통신망의 혼잡제어를 위한 변형된 slow start 알고리즘

남 지 승† · 조 진 용†† · 임 정 균††

요 약

네트워크의 발달에 의해 현대 사회는 정보의 국경이 사라졌다. 컴퓨터 네트워크를 이용해 현대인들은 대용량의 데이터를 빠른 시간 내에 전송할 수 있게 되었고, 수많은 정보를 보다 쉽게 얻을 수 있는 환경이 조성되었다. 그러나, 네트워크 상에 데이터의 흐름이 많아지면서 혼잡 현상이 발생하게 되어 이에 대한 관리가 중요하게 되었고, 그 중의 한 방법이 slow start 이다. 본 논문에서는 slow start의 변형을 통해 네트워크 상의 혼잡을 보다 효율적으로 제어하고, 완화시킬 수 있는 방법을 제안하고 그 성능을 평가하였다.

Modified Slow Start Algorithm for Network Congestion Control

Ji-Seung Nam† · Jin-Yong Jo†† · Jung-Kyun Lim††

ABSTRACT

The improvement of network systems makes the world be near at hand. Due to the technological progress, we can send or receive a large amount of data in a short time over computer networks. However, The data overflow on the network can cause severe congestion. So we need to control it more efficiently. One of the control methods is a slow start. In this paper, we proposed a amended slow start for network congestion and tested its efficiency.

1. 서 론

정보 사회에서는 통신, 방송, 대화형 화상 시스템과 같은 다양한 멀티미디어 서비스들이 서로 융합되어 있고, 이들 서비스에 대한 수요도 날로 증가하고 있다. 또한, 서비스들의 원활한 제공을 위한 통신망의 발달은 대용량, 초고속화되어 왔으며, 지난 수년간 과히 폭발적인 성장을 해 왔다.

하지만, 통신망의 속도는 정보 수요자들이 요구하는 수준에 도달하지 못했다. 또, 전송되는 정보를 처리해야 할 송/수신 PE(Processing Element)간의 처리 속도 불균형, 데이터 흐름의 집중 현상, 통신망의 용량을 초과하는 데이터 유입 등으로 인해 과다한 패킷들이 네트워크 상에 올려 지게 되고, 이런 패킷들의 범람은 네트워크 통신 효율을 떨어지게 해 결국, 혼잡 현상을 초래하게 되었다.

단편적인 예로, 인터넷 gateway에서 버퍼 범람(buffer overflow)으로 인해 약 10%의 처리되어야 할 패킷들이 버려지고 있다고 한다[1]. 이런 패킷의 손실이 대규모의 정보 교환과 실시간 서비스를 필요로 하는 시스템 사이에서 원활한 정보 교환을 방해하고 있는 실정

※ 이 논문은 전남대학술연구비의 지원에 의하여 연구되었음.
† 종신회원 : 전남대학교 컴퓨터공학과 교수
†† 준 회 원 : 전남대학교 대학원 컴퓨터공학과
논문접수 : 1998년 12월 3일, 심사완료 : 1999년 7월 8일

이다. 따라서, 시스템간의 흐름 및 혼잡 제어는 네트워크 통신에 있어 필수적인 요소가 되었다.

이에, 본 논문에서는 TCP/IP(Transmission Control Protocol/Internet Protocol)의 slow start, 혼잡회피(congestion avoidance), 빠른 재전송(fast retransmit) 그리고 빠른 복구(fast recovery) 알고리즘을 살펴 본 후, 음성과 화상 통신에 많이 사용되는 UDP(User Datagram Protocol)[2]를 이용해 망 혼잡을 보다 효과적으로 처리 할 수 있는 흐름제어 알고리즘을 구현하고, 성능 평가함으로써 TCP/IP에 적용시키고자 한다.

2. 기존의 혼잡제어 기법

1980년대 IP 프로토콜을 이용하는 비 연결형 끝점간 패킷 전송 서비스에 기초한 Internet protocol 구조에서, 패킷 전송 경로 추적에 대한 주의 부족으로 인해 네트워크 서비스에 혼란을 초래하게 됐는데, 이로 인해 흐름 및 혼잡 제어에 관한 연구가 활발하게 이루어지는 계기가 되었다[3].

1986년 초, Van Jacobson은 통신망 상의 혼잡 현상을 막기 위해 TCP에 구현할 수 있는 혼잡 회피법(congestion avoidance mechanism)을 발전시켰다. 이 메커니즘은 호스트 내에서 TCP 연결점들로 하여금, 혼잡이 발생하는 동안 'back off' 시키게 한 것으로, 그후, Internet에서 널리 사용하는 기법으로 발전되었다[4].

1988년 이후, Internet이 급속도로 성장하였고, Van Jacobson의 혼잡 회피법이 모든 네트워크 환경에 적합하지 않아 충분한 흐름 및 혼잡 제어를 하기에는 부족하다는 인식과 함께, 이에 대한 연구 결과가 나오기 시작했으나, 네트워크의 끝점(end point)이 수행해야 할 흐름제어의 강도에 대해서는 한계가 있었다. 이에 따라 라우터에서의 끝점 혼잡 보완을 위한 메커니즘이 필요하게 되었다.

라우터에서 흔히 행해지는 흐름 또는 혼잡제어에 관한 알고리즘은 크게 '큐 관리'와 이를 통한 'scheduling'을 들 수 있는데 '큐 관리'란 필요한 시기에 패킷들을 버림으로써, 패킷 큐의 길이를 관리하는 방법을 말하며, 'scheduling'은 패킷의 사용, 전송, 및 이들의 흐름간 대역폭(bandwidth)등을 할당하는 방법이다[5].

각 큐는 설정된 최대 크기 이하로 패킷을 받아 들인 후, 큐의 최대값에 도달하게 되면 큐 내부로 들어오려는 패킷들을 버리게 되는데 이 방법을 'Tail drop'이라고 한다. 이는 과거 몇 년간 인터넷에서 널리 사용되어 졌으며, 수행시키는데도 큰 문제는 없었으나, 2가지 결점을 가지고 있다.

첫째, 'Lock-out'현상을 들 수 있는데, 이는 소수의 흐름에 의해 큐 공간이 독점되고, 이에 따라 네트워크 서비스를 원하는 PE들이 큐 공간을 획득하는데 어려움을 갖게 되는 현상이다.

둘째, 큐가 가득 채워졌을 때에만 혼잡을 알리게 되므로, 오랜 기간 동안 큐는 여유 공간 부족 상태에 놓이게 된다는 점이다.

위 두가지 문제점의 해소를 위해, 끝점간 지연에 대한 부담을 감수해야 한다.

이와 더불어 큐 관리에서 사용되어지는 2가지 패킷 버림(drop)법이 있다. 이는 큐가 가득 채워져 있을 때, '무작위 추출'을 통한 버림과 '큐 전면부'에 있는 패킷을 버리는 방법으로, 이 두 방법을 통해 'Lock-out' 현상은 해결될 수 있지만, 오랜 시간 동안 큐의 여유공간이 부족한 상태에 놓이게 되는 문제는 해결할 수 없다[6].

여유 공간 확보 문제를 해결하기 위해서, 라우터는 큐가 가득 차기 전에 패킷을 버려야 하고, PE는 버퍼가 넘치기 전에 혼잡에 대해 대응해야 하는데 이 방법을 '능동적 큐 관리'라고 한다[7].

지금까지 설명했던 라우터 내에서의 기본적인 흐름제어 방법과 더불어 TCP/IP에서는 slow start, 혼잡회피, 빠른 재전송, 그리고 빠른 복구 등의 4가지 알고리즘을 함께 묶어 사용한다[8].

2.1 Slow start

구 TCP/IP에서는 수신측에서 알려온 윈도우 크기만큼의 패킷들을 네트워크 상에 올리곤 했다.

송신측과 수신측에 연결된 라우터간의 대역폭이 다를 경우, 중간 라우터는 손실을 막기위해 패킷을큐에 넣게 된다. 패킷의 범람으로 인해 큐공간이 부족해지면 처리율이 감소되는데, 이를 막기 위해 slow start를 사용한다[9].

이 기법은 새로운 패킷이 네트워크에 올려지는 비율과 수신측에서 보낸 ACK신호를 관찰하면서 동작하게 된다. slow start는 cwnd라는 혼잡 윈도우를 사용하는데, 이는 새로운 연결이 설정될 경우, 그 크기를 하나의 패킷 크기로 초기화하고 송신측에서 보낸 신호에 대하여 수신측에서 ACK신호를 보내 올 때마다 혼잡

윈도우 cwnd의 크기를 늘려 가는 방식이다.

이때 송신측에서는 네트워크 상태를, 수신측에서는 버퍼 크기에 대한 정보를 관리하게 되며, 수신측은 혼잡 윈도우의 크기와 수신측의 버퍼 크기 중 작은값으로 다음에 보내질 패킷의 수를 결정하게 된다.

2.2 혼잡 회피

두 번째 방법으로는 혼잡 회피를 들 수 있으며 이는 패킷의 손실에 관해 다루는 것이다. 패킷이 손실되었다고 가정할 수 있는 경우는 타이머가 설정해 놓은 시간 내에 ACK신호를 받지 못했거나, 수신측에서 중복된 ACK신호를 보냈을 경우에 해당된다.

이와 같은 방법으로 혼잡 발생이 인식되면, 패킷의 전송률을 떨어뜨린 후, slow start 알고리즘을 실행하게 된다.

2.3 빠른 재전송 및 빠른 복구

TCP는 이 중복된 ACK신호가 패킷 손실에 의한 것인지, 잘못된 순서에 의한 것인지를 알 수 없으므로, 하나 또는 두 개의 중복된 ACK신호가 올 경우에는 패킷 순서에 오류가 발생한 것으로 가정하고, 그 오류를 수신측에서 처리하게끔 한다.

만약, 세 개 이상의 ACK신호를 받는다면 패킷 손실로 판단, 손실된 패킷을 즉시 재 전송하게 되는데 이를 빠른 재전송이라고 한다. 패킷 손실로 간주되어 빠른 재전송을 한 후에는 slow start를 수행하지 않는데, 이를 빠른 복구라고 한다.

이 경우 slow start를 실행하지 않는 이유는 중복된 ACK신호를 통해, 하나 이상의 패킷이 분실되었음을 이미 인식하고 있기 때문이고 데이터가 네트워크 상에서 이동되고 있는 상황에서 slow start를 수행함으로써 발생할 수 있는 전체적인 흐름을 감소시키지 않기 위해서이다.

3. Slow start의 구현 및 문제점

지금까지 TCP/IP에서 사용되고 있는 혼잡제어 알고리즘에 대해서 간략히 살펴보았다. 지금부터 자바언어를 이용해 UDP상에 sliding window, slow start 등의 알고리즘을 구현해 보고, 각 알고리즘의 문제점들을 살펴 본다.

먼저, 자바 언어에서 사용되는 데이터그램 소켓과 패킷의 생성법, 패킷 송/수신에 관한 인스턴스들을 살펴본다[10].

- **Datagram socket의 생성**

```
try{
    DatagramSocket s
            = new DatagramSocket();
    }catch(Exception e){
                .....
    }
```

- **Datagram packet의 생성**

```
DatagramPacket spack
= new DatagramPakcet(Bytes,Bytes.length,
                    targetHost,port);
```

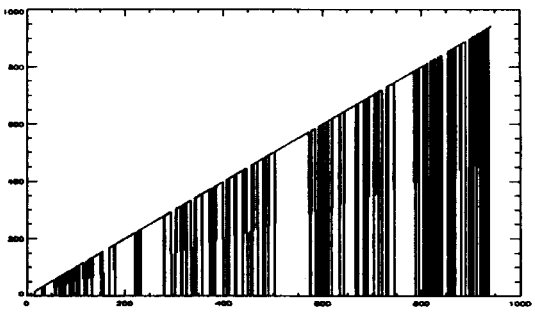
- **패킷 송신**

```
(DatagramSocket)s.send((DatagramPacket)
                                packet);
```

- **패킷 수신**

```
(DatagramSocket)s.receive((DatagramPacket)
                                packet);
```

위의 기본적인 인스턴스 외에 본 논문에서 사용된 자바 언어 함수들의 구체적인 설명에 대해서는 생략한다. 이렇게 만들어진 패킷들이 송신측과 수신측의 아무런 제어 없이 LAN 상에서 이동 될 때, 약 20.7%의 패킷 손실을 가져 왔는데, 이는 PE에서의 패킷 추적만으로는 그 이유를 밝힐 수 없었고 단지, 수신측 포트에서 받아들일 수 있는 양 이상의 패킷 송신으로 인해 손실된 것으로 생각되어 진다.[1]



* x축 : 순서수, y축 : 순서수와 손실 유무

(그림 1) 제어가 없었을 때, LAN 상의 패킷 손실 수신측에서 ACK신호를 받았을 경우 순서수에 해당하는 y축 값을 갖게 되며, 손실된 경우 0값을 갖는다.

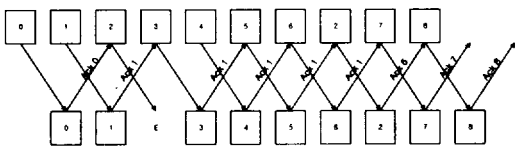
1) 송신측은 패킷의 생성과 송신만을 하며, 수신측은 패킷을 받아 순서수(Sequence Number)만을 기록한다.
두 PE 양단에서 행해지는 어떠한 제어도 없으며, 수신측은 송신측에게 ACK신호를 보내지 않는다. 20.7%는 네트워크 상황에 따라 다소 차이가 있음.

(그림 1)은 아무런 제어가 없었을 때 나타난 패킷 손실에 관한 그래프이다.

위와 같이 네트워크의 성능이 떨어지거나, 송신측과 수신측의 시스템간 처리 속도가 맞지 않는다면, 데이터 손실이라는 큰 문제를 야기 시키게 된다. 이 때문에 PE 양단의 처리 속도 차이를 시간 지연을 통해 해결하는 흐름제어가 필요하게 되며, 이를 통해 패킷 손실에 의한 송신측의 재 전송률을 감소시킬 수 있다.

하지만, 버퍼 관리 없이 시간 지연에만 의해 혼잡제어가 이루어진다면, 송신측은 재전송 해야 할 패킷에 대한 부담을 갖게 된다. 이러한 부담을 없애기 위해서 버퍼 관리는 필수적이다.

버퍼 관리 기법중의 하나로, (그림 2)와 같은 sliding window를 들 수 있다.



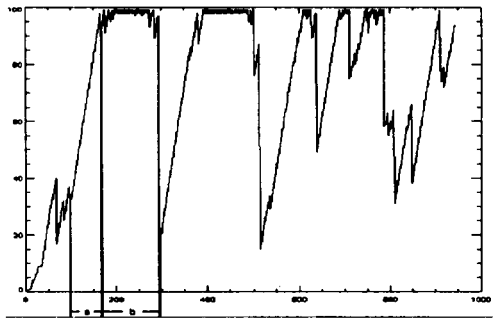
(그림 2) sliding window

송신측은 수신측과 동일한 초기 버퍼를 가지게 되며, 위 (그림 2)에서 볼 수 있듯이 패킷 손실이나 오류로 인해 잘못된 패킷을 수신했을 경우, 가장 최근에 정상적으로 수신됐던 패킷의 순서수를 ACK신호로 송신측에 전송하게 된다. 송신측은 timeout이 발생했거나, 동일한 순서수를 갖는 ACK신호를 연속적으로 수신할 경우, 패킷 손실로 간주해 ACK신호 내의 순서수 +1번째 패킷을 재전송하게 된다. 또한, 송신측과 수신측에서는 중복된 ACK신호나 데이터 오류에 대한 책임을 갖는다.

(그림 3)은 가상 환경 하에서 sliding window를 이용한 파일 전송에서 나타난 송신측의 버퍼 크기 변화를보여준다.[2)]

실제 LAN상에서 sliding window는 패킷 전송을 모두 마치기까지 걸린 시간 면에서 최고의 성능을 나타낸다. 하지만, (그림 3)의 a나 b구간에서와 같은 속성으로 인해 문제점을 야기시킬 수 있다.

2) 가상 환경
- 무작위 시간 지연 : 10~50 mS/packet
- 무작위 패킷 버림 : 전체 패킷의 3%이내
- 송신측이 갖는 최대 버퍼 크기 : 100 × 1026 바이트



* x축 : 수신측에서 받은 패킷의 수, y축 : 버퍼 크기

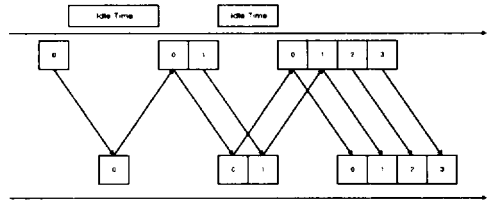
(그림 3) 송신측 버퍼 크기 변화

먼저, a구간은 수신측에서 인위적으로 가해졌던 시간 지연이나 패킷 버림에 의해 송신측의 버퍼 크기가 증가되고 있음을 보여준다. 이는 실제 WAN상에서 중간 노드의 혼잡을 가상한 것으로, 이 구간은 중간노드의 혼잡으로 인해 ACK신호를 RTT내에 받지 못하게 되므로 버퍼 크기가 증가된 것이다.

이런 현상이 발생할 경우, 네트워크 상의 중간 노드들은 들어오는 패킷을 처리할 능력을 상실하게 되며, 이와 같은 패킷 손실로 인해 송신측은 재전송에 대한 부담이 늘어나게 된다.

또, b구간은 a구간에서 발생했던 묶음 단위의 패킷 손실로 인해 버퍼내의 여유 공간이 없는 상태를 나타낸다. 이 때는 망 혼잡이 해결 될 때까지 stop and wait의 속성을 보이므로, 전송 효율이 떨어지게 된다.

따라서, 송신측은 수신측에서 처리할 수 있는 만큼의 패킷을 보내야 하는데, 이를 구현한 알고리즘이 (그림 4)와 같은 slow start이다.



(그림 4) slow start

slow start 알고리즘을 구현하는데 있어 timeout의 설정이 중요한데, 그 이유는 아래와 같다.

(1) Timeout이 실제 RTT보다 크게 설정되어 있고, 수신측으로 부터 ACK신호를 받지 못 할 경우

- 버퍼가 가득찬 상태로 ACK신호를 기다리게 되므로, 불필요한 시간을 낭비하게 되고 빠른 재전송이 불가능해지며, 이에 따라 전체 네 트워크 효율이 떨어진다.

(2) Timeout이 실제 RTT보다 작게 설정되어 있고, 수신측으로 부터 ACK신호를 받지 못할 경우

- 실제 손실되지 않은 패킷들에 대해, 손실된 것으로 간주하고 재전송을 하게 되므로, 중복된 패킷을 네트워크 상에 올려 중간 노드들 사이의 혼잡을 가중시킨다.

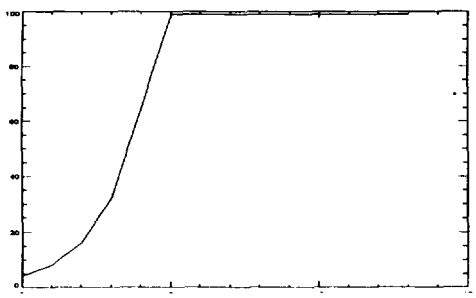
위 현상들을 해결하기 위해 TCP 프로토콜 사양[11]에서는 RTT의 평형성 유지를 위해 다음과 같은 timeout 설정법을 제안하였으며, 본 논문에서도 같은 방법을 이용해 구현하였다.

$$R' \leftarrow \alpha R + (1-\alpha)M$$

여기서, R은 현재까지 보낸 패킷에 대한 RTT의 평균값이며, α는 상수, M은 가장 최근 보내진 패킷에 대한 RTT이다. α값을 0.9로 제안하였고, 이 R'값을 이용해 다음 패킷에 대한 timeout은 2R'의 값으로 설정되어진다.

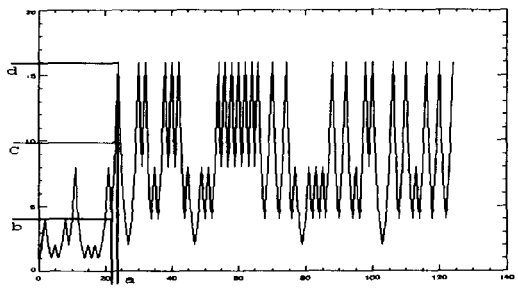
또한, timeout이 되기 전, 수신측에서 버퍼에 존재하지 않는 순서 수를 갖는 ACK신호를 계속해서 보낼 때, 송신측은 패킷 손실로 간주하고 수신측에서 보내온 순서수+1의 순서를 갖는 패킷을 재전송을 하게 된다. 이때 송신측은 중복된 ACK신호에 대한 책임을 가지고 있다. 이는 네트워크 상에 불필요한 패킷들을 올려놓지 않기 위해서 실행되어지는 알고리즘이다.

(그림 5)와 (그림 6)은 실제 LAN상과 가상환경하에서, slow start를 수행했을 때 나타나는 송신측의 버퍼 크기 변화이다.



* x축 : 버퍼크기가 설정된 회수, y축 : 버퍼크기

(그림 5) 실제 LAN상의 송신측 버퍼 크기 변화



* x축 : 버퍼크기가 설정된 회수, y축 : 버퍼크기

(그림 6) 가상 환경의 송신측 버퍼 크기 변화

(그림 5)와 (그림 6)에서 볼 수 있듯이 slow start 알고리즘은 송신측이 수신측에서 처리할 수 있는 만큼의 패킷을 보내게 되므로 수신측에 적은 부담을 주고, 중간 노드에서 혼잡을 야기시킬 수 있는 가능성을 그만큼 감소시켜 준다.

하지만, slow start도 다음과 같은 약간의 문제점이 있을 수 있다. (그림 6)의 x축 a구간에 해당되는 y축 b~d 구간은 이전 버퍼크기에서 timeout이 발생하지 않아 버퍼가 2배 증가하고 있는 상태이다. 이렇게 2배 증가된 상황에서 송신측이 보낸 패킷이 손실되거나 timeout이 발생했을 경우에 그 다음으로 보내졌던 패킷들은 손실되거나 timeout될 가능성이 높다.[3)]

예를 들어, y축의 c지점에서 timeout이 발생했을 경우, c~d구간에서는 중간 노드의 혼잡에 의한 패킷 손실량이 늘어나게 된다. 이럴 경우, 송신측 버퍼에는 ACK신호를 받지 못한 패킷들로 채워지게 되며, 이런 패킷의 범람으로 인해 중간 노드는 혼잡에 대한 부담을 갖게 된다.

4. 수정 알고리즘 제안

(그림 6)의 c~d 구간과 같은 문제점으로 인해 발생할 수 있는 중간 노드의 혼잡을 해결하기 위한 몇 가지 방법을 생각해 볼 수 있다.

(1) 송신측에 시간 지연을 가함으로써, 중간 노 드들 사이의 혼잡을 완화 시킬 수 있는가?

(2) 혼잡 상황을 미리 예측한 후, 혼잡 발생이 예측될 경우 송신측에서 패킷 전송을 중지시 킴으로써 혼잡을 완화시킬 수 있는가?

3) 송신측에서 패킷을 네트워크 상에 올릴 때의 시간 지연에 비해 중간 노드에서 발생하는 지연이 크기 때문이다.

(1)의 경우, 중간 노드에서 발생할 수 있는 혼잡에 대해서는 효과적으로 대처 될 수 있다. 하지만, 네트워크 상황이 원활할 경우, 송신측의 시간 지연에 의해, 올려지는 패킷의 수가 제한되므로, 전송 효율면에서 뒤쳐지게 된다.

(2)의 경우, timeout시간과 실제 ACK신호가 도착한 시간에 대한 통계를 이용한다면, 혼잡 상황에 대한 예측이 어느 정도 가능하다. 하지만, 송신측의 버퍼 크기가 작을 경우, 첫 번째 보낸 패킷에 대한 ACK신호가 도착하기 전에 송신측 버퍼에 여유 공간이 없게 되므로, 문제가 발생된다.

또, 혼잡 예측을 위해서는 정확한 timeout 시간설정이 필요하지만, 네트워크 traffic이 일정하지 않으므로 timeout 간격에 의존해 혼잡을 예측하기는 어려운 일이다.

위와 같은 제한 때문에 시간 지연 또는 예측에 의한 혼잡제어는 효율이 떨어지게 된다.

따라서, (그림 6)의 c~d 구간에서 발행할 수 있는 패킷 손실량을 줄이기 위해서는 네트워크 상황에 유동적으로 대응할 수 있고, 버퍼 크기의 증가폭을 완화시켜 줄 수 있는 새로운 인자의 도입이 필요하게 된다.

평균 버퍼크기는 네트워크 혼잡 정도에 대한 정보를 함축하고 있으며, 이 정보를 이용하면 버퍼크기 증가폭을 네트워크 상황에 맞춰 완화시킬 수 있으므로, 위의 두 요건을 만족시키게 된다. 즉, 갑작스런 버퍼크기 증가로 인한 패킷 손실을 줄이기 위해, 버퍼크기 증가시 기존 버퍼크기 설정법(2×현재 버퍼크기)에 평균 버퍼크기를 도입해 각각에 가중치를 줌으로써 증가폭을 완화시킬 수 있다.

본 논문에서 제안하는 slow start의 변형은 다음과 같다.

(1) 이전 버퍼 크기에서 timeout이 발생하지 않 았을 경우(버퍼 크기 증가)

$$B_{next} = \alpha B_{avg} + (1-\alpha) \times (2B_{cur})$$

(2) 이전 버퍼 크기에서 timeout이 발생한 경우(버퍼크기 감소)

$$B_{next} = B_{cur}/2$$

B_{next}는 다음에 확보되어야 할 버퍼크기를 나타내며, B_{avg}는 현재까지의 평균 버퍼 크기, B_{cur}은 현재 버퍼크기를 나타낸다. B_{avg}와 B_{cur}은 각각 패킷 손실을 줄이고, 대역폭 낭비를 막기 위해 사용되어진다.

변형된 식에서 B_{avg}를 사용함으로써, 전체 버퍼크기 설정에 미치는 영향은 다음과 같다.

(1) $B_{avg} < 2B_{cur}$일 경우

변형 slow start가 갖는 버퍼 크기는 기존 slow start가 갖는 버퍼크기 보다 작게 된다.

$$\alpha B_{avg} - 2\alpha B_{cur} + 2B_{cur} < 2B_{cur}$$
$$(\because 2\alpha B_{cur} > \alpha B_{avg})$$

즉, 평균 버퍼크기에 비해 현재 버퍼크기가 클수록 Bavg값이 버퍼크기 설정에 많은 부하를 가하게 되므로 갑작스런 버퍼크기 증가로 인한 패킷 손실을 줄일 수 있다.

(2) $B_{avg} > 2B_{cur}$

변형 slow start가 갖는 버퍼 크기는 기존 slow start가 갖는 버퍼 크기보다 크게 된다.

$$\alpha B_{avg} - 2\alpha B_{cur} + 2B_{cur} > 2B_{cur}$$
$$(\because 2\alpha B_{cur} < \alpha B_{avg})$$

이 때는 네트워크 혼잡으로 인해 낮은 버퍼 크기를 유지하게 된다. 혼잡이 완화될 경우 버퍼 크기가 증가하게 되므로, 버퍼 크기의 증가폭을 기존 slow start에 비해 크게 함으로써 대역폭 낭비를 줄일 수 있다.

다음으로 버퍼 크기 설정에 이용했던 가중치 α의 도출을 위해 평균 버퍼 크기와 현재 버퍼 크기와의 관계를 따져볼 필요가 있다. 패킷 손실이 전혀 없을 경우 기존 slow start에서는 2n으로 증가 하게 되므로 평균 버퍼 크기는 (2n-1)/n이 되고, 현재 버퍼 크기는 2n-1이 된다.

현재 버퍼 크기를 1로 봤을 때, 이에 대한 평균 버퍼크기는

$$\frac{2^n-1}{n} : 2^{n-1} = x : 1$$
$$x = \frac{2(2^n-1)}{n \times 2^n} \fallingdotseq \frac{2}{n}$$

($\because$n은 현재까지 버퍼크기가 설정된 회수로, 패킷 손실이 없을 때 버퍼크기는 2^{n-1}이 된다.)

즉, 현재 버퍼 크기의 약 $2/n$이 된다.

$$B_{avg} \fallingdotseq 2B_{cur}/n = \frac{2^n}{n}$$

또, 버퍼크기가 증가될 때 기존 slow start는 $2\times B_{cur}$의 크기를 갖게 되고, 변형된 slow start는

$$\alpha B_{avg}+(1-\alpha)\times(2B_{cur}) \fallingdotseq \alpha \frac{2B_{cur}}{n}+(1-\alpha)\times(2B_{cur})$$
$$= (\frac{\alpha}{n}+1-\alpha)\times 2B_{cur}$$

의 크기를 갖게 된다.

따라서, 변형된 slow start가 기존 slow start에 비해 낭비할 수 있는 대역폭은

$$(\frac{\alpha}{n}+1-\alpha)\times 2B_{cur} : 2B_{cur} = x : 1$$

$$x=\frac{\alpha}{n}+(1-\alpha)$$

($\because 0 < \alpha < 1$, α는 가중치이고, 평균 버퍼 크기와 기존 버퍼 크기 설정법을 같이 활용하기 위해서는 0이나 1값을 가질 수 없다.)

x가 된다.

마지막으로, 네트워크 상황이 원활할 때 기존 slow start에서 사용되는 대역폭의 90% 이상을 변형 slow start에서 사용할 수 있게 하기 위해 α는

$$0 < \alpha \leq 0.1$$

의 값을 갖는다.

실험에서 사용된 α값은 0.1로, 0.1이하의 값을 선택했을 경우 Bavg값이 버퍼 크기 설정에 미치는 영향이 너무 적기 때문에 최대값을 이용했다.

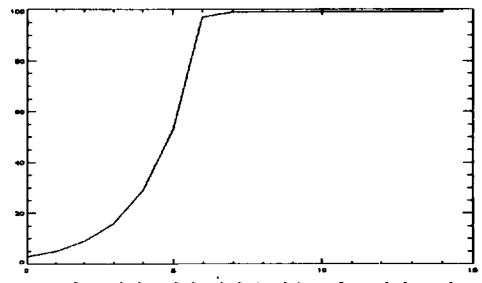
위의 정보를 이용해 변형 구현된 slow start 알고리즘에 의한 송/수신 특성 중, 송신측의 버퍼크기 변화를 관찰한 결과는 (그림 7)과 같다.

(그림 5)와 비교해 (그림 7)의 버퍼 증가폭이 완화되었음을 알 수 있다. 이는 패킷 손실이 전혀 없는 상황에서 기존 slow start에 비해 대역폭을 낭비할 수 있는 여지를 가지고 있지만 그 차이는 극히 미소하다.[4]

(그림 8)은 변형 slow start에서 볼 수 있는 송신

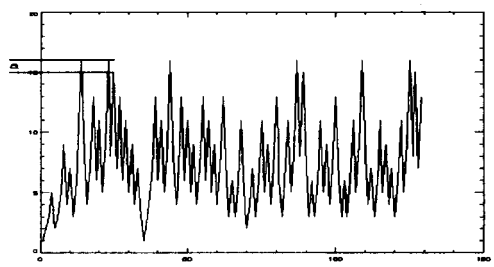
측의 버퍼 크기 변화이다. (그림 6, 기존 slow start)에 비해 (그림 8, 변형 slow start)의 y축 a구간을 주목할 필요가 있다.

만약 기존 slow start를 이용했다면, a하단의 선 위치가 a상단의 선 위치로 이동했을 것이다. 즉, 변형된 slow start는 기존 slow start에 비해 (그림 8)의 a구간과 같은 곳에서 발생할 수 있는 패킷 손실을 줄여 준다.



* x축 : 버퍼크기가 설정된 회수, y축 : 버퍼 크기

(그림 7) 실제 LAN상에서 버퍼 크기 변화



* x축 : 버퍼크기가 설정된 회수, y축 : 버퍼 크기

(그림 8) 가상 환경하에서 송신측의 버퍼 크기 변화

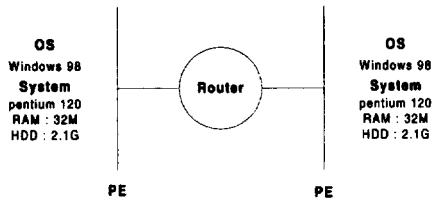
5. 알고리즘 성능 평가

지금까지 기존 slow start와 변형된 slow start에 대해 알아보았다. 성능 평가에 앞서 두 알고리즘 모두에 공평성을 부여하고, WAN 환경에 최대한 유사하게 만들기 위해 다음과 같은 조건하에서 실험하였다.

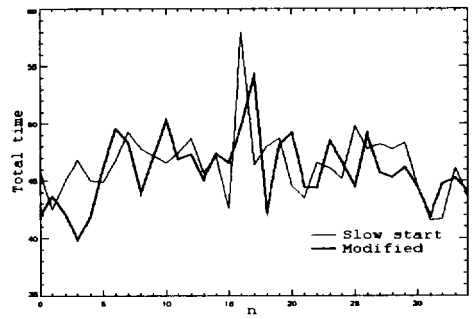
- 수신측의 시간 지연은 10~50 (×0.001 sec)사이, 패킷 버림은 전체 패킷의 3% 범위 내에서 무작위 발생시켰다.
- 위 사항에 공평성을 기하기 위해 자바 언어의 Random 함수에 seed값을 주어 두 알고리즘 모두 같은 무작위 순서를 갖게 하였다.
- 네트워크 상황 변화를 고려해 두 알고리즘은 번갈아 테스트했다.
- 송신측과 수신측 모두 최대 우선 순위를 갖는 thread에 의해 실행 된다.

다음 (그림 9)는 실험이 수행된 환경을 보여 준다.

4) 패킷 손실이 없는 LAN환경에서 테스트 한 결과, 오히려 변형 slow start가 942 Kbyte의 파일 전송을 마치기까지 걸린 시간면에서 기존 slow start보다 빠를 수도 있음을 확인했다.(기존: 5.490 sec, 변형 : 5.270 sec)

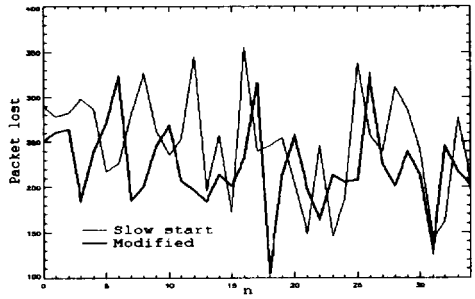


(그림 9) 실험 환경



(그림 10) total time 비교

(그림 10)은 시도수[5]에 대한 전송 시간(초)을 나타내는 그래프로, 전송 시간은 인위적인 시간 지연, 네트워크 traffic, 패킷 손실에 의한 재전송량 등에 의해 결정되어 진다.



(그림 11) 패킷 손실 비교

또, (그림 11)은 변형된 알고리즘과 기존 알고리즘에 의해 전송된 패킷의 손실량으로, 변형 slow start는 기존 slow start를 실행했을 때보다 평균적으로 낮은 패킷 손실을 보이고 있음을 알 수 있다.

실제, 네트워크 traffic을 감안하더라도 변형 알고리즘은 total time 측면에서 기존 알고리즘에 비해 약 1.16%의 성능 향상이 있었고, 전체 패킷 손실 측면에서도 기존 slow start법에 비해 10.3%정도의 패킷 손실이 덜 발생하는 것을 확인할 수 있었다.

5) (그림 10)과 (그림 11)은 네트워크 traffic이 일정하지 않은 LAN 환경에서 가상환경을 조성한 후, 테스트한 결과이다. 12시부터 16시까지 약 5분 간격으로 총 35회 실시했으며, 각 그림의 시도수는 942Kbyte 크기의 파일전송이 끝났을때 1회로 설정되어진다.

6. 결 론

네트워크의 효과적인 관리와 이용을 위해서는 패킷의 범람으로 인해 야기될 수 있는 중간 노드들 사이의 혼잡에 대한 효과적인 제어가 필요하다. 따라서, 송/수신측 PE는 네트워크 상에 올려지는 패킷의 수를 조절할 필요가 있다.

현재 TCP/IP에서는 slow start, 혼잡 회피, 빠른 복구, 빠른 재전송 등의 기법들을 이용하여 중간 노드에게 적은 부담을 주면서, 효율적인 전송을 책임지게 한다.

slow start는 그 이름과 달리 파일의 전송 속도와 네트워크에 주는 부담 등의 전반적인 망 혼잡 문제를 해결하는데 적절히 이용되고 있다. 하지만, 중간 노드가 작은 파이프로 구성되어 있고, 송신측에서 큰 버퍼 크기를 가지고 slow start를 이용해 데이터를 전송한다면, 묶음 단위의 패킷 손실을 초래할 수도 있다.

이점을 해결하기 위해서 송신측은 버퍼 크기에 관한 정보를 가지고 있을 필요가 있으며, 이 정보를 적절히 이용해 버퍼 크기를 변화시킬 경우, 묶음 단위의 패킷 손실을 어느 정도 예방할 수 있다.

본 논문에서는 버퍼 크기 변화를 관찰하면서 얻은 정보를 이용해, 기존 slow start에서 변형된 버퍼 크기 설정법을 제안하고 테스트 했다.

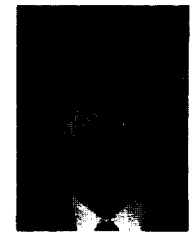
제안된 알고리즘에 의해 데이터를 전송했을 때, 패킷 손실량과 total time의 측면에서 각각 10.3%와 1.16%의 성능 향상이 있음을 확인했다. 보다 효과적인 혼잡 제어를 위해서는 버퍼 관리와 더불어 timeout 측정에 관한 연구가 계속되어야 할 것이라고 생각된다.

참 고 문 헌

[1] Van Jacobson and Michael J. Karels, "Congestion Avoidance and Control," Computer Communication Review, Vol.18, No.4, pp.314-329, Aug. 1988.

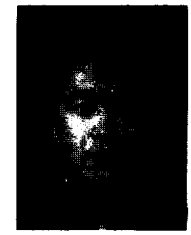
[2] J. Postel, "User Datagram Protocol," RFC 786, pp.1-3, Aug. 1980.

[3] J. Nagle, "Congestion Control in TCP/IP," Internet Draft, pp.1-4, Jan. 1998.
[4] R. Braden, ED, "Requirements for Internet Hosts Communication Layers," RFC 1122, Oct. 1989.
[5] Karol, M.J., Hluchyj, M.G., and Morgan, s.p. "Input Versus Output Queueing on a Space Division Packet Switch," IEEE Trans. on Comm., Vol.35, pp.1347-1356, Dec. 1987.
[6] Raj Jain, "A Timeout-Based Congestion Control Scheme for window Flow-Controlled Network," IEEE Journal on Selected Areas in Comm, Vol. SAC-4, No.7, pp.1162-1167, Oct. 1986.
[7] Bob Braden and Dave Clark, "Recommendation on Queue Management and Congestion Avoidance in the Internet," pp.1-10, Feb. 1998.
[8] W. Richard Stevens and Gary R. Wright, "TCP/IP Illustrated, Volume1 and Volume2," Addison-Wesley, 1994, 1995.
[9] Raj Jain, "Myths about Congestion Management in High-Speed Networks," Telecommunication and Networks, DEC, 550 king st, littleton, MA 01460, DEC-TR-726, pp.1-10, Oct. 1989.
[10] Campione, M., and Walrath, K. "The Java Language Tutorial : Object-Oriented Programming for the Internet," MA, Addison-Wesley, 1996.
[11] Postel, J. Ed. "Transmission Control Protocol Specification," SRI International, Menlo Park, CA, RFC 793, Sep. 1981.



남 지 승

e-mail : jsnam@chonnam.chonnam.ac.kr
1981년 인하대학교 전자공학과 졸업(공학사)
1985년 University of Alabama, Electrical Engineering 졸업(공학석사)
1992년 University of Arizona, Electrical & Computer Engineering 졸업(공학박사)
1992년~1995년 한국전자통신연구소 선임연구원
1995년~현재 전남대학교 컴퓨터공학과 부교수
관심분야 : 통신 프로토콜, 실시간 통신 서비스, 라우디 등



조 진 용

e-mail : jiny92@hotmail.com
1992년~1999년 전남대학교 컴퓨터 공학과
관심분야 : 컴퓨터 네트워크, 영상처리 등

임 정 균

e-mail : limxpark@logic.co.kr
1992년~1999년 전남대학교 컴퓨터 공학과
1999년~현재 LG정보통신
관심분야 : 컴퓨터네트워크, 멀티미디어 등