라우터 버퍼 관리 기반 체증 제어 방식의 최적화를 위한 자체 적응 알고리즘

홍 석 원† · 유 영 석††

요 약

최근 라우터에서 능동적인 큐 관리를 통해서 인터넷의 체증을 회피하는 방식으로 Random Early Detection(RED)에 대해서 많은 논의가 이루어지고 있다. 본 논문에서는 먼저 RED 알고리즘과 현재까지 제안된 RED 변형 알고리즘의 특징을 살펴본다. 그리고 RED의 동작을 이해하기 위한 근본적인 과제로서 RED에서 사용하고 있는 4개의 파라메터가 라우터 큐의 동작에 미치는 영향을 분석한다. RED 파라메터의 분석을 통해서 고정된 파라메터 값을 사용할 경우 다양한 트래픽 상황에 적절히 반응하지 못하는 것을 보여주고 이러한 문제를 해결하기 위한 자체 적응 RED 알고리즘을 제안한다. 제안된 알고리즘에서는 두 가지 파라메터의 값을 조정할 수 있도록 하였다. 먼저 라우터의 큐 상태에 따른 패킷을 폐기하기 위한 최대 확률을 자체적으로 조정하여 트래픽 버스트가 지속되는 경우에 대처할 수 있도록 하였다. 그리고 체증 상태에서 정상 상태로 이행할 경우 평균 큐 길이가 실제 큐 길이에 가능한 빨리 접근할 수 있도록 큐 가중치의 값을 조정할 수 있도록 하였다.

A Self-Adaptive Algorithm for Optimizing Random Early Detection(RED) Dynamics

Sug-Won Hong† · Young-Suk You††

ABSTRACT

Recently many studies have been done on the Random Early Detection(RED) algorithm as an active queue management and congestion avoidance scheme in the Internet. In this paper we first overview the characteristics of RED and the modified RED algorithms in order to understand the current status of these studies. Then we analyze the RED dynamics by investigating how RED parameters affect router queue behavior. We show the cases when RED fails since it cannot react to queue state changes aggressively due to the deterministic use of its parameters. Based on the RED parameter analysis, we propose a self-adaptive algorithm to cope with this RED weakness. In this algorithm we make two parameters be adjusted themselves depending on the queue states. One parameter is the maximum probability to drop or mark the packet at the congestion state. This parameter can be adjusted to react the long burst of traffic, consequently reducing the congestion disaster. The other parameter is the queue weight which is also adjusted aggressively in order for the average queue size to catch up with the current queue size when the queue moves from the congestion state to the stable state.

※ 본 논문은 정보통신부의 대학기초연구지원사업의 지원으로 이루어졌음.
† 종신회원 : 명지대학교 전자정보통신공학부 교수
†† 정 회 원 : (주)디지텔 연구소 연구원
논문접수 : 1999년 4월 7일, 심사완료 : 1999년 9월 22일

1. 서 론

현재 인터넷에서 체증을 해결하는 방법은 이미 발생한 체증에 반응하여 송신 호스트의 윈도우 크기(window size)를 통해 전송율을 조절하는 것으로 오로지 양 종단 TCP 호스트에만 의존하는 방법이 사용되고 있다[1, 2]. 하지만 이미 발생한 체증에 대하여 수동적으로 동작하는 이 방법으로는 현재의 인터넷이 경험하고 있는 폭발적인 트래픽 증가 상황에 대처하기에는 크게 미흡한 점이 있다. 체증이 이미 발생한 상태에서는 발생 지점의 라우터 버퍼내의 큐 길이는 이미 최대치에 도달해 있으며, 따라서 이후에 도착하는 모든 패킷은 폐기되므로 이 패킷들을 전송한 모든 송신 호스트들은 거의 동시에 윈도우 크기를 줄이는 slow start 단계에 들어가게 되어 일시적으로 링크 사용율이 떨어지는 global synchronization이 발생하게 된다. 이러한 문제를 해결하기 위해서는 망 종단 송신 호스트의 전송율을 조절하는 현재의 소스 기반(source-based)의 체증 제어와 함께 망 중간 노드인 라우터에서 버퍼를 관리함으로써 체증이 발생하기 전에 능동적으로 대처하는 방안이 필요하다. 그렇게 함으로써 global synchronization에 의해 순간적으로 링크의 사용율이 극단적으로 감소하는 것을 방지할 수 있을 뿐만 아니라 라우터 내의 큐 길이를 항상 작은 크기로 유지할 수가 있게 된다. 고속망으로 갈수록 망 중간 노드에서의 큐잉 지연(queueing delay)이 전체 패킷 처리율(throughput)에 매우 큰 영향을 미치기 때문에 중간 노드의 큐 길이를 작게 유지하는 것도 체증 제어의 중요한 목적의 하나로 인식되고 있다[3].

이와같이 라우터의 버퍼 관리를 통한 체증 회피(congestion avoidance)를 수행하는 방식으로 현재 주목을 받고 있는 것이 RED(Random Early Detection) 방식이다[4]. RED는 평균 큐 길이에 따라 망의 체증 정도를 결정하여 이에 따라 패킷을 폐기(drop)하는 확률을 결정한다. RED를 사용할 경우 현재의 라우터에서 사용하는 Drop-Tail 방식과 비교하여 링크의 사용 효율을 높일 수 있다는 것은 이미 여러 문헌에서 보여주고 있다[3, 4, 5, 6].

한편 RED를 다양한 트래픽 상황에서 적용할 경우 발생하는 문제점을 지적하고 이를 개선하기 위한 시도가 제안되었다[7, 8]. 이러한 제안은 크게 두 가지 점으로 요약할 수 있다. 먼저 상이한 특성을 갖는 트래픽 유형에 대해서 동일한 패킷 폐기 확률을 사용할 경우 초래되는 링크 사용의 공평성에 대해서 지적하고 있다. 또 다른 지적은 고정된 패킷 폐기 확률(deterministic probability)을 사용할 경우 트래픽의 버스트 상황의 변화에 대해서 제대로 반응하지 못한다는 점이다.

본 논문에서는 RED를 사용하는데 있어서 RED 파라메터의 값을 어떻게 정하느냐는 근본적인 문제에서부터 출발하여, RED 성능의 최적화를 위한 바람직한 파라메터 사용의 방향을 제시해보도록 한다. 이를 위해 먼저 RED 파라메터 값이 라우터 큐에 미치는 영향을 분석하고, 잘못된 파라메터 값을 사용했을 경우에는 오히려 RED 알고리듬이 실패하는 트래픽 상황도 발생할 수 있음을 보여 주었다. 그리고 이와 같은 분석을 토대로 하여 트래픽의 변화에 따라 최적의 파라메터 값을 찾아가는 자체 적응 알고리듬을 제안하여 RED의 문제점을 해결해 보고자 하였다. 제안된 알고리즘을 사용하였을 때의 성능 향상을 시뮬레이션을 통해 살펴보았다.

2. RED와 RED 변형 알고리즘

RED는 버퍼에서의 평균 큐 길이(이후 $Qavg$로 표기)를 이용하여 체증 제어를 수행한다. RED는 매 패킷이 도착할 때마다 $Qavg$를 구하고, 이것을 미리 정해 놓은 파라메터인 최소 큐 한계값(minimum threshold, 이후 min_{th}로 표기)과 최대 큐 한계값(maximum threshold, 이후 max_{th}로 표기)과 비교한다. $Qavg$가 min_{th} 보다 작을 때에는 망의 링크의 사용이 낮은 수준에 머물러 있다고 판단할 수 있으므로 모든 패킷은 정상적으로 처리된다. 하지만 $Qavg$가 min_{th}와 max_{th}사이에 있을 때 도착하는 패킷은 확률, p_a에 의해 랜덤하게 폐기되든지 체증 통보(congestion notification)를 위한 비트(ECN bit)를 마크한다 [11, 12]. 이 확률은 0에서부터 RED내에 정의된 상수인 최대 확률(maximum probability, 이하 max_p로 표기)까지의 범위 내에 존재한다. $Qavg$가 max_{th} 보다 클 때에는 $Qavg$가 max_{th} 밑으로 떨어질 때까지 도착하는 모든 패킷을 폐기하거나 ECN 비트를 마크한다.

$Qavg$는 EWMA(Exponential Weighted Moving Average)를 통한 low-pass filter를 사용하여 다음과 같이 계산한다.

$$Qavg \leftarrow (1-w_q)*Qavg + w_q*q$$

$Qavg$를 구할 때 현재의 큐 길이가 $Qavg$에 미치는 영향은 큐 가중치(Queue Weight, 이하 w_q로 표기)로 결정된다. RED에서는 w_q를 이용하여 일시적인 버스트 트래픽의 영향으로 인해 실제 큐의 길이가 짧은 시간 동안 증가하더라도 $Qavg$가 바로 영향을 받지 않도록 하고 있다.

RED의 목적은 망에 체증이 발생하여 라우터의 큐에 도착하는 패킷을 전부 폐기하기 전에 능동적으로 라우터의 큐 관리를 통해서 체증을 회피하고, 각 TCP 연결에 대해서 링크의 대역을 사용하는 양에 비례하여 패킷을 폐기하도록 함으로써 모든 TCP 연결이 slow start 단계에 들어가서 링크의 사용 효율이 순간적으로 급격히 떨어지는 global synchronization 현상을 피하기 위함이다.

최근의 연구에서는 RED가 갖는 문제점을 지적하고 이를 개선하기 위한 변형된 RED 알고리즘을 제시하고 있다. RED가 갖는 문제점 중의 하나로 지적되는 것은 라우터의 링크를 공유하고 있는 여러 트래픽 형태에 따른 구분 없이 모두 동일한 기준에 의해서 패킷의 폐기를 결정한다는 점이다. FRED(Flow RED)에서는 트래픽 유형을 UDP를 사용하는 오디오나 비디오 트래픽과 같이 TCP 체증 제어 방식에 순응하지 않는 트래픽과 TCP 체증 제어에 순응하지만 허락하는 범위에서 가용 대역을 모두 차지하는 트래픽, 그리고 telnet 서비스와 같이 가용 대역을 사용하는데 민감하게 반응하지 않는 트래픽으로 구분하고 있다[7]. FRED에서는 라우터의 큐에서 각 TCP 연결이 차지하고 있는 큐 길이(per-flow buffer count)를 추적함으로써 트래픽 유형에 상관없이 모두 공평하게 링크를 사용할 수 있도록 하고 있다.

RED의 또 다른 문제점으로는 패킷을 폐기 혹은 ECN 비트를 마크하기 위해서 사용하는 최대 확률 max_p을 고정된 값으로 사용하는데서 비롯되는 문제이다. 만약 라우터 큐에서의 변화에 따라 max_p값을 조정하면 더 높은 링크 사용 효율을 얻을 수 있다는 점이다. ARED(Adapted RED)는 라우터가 처리하는 트래픽의 부하에 따른 큐 길이의 변화를 추적하여 최대 확률을 변화함으로써 링크의 효율을 높이는 방안을 제안하고 있다[8].

FRED와 ARED가 RED가 갖고 있는 문제점을 개선하기 위한 변형 알고리즘이라면 WRED(Weighted RED)는 트래픽 클래스에 따라서 다른 패킷 폐기 확률을 적용하여 우선 순위가 낮은 패킷을 먼저 폐기할 수 있도록 하고 있다. 즉, WRED는 IP precedence 비트에 의해 구분되는 트래픽 클래스에 대해서 별도의 RED 큐를 운영하며 각 큐에 대해서 다른 RED 파라메터를 적용한다. 따라서 WRED는 RED의 능동적인 라우터 큐 관리에 의한 체증 제어와 함께 트래픽 클래스 간의 우선 순위에 따라 패킷의 폐기를 차별화하는 RED의 확장된 시도라고 볼 수 있다[9].

RIO(RED with In and Out)도 WRED와 같이 RED의 확장된 알고리즘으로 볼 수 있다[10]. RIO에서는 서비스 사용 계약(Service Level Agreement)를 준수하는 패킷은 In-profile 패킷으로, 그리고 위반하는 패킷은 Out-profile 패킷으로 구분하여 두 개의 큐를 만들어 RED 알고리즘을 적용한다. 만약 큐 길이가 min_{th}와 max_{th} 사이에 있을 경우 In-profile 큐의 패킷은 폐기없이 전송하지만 Out-profile 큐의 패킷은 랜덤하게 선택되어 폐기된다. 만약 큐 길이가 max_{th}를 넘을 경우 두 개의 큐의 패킷이 모두 랜덤하게 폐기되는데 이 때 Out-profile 큐의 패킷이 더욱 높은 확률로 폐기된다. 이와 같이 RIO 알고리즘은 지나치게 대역을 남용하는 사용자로부터 규약을 준수하는 선량한 사용자가 공정한 링크 대역을 사용하는 것을 보장하려고 시도하고 있다.

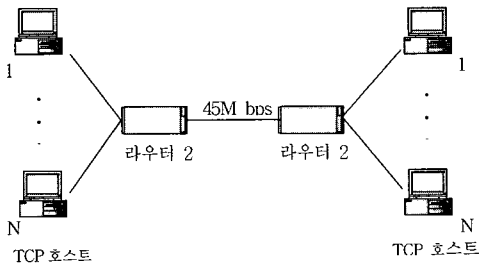
3. RED 파라메터에 대한 동작 특성

3.1 시뮬레이션 구성

RED를 적용하기 위해서는 RED 내에서 정의되어 있는 4개의 파라메터(w_q, max_p, min_{th}, max_{th})가 RED 라우터의 동작에 어떠한 영향을 미치고 있는지를 먼저 정확히 이해할 필요가 있다. 본 논문에서는 (그림 1)과 같은 망 구성에 의해 시뮬레이션을 수행하였다. 각 TCP 호스트들은 2KB 크기의 패킷을 현재의 윈도우 크기(cwnd) 만큼 계속 전송하도록 하였다. 각 호스트들은 자신이 전송한 패킷이 체증의 영향으로 폐기되었을 경우 TCP의 체증 제어 방식인 slow start와 congestion avoidance를 행하게 된다[1, 2]. 그리고 송신 호스트에서 수신 호스트로 단방향 전송만을 가정하였다. 각 호스트의 최대 윈도우 크기(advertised window size)는 모두 64KB로 하였다.

연결된 호스트의 수는 5개, 10개, 20개로 하였으며, 연결 호스트가 증가할수록 트래픽의 버스트 효과가 커져 라우터에서 체증이 심해지게 된다. 송신 호스트에

서 라우터 1까지의 전송 지연 시간(propagation delay)은 1.0ms에서 3.0ms까지의 범위에서 정해지며 라우터 1로부터 수신 호스트까지의 전송 지연 시간은 5ms로 고정시켰다. 각 호스트들은 시간의 차이를 두고 전송을 시작하도록 하였으며, 시뮬레이션이 시작 후 10ms 이내에서 전송을 시작하도록 하였다.



(그림 1) 시뮬레이션 모델

3.2 큐 가중치와 평균 큐 길이

평균 큐 길이(*Qavg*)를 구할 때 실제 큐 길이가 *Qavg*가 변하는데 미치는 영향을 나타내는 값이 큐 가중치(w_q)이다. w_q가 작으면 작을수록 현재의 큐의 실제 길이가 *Qavg*가 변하는데 미치는 영향이 작게 된다. 이런 경우에는 짧은 시간 동안 큐의 길이가 증가되더라도 *Qavg*가 증가하는데 미치는 영향이 작기 때문에 일시적인 트래픽의 증가로 인한 오류를 피할 수 있다.

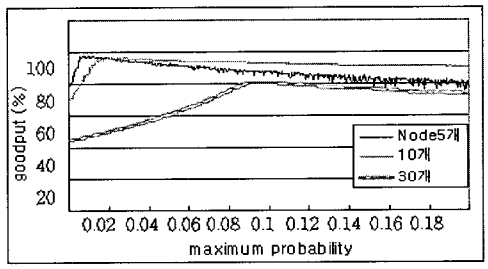
하지만 w_q를 마냥 작게만 주는 것도 문제가 있다. *Qavg*가 max_{th}를 넘었을 경우가 그것이다. *Qavg*가 max_{th}를 넘었을 경우에는 도착하는 모든 패킷을 폐기하기 때문에 이 패킷에 해당하는 TCP 연결을 갖는 송신 호스트는 모두 slow start 단계에 들어가기 때문에 전송 효율이 크게 떨어지게 된다. 그래서 *Qavg*가 max_{th}를 넘었을 때에는 가능한 한 빨리 *Qavg*가 max_{th}밑으로 떨어져야 하는데 실제 큐 길이는 매우 감소했다 하더라도 w_q가 너무 작기 때문에 이미 올라간 값이 떨어질 때에도 많은 시간이 필요하게 된다.

w_q가 너무 클 경우에는 이와는 반대로 *Qavg* 값이 실제 큐의 길이의 영향을 많이 받게 된다. *Qavg*가 실제 큐 길이의 영향을 받아 기복이 크다는 것은 RED를 사용하는 의의를 줄이는 일이다. 일시적인 버스트 트래픽에 대해서도 바로 반응을 해버린다면 이는 Drop-Tail 방식과 별반 다를 것이 없기 때문이다. w_q는 [4]에서는 0.001이상을 쓰도록 권하고 있고, [4, 5, 6]에서 모두 0.002를 쓰고 있다.

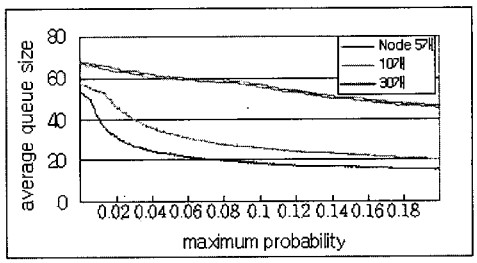
3.3 최대 확률(maximum probability)

max_p의 값을 조절할 경우 가장 영향을 받는 부분은 *Qavg*이다. max_p가 커질 경우에 패킷 폐기 확률(packet drop probability)이 커져 폐기하는 횟수가 많아지므로 실제 큐 길이나 *Qavg*가 전체적으로 줄어들게 된다.

(그림 2)와 (그림 3)은 max_p를 0.0001에서 0.2까지 변화시키며 goodput과 *Qavg*의 평균을 살펴본 것이다. 여기서 goodput은 라우터를 통해 폐기되지 않고 전송되는 패킷을 의미하며 실제 처리율(effective throughput)이라고 할 수 있다. (그림 2)를 보면 망의 버스트가 심해질수록 큰 max_p에서 최대 goodput이 나오는 것을 볼 수 있다. (그림 3)은 max_p에 대한 *Qavg* 값을 보여주고 있다. 이 값은 당연히 max_p가 커질수록 점차 작아지게 된다.



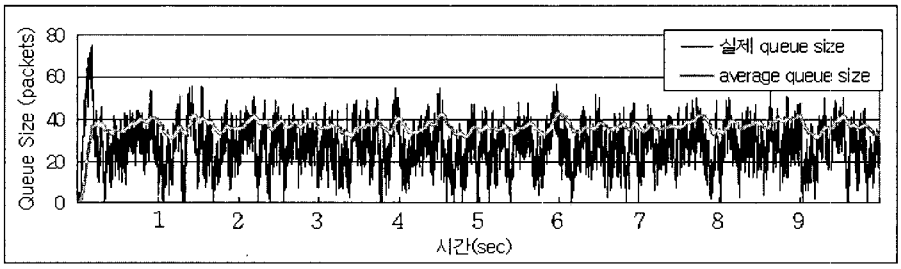
(그림 2) maxp vs goodput



(그림 3) maxp vs 평균 큐 길이

3.4 최소 한계값(minth), 최대 한계값(maxth)

실제 망에 RED를 적용했을 때 min_{th}, max_{th} 값을 설정하는 것은 상당히 중요한 일이지만, 이것은 수학적으로나 시뮬레이션 만으로 정할 수 있는 파라메터는 아니다. [4]에서 이 파라메터의 범위를 다음과 같이 정하고 있다.



(그림 4) 실제 큐 크기와 평균 큐 크기의 변화 (TCP 호스트 10개, maxp = 0.02)

트래픽이 버스트한 특성을 갖을수록 min_{th}는 크게 주어 링크의 높은 사용 효율을 유지할 수 있도록 해준다. max_{th}는 가급적 작게 주는 것이 유리하다. max_{th}는 Drop-Tail의 최대 버퍼 크기와 비슷하게 동작하므로 이 파라메터의 값이 클 경우는 큐 길이가 전체적으로 높게 유지되기 때문에 큐잉 지연 시간(queueing delay)이 길어지기 때문이다. max_{th}와 min_{th}의 차이는 왕복 시간(one roundtrop time) 동안에 계산되는 평균 큐 길이의 추정치 보다 커야 RED 라우터는 효율적으로 동작할 수 있다. 대략 max_{th}를 최소한 min_{th}의 두배 이상으로 할 것을 권고하고 있다.

본 논문의 시뮬레이션을 통해 볼 때 max_{th}는 이전 Drop-Tail 시 최대 큐 길이의 60~70%로 정해 주면 별 무리가 없고, min_{th}는 가능한 크게 해주는 것이 좋겠지만, global synchronization을 없애기 위한 min_{th}와 max_{th}와의 폭을 넓게 해주기 위해 max_{th}의 20% 정도로 정해주었을 때 상대적으로 좋은 결과를 얻을 수 있었다.

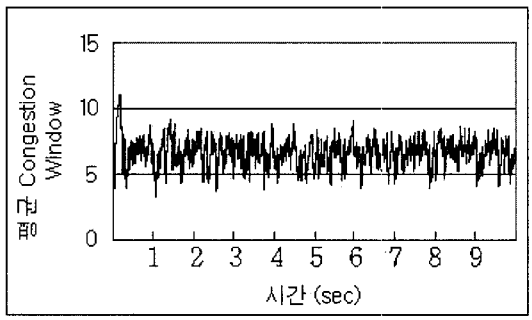
3.5 RED 파라메터 적용의 문제점

위에서 설명한 바와 같이 RED의 4가지 파라메터는 그 값을 어떻게 정하느냐에 따라 RED의 성능에 중대한 영향을 미친다. 하지만 어떠한 트래픽 상황에서도 범용적으로 적용할 수 있는 파라메터를 결정하는 것은 어려운 과제이다. 이 파라메터의 최적값은 망의 트래픽 상황에 따라 달라지기 때문에 결국은 트래픽 상황의 변화에 따라 그 값을 변화할 수 있어야 할 것이다.

RED의 파라메터를 고정시켜놓고 호스트의 수를 계속 증가시키면 어느 순간에는 효율이 극히 나빠지는 상황이 발생되었다. 그 이유는 트래픽의 버스트 기간이 증가됨으로써 심한 체증 현상을 초래하게 되며 이러한 트래픽 상황에 대해서는 고정적으로 정한 RED의 파라메터가 원래의 의도대로 동작하지 못하기 때문이다.

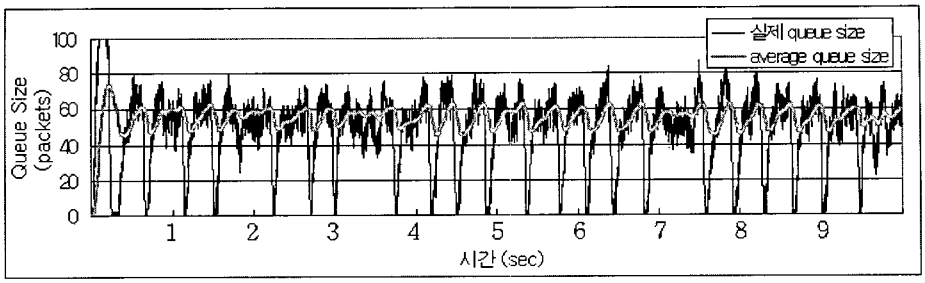
RED에서는 패킷을 마크하는 확률은 0에서 max_p 범위에서 존재한다. 그러므로 max_p로도 $Qavg$를 max_{th}이하로 유지하지 못할 만큼 트래픽의 버스트가 지속되면 $Qavg$는 결국 max_{th}를 넘게되고, 버스트 상황이 끝나고 정상적인 트래픽 상황이 되어 실제 큐의 길이가 max_{th}보다 작아지더라도 $Qavg$는 오랫 동안 max_{th}를 넘어서는 경우가 발생한다. 이러한 경우 $Qavg$가 max_{th} 밑으로 떨어질 때까지 모든 패킷을 폐기하며 결국 global synchronization이 발생하게 된다.

다음의 (그림 4~9)는 max_p를 제외한 파라미터를 [4]에서 권장한 값으로 설정하고 시뮬레이션을 실행한 결과이다.($w_q = 0.002$, $max_{th} = 60$ 패킷, $min_{th} = 9$ 패킷) (그림 4, 6, 8)에서 검은 선은 실제 큐의 길이를, 흰 선은 $Qavg$를 나타낸다.

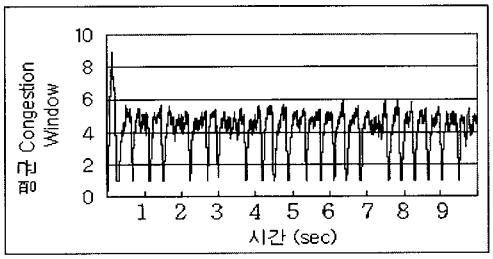


(그림 5) Congestion Window(cwnd) 크기의 변화 (TCP 호스트 10개, maxp = 0.02)

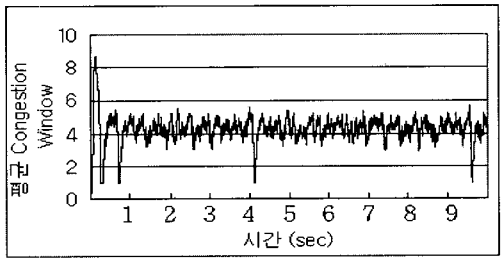
(그림 4)와 (그림 5)는 10개의 호스트가 연결되어 있는 경우이다. 이때에는 (그림 4)에서 보는 바와 같이 $Qavg$가 max_{th}(60패킷) 보다 항상 작게 유지되어 정상적으로 동작하게 된다. 하지만 호스트의 수를 20개로 늘리게 되면, (그림 6)과 (그림 7)에서 보는 바와 같이 max_p가 너무



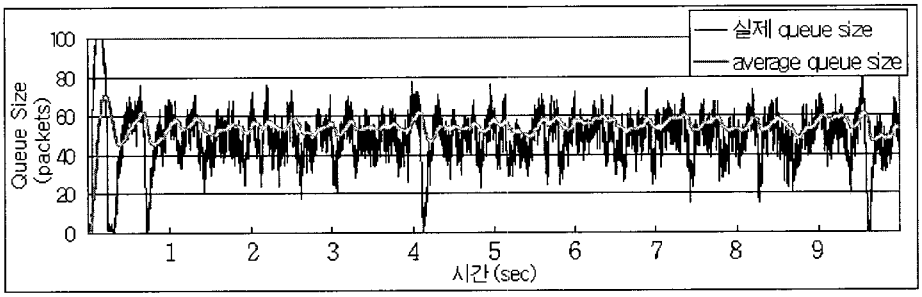
(그림 6) 실제 큐 길이와 평균 큐길이의 변화(TCP 호스트 20개, maxp = 0.02)



(그림 7) Congestion Window(cwnd) 크기의 변화
(TCP 호스트 20개, maxp = 0.02)



(그림 9) Congestion Window(cwnd)의 변화
(TCP 호스트 20개, maxp = 0.03)



(그림 8) 실제 큐 길이와 평균 큐길이의 변화(TCP 호스트 20개, maxp = 0.03)

작기 때문에 $Qavg$가 max_{th}(60패킷)를 자주 넘어 수시로 global synchronization이 발생하여 모든 호스트가 slow start 단계에 들어가게 된다. 이렇게 발생한 체증의 영향은 Drop-Tail 보다 심각하다. RED는 $Qavg$가 max_{th} 보다 클 경우 체증이 발생한 것으로 판단하여 모든 패킷을 폐기하고 이에 응답하는 호스트들이 slow start에 들어가 실제 큐 길이가 크게 떨어지더라도 $Qavg$는 w_q에 의해 서서히 떨어지게 되므로 실제 체증은 해결되었지만 RED는 체증 상황으로 오인하여 도착하는 패킷을 계속 폐기하게 된다. (그림 8)과 (그림 9)는 max_p=0.03으로 실행한 결과이다. 이 경우 이전과는 달리 max_p를 크게 하면 RED가 제대로 동작할 수 있음을 보여주고 있다.

4. RED 최적화를 위한 자체 적응 알고리즘

RED는 w_q와 max_p의 값이 고정되어 있기 때문에 트래픽 상황이 큰 폭으로 변할 때에는 유연하게 대처하지 못하는 문제점을 갖고 있다. 이러한 문제를 해결하기 위해서는 w_q와 max_p를 트래픽의 상황에 맞추어 적절하게 변화시킬 필요가 있다. 본 논문에서는 RED의 이러한 단점을 보완하기 위해서 트래픽 버스트 상황에 따른 큐의 변화에 따라 w_q와 max_p을 자체 적응하는

RED 알고리듬을 제안한다.

4.1 최대 확률과 큐 가중치의 자체 적응

RED에서 체증 회피(congestion avoidance)를 위해서 $Qavg$에 따라서 0부터 max_p까지의 범위를 갖는 확률에 의해 패킷을 폐기함으로써 체증 발생 전에 미리 전송률을 조정한다. 하지만 max_p를 계속 적용하더라도 트래픽의 버스트가 길어질 경우에는 결국 $Qavg$가 max_{th}를 넘게 되고, $Qavg$가 max_{th} 밑으로 떨어질 때까지 도착하는 모든 패킷이 폐기되어 버리는 사태가 발생할 수가 있다. 이렇게 되면 결국 global synchronization이 발생할 수 밖에 없다. 이러한 상황을 방지하기 위해서는 트래픽의 상황에 따라 max_p를 증가시켜 $Qavg$가 max_{th} 보다 작게 유지될 수 있도록 해야 한다. 이를 위해서 본 논문에서는 max_p를 트래픽의 버스트의 정도에 따라 증가 또는 감소하여 최적화된 값을 찾아가도록 하였다.

RED에서 체증이 발생했다고 판단할 수 있는 시점은 $Qavg$가 max_{th}를 넘어서는 순간부터라고 할 수 있다. 따라서 그 순간에 max_p를 높여주면, 앞으로의 트래픽 상황에 대해서 능동적으로 대처할 수 있게 된다. 또한 상당히 심한 버스트한 상황이어서 이렇게 올린 max_p로도 부족해서 계속 체증이 발생하면, 그 순간 또 다시 확률을 높여 준다. 이와 같이 시간이 지날수록 그 상황에 알맞는 max_p가 될 때까지 증가되어 결국은 알맞은 값의 max_p에 도달하게 된다.

그리고 체증이 발생한 상황, 즉 $Qavg$가 max_{th}를 넘었을 때에는 모든 패킷을 폐기하여 1 RTT 후에는 송신 호스트가 slow start에 들어가게 되므로 실제 큐 길이는 max_{th} 밑으로 떨어지게 된다. 하지만 $Qavg$는 w_q의 영향으로 아주 완만하게 움직이므로 실제 큐의 길이는 아주 작은 값임에도 불구하고 $Qavg$는 max_{th} 위에 지속적으로 머물게 되어 계속해서 도착하는 모든 패킷을 폐기하게 된다. 이러한 상황을 피하기 위해서 $Qavg$가 max_{th}를 넘어있는 기간 동안에는 w_q를 증가하여 가능한 빠르게 $Qavg$를 max_{th} 밑으로 떨어질 수 있도록 하였다.

버스트한 상황이 해소되어서 더 이상 $Qavg$가 max_{th}를 넘어서는 일이 없을 때에는 max_p를 낮추어 주어야 한다. 이를 위해서는 $Qavg$가 min_{th}, max_{th} 사이에서 미리 정의된 안정 시간(stable period, 이하 s_period로 표기) 이상 머물면 max_p를 증가시키는 폭보다는 작은 폭으로 서서히 max_p를 낮추어 준다. max_p를 감소시키는 시간 간격 s_period는 매번 증가된다. 그 이유는 $Qavg$가 정상 상태($Qavg$가 min_{th}와 max_{th} 사이에 머무는 상태)에 있으면서 최적의 max_p를 찾아주기 위해 계속 max_p를 감소시키다 보면, 어느 순간에는 결국 체증이 발생할 수 있으므로 이것을 가능한 피하기 위해서 s_period를 증가시키며 max_p를 감소시켜준다.

4.2 알고리즘

w_q와 max_p의 값을 큐의 길이에 따라 자체 적응하는 RED 알고리즘이 아래에 나와 있다. 원래의 RED 알고리즘과 사용된 변수와 파라메터는 [4]에서 그대로 인용하였다. 그리고 원래의 알고리즘에서 새로 추가된 부분은 코멘트 라인으로 표시하였다.

이 알고리즘에서 보는 바와 같이 $Qavg$가 max_{th}를 넘어서는 순간을 위해서 $IsBurst$를 정의했고, max_p를 감소시킬 때를 위해 변수 $IsNonBurst$, s_period와 상수 min_s_period을 정의했다. min_s_period는 $Qavg$가 체증 발생 후 max_{th} 이하로 떨어질 때 s_period를 초기화해주는 시간으로 $Qavg$가 max_{th} 이하로 떨어진 후 정상 상태가 s_period까지 지속된 후에는 max_p를 감소시키고 s_period는 증가된다.

또한 $Qavg$가 min_{th}와 max_{th}사이에 있을 때는 두 가지의 일을 하게 된다. 하나는 $Qavg$가 max_{th}위에서 밑으로 떨어지는 순간에 w_q을 원래의 값으로 낮추고, max_{th}를 낮추기 위한 준비 작업을 하게 된다. 두 번째는 $IsNonBurst$와 s_period를 이용해서 현 시간 $time$과 비교하며, max_p를 낮추는 일을 한다. $IsNonBurst$에는 체증이 끝나는 시간 또는 확률을 떨어뜨렸을 때의 시간을 갖고 있으므로 두 한계값(threshold) 사이에 머무르고 있는 시간 즉 $time - IsNonburst$가 s_period 만큼 지속되었을 경우에는 s_period는 증가시키고 max_p은 낮추어 준다. 이러한 일은 정상 상태인 동안 계속 반복된다.

Saved Variables :

$Qavg$: average queue size
q_time : start of the queue idle time
$count$: packets since last marked packet
$IsBurst$: flag for burst state
$IsNonBurst$: flag for nonburst state
max_p : maximum value for p_b
w_q : queue weight

s_period : time interval to decrease max_p

Fixed parameters:

min_{th} : minimum threshold for queue
max_{th} : maximum threshold for queue
min_s_period : minimum time to decrease max_p

Other:

p_a : curent packet-marking probability
q : current queue size
time : current time
$f(t)$: a linear function of the time t

Initialization:

$Qavg \leftarrow 0$
count ← -1
IsBurst ← FALSE
IsNonBurst ← 0

for each packet arrival
calculate the new average queue size *Qavg*:
if the queue is nonempty
$Qavg \leftarrow (1-w_q)*Qavg + w_q*q$
else
$m \leftarrow f(time-q\_time)$
$Qavg \leftarrow (q-w_q)^m*Qavg$
if $min_{th} \leq Qavg < max_{th}$
increament *count*

// 평균 큐 길이가 min_{th}, max_{th} 사이에 있을 때 w_q와 max_p의 조정 시작//

if *IsBurst* == TRUE
IsBurst ← FALSE
decrease w_q
IsNonBurst ← *time*
else if *time-IsNonBurst* > *s_period*
decrease max_p
increase *s_period*
IsNonBurst ← *time*

//평균 큐 길이가 min_{th}, max_{th} 사이에 있을 때 wq와 max_p의 조정 끝//

calculate probability p_a :
$p_b \leftarrow max_p(Qavg-min_{th})/(max_{th}-min_{th})$
$p_a \leftarrow p_b / (1-count*p_b)$
with probability p_a :
mark the arriving packet
count ← 0
else if $max_{th} < Qavg$
mark the arriving packet
count ← 0

// 평균 큐 길이가 max_{th}를 넘었을 때 w_q와 max_p의 조정 시작 //

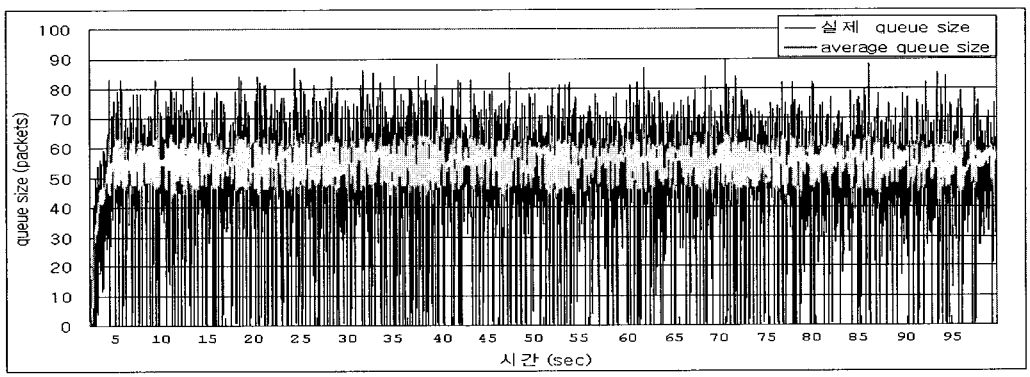
if *IsBurst* == FALSE
increase max_p
increase w_q
s_period ← *min_s_period*
IsBurst ← TRUE
IsNonBurst ← 0

// 평균 큐 길이가 max_{th}를 넘었을 때 w_q와 max_p의 조정 끝 //

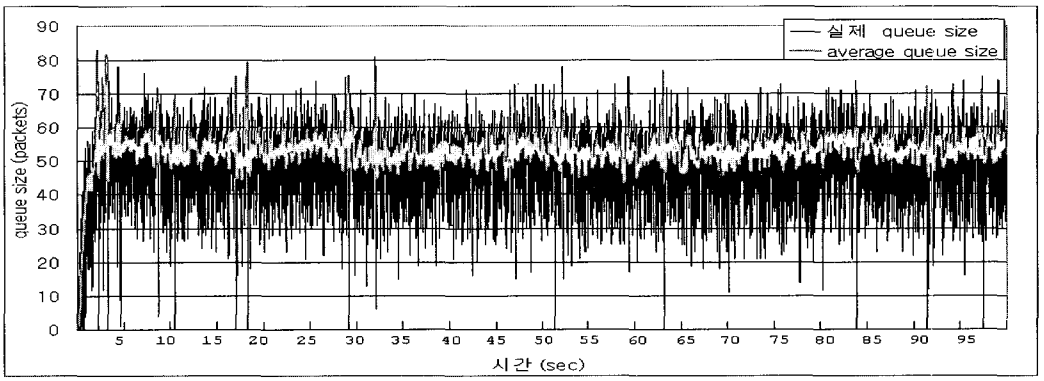
else *count* ← -1
when queue becomes empty
q_time ← *time*

4.3 시뮬레이션 결과

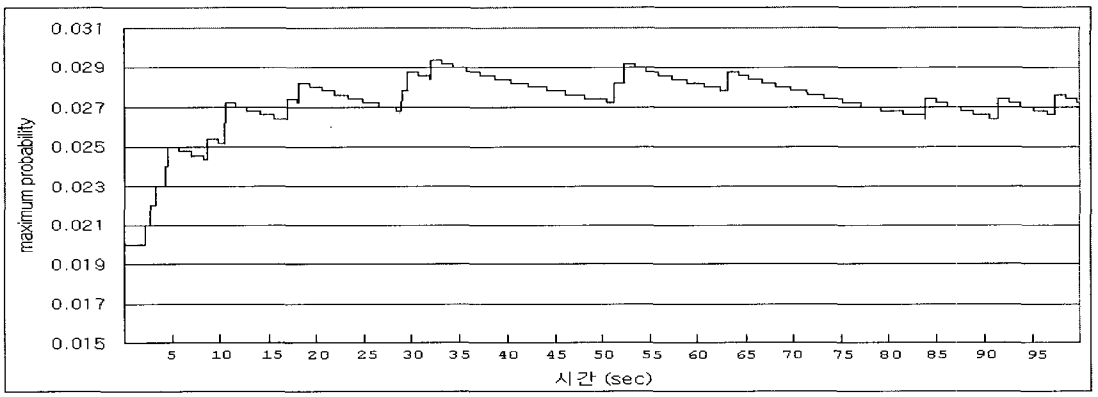
이 시뮬레이션에서도 각 파라미터의 값들은 전과 같은 값을 사용했으며 max_p는 0.001 씩 증가시켰으며 감소시킬 때에는 0.0002 씩 감소를 시켰다. 감소시키는 시간폭 *s_period*의 초기값 *min_s_period*은 1초, 증가폭은 0.1초씩 하였다. 그리고 이 경우에는 시뮬레이션 시간을 100초로 다른 경우에 비해 길게 하였다.



(그림 10) RED를 사용할 때의 큐 길이



(그림 11) 자체 적응 RED를 사용할 때의 큐 길이



(그림 12) 자체 적응 RED를 사용할 때의 최대 확률(maxp)의 변화

(그림 10)은 20개의 호스트가 연결된 RED 경우를 보여 주고 있다. 이 경우 수시로 체증이 발생하여 global synchronization으로 인해 큐 길이가 0까지 떨어지는 지점이 많이 보이고 있다. (그림 11)과 (그림 12)는 같은 상황에 대해 자체 적응 RED를 적용했을 경우이다. (그림 11)을 보면 초기에는 max_p가 너무 작은 값이기 때문에 여러 번 체증이 발생하며 max_p가 증가되는 것이 보인다. 하지만 max_p가 어느 정도 커지게 되면 체증이 상당히 큰 간격으로 발생해 RED에 비해 매우 좋아진 것을 알 수 있다. 그리고 w_q값을 조정했기 때문에 (그림 11)에서 $Qavg$가 $max_{th}(60)$를 넘는 시간이 오래 지속되지 못하므로 (그림 10)과 비교하여볼 때 평균 큐 길이(그림의 흰 선)의 그래프가 날카롭게 치솟은 것을 볼 수가 있다.

(그림 12)의 max_p의 변화를 보면 체증이 발생할 때 max_p가 증가되는데, max_p가 0.29정도로 커진 후부터는 체증의 발생 회수가 줄어들므로 다음 max_p가 증가될 때까지의 간격이 길어진 것을 볼 수 있다. 이는 이 값이 이러한 트래픽 상황에서는 최적값에 접근한 값임을 보여주고 있는것이다.

5. 결 론

본 논문에서는 능동적인 큐 관리를 통한 체증 회피 방식으로 주목받고 있는 RED 알고리즘과 RED의 문제점을 개선하거나 기능을 확장하기 위해 최근 제안된 RED 변형 알고리즘에 대해서 설명하여 현재까지 RED에 관련되어 이루어지고 있는 연구 결과를 살펴보았다. 그리고 이러한 연구의 근본적인 과제로서 RED 파라메터가 RED 큐의 동작에 미치는 영향을 분석하였다. RED 파라메터의 특성 분석을 통해서 파라메터들

을 고정된 값으로 사용할 경우 트래픽의 버스트가 매우 길어지는 상황에서는 RED가 제대로 기능을 발휘하지 못하는 상황이 발생할 수 있음을 시뮬레이션을 통해 보여 주었다. 하지만 모든 트래픽 상황에 적용될 수 있는 최적의 파라메터 값은 존재하지 않으며, 결국 RED 성능의 최적화를 위해서는 트래픽 상황의 변화에 따라 능동적으로 RED 파라메터의 값을 변화할 필요성이 있음을 보여주었다.

본 논문에서는 라우터의 큐 길이에 따라서 정상 상태(stable state)와 체증 상태(congestion state)로 구분하고 큐 상태에 따라 max_p를 변화함으로써 최적의 패킷 폐기 확률을 찾아가도록 하였다. 또한 실제 큐의 길이가 체증 상태에서 정상 상태로 이행하였을 경우 $Qavg$도 가능한 빨리 정상 상태로 이행할 수 있도록 w_q를 조정하는 방안을 제시하였다.

하지만 이러한 휴리스틱(heuristic) 알고리즘의 한계는 이와 같이 max_p와 w_q를 조절함으로써 모든 트래픽 상황에 대해서 반드시 최적의 효율을 얻을 수 있다는 보장은 할 수 없다는 점이다. 이를 위해 더욱 다양하고 광범위한 트래픽 상황에 대한 시뮬레이션과 분석을 통해 이 방식의 타당성을 입증해야 할 것이다. 이것은 결국 모든 RED 변형 알고리즘이 갖는 한계라고 할 수 있다. 이러한 한계에도 불구하고 RED의 올바른 사용을 위해서는 큐의 상태 즉 트래픽 상황에 따른 파라메터의 자체 적응 방식은 필요하다고 할 수 있다.

참 고 문 헌

[1] V. Jacobson, "Congestion Avoidance and Control," *Computer Communication Review*, 18(4) : 314-329, August 1988.

[2] W. Stevens, "TCP Slow start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," RFC 2001, January 1997.

[3] B. Braden et al., "Recommendations on Queue Management and Congestion Avoidance in the Internet," RFC 2309, April 1998.

[4] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," IEEE/ACM Transactions on Networking, 1(4) : 397-413, 1993.

[5] M. Gaynor, "Proactive Packet Dropping Methods for TCP Gateways," http://www.eecs.harvard.edu/~gaynor/final.ps.

[6] 유영석, 홍석원, "체증 제어를 위한 Random Early Detection(RED) 알고리즘의 파라메터 분석", 통신학회 추계학술발표 논문집 p.41-44, 1997년 11월.

[7] D. Lin and R. Morris, "Dynamics of Random Early Detection," Proc. of ACM SIGCOMM, September 1997.

[8] W. Feng et al., "Technique for Eliminating Packet Loss in Congested TCP/IP Networks," U. Michigan CSE-TR-349-97, http://www.eecs.umich.edu/~wuchang/ared.

[9] "Advanced QoS Services for the Intelligent," Cisco white paper, http://www.cisco.com/warp/public/732/net_enabled/qos_wp.htm#HDR7.

[10] D. Clark and J. Wroclawski, "An Approach to Service Allocation in the Internet," Internet Draft <draft-clark-different-svc-alloc-00.txt>, July 1997.

[11] S. Floyd, "TCP and Explicit Congestion Notification," *Computer Communication Review*, 34(3) : 10-23, October 1994.

[12] K. Ramakrishnan, and J. Raj, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks," ACM Transactions on Computer Systems, 8(2) : 158~181, 1990.

홍 석 원

e-mail : swhong@wh.myongji.ac.kr

1979년 서울대학교 물리학과 (학사)

1988년 North Carolina State University 전산학과(석사)

1992년 North Carolina State Univerisity 전산학과(박사)

1993년~1995년 전자통신연구원 광대역 통신망 연구부 선임연구원

1995년~현재 명지대학교 전자정보통신공학부 부교수

관심분야 : 네트워크 기술 및 프로토콜, 성능 분석

유 영 석

e-mail : ysy@dgtel.co.kr

1998년 명지대학교 전자정보통신 공학부 졸업

1998년~현재 (주)디지텔 연구소 연구원

관심분야 : 인터넷 접속 기술, 인터넷 체증 제어