폭주제어를 위한 효율적인 RED 알고리즘

고동엽*, 임석구
백석대학교 정보통신학부
e-mail: dengx2.com@gmail.com, sklim@bu.ac.kr

Effective RED Algorithm for Congestion Control

Dong-Yub Ko*, Seog-Ku Lim
*Div. of Information and Communications, BaekSeok University

요 약

인터넷에서 폭주상황이 발생한 후에는 네트워크 성능이 급격하게 저하되는 문제점을 개선하고자 RED 알고리즘이 제안되었으며, 현재 IETF에서는 표준으로 사용되고 있다. 그러나 RED 알고리즘은 고정된 파라미터 값을 사용하여 가변적인 네트워크 트래픽 상황에 능동적으로 대처하지 못하는 문제점을 가지고 있다. 본 논문에서는 기존의 RED 알고리즘의 문제점인 고정된 파리미터 값 설정으로 인한 문제점을 해결하기 위하여 효율적인 RED 알고리즘을 제안한다. 제안한 알고리즘은 현재의 가변적인 트래픽 상태에 따라 패킷폐기확률을 조정하여 라우터와 같은 게이트에서의 폭주상황을 효율적으로 제어하도록 하였다. 또한 제한안 알고리즘은 NS-2를 이용하여 시뮬레이션을 수행함으로써 성능을 검증하였다.

1. 서 론

초기 인터넷에서 폭주를 제어하는 방법은 이미 발생한 폭주에 반응하여 송신측의 원도우 크기(Window Size)를 이용하여 전송률을 조절하는 방식이 사용되었다[1,2]. 하지만 이미 발생한 폭주에 대해서 수동적으로 동작하는 이 방법으로서 폭발적으로 트래픽이 증가하는 현재의 인터넷에서는 미흡한 점이 있다. 이러한 문제를 해결하기 위해서 네트워크 종단간 송신 호스트의 전송률을 조절하는 현재의 소스기반의 폭주제어와 함께 네트워크 중간 노드인 라우터에서 버퍼를 관리함으로써 폭주가 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 적절한 크기로 유지할 수 있게 된다.

이와 같이 라우터의 버퍼관리를 통한 폭주회피(Congestion Avoidance)를 수행하는 방식으로 현재 인터넷의 표준으로 사용되고 있는 방식이 RED(Random Early Detection)이다[3,4]. RED는 평균 큐 길이에 따라 네트워크의 폭주 정도를 결정하여 이에 따라 입력되는 패킷을 폐기(Drop)하는 확률을 결정한다. RED 방식은 Drop-Tail 방식과 비교하여 링크의 사용 효율을 높일 수 있다[4].

한편, RED를 다양한 트래픽 상황에서 적용할 경우 발생하는 문제점을 지적하고 이를 개선하기 위한 시도가 있었다[5,6]. 이러한 시도는 크게 두 가지 측면으로 요약할 수 있다. 먼저 상이한 특성을 갖는 트래픽 유형에 대해서 동일한 패킷 폐기확률을 사용할 경우 초래하는 링크 사용의 불공평성을 지적하고 있다. 다른 측면은 고정된 패킷 폐기확률을 사용할 경우 트래픽의 버스트 상황의 변화에 대해서 올바르게 반응하지 못한다는 점이다.

본 논문에서는 트래픽의 변화에 따라 패킷의 폐기확률을 조정하여 라우터에서의 폭주상황을 효율적으로 제어하기 위한 방안을 제안하였다. 또한 제한안 알고리즘은 NS-2를 이용하여 시뮬레이션을 수행함으로써 성능을 검증하였다.

서론에 이어 2장에서는 Drop-Tail과 RED의 알고리

증을 간략하게 설명하고, 3장에서는 RED 파라미터의 설정, 시뮬레이션 모델, 그리고 시뮬레이션 결과를 분석하며, 마지막으로 4장에서는 결론을 맺는다.

2. Drop-Tail과 RED 알고리즘

2.1 Drop-Tail 알고리즘

Drop-Tail 알고리즘은 가장 단순한 큐 관리 메커니즘으로서 라우터 큐의 가용 한계까지 패킷을 저장하고 큐가 overflow 상태가 되면 그 이후에 들어오는 패킷을 폐기하는 방법이다.

$$\begin{aligned} &\text{if}(q\_len \geq q\_limit) \\ &\quad q \rightarrow remove() \end{aligned} \tag{1}$$

2.2 RED 알고리즘

RED 알고리즘은 평균 큐의 크기를 이용하여 네트워크의 폭주를 감지하여 제어하는 기능을 가지고 있다. RED는 라우터의 큐 상태에 따라서 도착하는 패킷에 대하여 선택적 폐기를 함으로서 네트워크가 폭주상태로 이르는 것을 방지한다.

RED 알고리즘에서는 새로운 패킷이 도착할 때 마다 식(2)와 같은 EWMA(Exponentially Weighted Moving Average)에 의해 평균화된 큐의 길이를 구하여 확률적으로 패킷을 폐기한다. 일반적으로 RED는 폭주상태를 예측하여 네트워크에 걸리는 부하를 줄임으로서 Drop-Tail 알고리즘보다 성능이 우수하다.

$$Q_{avg} \leftarrow (1-w_q) \times Q_{avg} + w_q \times Q_{size} \tag{2}$$

여기서 Q_{avg}는 평균 큐 길이, w_q는 큐의 가중치, Q_{size}는 현재 큐 길이를 의미한다. 식 (2)에서 산출된 Q_{avg}를 미리 정한 큐의 임계값 min_{th}, max_{th}와 비교하여 패킷을 폐기한다.(식 (3) 참조)

$$\begin{aligned} &\text{if}(min_{th} \leq Q_{avg} < max_{th}) \\ &\quad \text{폐기 확률 } P_b\text{를 계산하고} \\ &\quad P_b\text{확률로 패킷을 폐기 또는 } mark\text{한다.} \\ &elseif(Q_{avg} > max_{th}) \\ &\quad \text{패킷을 폐기 또는 } mark\text{한다.} \end{aligned} \tag{3}$$

패킷 폐기확률(P_b)은 식 (4)와 같이 0~max_p 사이의 값을 가지게 되고 Q_{avg}가 최소 임계값(min_{th})에서 최대 임계값(max_{th})에 이르기까지 선형적으로 변한다. Q_{avg}가 max_{th}에 근접할수록 더 높은 확률로 패킷을 폐기한다. [그림 1]에 RED에서의 패킷 폐기확률을 나타내었다.

$$P_b = P_{max} \times \frac{Q_{avg} - min_{th}}{max_{th} - min_{th}} \tag{4}$$

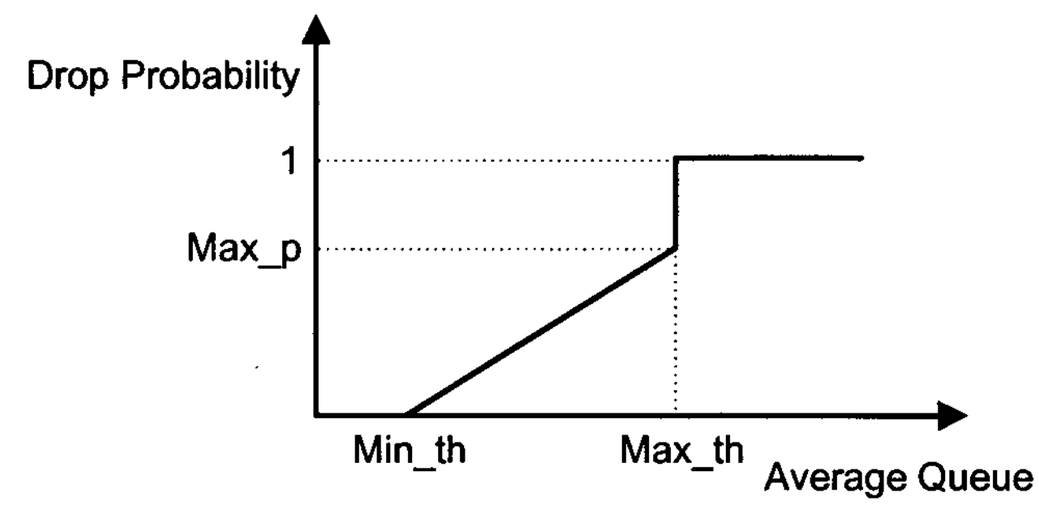


그림 1. RED에서의 패킷 폐기확률

패킷이 폐기될 확률은 max_p값에 따라 변하게 되므로 이 값의 설정은 RED 알고리즘의 성능에 큰 영향을 끼친다. RED 알고리즘은 다양한 파라미터를 가지고 있으며 주요 파라미터는 다음 <표 1>과 같다.

〈표 1〉 RED에서의 주요 파라미터

w_q	Q_{avg}를 계산하는 가중치 값
min_{th}	Q_{avg}의 최소 임계값
max_{th}	Q_{avg}의 최대 임계값
max_p	큐에서의 폐기 확률

w_q는 과거 큐의 값을 얼마나 반영하는지에 대한 가중치 값으로 w_q값이 작을 경우 Q_{avg}의 변화 폭이 작아지게 되어 현재 큐의 크기는 줄었으나 Q_{avg}의 크기가 줄어드는 시간이 오래 걸리게 되므로 불필요한 폐기가 계속 일어나게 된다. 반면 w_q값이 클 경우에는 Q_{avg}의 변화폭이 커지게 되므로 일시적으로 폭주하는 패킷을 모두 수용하지 못하게 된다. RED 알고리즘은 이러한 다양한 파라미터 값에 따라서 성능이 달라지므로 네트워크의 특성에 따라 파라미터 값을 설정해 주는 것이 중요하다. 적절하지 못한 파라미터 값의 설정은 Drop-Tail 알고리즘보다 떨어진 성능의 결과가 나오게 된다. min_{th}값은 Q_{avg}값이 min_{th}값보다 큰 경우부터 RED 알고리즘이 적용되게 하는데 이 값이 작을 경우엔 빈번한 패킷 폐기 현상이 일어나게 되고 max_{th}값이 너무 작을 경우 큐의 자원 활용률이 떨어지게 되며 반대로 너무 큰 경우 Drop-Tail과 비슷한 결과를 가져오게 된다. RED 알고리즘 중에는 동적으로 max_p값을 변화시

켜 일반 RED보다 더 높은 성능을 가진 Adaptive RED가 있다[4].

Adaptive RED는 큐 길이 변화를 추적하여 가장 적절한 max_p값을 적용하는 것으로 max_p값을 어떻게 변화시키느냐에 따라 AIMD(Addictive Increasing /Multiplicative Decreasing)와 MIMD(Multiplicative Increasing/Multiplicative Decreasing) 방식이 있다.

큐 관리에 RED 알고리즘을 사용하는 주요 목적은 평균 큐 크기를 제어함으로써 폭주를 회피하는 것이고 추가적으로는 global synchronization을 방지하고 bursty한 트래픽 상황에 적용하는데 있다.

2.3 제안 RED 알고리즘

본 논문에서 제안한 RED 알고리즘은 min_{th}와 max_{th}의 중간값(M) $(min_{th}+max_{th})/2$을 전후로 P_b 기울기를 다르게 설정한다. RED의 기본적인 알고리즘(3)에서 수정한 알고리즘은 다음과 같다.

$$\text{if}\left((min_{th}+max_{th})\times\frac{1}{2}\le Q_{avg}<max_{th}\right) \quad (5)$$

$P_b = x\times v_c\times v_{ave}+v_d$; [$x$: 기울기를 결정할 값]
P_b 확률로 패킷을 폐기 또는 $mark$한다.

$$elseif\left(min_{th}\le Q_{avg}<(min_{th}+max_{th})\times\frac{1}{2}\right)$$

$P_b = v_c\times v_{ave}+v_d$;

$elseif(Q_{avg}\ge \max_{th})$
패킷을 폐기 또는 $mark$한다.

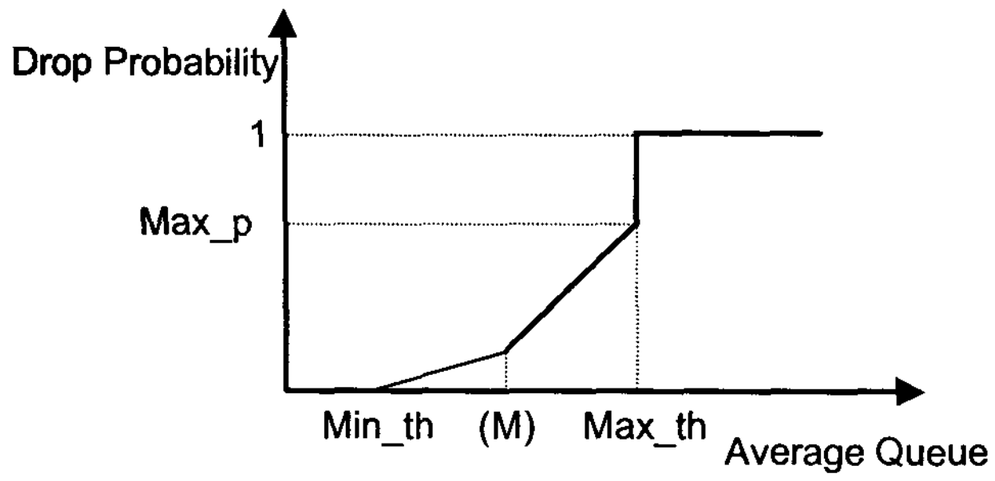


그림 2. Proposed-RED에서의 패킷 폐기확률

제안된 RED 알고리즘은 (5)의 식을 이용하여 min_{th}와 max_{th}의 중간 값을 기준으로 Q_{avg}의 값이 M보다 크고 max_{th}값 보다 작으면 v_c값에 x를 곱한다. 본 논문에 적용한 x 값은 5.0 이다.

3. 시뮬레이션 모델 및 결과분석

3.1 RED 파라미터 설정

가중치 파라미터 w_q = 0.002

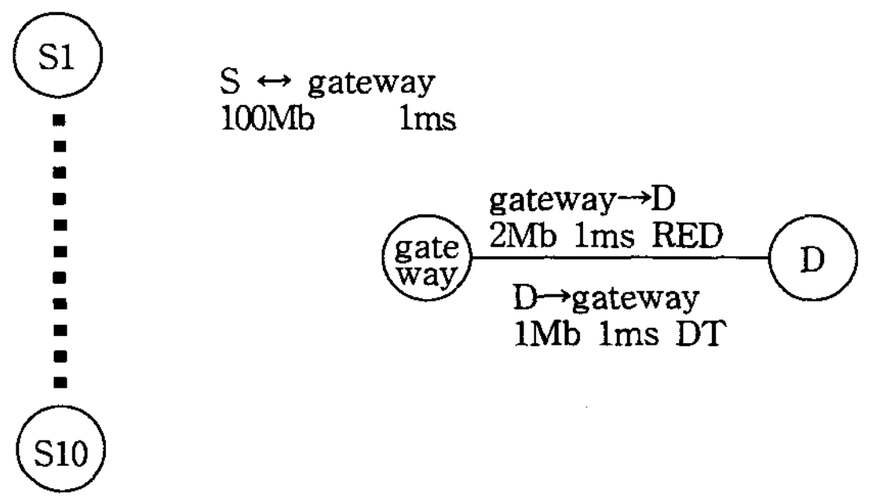
최대 폐기 확률 max_p = 1/30

최소 임계값 min_{th} = 5

최대 임계값 max_{th} = 15

3.2 시뮬레이션 모델

본 논문에서 사용한 시뮬레이션 모델은 [그림 3]과 같이 구성하였고 소스노드는 1개에서 10개 까지 증가시키며 트래픽을 증가하였다. 각 소스노드에서는 평균 0.02sec 간격의 지수분포와 평균 1000byte 크기의 파레토(Pareto) 분포로 패킷 전송을 시작한다. 소스노드와 gateway 사이의 링크는 100Mb의 대역폭과 1ms의 링크 delay이며 gateway 와 목적지노드 사이의 링크는 2Mb의 대역폭과 1ms의 링크 지연을 가정하였다. gateway에서 D사이 링크의 큐 크기는 100으로 설정하였다.



S : Source Node
D : Destination Node

그림 3. 시뮬레이션 모델

3.4 시뮬레이션 결과분석

시뮬레이션 모델은 Linux 환경 기반에 ns2-2.30 과 Tcl ,Tk 를 이용하여 수행하였다. 각 시뮬레이션에서 성능 파라미터는 지연(delay)과 수율(throughput)이다.

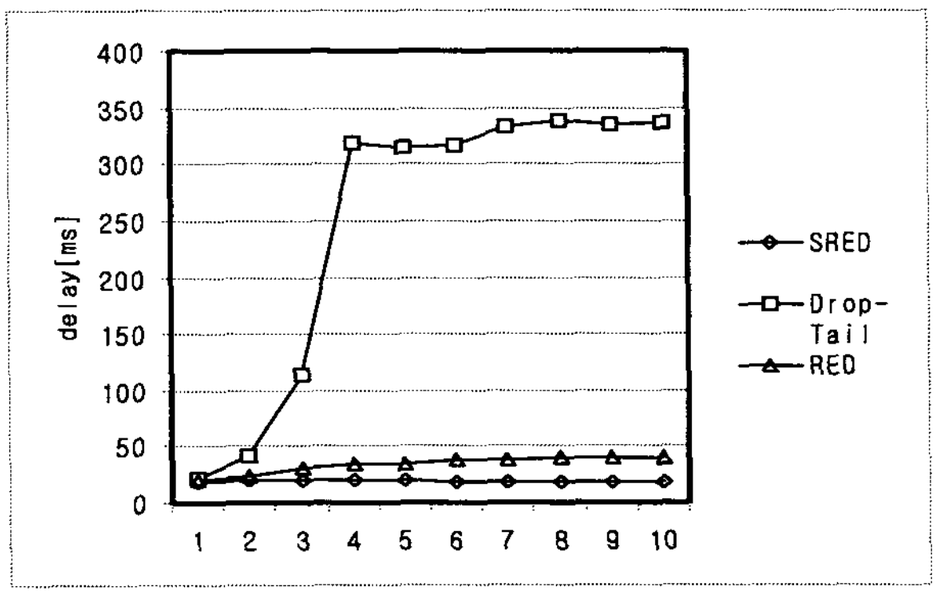


그림 4. delay 비교

〈표 2〉 각 알고리즘의 delay 수치 값

Src num	Proposed RED	Drop-Tail	RED
1	17.7017	22.0843	19.1322
2	20.0271	41.0739	23.8667
3	20.2183	114.551	30.9226
4	19.5534	319.24	34.446
5	19.4295	315.101	33.7746
6	18.9947	317.087	37.9239
7	18.8532	333.702	38.6102
8	18.894	338.515	39.1982
9	18.6715	335.022	39.5056
10	18.6416	337.233	39.7633

<표 2>에서 RED가 Drop-Tail 보다 delay측면에서 우수한 면을 보여준다. 소스 노드의 개수가 1개일 때 RED의 평균 delay는 19.13[ms], Drop-tail의 delay는 22.08[ms] 으로 큰 차이는 없었으나 소스 노드의 개수를 6개로 늘렸을 때 RED의 평균 delay는 37.92[ms], Drop-Tail의 delay는 317.82[ms]로 큰 차이를 보였다. 트래픽의 양이 늘어날수록 RED가 Drop-Tail보다 delay 면에서 좋은 성능을 나타내는 것을 나타냈다.

Proposed-RED와 RED의 delay의 결과를 비교해보면 전체적으로 Proposed-RED의 delay 값이 낮게 측정 되었다. delay면에서 Proposed-RED의 성능이 RED보다 우수하다고 볼 수 있다.

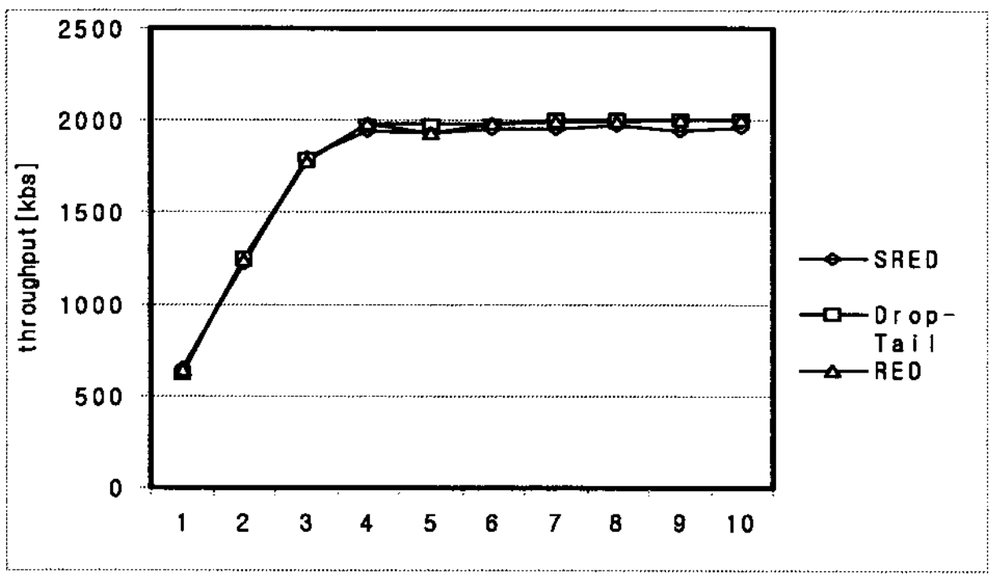


그림 5. throughput 비교

〈표 3〉 각 알고리즘의 throughput 수치 값

Src num	Proposed RED	Drop-Tail	RED
1	646.908	618.725	639.215
2	1217.32	1248.24	1240.04
3	1803.84	1793.38	1781.15
4	1939.56	1977.18	1981.33
5	1935.52	1978.8	1929.48
6	1946.8	1985.08	1982.66
7	1955.33	1999.9	1989.66
8	1971.18	1999.93	1991.87
9	1941.32	1999.9	1999.21
10	1959.88	1999.88	1999.83

[그림 5]에서 Drop-tail, RED, Proposed-RED의 throughput 결과는 거의 차이가 없었다. 즉 throughput이 비슷한 결과에서 delay 값이 가장 작은 Proposed- RED가 RED나 Drop-tail보다 좋은 성능을 내는 것을 알 수 있다.

4. 결론 및 추후연구

네트워크에서 폭주 상황을 극복하기 위해서는 여러 가지 방법이 제안되고 있으나 다양한 네트워크 환경에 따라 그 네트워크에서 필요로 하는 기법이 다르다. 그러므로 어떠한 네트워크의 망 상태에도 적응 가능한 큐 관리가 필요로 한다. RED알고리즘은 각 네트워크 환경에 따라 파라미터 값을 어떻게 설정하는가, 혹은 어떤 알고리즘을 사용했는가에 따라서 결과에 큰 영향을 미친다. 그러므로 RED 알고리즘을 사용 시에는 네트워크의 특성을 분석한 후 그에 맞는 파라미터 값이나 알고리즘을 적용해야 한다. 본 논문에서는 파라미터 값을 고정으로 하고 소스 노드의 수를 증가시켜 폭주 상태를 만들었으며 각 폭주상태에서 세 가지 알고리즘의 성능을 분석해 보았다.

향후 연구 과제로는 동적으로 max_p, w_q, max_{th}값을 변화시켜 전송 지연시간을 최소화 하고 처리 율을 올릴 수 있는 파라미터 분석이 필요하다.

참고문헌

[1] V. Jacobson, "Congestion Avoidance and Control," Computer Communication Review, Vol. 18, No. 4, pp314-329, Aug. 1988.

[2] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithm," RFC 2001, Jan. 1997.

[3] C. Lefelhocz, "Congestion Control for Best-Effort Service: Why We Need a New Paradigm," IEEE Network, January, 1996

[4] Floyd. S and Jacobson, V, "Random Early Detection gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking. Vol1 No.4, p.397-413, Aug. 1993,

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

[6] W. Feng et al., "Technique for Eliminating Packet Loss in Congested TCP/IP Networks," Univ. Michigan CSE-TR-349-97.

[7] Sally Floyd. Ramakrishna Gumma, and Scott Shenker, "Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management, Aug. 2001.