濟州大學校 尖端技術硏究所 論文集, 第12卷, 2號, pp.118-126, 2001. 12
Cheju Nat'l Univ. Res. Inst. Adv. Tech. Jour., Vol.12, No.2, pp.118-126, 2001. 12

예측기반 RED 알고리즘

김 복 심* · 김 동 춘* · 김 대 영* · 안 기 중**

Prediction based RED Algorithm

Bog-Sim Kim*, Dong-Choon Kim*, Dae-Young Kim* and Khi-Jung Ahn**

ABSTRACT

In Internet, the management of traffic congestion involves the policy which drops a packet in advance. In this case, router drops one or more packets to improve the network performance before when its output buffer becomes full. RED(Random Early Detection) is a good example for this technique. A new congestion avoidance algorithm, predictive RED is proposed to estimate the queue length variation and applied to the packet drop probability to control the traffic congestion proactively. And it showed me it can improve the performance by simulation.

Key Words : Traffic congestion, RED, congestion avoidance, performance

I. 서 론

인터넷에서 주로 사용되고 있는 TCP(Transmission Control Protocol)는 송신단에 일정 시간동안 Ack (Acknowledgment) 패킷이 도착하지 않아 타임아웃이 발생하거나, 데이터 패킷의 손실을 의미하는 중복 Ack을 받으면, 네트워크에 혼잡이 일어났다고 가정한다. 그리고 혼잡 윈도우(congestion window)의 크기를 줄이고 전송 속도를 천천히 증가시키는 저속출발(Slow start)과 혼잡회피(Congestion Avoidance)를 실행하여 네트워크 혼잡을 완화시키고 패킷 손실을 줄인다. 즉, 전송 중에 발생하는 패킷 손실의 원인을 네트워크 혼잡에 두고 이를 줄이기 위한 방향으로 개선되어 온 TCP 프로토콜은 통신의 양 종단간 신뢰성 있는 데이터 전송을 보장하며 인터넷에서 가장 보편적으로 쓰인다.

TCP 통신망에서의 혼잡 제어는 크게 두 가지로 나눌 수 있다. 첫째로는 단말 혼잡 제어(end-to-end congestion control)를 들 수 있다. 이는 TCP connection의 양단, 즉 전송 측과 수신 측에서 패킷을 제어하는 것을 의미한다. 두 번째로는 게이트웨이 혼잡 제어를 들 수 있다. 이는 단말 혼잡 제어를 보완하기 위해서 통신망의 게이트웨이 상에 혼잡 제어 기법을 구현하는 방법이다. 가장 대표적인 방법으로는 RED (random early detection)를 들 수 있다. RED는 기존의 Tail-Drop 방식의 게이트웨이의 단점을 보완하여 보다 나은 성능을 나타낸다.[1,2]

RED(Random Early Detection)알고리즘은 망으로부터 TCP 소스로 전해지는 망 상태 신호를 적절히

* 제주대학교 대학원
Graduate School, Cheju Nat'l Univ.
** 제주대학교 통신 · 컴퓨터 공학부, 첨단기술연구소
Faculty of Telecommunication and Computer Engineering, Res. Insti. Adv. Tech., Cheju Nat'l Univ.

조절함으로써 TCP프로토콜이 갖는 혼잡 회피 기능의 효과를 극대화한다. 망으로부터 TCP 소스로 전해지는 묵시적인 망 상태 신호라 함은 패킷의 손실을 의미하는데 RED 알고리즘이 망 내부의 라우터에 구현되면 라우터 버퍼가 넘칠 때(체증)까지 기다리지 않고 사전에 그 조짐을 감지하여 인위적으로 패킷을 폐기해 줌으로써 TCP 프로토콜이 미리 트래픽의 망 내 유입을 조절할 수 있게 한다. 이렇게 함으로서 체증을 사전에 예방하고 결과적으로 망의 수율을 높이고 지연을 줄이게 된다. 그러나 TCP의 윈도우 흐름 제어에 기반하는 혼잡 회피/제어 알고리즘은 TCP연결간 성능향상의 문제점을 가지고 있다.

TCP 체증 회피/제어 알고리즘이 망 내부의 RED 알고리즘과 결합될 경우, 이 불공정성이 어느 정도 해소되는 것으로 연구 결과가 보고되어 있다. 그러나 이는 단일 라우터 내부의 정보에만 전적으로 의존하는 것이므로 그 효과에 있어 일정한 한계가 있다.[3]

본 논문에서는 패킷 손실율을 적게 하고, 지연을 최소화하면서 처리량을 증대시키고 더 나아가서는 Link의 효율을 증가시키기 위한 해결책으로, 패킷 폐기 확률을 계산, 적용하는데 있어 다음에 오는 큐 길이를 예측하여 예측치를 적용시킨 알고리즘을 사용함으로써 처리율의 성능향상을 보였다.

2장에서는 TCP/IP 망에서의 트래픽 제어기술과 기존 RED알고리즘에 대해서 기술하며, 3장에서는 본 논문에서 사용된 개선된 RED 알고리즘 즉, 예측기반 RED알고리즘을 소개한다. 4장에서는 시뮬레이션 및 성능분석으로 개선된 알고리즘을 통해 나타난 모의 실험 결과를 제시하며, 5장에서는 요약 및 결론을 맺는다.

II. TCP/IP 망에서의 트래픽 제어기술

2.1. 종단간 트래픽 제어기술

엔드 시스템들과 전체적으로 연결되어 있는 네트워크들이 좋은 성능을 얻으려면 트랜스포트 프로토콜의 설계와 구현이 아주 중요하게 된다. 트랜스포트 프로토콜은 애플리케이션들과 네트워킹기능 사이에 인터페이스를 제공하며 애플리케이션들은 원하는 QoS를 요구할 수 있다. TCP와 같은 커넥션중심(connection-oriented)의 트랜스포트 프로토콜은 애플리케이션 데이터의 전체 흐름을 논리적인 스트림들로 나누고 이들 스트림에 자원들을 다르게 할당할 수 있다. 마지막으로 데이터단위들의 전송과 재전송을 위한 트랜스포트 프로토콜의 방식들은 네트워크의 혼잡(congestion)에 심각한 영향을 미친다.

2.2. 흐름제어와 오류제어

통신 링크, 네트워크, 인터네트워크의 성능을 결정하는 기본적인 메커니즘이 흐름제어와 에러제어이다.

네트워크나 인터네트워크 전체로 흐름과 에러 제어를 다룰 때에는(예, TCP 수준에서) 분석이 아주 복잡하게 되며 가변적인 전파 지연, 가변적인 데이터 속도, 네트워크에 들어가는 다른 트래픽 원천들이 만든 혼잡 영향, 동적인 라우팅 결정의 영향, 여러 논리 커넥션 사이의 상대적인 우선 순위와 그 밖의 요인들도 고려해야 한다.

2.2.1. 슬라이딩 윈도우 기법

슬라이딩 윈도우 기법은 Fig. 1과 같이 나타낼 수 있다. 이 그림에서 바이트 1에서 11까지 번호를 붙였다. 수신 호스트에 의해 알려지는 윈도우는 광고된 윈도우(Advertised Window: awnd)라 불리고, 바이트 4에서 9까지를 차지하고 있다. 이것은 수신 호스트는 3까지의 모든 바이트에 Ack를 전송했다는 것과 6바이트의 윈도우 크기를 알려 준다는 것을 의미한다. 윈도우의 크기는 Ack 번호와 관계가 있다. 송신 호스트는 이용 가능한 윈도우를 계산하는데, 이것은 즉시 송신할 수 있는 데이터가 얼마인가를 나타낸다.

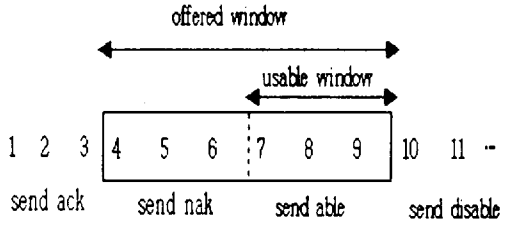

Fig. 1. Sliding Window Mechanism.

시간이 경과하여 수신 호스트가 Ack를 전송하는 것은 슬라이딩 윈도우를 오른쪽으로 이동하게 한다. 윈도우의 양쪽 끝이 이동하므로써 윈도우 크기는 증가하거나 감소된다. 윈도우의 양쪽 끝이 이동하는 것은 닫힘(close), 열림(open), 수축(shrink)으로 표현된다.[4]

2.3. 혼잡제어(Congestion Control)

TCP는 종단간 흐름제어만을 지원하고 간접적인 방법으로 인터넷의 혼잡을 추정한다. 더욱이 네트워크이나 인터넷에서의 지연은 변할 수도 있고 매우 클 수도 있기 때문에 TCP 개체의 혼잡 정보를 믿을 수만은 없다. 다양한 TCP 개체들은 상호간의 협조관계가 없다. 그래서 네트워크의 상태를 유지하기 위해 상호 협력하지 않는다. 단지 이용할 수 있는 자원을 더 차지하려고 하는 best-effort 서비스를 수행할 뿐이다. 그래서 TCP에서의 혼잡 제어는 매우 어렵고 복잡하지만 지금까지의 많은 연구들에 의해 상당한 성과를 보이고 있다. 혼잡제어 알고리즘들은 크게 재전송 타이머 관리와 윈도우 관리로 나누어 볼수 있다.[5]

2.3.1. 재전송 방식

링크제어 프로토콜과 마찬가지로 TCP도 흐름제어 방식을 갖고 있을 뿐만 아니라 에러제어도 하고 있다. 그러나 TCP의 경우 링크제어 프로토콜에 있는 명시적인 부정 애크날리지는 없고 긍정 Ack만 있다. 이 긍정 Ack만 이용하면 주어진 타임아웃기간 내에 Ack가 도착하지 않았을 시 재전송을 한다.

2.3.2. 저속출발(Slow Start)

송신 호스트는 수신 호스트의 광고윈도우 크기에 따라서 다수의 세그먼트를 네크웍에 전송하면서 연결을 시작한다. 그런데 이것은 2대의 호스트가 같은 LAN상에 있으면 문제가 없지만, 송 · 수신 호스트 사이에 라우터와 느린 링크가 존재한다면 문제가 발생한다. 즉, 패킷들은 중간 라우터에서 빨리 처리되기 어렵기 때문에 중간 노드에서 기다려야 한다. 그런데 중간 라우터의 저장 공간이 부족하게 되면 버퍼 오퍼플로우(overflow)가 발생하게 되어 처리율을 상당히 떨어뜨리게 된다. 그래서 slow start라는 알고리즘이 나오게 되었다.[4,6] 이것은 네트워크에 전송되는 패킷의 통신 속도를 다른 쪽 종단으로부터 되돌아온 Ack의 통신 속도와 맞추기 위한 것이다.

2.4. 혼잡 회피(Congestion Avoidance)

데이터는 빠른 LAN에서 느린 WAN으로 전송되거나 다수의 입력 스트림(stream)이 출력 수용량이 적은 라우터에 도달할 때 제대로 전송이 이루어지지 못하고 혼잡이 발생한다. Congestion Avoidance는 패킷 손실을 다루기 위한 방법이다.

Congestion Avoidance와 slow start는 서로 다른 목적을 가진 독립적인 알고리즘이다. 그러나, 혼잡이 발생하면 TCP는 망의 패킷 전송률을 감소시키고 다시 slow start를 수행한다. 그리고 Congestion Avoidance와 slow start는 함께 구현되고 각 연결을 관리하기 위해서는 다음 두 개의 변수를 필요로 한다. 혼잡 윈도우(cwnd)와 slow start 한계값(ssthresh: slow start threshold)이 그것이다.[7]

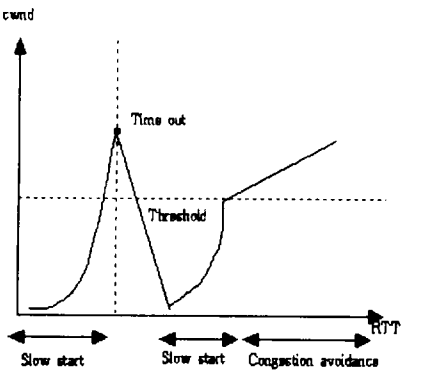

Fig. 2. Cwnd Variation of Slow Start and Congestion Avoidance.

2.5. 빠른 재전송 및 복구 (FRR ; Fast Retransmit and Recovery)

빠른 재전송 및 복구(FRR) 알고리즘은 같은 번호가 붙은 확인 메시지 3개를 연속해서 받으면 그 번호로 시작하는 패킷이 분실되었다고 생각하고 해당 패킷만 즉시 보낸다. 이렇게 되면 분실된 패킷이 빠

르게 복구되고 망이나 다른 패킷에 커다란 영향을 주지 않는다. 이러한 FRR이 성공하게 된 배경에는 망 속도가 빨라지고 에러가 일어날 확률이 작아져 문제가 된 단 하나의 패킷만 재 전송함으로써 복구가 빨라질 수 있다는 생각이다. 하지만 여러 패킷이 분실된 경우라면 이 알고리즘은 별 효과를 보지 못할 것이다.[8]

2.6. RED 알고리즘

RED는 버퍼에서의 평균 큐 길이를 이용하여 체증제어를 수행한다. 일반적으로 패킷이 도착할 때마다 RED 알고리즘은 다음과 같은 절차를 수행한다.

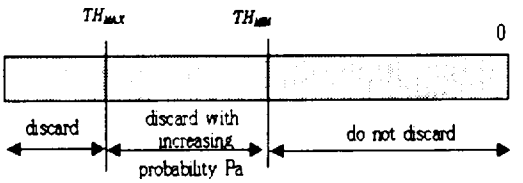

Fig. 3. RED Buffer.

이 알고리즘은 FIFO 출력큐에 새로운 패킷이 도착할 때마다 두 기능을 수행하는데 첫 번째 절차는 평균 큐길이 avg는 EWMA(Exponentioal Weighted Moving Average)를 통한 low-pass filter를 사용하여 다음과 같이 계산한다.

$$avg \leftarrow (1-w_q)avg + w_q q \quad (1)$$

(avg : average queue size
w_q : queue weight
q : current queue size)

이 평균 큐 길이를 두 한계치 Fig. 3과 같이 비교한다. avg가 하한치 TH_{min}보다 작으면 체증이 최소이거나 없는 것으로 가정하여 이 패킷을 큐에 넣는다. avg가 상한치 TH_{max}보다 크면 체증이 심각하다고 가정하며 이 패킷을 버린다. avg의 정확한 값에 따라 avg를 상한선에 이르게 증가시키는 확률 P_a를 계산한다. 큐가 이 영역에 있으면 이 패킷을 확률 P_a로 버리며 확률(1- P_a)로 큐에 들어가게 한다.

기본적으로 알고리즘의 첫 부분(큐 크기 계산)은 허용할 수 있는 폭주정도를 결정하며 알고리즘의 두 번째 부분(패킷 폐기결정)은 현 수준의 체증이 있을 때 폐기되는 패킷들의 빈도를 결정한다.

Fig. 3을 이용하여 이제 RED 알고리즘을 보다 상세히 살펴보면 다음과 같다.

2.6.1. 평균 큐 크기의 계산

앞 큐 길이들의 지수가중(exponentially weighted) 평균을 이용하여 식(1)에서와 같이 평균 큐 길이를 계산한다.

2.6.2. 패킷 폐기의 결정

avg가 TH_{min}보다 작으면 들어오는 패킷을 큐에 넣으며 avg가 TH_{max}보다 크면 들어오는 패킷을 자동적으로 폐기한다. 임계영역은 avg의 값이 두 한계치 사이에 있을 때이다. 이 영역에서는 RED가 들어오는 한 패킷에 두 요인에 의존하는 폐기확률을 할당한다.

- ⊙ avg가 TH_{max}에 가까워질수록 폐기확률이 더 높아진다.
- ⊙ avg가 임계영역에 있으면 얼마나 많이 연속적으로 패킷들을 폐기 않도록 하는가 하는 count를 유지하는데 이 count값이 클수록 폐기할 확률이 더 높게 된다.

$$F = \frac{avg - TH_{min}}{TH_{max} - TH_{min}} \quad (2)$$

$$P_b = F \times P_{max} \qquad 0 \le F \le 1 \quad (3)$$

$$P_a = \frac{F \times P_{max}}{1 - count \times F \times P_{max}} = \frac{1}{\frac{1}{F \times P_{max}} - count} = \frac{P_b}{1 - count \times P_b} = \frac{1}{\frac{1}{P_b} - count} \quad (4)$$

RED는 종단간 체증 제어 알고리즘과 연계하여 평균 큐 사이즈를 낮게 유지하려는 알고리즘으로서, 가끔 발생하는 burst traffic은 허용한다. 즉, 평균 큐 사이즈를 관찰하는 도중에 체증 발생 조짐이 감지되면 인위적으로 패킷을 폐기하여 TCP의 혼잡 회피/제어

기능이 동작하게 된다.

III. 예측 기반 RED 알고리즘

3.1. 큐 가중치와 평균 큐 길이

평균 큐 길이(Q_{avg})를 구할 때에 실제 큐 길이가 Q_{avg}가 변하는데 미치는 영향을 나타내는 값이 가중치(W_q)이다. W_q가 작으면 작을수록 현재의 큐의 실제 길이가 Q_{avg}가 변하는데 미치는 영향이 작기 때문에 일시적인 트래픽의 증가로 인한 오류를 피할 수 있다.

3.2. 큐 변화량과 평균 큐 길이

기존 RED 알고리즘이 패킷을 폐기하는 방법은 Q_{Avg}를 식1과 식4를 이용하여 확률 P_a를 적용하여 패킷을 처리한다.

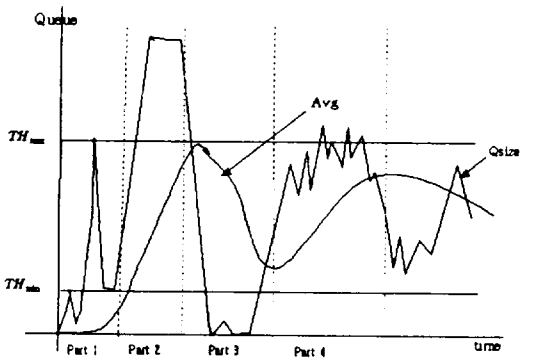

Fig. 4. Predictive RED.

RED를 사용한 환경에서는 위 Fig. 12에서 같이 살펴 볼 수 있다.

우선 Fig. 4에서 part1. part2. …에서 문제점들을 발견할 수 있는데 아래와 같다.

- part 1에서 실제 큐 길이는 최대 Q길이에 도달하고 있어 패킷 폐기를 해서 제어가 필요하나 Q_{Avg} 값보다 작은 상태에서 폐기확률을 적용할 수가 없다.
- part 2는 체증이 발생하여 패킷이 폐기되고 있으나 Q_{Avg}값이 TH_{max} 밑에 있어 체증을 인식하지 못한 상태이다.
- part 3에서는 전역동기화(Global Synchronization)가 발생하여 현재 Q가 비어 있음에도 불구하고 Q_{Avg}가 TH_{max} 가까이에 있어 도착하는 패킷에 대해 확률 P_a가 적용되어 폐기 처리한다.
- 순간적인 버스트한 데이터에 대해 민감하게 반응하는 것을 억제해야 한다. 체증이 발생시킬 수 있는 적당한 버스트한 데이터에 대해서는 폐기확률을 적용하여 패킷을 폐기하여야 체증을 막을 수 있다.

이를 보완하기 위하여 본 논문에서는 큐 변화량을 적용한 가중평균 길이와 큐차이를 적용하여 폐기확률을 계산하고자 한다.

3.3. 예측 큐 가중치와 평균

- 구간1. 2의 급격하게 변하는 큐의 변화를 Q_{avg}가 인식할 수 있으려면 W_q를 크게 해야 하나 이 방법은 전체 효율에 지장을 초래할 수 있으므로 새로운 변수를 식1과 유사하게 도입하여 폐기확률에 적용하여 체증을 방지할 수 있도록 하였다.

$$D_{Avg} = (1 - W_d)D_{Avg} + W_d(Q_{size} - Q_{Avg}) \quad (5)$$

(D_{avg} : Q_{avg}와 Q의 가중평균값
W_q : 가중치)

$$P_a = \frac{F \times P_{max}}{1 - count \times F \times P_{max}} \quad (6)$$

(단, $F = \frac{(Avg + D_{Avg}) - TH_{min}}{TH_{max} - TH_{min}}$)

- 구간3은 현재의 Q가 TH_{min} 보다 작을때는 들어오는 패킷을 폐기하지 않아야 한다. Q_{Avg}가 TH_{max}에 가깝게 있어 높은 폐기 확률을 적용하고 있다. 이를 해결하기 위해 식 7과 같은 방법을 적용하여 패킷 폐기 여부를 결정한다. 여기서는 식 6의 P_a도 역시 사용되어진다.

$$D_{avg} = -Q_{Avg} \tag{7}$$

즉, $$Q_{Avg} + D_{Avg} = Q_{Avg} + (Q - Q_{Avg}) \tag{8}$$

- 체증을 야기할 만큼 버스트한 데이터가 지속되면 (순간적인 버스트가 아님) 이를 인식하여 식 9를 이용하여 폐기확률을 높여 적용한다.

$$D_{Avg} = (1 - W_d)D_{Avg} + W_d(Q - Q_{avg}) \tag{9}$$

RED는 새로운 패킷이 Router에 들어왔을 때마다 평균 큐 길이를 계산하게 된다. 현재 큐 길이에 대하여 가중치를 두어 평균 큐 길이가 계산되었다.[1] 여기에는 다음에 오는 패킷에 대해서 한 개의 버스트한 패킷에 대해서는 폐기확률을 적용하지 않지만 어느 정도 연속적인 버스트한 패킷에 대해서는 체증을 야기할 가능성이 있으므로 높은 폐기 확률을 적용시켜야 한다.

또한 Router에서 패킷을 폐기하게되면 큐 길이에 영향을 주게 되는데 지금 폐기했다 하더라도 당장 큐의 길이가 줄어들지 않고, 어느 정도 시간이 지나야 큐 길이가 줄어들게 된다. 따라서 예측 평균 큐 길이를 적용하면 이러한 단점을 보완하여 폐기확률이 평균 큐 길이에 적응할 수 있도록 하고자 한다.

본 논문의 알고리즘은 예측기반 RED(predictive RED : PRED)라 하였으며, 제안되는 알고리즘은 Fig. 5와 같다.

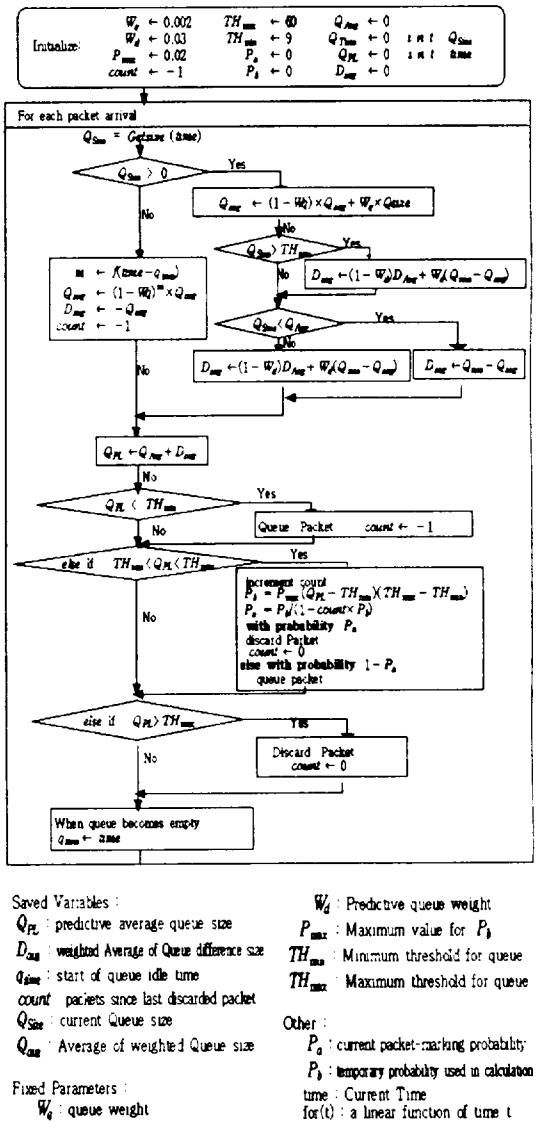

Fig. 5. Predict RED Algorithm.

Ⅳ. 시뮬레이션 및 성능분석 결과

4.1. 시뮬레이션 모델

본 논문에서 사용한 Network 환경은 Fig. 6과 같이 구성하였으며 다음과 같은 전제조건을 두었으며, 일반적인 사항은 고려하지 않았다.

① 모든 링크의 대역폭은 80Mbps

② 링크와 gateway 사이의 delay time은 1. 2. 3. 5. 4. 4 … msec으로 반복적용

③ Sink 노드와 Router 사이의 delay time : 2msec

시뮬레이션은 링크 수를 늘여가면서 Gateway 형태를 RED, PRED의 방식을 비교하였으며, 시뮬레이션 시간은 10초, Gateway 큐 길이는 100패킷, 1패킷은 1000byte로 하였고 Ack packet은 40byte로 하였다.

체증인지 방법은 gateway가 패킷을 폐기하였다는 것을 링크 송신노드가 10msec 지나서 알 수 있도록 하였다.

논문에서 사용된 파라미터 중 예측 큐 평균 길이를 구하기 위하여 사용된 예측가중치는 평균 큐 길이를

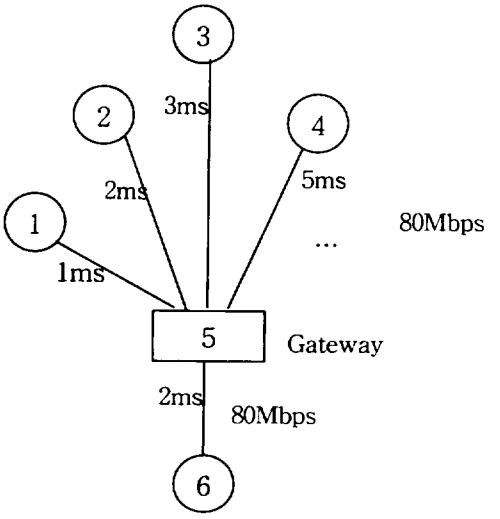

Fig. 6. Simulation network.

큐 변화량에 적응하기 위하여 0과 1사이의 값으로 정하였으며 시뮬레이션을 통하여 적절한 값으로 0.03를 정하였다. 이 값은 일반적인 것은 아니다.

시뮬레이션 파라미터는 RED인 경우 P_{max}는 0.02. Wq는 0.002. TH_{max}sms 60. TH_{min}는 9. 패킷 크기는 1000. 윈도우 크기는 1024이고 PRED인 경우에는 RED의 파라미터를 그대로 사용하면서 Wd를 0.03으로 사용하였다.

수정된 RED 알고리즘과 기존 RED 알고리즘의 성능을 비교하기 위해 Microsoft Visual C++ 6.0을 사용하여 시뮬레이션을 실행하였다.

4.2. 성능측정 방법

시뮬레이션 성능측정은 링크의 효율성, 패킷 손실율, 평균 큐 길이와 예측 평균 큐 길이에서 폐기 확률을 이용하여 성능을 측정하였으며, 계산방법은 다음과 같다.

• Link utilization

$$Link\ utilization = \frac{\text{전송한 } total\ packet}{total \text{ 시뮬레이션 시간}} \quad (10)$$

• Packet Loss Rate

$$Packet\ Loss\ Rate = \sum_{(i=1)}^{n} \frac{(drop\ packet\ count)}{n} \quad (11)$$

drop packet count = congestion packet + marked packet with packet Pa

n : simulation 횟수

4.3. 폐기확률의 비교

- 평균 큐 길이의 이용한 폐기 확률

$$Q_{avg} \leftarrow (1 - Wq) \times Q_{avg} + W_q \times Qsize \quad (12)$$

$$P_b = P_{max}(Qavg - TH_{min})(TH_{max} - TH_{min})$$
$$P_a = P_b/(1 - count \times P_b) \quad (13)$$

- 예측 평균 큐 길이의 이용한 폐기 확률

$$D_{avg} \leftarrow (1 - Wd) \times Davg + (Qsize - Q_{Avg}) \times Wd \quad (14)$$

$$Q_{PL} \leftarrow Q_{avg} + D_{avg} \quad (15)$$

$$P_b = P_{max}(Q_{PL} - TH_{min})(TH_{max} - TH_{min})$$
$$P_a = P_b/(1 - count \times P_b) \quad (16)$$

4.3. 시뮬레이션 결과

시뮬레이션 결과는 Fig. 7에서 Fig. 10에 걸쳐 나타내었다. 그림에서 시뮬레이션은 커넥션 5개부터 20를 갖고 나타내었고 Fig. 7 와 Fig. 8은 커넥션이 5개 일 때의 RED와 PRED 결과를 표현하였다.

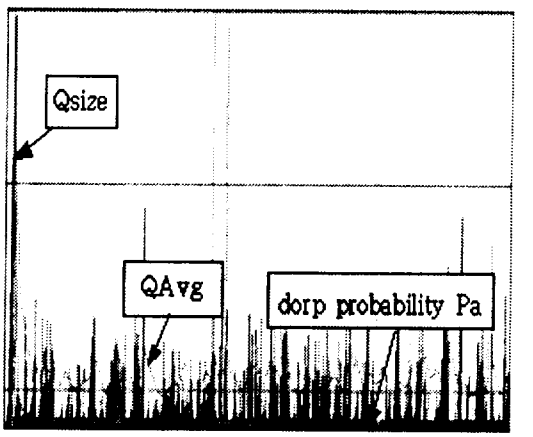

Fig. 7. Performance of RED.

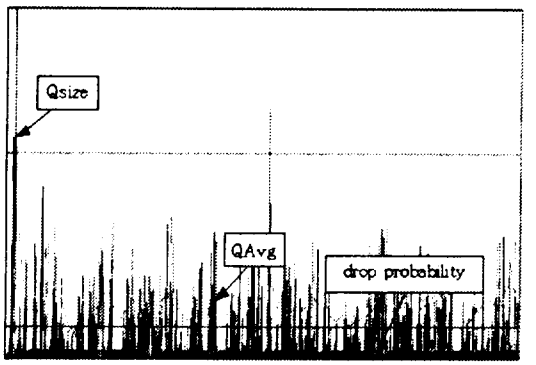

Fig. 8. Performance of PRED.

패킷 손실율도 random하게 처리하는 특성으로 인하여 측정된 값을 사용하였으며, 시뮬레이션의 정확도를 위하여 시뮬레이션 시간을 10sec로 설정하였다.

전송률은 RED는 49.128 PRED 55.627 이며, 손실률(혼잡)은 RED는 232이고, PRED 200 이다. 결과적으로 개선된 알고리즘이 훨씬 성능이 좋은 것으로 나타나고 있다.

Table 1. Simulation Result

conection	Translation Rate		Ross Rate	
	RED	PRED	RED	PRED
5	49.128	55.627	232	200
6	54.569	61.082	363	333
7	57.069	66.803	446	287
8	64.123	70.428	446	391
9	68.044	73.987	494	485
10	69.876	78.100	605	553
11	72.185	82.658	697	609
12	76.904	85.462	687	682
13	80.961	85.205	711	681
14	84.010	87.664	895	759
15	86.387	89.947	943	868
16	87.799	91.119	995	940
17	90.097	92.002	1.151	1.066
18	91.742	93.340	1.180	1.162
19	93.264	94.867	1.226	1.165
20	94.240	95.759	1.370	1.305

위의 표는 커넥션 5개부터 20개까지의 시뮬레이션 상황을 표로 나타내었다. 전송률에 있어서는 PRED가 RED보다 훨씬 전송률이 높은 차이를 볼 수 있으며 손실률에서는 PRED가 RED보다 손실이 적은 것을 볼 수 있다.

아래 Fig. 9 Fig. 10은 위 Table. 1에서 전송률과 손실률 기준으로 RED와 PRED의 상관관계에 대해서 살펴보았다.

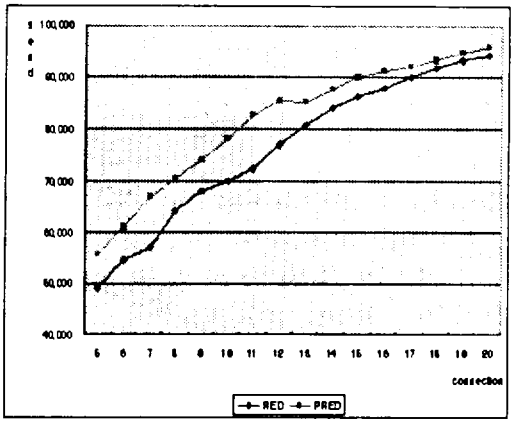

Fig. 9. Comparision of the Performance.

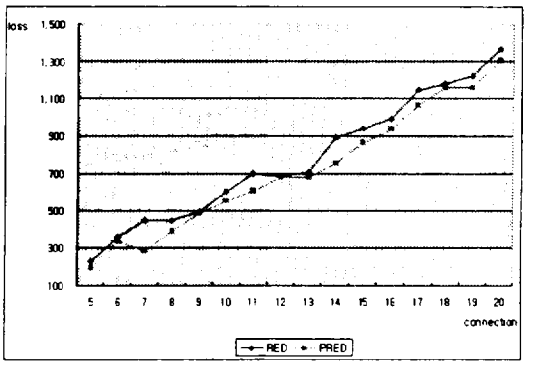

Fig. 10. The number of droped Packets.

V. 결 론

본 논문에서는 RED 알고리즘의 전송률과 손실율 문제를 해결하기 위해 기존 RED의 큐평균과 예상 큐평균을 구하여 서로 더함으로써 큐길이를 예상할 수 있도록 하였다.

그 결과 전송량과 손실률에 있어서 개선됨을 살펴

보았다.

앞으로 연구과제는 RED가 효율적이면서도 간단하다는 장점이 있다고 해도, 개선의 여지가 있다는 것을 본 연구를 통해서 입증해 보였다. 그리고, 측정 파라미터를 가변적으로 변화를 시키는 방안과, Q_{PL} 설정에 대해서도 보다 더 연구가 필요하고 다양한 방식으로 접근할 필요가 있다.

참고문헌

1) S. Floyd and V. Jacobson. 1993. "Random Early Detection Gateways for Congestion Avoidance". IEEE/ACM Transactns on Networking

2) 유영석, 홍석원. 1997. 11. "체증 제어를 위한 Random Early Detection(RED) 알고리즘의 파라미터 분석". 통신학회 추계발표 논문집 pp.41-44

3) 이지형, 이상연, 정충교. 1999. 7. "RED알고리즘의 공정성 개선 방안". 통신학회 하계종합학술발표회 논문집 pp.970-973

4) W.Stevens. 1997. 1. "TCP Slow start, Congestion Avoidance, Fast Retransmit and Fast Recovery algorithm". Internet RFC 2001

5) 강문칠 편역. 2001. 1."최신 네트워크 설계". 세명서관

6) V.Jacobson, August 1988. "Congestion Avoidance and Control". Proccding of SIGCoMM '88. pp.314-329

7) 전인재. 1999.12. "네트웍의 혼잡 정보를 이용한 TCP 연결간의 공정성 개선 방안". 강원대학교 논문집

8) 장혁수, 주우석. 2000. 12. "인터넷 통신 프로토콜 및 응용" 도서출판