 1, CWND =1
   set state to slow-start

5. Three duplicate ACK가 수신된 경우
   CongestionRTT = MaxRTT
   MaxRTT = 0
   reset MaxRTT to BaseRTT
   set state to congestion avoidance

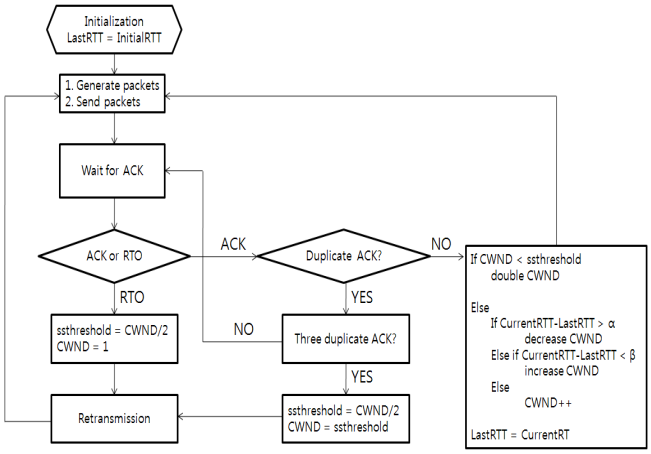
- 용어설명 -

(1) CongestionRTT : 혼잡 발생시의 RTT. 초기 상태는 ∞이며 Three duplicate ACK가 수신될 경우 업데이트된다.
(2) MaxRTT : 수신된 RTT 중 최대 값
(3) CurrentRTT : 현재 수신되는 ACK 패킷의 RTT
(4) Wth : slow-start 임계치
(5) CWND : 혼잡 윈도우

3. TCP AD-Vegas 알고리즘 제안

TCP AD-Vegas는 효율적인 RTT 측정 방식을 추가하여 TCP의 성능을 개선한다.

기존의 TCP Reno 방식이 패킷 손실에 따른 혼잡 윈도우 크기 제어로 효율성의 문제를 발생시킨 것에 비해 TCP AD-Vegas 방식에서는 혼잡 발생으로 패킷 손실이 일어나기 전 혼잡 윈도우 크기를 조정하여 패킷 손실을 막는 것에 초점을 둔다. RTT는 혼잡 발생 가능성을 알려주는 지표가 된다.



(그림 2) TCP AD-Vegas 알고리즘 순서도

제안된 알고리즘에 의하면 CurrentRTT는 항상 LastRTT와 비교가 이루어진다. 만약 CurrentRTT가 LastRTT보다 일정치 이상 크다면 혼잡 윈도우의 크기를 미리 설정해 놓은 양만큼 줄인다. α를 특정한 양수라고 한다면 CWND는 식(2)와 같이 변한다.

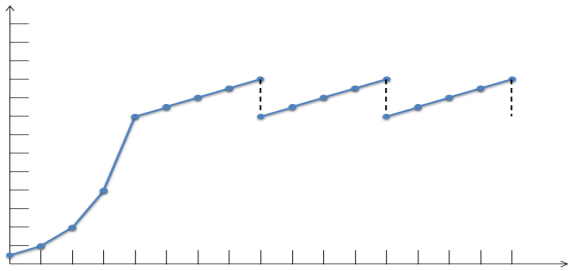
$$\text{If } (\text{CurrentRTT} - \text{LastRTT} > \alpha) \quad \text{decrease CWND} \tag{2}$$

만약 CurrentRT가 LastRTT보다 일정치 이상 작다면 혼잡 윈도우의 크기를 미리 설정해 놓은 양만큼 늘린다. β를 특정한 음수라고 한다면 CWND는 식(3)과 같이 변한다.

$$\text{If } (\text{CurrentRTT} - \text{LastRTT} < \beta) \quad \text{increase CWND} \tag{3}$$

만약 CurrentRTT - LastRTT 값이 α와 β 사이라면 기존의 Reno 방식과 마찬가지로 CWND의 값을 1 증가시킨다. 타임아웃이나 Three duplicate ACK를 수신 했을 경우에도 기존의 Reno 방식을 취한다.

이 알고리즘을 이용하면 패킷 손실을 최대한 줄일 수 있다. CurrentRTT - LastRTT 값의 차이가 α와 β 사이라면 CWND를 계속 높여 대역 효율을 높인다. 혼잡에 의해 RTT의 값이 직전 RTT에 비해 α 이상 높아진다면 CWND의 값을 줄여 혼잡을 완화할 수 있다. 또한 새로운 TCP가 추가되는 이유로 인한 급격한 트래픽 증가가 이루어질 경우에도 미리 혼잡을 예방할 수 있다. 반면에 기존에 존재하던 TCP가 연결을 끊어 혼잡이 완화되면 다른 TCP들도 그만큼 혼잡 윈도우의 크기를 늘려 높은 대역을 유지할 수 있다. (그림 3)에서 보면 TCP 대역의 높고 낮음이 기존의 TCP에 비해 차이가 크지 않으므로 거의 일정한 대역을 유지할 수 있다.



(그림 3) 제안된 알고리즘의 혼잡 윈도우 변화

4. 성능평가

4.1 RTT 변화요인

TCP AD-Vegas 방식은 혼잡 감지를 위해 RTT의 변화를 이용한다. 따라서 RTT의 변화는 알고리즘의 성능평가에 중요한 요인이다.

4.1.1 Congestion

라우터 버퍼에 데이터가 많아지면 큐잉 시간이 길어지므로 지연 시간이 생긴다. 혼잡이 발생하면 지연시간이 길어져 RTT 변화폭이 증가한다.

4.1.2 라우팅 테이블의 변화

라우터가 패킷의 이동 경로를 지정할 때 라우팅 테이블을 참조한다. 라우터는 라우팅 프로토콜을 이용해 라우팅 테이블을 업데이트 하는데 기존의 경로에 이상이 생겨 라우팅 테이블이 바뀌면 패킷의 이동경로가 바뀐다. 이 경우 RTT가 변하게 된다.

위의 두 가지 경우 중 두 번째 경우는 흔하지 않으므로 RTT의 변화는 Congestion의 영향으로 간주한다.

4.2 공정성 평가

4.2.1 Reno와의 공정성

TCP AD-Vegas 방법은 slow-start 임계값에 이르기 전까지 CWND의 값을 지수적으로 증가시킨다. 임계값이 넘어선 이후에도 Congestion에 의해 CWND 값을 줄여야 하는 상황이 오기 전까지는 CWND를 점진적으로 높일 수 있다. 따라서 Vegas 방식에서 발생하는 Reno와의 공정성 문제가 해소된다. 또한 Congestion 발생 시 Reno와는 달리 CWND의 값을 줄이는 폭이 작기 때문에 대역의 효율적인 면도 증대된다.

4.2.2 TCP AD-Vegas 간의 공정성

TCP AD-Vegas를 이용한 TCP는 네트워크에서 이용할 수 있는 최대의 대역을 유지한다. 이 때 새로운 TCP가 추가된다면 트래픽의 증가로 인해 기존의 TCP의 대역은 줄어들고 새로운 TCP의 대역이 늘어난다. 두 TCP간 대역이 균형을 이루면 RTT의 변화폭이 작아져 서로 비슷한 대역을 나눠 갖게 된다. 따라서 TCP AD-Vegas를 이용한 TCP 간에 공정성이 이루어진다.

5. 결론

본 논문에서는 현재 사용하는 TCP 혼잡 제어 방식의 효율을 향상시키기 위한 TCP AD-Vegas를 제안하였다. TCP AD-Vegas는 TCP Reno 방식에 기반을 두지만 기존 Reno와 다르게 RTT를 이용해서 패킷 손실을 미리 예방하고자 하였다. Vegas 방식처럼 RTT를 이용하지만 혼잡의 기준을 BaseRTT로 하지 않고 직전 RTT와의 차로 한다는 점에서 차이가 난다. 이러한 방법으로 Reno가 가지는 문제점인 패킷 손실로 인한 효율성 감소와 Vegas에서 발생하는 TCP 플로우 사이의 공정성 문제를 해결하고자 하였다. 성능 평가에서는 제안한 알고리즘에 중대한 영향을 끼치는 RTT의 변화 요인을 분석하고 공정성을 정성적으로 평가하였다. 향후 연구 방향으로 시뮬레이션을 통해서 제안한 방식의 성능을 정확하게 평가할 계획이며, 혼잡 윈도우의 증감을 결정할 파라미터의 연구를 진행할 계획이다.

참고문헌

[1] Behrouz A. Forouzan, “TCP/IP Protocol Suite 3rd edition”, McGraw-Hill

[2] Mo J., La R.J., Anantharam V., and Walrandi J., "Analysis and comparison of TCP Reno and Vegas", Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE, pp.1556-1563, vol.3, 1999

[3] Srijith K.N., Jacob L., and Ananda A.L., "TCP vegas-A: solving the fairness and rerouting issues of TCP Vegas", Performance, Computing, and Communications Conference, 2003. Conference Proceedings of the 2003, IEEE International, pp.309-316, 2003

[4] Tsang E.C.M, and Chang R.K.C, “A simulation study on the throughput fairness of TCP Vegas", Proceedings Ninth IEEE International Conference on Network, 2001, pp.469-474 Aug, 2002

[5] Yuan-Cheng Lai, and Chang-Li Yao, "The performance comparison between TCP Reno, and TCP Vegas", Parallel and Distributed System: Workshops, Seventh International Conference on 2000, pp.61-66, 2000

[6] 박태준, 이재용, 김병철, “고정대역 네트워크에서 혼잡 윈도우 제한에 의한 TCP 성능개선”, 전자공학회논문지-TC, 제42권 제12호, pp.149-158, Dec. 2005

[7] 이선헌, 송병훈, 정광수, “TCP Vegas에서 공정성 향상을 위한 혼잡제어 알고리즘”, 정보과학회논문지:정보통신 제31권 제3호, pp.583-592, 2004

[8] 최지현, 김대영, 김관웅, 정계택, 전병실, “네트워크 효율 향상을 위한 개선된 TCP 혼잡제어 알고리즘”, 전자공학회논문지-TC 제40권 8호, pp.31-39, Aug. 2003