Journal of Digital Contents Society Vol. 8 No. 4 Dec. 2007(pp. 573-578)

버스트 토큰을 이용한 리키 버킷에 관한 연구

김동철*

요 약

리키버킷 사용에 관해서 많은 연구가 있었다. 고속네트워크에서 제어기능을 다루는데 리키버킷 방식의 사용에 관해서 많은 관심을 가지고 있다. 본 논문에서 제안한 버스트 토큰 fill 방식은 최고율을 유지하고 버스트 크기도 조절하며 평균율에 그다지 큰 변화를 주지 않으면서 좋은 효과를 기대 할 수 있는 방법을 제안 하였다. 이러한 방법을 사용함으로써 어떤 효과가 있을지를 보여 주었다. 그러나 이방법의 사용에 있어서 on-off 방식과 같은 버스트 데이터의 특별한 형태에 보다 효율적이며 모든 데이터에 적용하는 데는 더욱 세심한 주의가 필요하다.

A Study of Leaky Bucket using Burst Tokens

Dong-chul Kim*

Abstract

The Leaky Bucket mechanism has been studied in many areas. There has been much interest in flow control in High Speed Networks using the Leaky Bucket. In this paper, We propose the Burst Tokens Fill method that performs well in Leaky Bucket to control the average rate, peak rate, and burst size. This scheme show the improvement of performance. However, This method does not apply all area of applications. Especially application of on-off shape is appropriate and the other area must need to care for appling this scheme.

Keywords : Leaky Bucket, Flow control

1. 서론

최근 VLSI, 광통신의 발전과 더불어 고속네트워크의 발달을 이루어 왔다. 초고속네트워크 ATM을 기반으로 하는 다양한 서비스가 요구되어졌다. 전통적인 패킷 스위칭 네트워크에서는 패킷의 흐름을 제어하기 위해서 윈도우방식을 기본으로 하는 방식을 사용하였으나 고속네트워크 통신에서는 이러한 방식만으로는 여러 가지 어려움이 있다. 그러므로 좀 더 간단하며 효과적으로 방법을 이용하는 것이 제안되었다. 입력 율을 조절함을 통해서 좋은 성과를 이루는 방식이다. 이러한 방식의 대표적인 방식중의 하나가 리키버킷을 사용하는 방식이다. 어느 정도의 입력 데이터의 버스트를 허락하면서 동시에 제한된 입력율 이하로 패킷을 입력하는 방식이다. 네트워크로 패킷을 보낼 수 있는 속도를 규제해 주는 제어는 QoS 구조의 중요한 요소 중에 하나이다. 패킷 플로우에서 패킷율 제어하는 중요한 제어기준에서 패킷흐름의 평균율, 최고율 그리고 버스트 크기가 있다.

리키버킷 관한 많은 연구가 이루어져 왔다. 리키 버킷의 연구 중에 큐잉 모델을 통한 분석이 사용되었다. 특히 MM1과 MD1을 사용하여 패킷 손실율과 쓰루풋을 분석하였다. 이들의 논문에서 서비스율을 일정하게 하는 것이 더욱더 좋은 성능이 있었음을 보였다[1][2]. Venket는 출력 흐름의 버스트네스를 토큰 버킷 사이즈 의 함수로 나타내었고 특히 버스트네스는 토큰 버킷 사이즈와 함께 증가함을 해석적인 방식을 통하여 보였다[3]. Dirk은 리키버킷을 사용하여 공정성과 스루풋의 트래이-오프 문제를 해결하였다[4]. Philippe는 주기적인 데이터 패턴은 최근까지 여러 자료에서 보여준 on-off 혹은 트라이

※ 제일저자(First Author) : 김동철
접수일자:2007년10월10일, 심사완료:2007년10월25일
* 평택대학교 컴퓨터학과 교수
kim@ptuniv.ac.kr

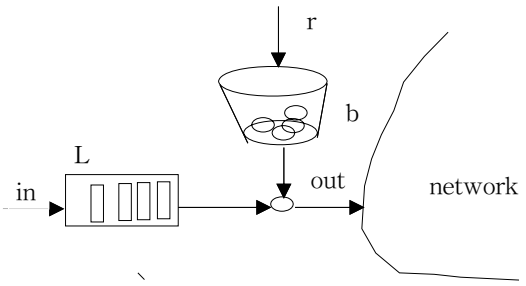
상태 패턴에서 보다도 더 나쁜 손실율이 발생함을 시뮬레이션을 통하여 보였다[5]. Gan은 멀티미디어 트래픽을 갖는 ATM 네트워크에서 수퍼리키버킷을 사용하여 각각의 QoS의 요구에 따라 3개의 다른 데이터 타입을 다룰 수 있는 방법을 제시하였다[6]. Mohamad는 다이나믹 리키버킷 알고리즘을 사용하여 고성능 지능 트래픽 제어기 퍼지 함수를 사용하여 구현 하였다. 제어는 실제 평균 셀율의 실시간 평가를 통하여 취해졌다. 성능의 향상을 이루었다[7]. Michell은 다양하게 서로 관계하는 도착 셀과 결정된 토큰 도착 분포아래서 출력과정의 2차 통계를 적용하여 리키버킷 매카니즘의 행동을 연구하였다[8]. Choi는 입력데이터 버퍼에 스레쉬홀드를 갖는 리키버킷을 사용하였고, MMPP(Markov Modulated Poisson process)를 버스트 입력 트래픽으로 사용하였다. 상당히 성능에 향상이 있었다[9].

본 논문에서는 리키버킷시스템에서 버스트 데이터 크기와 입력버퍼와 토큰버퍼 크기의 관계를 연구 하였다. 특히 버스트 크기의 로컬리티 성질이 있을 경우 버스트 데이터 안에 패킷의 지연시간과 드롭 율을 줄이는 버스트 토큰 fill 방법을 제시하였다.

2. 리키버킷의 구조

2.1 리키버킷 모델

리키버킷기법은 (그림 1)처럼 구성될 수 있다. 리키 버킷은 b 개의 토큰을 담을 수 있는 버킷이다. 토큰은 초당 r 개씩 토큰 버킷으로 들어간다. 만일 버킷 안에 토큰이 꽉 차 있는 경우 토큰은 넘쳐흘러서 버려진다. 만약 b 개 보다 적은 수의 토큰이 채워져 있다면 새로 생성된 토큰이 버킷 안으로 추가 된다. 노드에서 받아들인 패킷은 이미 전송되는 패킷이 없으면 패킷은 네트워크로 들어갈 준비가 된다. 그러나 반드시 패킷에서 토큰을 1개 받아야 네트워크 안으로 전송될 수 있다. 만약 토큰 버킷에 토큰이 비어 있으면 패킷은 토큰이 새로 생성 될 때 까지 기다려야 한다. 그러므로 토큰이 기다림을 위한 대기 영역이 필요하다. (그림 1)에서 각각의 부분에 대한 내용이 보여 준다.



(그림 1) 리키버킷

(그림 1)의 리키 버킷에서 패킷이 토큰 대기 영역인 입력 버퍼에서 기다리고 버퍼의 크기를 L로 한다. 입력패킷의 입력율을 in (packet per sec) 로 들어온다고 하자. 리키 버킷의 크기는 토큰이 최대 b 개 담을 수 있고 토큰의 생성율은 r (tokens per sec) 이라고 하자. 출력율 또한 out (packet per sec)로 하자.

네트워크에서 입력 패킷의 흐름을 제어 하는데 시간에 따라 차이가 있다. 평균율은 패킷의 흐름에서 긴 시간동안 패킷들의 평균 양을 측정하는 반면에 최고율은 상대적으로 짧은 시간동안 전송할 수 있는 최대 패킷 수를 측정한다. 버스트 크기는 최고율일 때의 짧은 시간보다도 아주 적게 즉 시간의 간격을 0으로 접근 시키면서 네트워크를 통하여 순간적으로 전송 할 수 있는 패킷의 수로 제한 할 수 있다. 토큰을 생성하는 방식은 여러 가지가 있다. 이 모델에서는 그 중에서 일정한 같은 시간에 토큰을 생성하는 방식을 고려하였다.

이 논문에서는 버스트 패킷들이 있을 경우 리키버킷 모델의 문제점에 대해서 다루기로 한다. 리키 버킷의 모델의 경우 토큰버킷의 크기의 패킷이 들어 올 경우 리키 버킷의 토큰이 모두 소비되며 순식간에 출력을 통하여 네트워크를 통하여 빠져나간다. 이 경우 입력된 버스트 데이터가 토큰버킷의 양보다 큰 경우 나머지 패킷은 입력의 대기 버퍼 안에서 기다려야 한다. 이때 대기 버퍼의 공간이 충분히 있다면 기다리면 되지만 그렇지 않을 경우의 패킷들은 기다릴 장소가 없기 때문에 손실된다. 이렇게 손실된 패킷은 여러 가지 방법을 통하여 재전송 된다면 상당한 오버헤드를 생성하게 된다. 그러므로 토큰 버퍼의 크기와 대기버퍼의 공간과 버스트 입력의 크

기의 관계는 상당히 성능 면에서 중요하다.

많은 응용 프로그램에서 생성되는 버스트 패킷들에서 locality 개념이 적용된다. 여기서 버스트 된 패킷들의 locality 개념이 적용되어 클러스트 방식으로 왔을 경우 오히려 전체적인 평균 패킷율은 낮더라도 최고율은 상당히 높을 것이다. 이 경우 리키 버킷 방식은 다음과 같은 문제가 있다.

리키 버킷 안에 패킷이 몇 개 있다면 네트워크로 들어가는 패킷의 양이 리키 버킷의 토큰 생성 율로 네트워크로 들어간다. 그러므로 만약 몇 개의 버스트 데이터가 연속해서 올 때 대부분은 손실되므로 성능 면에서 저조한 결과를 초래할 것이다. 만약 리키 버킷에 토큰이 꽉 차 있을 경우에도 토큰 수의 패킷들은 즉시 네트워크로 입력된다고 할지라도 뒤 따라 오는 버스트 데이터는 입력대기 버퍼의 공간의 크기에 따라 손실 율이 결정될 것이다. 또한 패킷에 있는 데이터도 입력 큐 안에서 상당한 지연시간을 가져야 할 것이다. 이러한 경우는 on-off 형태의 데이터가 들어오는 경우 평균 패킷 입력은 낮으나 버스트 율이 높을 경우 문제가 생긴다.

만약 버스트 데이터가 올 경우에는 오직 상당히 네트워크의 혼잡이 증가한 경우라면 즉 평균 패킷 율이 이미 상당히 네트워크의 최대 밴드폭에 접근한 경우에는 자연히 입력되는 패킷의 양을 줄여야 한다. 즉 상당히 패킷의 손실이 있다고 하더라도 패킷을 네트워크로 주입되는 것을 제안해야 한다. 그러므로 리키 버킷을 사용할 여러 경우를 구별하여 상태에 따라 리키버킷을 사용 할 수 있는 알고리즘이 필요하고 이러한 방법을 적용하면 성능을 상당히 높일 수 있다.

그러므로 on-off 방식의 버스트 데이터가 들어 올 경우 몇 개의 버스트 데이터에 대비하여 버스트 토큰들을 준비한다면 상당히 좋은 반응을 얻을 것이다. 여기서 우리는 토큰이 일정하게 채워지는 방식에서 버스트 토큰이라는 새로운 방식을 제안 하고 이러한 방식의 이름을 버스트 토큰 fill 방식으로 한다. 이러한 방법이 얼마나 효과 적인가를 다음 절에서 보인다.

앞으로 설명을 쉽게 하기위해서 초당 패킷의 생성율을 in, out 그리고 r로 하고 bs는 입력되는 버스트 크기를 나타낸다. b는 리키 버킷 안에 들어 갈수 있는 토큰의 크기를 나타내었고, L은 패킷의 대기 영역 즉 입력버퍼의 크기를 나타내었다. 우선 버스트 패킷들은 짧은 시간에 노드 안으로 들어오므로 패킷 입력율이 순간적으로 상당히 높아진다. 여기서 우리는 토큰이 있을 경우 출력율 out은 입력율 in 보다 상당히 크다고 가정하여야 한다. 그러므로 버스트 패킷들이 순간적으로 네트워크 안으로 주입된다고 생각한다. 이 논문에서 입력율과 출력율의 관계는 in << out 의 관계를 모든 경우의 예에서 유지된다. 여기서 << 기호는 상당히 큰 차이가 있음을 나타낸다.

입력율 in 이 토큰 생성율 r 보다 적으면 네트워크에서 혼잡에 관해서 r로 제어하는 기능이 없다. 그러므로 in > r 의 관계를 유지하는 경우를 다룬다. 본 논문에서 토큰 생성율 r에 의해서 혼잡율을 줄이는 경우를 다룬다. 토큰의 생성율은 일정한 시간에 따라 토큰을 생성한다고 가정한다. 그러므로 이 논문에서 다루는 경우는 단지 r < in << out 인 경우만을 고려한다. 여기서 r과 in은 평균 패킷율을 다룬 것이고 in 의 경우 버스트 데이터가 들어 올 경우 순간 패킷 입력율은 in 보다 상당히 크다. 이 경우도 토큰 생성율 r 보다는 순간적으로 상당히 큰 버스트 토큰 생성율이 될 것이다. 리키버킷의 크기가 b 이므로 출력되는 버스트 크기는 b 보다 클 수 는 없다. 그러므로 만약 버스트 토큰 방식을 사용하면 출력 버스트 크기를 b로 하면서 입력의 큰 버스트를 작은 버스트들로 잘라서 나누는 역할도 할 수 있다. 본 논문에서 새로 제안하는 토큰 버스트 fill 방식을 제안한다.

2.2 버스트 토큰 fill 모델

버스트 토큰 fill 방식은 기존에 일정한 시간에 따라 토큰을 생성하는 것에서 벗어나 필요할 때는 순간적으로 리키 버킷에 토큰을 모두 채우는 방식이다. 만약 현재 리키버킷이 비어 있을 경우 fill 함수를 1로 세트하면 순간적으로 버킷을 1번 채우라는 것을 의미하며 fill 함수가 2이면 1번 채운 후 출력한 후 즉시 한 번 더 토큰 버킷을 채우라는 의미이다. 그러므로 즉시 버킷을 채우므로 이름을 버스트 토큰 fill 이라고 하였다.

본 논문에서는 평균 패킷 도착율은 낮은데 입력 버스트 성향이 강한 경우이다. 그러나 어떻게 on-off형태의 데이터인지 아니면 일반적인 혼잡

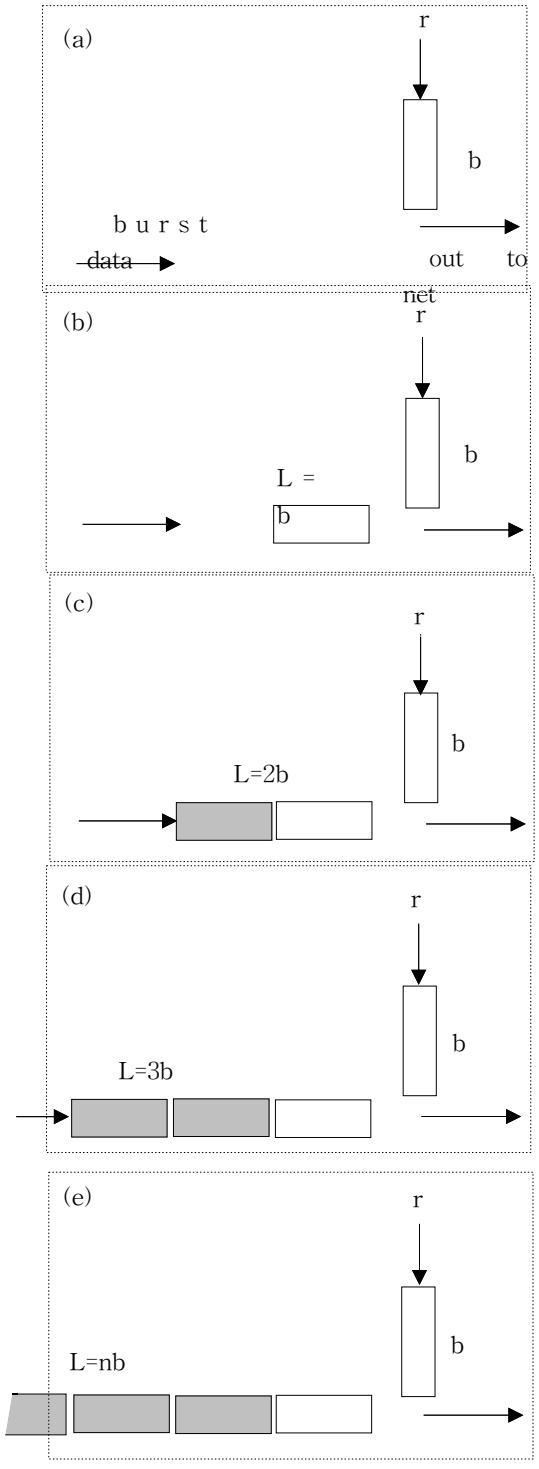
한 경우를 찾는 방법 보다는 on-off 경우 어떤 효과가 있는지를 나타내는 데 초점을 맞추었다. 다음 (그림 2)은 입력 버스트 크기와 토큰 버퍼의 크기와의 관계를 나타내며 이러한 관계에 있을 경우 패킷의 지연과 손실을 보여 준다.

(그림 2)의 (a)에서는 버스트 데이터 (bd : burst data)가 들어오는데 토큰 대기영역의 입력 버퍼가 없는 경우이다. 다음 3가지 경우를 생각해 보자. 토큰이 버킷 안에 있는 최대 토큰의 수 b와 bd 안에 패킷의 수가 b 보다 적을 경우와 같을 경우에는 각각의 버스트 패킷들에 토큰이 주어지므로 bd 크기의 패킷들이 네트워크 안으로 바로 주입된다. 그러나 b 만한 크기의 버스트 입력 패킷들이 또 도착 할 경우에는 다음과 같은 경우가 생긴다. 즉 다음 토큰이 생성되기 전에 도착한 모든 버스트 패킷은 모두가 버려진다. 또한 1개의 토큰이 생성되었다고 해도 1개의 패킷을 네트워크로 보내고 나머지 모든 패킷은 버려진다.

이런 경우에 버스트 토큰 fill 방식을 사용하면 다음과 같다. 즉 버스트 토큰 fill 방식에 유일한 함수는 fill 함수 이다. 즉 fill을 1이라고 하면 즉시 b 개의 토큰을 토큰 버퍼에 모두 fill 하는 것이다. 이것은 locality의 환경에 따라 결정되는 파라메터와 유사하다. 여기서 이상적으로 적절한 값을 얻었다고 가정하면 불필요하게 패킷을 버리는 경우를 줄일 수 있다. 단 여기서 평균 패킷 도착율을 고려하여 결정한다고 가정 할 경우 이다.

(그림 2)의 (b)에서는 버스트 데이터(bd)가 들어오는데 토큰 대기 영역의 입력버퍼가 크기 L=b 인 경우 이다. 만약 bd의 크기가 b 이면 다음 상황이 벌어진다. 만약 토큰이 b개가 버킷 안에 있으면 처음의 버스트 버킷은 단숨에 네트워크로 들어가고 뒤에 오는 버스트패킷들은 토큰 대기 영역에 들어간다.

버스트 안에 있는 패킷의 길이를 pklength bits라 하면 버스트의 크기는 pklength * b 이다. 따라서 이 패킷들을 다 네트워크로 보내는 데는 (pklength * b)/r sec 가 걸린다. 또한 입력 버스트의 길이가 bs(burst size) = 3b 이면 b 크기의 버스트는 손실이 된다.



(그림 2) 리키버킷에 버스트 입력과 입력 버퍼

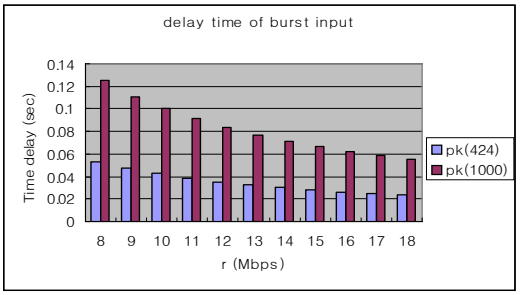
그러므로 버스트 토큰 방식을 사용하여 fill 함수를 조절하면 패킷 손실 율을 감소시킴을 알 수 있다. 즉 bs = 3이고 fill =1 이면 L=b 이므로 네트워크로 버스트 데이터를 내보내는데 (pklength *b)/r sec 가 걸린다.

(그림 2)의 (c)와 (d)의 경우를 생각해 보자. 같은 버스트 들이 연속적으로 온다고 생각 할 경우 (b)에서와 마찬가지로 생각 할 수 있다. 여기서 우리는 다음과 같은 일반화 된 경우를 생각 해 보자. 만약 입력 버스트의 크기 bs = n*b 일 경우 토큰 대기 영역의 크기 즉 L= n*b 일 경우이며 또한 리키 버킷이 토큰 b 개로 가득 차 있을 경우 이것을 출력하는데 걸리는 시간을 계산하면 (pklength * b * (n-1) / r)sec 이다. 그러나 버스트 토큰 fill 방식을 사용하면 fill의 선택에 따라 상당한 패킷 지연을 줄일 수 있다.

만약 L = b인 경우 입력 버스트의 크기 bs = n*b 인 경우에는 (n-1)b 패킷이 버려지므로 상당한 손실을 초래하게 된다. 이러한 경우도 버스트 토큰 fill 방식을 사용하면 fill의 선택에 따라 상당한 패킷 손실 을 줄일 수 있다.

3. 버스트 토큰 fill 모델의 분석

다음 그래프는 ATM cell의 예를 통하여 어떤 변화가 있는지 살펴보았다. 여기서 예는 IEEE에 제출한 Anantharam의 논문[3]에서 사용한 파라미터 값을 기준으로 하였다. 먼저 ATM cell 의 크기를 비트로 환산 하면 424 bit 가된다. 리키 버킷의 크기를 100으로 하고 n = 10으로 놓는다.

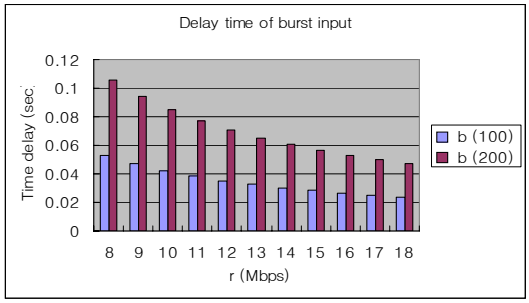


(그림 3) 패킷의 크기를 변화 시켰을 경우

토큰 생성율을 8Mbps에서 18M bps 로 했을 경우를 기본으로 하고 다음과 같은 요소를 변화 시켰을 경우 얼마나 변화가 있는지를 보여 주었다

(그림 3)에서는 패킷 크기를 424 bit에서 1Kbit 로 바꾸었을 경우 입력된 버스트를 네트워크로 내 보내는데 걸리는 시간을 보여 주었다. 패킷일 길수록 내 보내는 시간이 긴 것을 보여준다.

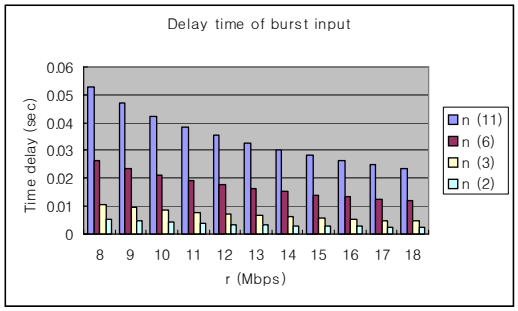
(그림 4)에서는 버킷의 크기의 변화를 보여주었다. b가 100일 경우에서 토큰 생성율이 변화하는 경우에 b가 200 경우 보다 훨씬 덜 민감한 것을 보여 준다.



(그림 4) 리키버킷의 크기 b를 변화 시킬 경우

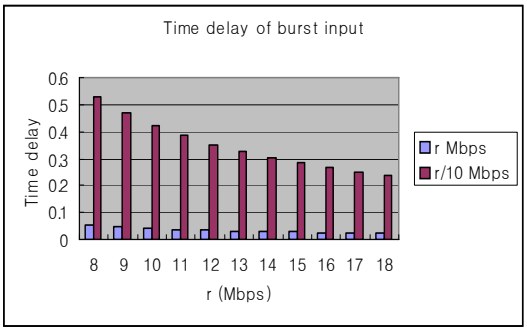
(그림 5)에서는 n의 변화에 따는 시간 지연을 보여 준다. 여기서 버스트 토큰 fill 방식을 사용하면 같은 조건에서 n의 크기에 따라 지연을 상당히 줄일 수 있다. fill 값을 어느 정도 적절하게 선택하느냐에 따라 시간 지연의 양을 현격하게 줄일 수 있다. 여기서 n의 크기는 시간에 비례하여 증가함을 볼 수 있다. 그러므로 버스트 패킷들이 모두 토큰대기영역에 있을 경우를 나타내는 경우이다.

그러나 만약 토큰 대기 영역이 부족할 경우 이들 대부분의 패킷들은 손실 되므로 어떤 응용 프로그램에서는 이들을 다시 보내기위해 상당한 오버헤드를 감수해야 한다.



(그림 5) b 버스트 크기가 n 개 일 때

(그림 6)에서는 r의 비트 율을 10배로 줄였을 경우 시간 지연이 어떻게 변화하는지 보여준다.



(그림 6) r bit rate를 변화시킴

버스토큰 fill 방식을 사용할 경우 네트워크의 로드가 별로 없을 경우에 fill의 값을 너무 많이 선택할 경우는 어떤 현상이 일어나는지 보면 다음과 같다. 5개의 버스트 입력이 들어 왔을 경우 fill 값을 7개로 선택한 경우에 다음에 들어오는 평균 입력 율에 따라 패킷이 네트워크로 입력되기 때문에 순간적으로 많은 양의 패킷이 네트워크로 들어가지는 않는다. 그러므로 비록 fill 값을 잘못 정했다고 해도 네트워크에 급진적인 혼잡현상을 초래 하지는 않는다.

그러나 이미 포화 상태인 경우 fill의 값을 선택하는 것은 매우 조심스럽다. 이미 네트워크에 로드가 많이 있을 경우 fill을 선택하는 알고리즘은 적은 fill의 값 중에서 선택하게 될 것이다 그러므로 1을 택할 것을 2로 택할 경우 순간적으로 b 패킷을 주입하는 경우가 생기므로 b의 크기가 너무 크지 않을 경우는 아주 심각한 혼잡을 초래 하지는 안으리라 생각이 된다. fill의 값을 1 보다 적은 값을 선택하므로 더욱 세심한 배려를 할 수 있을 것이다.

fill의 값을 선택하는 알고리즘은 로드가 약할 경우 2배로 증가 하지만 로드가 심할 경우는 10배로 감소시킴으로 fill 의 값을 선택하는 것도 하나의 방법이 될 수 있다. 하지만 이러한 값이 최선의 최적상태는 아니다.

4. 결 론

리키 버킷을 이용하여 패킷의 최고율과 평균율 그리고 버스트 크기를 조절할 수 있다. 여기서 본 논문은 이러한 근원적인 목적을 손상시키지 않고 기본 알고리즘을 변화 시켜서 상당한 성과를 얻었다.

본 논문에서 제안한 버스트 토큰 fill 방식은 최고율을 유지하고 버스트의 크기도 조절하며 평균율에 그다지 큰 변화를 주지 않으면서도 좋은 효과를 기대 할 수 있다. 버스트 토큰 fill 방식은 fill 값의 올바른 선택이 중요 하지만 약간의 오류에 아주 심각한 혼잡 현상은 초래하지 않는다. 하지만 어느 정도 적당한 값에서도 상당한 성능 향상에 기여 하리라 생각이 된다. 특히 재전송의 경우에는 더욱더 민감하게 반응하리라 생각이 된다.

이 방식은 입력 데이터의 형태에 따라 성능이 상당히 달라지리라 생각이 된다. 포아손 확률 함수를 따르는 입력일 경우 버스트가 일어날 확률을 생각하면 네트워크의 로드가 낮은 경우에는 긴 버스트 입력이 생길 경우가 아주 희박하다. 반면에 로드가 높은 경우에는 버스트가 일어날 확률이 많아지므로 이때 fill 의 값을 크게 잡을 경우 문제가 생길 수 있다 그러므로 fill의 값을 적당히 잡으면 여기서도 어느 정도 효과는 볼 수 있지만 이러한 알고리즘이 별로 의미가 없을 수 있다. 그러므로 여기서 사용되는 알고리즘은 입력데이터 성질에 따라 상당히 성능의 차이가 있으리라 생각된다. 하지만 많은 데이터가 이러한 극단적인 경우가 아닌 혼합한 경우일 경우 버스트 토큰 fill 방식은 의미가 있음을 알 수 있다.

on-off 소스에서 on 상태가 긴 상태를 생각할 경우 리키 버킷의 길이보다도 훨씬 커다란 버스

트 입력이 들어 올 경우 그러나 평균 패킷 율은 중간일 경우 혹은 낮을 경우를 생각해 보면 버스트 토큰 fill 방식은 상당한 효과가 있으리라 생각된다.

참 고 문 헌

[1] M. Sidi, W. Liu, I. Gopal, "Congestion Control Through Input Rate Regulation", IEEE GLOBECOM'89, pp.1764-1768, 1989

[2] L. Kleinrock, Queueing Systems, Vol. I: Theory. New York: Wiley, 1975.

[3] V. Anantharam and T. Konstantopoulos, "Burst Reduction Properties of the Leakey Bucket Flow Control Scheme in ATM Networks", IEEE Transactions on Communications, Vol. 42, NO, 12, December 1994

[4] D. Abendroth et al. Int. J. Electron. Commun. (AEU), volume 60, 2006, page 404-407.

[5] Ph. Oechslin. "Worst case arrival of leaky bucket constrained sources: The Myth of the on off source.", In Proceedings of the fifth IFIP International Workshop on Quality of Service, page 67-77, Columbia University, New York, USA, May 1997.

[6] D. Gan, S. McKenzie "Traffic policing in ATM networks with multimedia traffic: the super leaky bucket", Computer Communications, Volume 22, Issue 5, 15 April 1999, page 439-450.

[7] M. Yaghmaee, M. Safavi, M. Menhaj "An Intelligent usage parameter controller based on dynamic rate leaky bucket for ATM networks", Computer Networks volume 32, Issue 1, January 2000, page 17-35.

[8] K. Michell, A. Liefvoot, J. Place " Correlation properties of the token leaky bucket departure process", Computer Communications, Volume 21, Issue 11, August 1998, page 1010-1019.

[9] B. Choi, C. Park, D. Sung " Performance analysis of a Leaky Bucket scheme with a threshold in the data buffer", Compute Networks and ISDN Systems, volume 29, Issue 7, August 1997, page 781-795.

김 동 철

1989년 : University of Utah 학사
1993년 : Oregon State University 석사
1997년 : Brigham Young University 박사
1998년~현 재 : 평택대 컴퓨터학과 교수
관심분야 : 컴퓨터네트워크, QoS 관리, 분산 시스템, 네트워크성능분석