ATM 망에서 효율적인 TCP 폭주 제어 기법

최지현* · 김남희** · 김변곤*** · 전용일**** · 정경택*** · 전병실*

An Efficient TCP Congestion Control Scheme in ATM Networks

Jihyoun Choi* · Namhee Kim** · Byungon Kim*** · Yongil Jun**** · Kyungtaek Chung*** Byoungsil Chon*****

요 약

본 논문에서는 TCP에서의 트래픽 혼잡에 따른 영향을 최소화시키기 위하여 혼잡제어 윈도우 파라미터인 cwnd와 RTT을 이용한 향상된 TCP 폭주제어 알고리즘을 제안하였다. 제안된 기법에서는 혼잡의 주된 발생을 피할 수 있도록 하였고, 회복시간에 의해 발생하는 지연과 스위치 버퍼의 사용량을 감소시킬 수 있도록 하였다. 시뮬레이션을 통한 성능 평가결과 제안된 기법이 기존의 기법보다 성능이 우수함을 확인할 수 있었다. 버퍼효율에서는 기존의 기법보다 22.56%의 향상이 있었고, 패킷 드롭율은 약 0.1%로 기존의 알고리즘보다 적은 손실을 보여주었다.

ABSTRACT

In this paper, we proposed an enhanced TCP congestion control algorithm using RTT with congestion window parameter cwnd to minimize the effect of TCP congestion. The proposed scheme could avoid the occurrence of frequent congestion and decrease the delay caused by the recovery time and the using amount of switch buffer. Through the simulation, we showed that the proposed scheme can acquire higher performance than the existing scheme. There are 22.56% improvement at the average using buffer efficiency, and packet drop rate is 0.1% which is less than existing scheme.

keyword

ATM, congestion control, congestion window, TCP/IP

Ⅰ. 서 론

UBR 서비스 클래스는 전송망에 여분의 대역이 존재할 경우 여분 대역의 효율적인 사용을 위하여 ATM(Asynchronous Transfer Mode) Forum에서 제안된 서비스 클래스이다. 그러므로 UBR 클래스는 어떠한 대역이나 셀 전송율을 보장받지 못하므로 빈번한 셀 손실로 인한 사용자 터미널간의 재전송 기능이 가능한 애플리케이션에 유용한 서비스이다. ABR 서비스 클래스는 CBR과 VBR 서비스 클래스가 사용하고 남은 대역을 사용하며 전송 지연시간에 대해서는 민감하지 않으나 비교적 적은 셀 손실률을 필요로 하는 응용 서비스 간에 귀환 흐름제어 기법을 사용하여 전송 대역을 공평하고 효율적으로 분배하고자 하는 서비스 클래스이다[1-4]. 현재 TCP(Transmission Control Protocol) 분야에서의 주된 연구로 TCP가 가지는 특

*전북대학교 전자정보공학부 **군산대학교 컴퓨터정보과학과 ***군산대학교 전자정보공학부 ****한국전자통신연구원
접수일자 : 2003. 8. 1

성을 가능한 유지하고 기존의 TCP와의 호환성을 유지하며 이를 확장시키는 방안을 들 수 있는데 그 예로 TCP의 저 대역폭 서비스 문제를 해결하기 위해 옵션 필드를 확장하여 TCP 윈도우의 크기를 늘리거나, RTT(Round Trip Time) 측정의 정확도를 향상시키는 방법들을 들 수 있다[5,6]. 그 외 TCP의 프로토콜 기능 중 폭주 제어 기능을 정확히 수행하도록 함으로서 패킷 손실로 인한 성능 감소의 효과를 줄이려는 방법이 연구되고 있는데 이는 크게 두 가지로 나누어 생각할 수 있다[7]. 먼저 TCP의 폭주제어 기능을 강화시켜 통신망내의 폭주를 미리 예측하는 방법을 들 수 있고 다른 하나는 중간 노드들의 기능을 강화하여 TCP의 빈약한 폭주 제어 때문에 발생할 수 있는 TCP의 글로벌 동기화 문제를 해결하고 조기의 폭주 통지를 통해 불필요한 패킷의 손실이나 패킷 지연을 방지하는 것이다[8,9].

본 논문에서는 IP over ATM 망의 운용에서 서비스 품질을 보장하지 않는 "best effort" 방식인 ATM 클래스들 중 UBR 클래스를 사용하는 상황에서 TCP 계층에서의 흐름제어 시스템의 개선을 통해 네트워크의 수율을 향상시키기 위한 방안을 제안하였다. TCP에서 널리 사용되는 폭주제어방식은 slow-start와 congestion avoidance이다[9]. 제안된 알고리즘에서는 기존의 TCP의 폭주제어 방식에 폭주발생시의 RTT를 기억할 수 있도록 하고, 이를 다음 폭주상황 방지를 위한 파라미터로 사용한다. 초기 패킷 전송 후 목적지로부터 ACK가 오면 RTT를 알 수 있는데 네트워크에서 폭주가 발생하여 큐에 셀이 쌓이면 셀이 전송되는 시간이 늘어나며 자연적으로 RTT도 증가하게 된다. 제안된 방식에서는 패킷이 손실되어 재전송되면 이때 가장 긴 RTT 값을 기억하여 다음 전송부터는 RTT가 폭주발생시의 가장 긴 RTT 값을 넘지 않도록 제어하는 것에 기초한다. 이에 따라 네트워크 상에서 패킷의 버스트한 흐름에 따라 TCP의 폭주제어 메커니즘이 실행되어 계속적으로 발생하는 네트워크의 수율변화를 폭주상태에서의 상황을 기억하여 이후의 전송에서는 폭주상황 발생 이전에 이를 차단해 이에 따른 패킷 손실을 방지하여 안정적인 네트워크 수율을 확보할 수 있도록 하였다. 본 논문의 구성은 서론에 이어 제2장에서는 ATM망을 기반으로 한 TCP에서의 제안된 폭주제어기법에 대해 기술하였고, 3장에서는 시뮬레이션에 사용된 환경에 대해서 기술하였다. 그리고 4장에서는 결과를 통해 제안된 알고리즘의 성능을 평가·분석하고 마지막으로 5장에서 결론을 내렸다.

II. 제안된 TCP 폭주제어 기법

TCP의 타임아웃과 재전송은 연결에 따른 RTT 측정을 기본으로 한다. 따라서 TCP는 트래픽과 경로의 변화에 따른 시간변화를 감지하고 적절한 타임아웃 값을 유지해야 한다. 이를 위한 파라미터로 RTT 값을 사용하며 적당한 순서번호를 가진 바이트의 송신으로부터 ACK가 돌아올 때까지의 시간을 측정하게 된다. RTT의 지연은 링크지연과 패킷 처리 시간에 따른 패킷의 회신 시간과 큐잉 지연시간에 의해 결정된다. TCP 패킷의 버스트한 흐름은 스위치의 버퍼에 누적되는 데이터양을 급격하게 증가시키고 이를 처리하기 위한 큐잉 시간의 연장으로 인해 발생하는 전송 패킷의 손실을 유발한다. 기존의 알고리즘은 위와 같은 패킷의 손실과 전송상의 문제점으로 인해 야기되는 유실에 따른 다양한 효율감소 요인들을 비롯해 네트워크 내부의 폭주나 폭주상태를 상황의 발생 후 이를 처리하는 소극적인 제어방식임에 비해 제안된 알고리즘에서는 폭주발생시의 RTT를 기억하여 cwnd의 크기를 조절해 이를 다음 폭주상황 방지를 위한 파라미터로 사용하는 보다 적극적인 제어방식이라 할 수 있다. 제안된 알고리즘에서는 효율적인 네트워크 성능향상을 위하여 아래의 RTT 파라미터들을 사용한다.

(1) Congestion RTT : Congestion 발생시의 RTT를 의미하고 이의 재 발생을 방지하기 위한 파라미터이며 초기상태는 infinite
(2) Max RTT : 수신된 RTT중 최대값을 의미하며 재전송 타이머의 설정을 위해 일반적으로 사용되는 파라미터

(3) Current RTT : 현재 수신되는 RTT 값을 의미

다음은 네트워크의 효율향상을 위해 제안된 알고리즘의 TCP의 흐름상황에 따른 처리 규정이다.

(1) 패킷을 정상 수신했을 경우

```
TCPSender_ReceivePacket(Packet)
   current_RTT = Packet.RTT;
   max_RTT = max(current_RTT, max_RTT)
```

송신측으로부터 전송된 패킷을 해당 링크에 접속된 수신측에서 정상적으로 수신했을 경우 current_RTT 값을 수신된 패킷의 RTT 값으로 수정, 유지한다.

congestion_RTT와 재전송 타임아웃을 설정하기 위한 max_RTT 역시 수신된 패킷의 RTT와 비교하여 최대값으로 갱신한다.

(2) Slow-start 상태인 경우

```
if (current_RTT => cong_RTT)
   keep cwnd
else increase cwnd, ssthresh normally
```

Slow-start 상황에서는 수신된 현재 RTT 값을 Congestion_RTT 값과 비교하여 현재의 RTT 값이 congestion 발생시의 RTT 값 이상일 경우 cwnd의 증가를 방지한다. 그렇지 않은 경우는 기존의 알고리즘과 같이 처리한다.

(3) Congestion avoidance 상태인 경우

```
if (current_RTT => cong_RTT)
   keep cwnd
else increase cwnd, ssthresh normally
```

Slow-start의 경우와 마찬가지로 Congestion_RTT가 현재의 수신된 RTT 값보다 큰 경우에만 cwnd를 증가시킨다.

(4) Retransmit Timer out의 경우

```
set ssthresh = 1, cwnd = 1
set state to congestion avoidance
```

Timer out발생시 congestion avoidance를 실행하고 ssthresh와 cwnd를 초기화 하게 되며 네트워크 흐름 상태는 congestion avoidance로 설정된다.

(5) Three duplicate ACK가 수신된 경우

```
cong_RTT = max_RTT;
      max_RTT = 0;
reset max_RTT to base_RTT
```

세 개의 중복 ACK의 수신은 폭주상태를 의미한다. 따라서 폭주상태의 RTT를 기억하기 위한 congestion_RTT 값을 이때의 최대 RTT 값으로 갱신하고 Max_RTT 값을 초기화 한다. 초기 패킷 전송 후 목적지로부터 ACK가 오면 RTT를 알 수 있는데 네트워크에서 폭주가 발생하여 큐에 셀이 쌓이면 셀이 전송되는 시간이 늘어나며 자연적으로 RTT도 증가하게 된다. 제안된 방식에서는 위에서 설명한 흐름상황에 따른 처리 규정에 따라 패킷이 손실되어 재전송 되면 이때 가장 긴 RTT 값을 기억하여 다음 전송부터는 RTT가 폭주발생시의 가장 긴 RTT 값을 넘지 않도록 한다. 이에 따라 네트워크 상에서 패킷의 버스트한 흐름에 따라 TCP의 폭주제어 메커니즘이 실행되어 계속적으로 발생하는 네트워크의 전송률 변화를 폭주상태에서의 상황을 기억하여 이후의 전송에서는 폭주상황 발생 이전에 이를 방지하고 이에 따른 패킷 손실을 방지하여 안정적인 네트워크 수율을 확보할 수 있게 된다. 그림 1은 제안된 폭주제어 알고리즘의 의사코드이다.

```
state : TCP 연결의 상태(SlowStart or Congestion Avoidance)
cwnd : congestion window
ssthresh : slow start 임계치

/* TCP에서 패킷을 수신할 경우 */
TCPSender_ReceivePacket(Packet)
   // RTT를 갱신
      current_RTT = Packet.RTT;
   max_RTT = max(current_RTT, maxRTT);

   // 상태가 SlowStart인 경우
   if(state == SlowStart)
   {
      /* cong_RTT보다 current_RTT가 큰 경우 cwnd를 유지한다*/
      if (current_RTT => cong_RTT)
         keep cwnd; // cwnd를 유지
```

```
        else {
    duplicate cwnd, ssthresh normally
    if(cwnd => ssthresh) /* Congestion
    Avoidance로 상태 전이 */
    state = CongestionAvodance;
        }
    }
    /* 상태가 CongestionAvoidance인 경우 */
    else if(state == CongestionAvoidance)
    {
        /* cong_RTT보다 current_RTT가 큰 경우
    cwnd를 유지한다*/
          if (current_RTT => cong_RTT)
          keep cwnd
        else increase cwnd by 1, ssthresh normally;
    }
/* 재전송 타이머가 완료된 경우 */
OnRetransmitTimerOut
    ssthresh = 1;
      cwnd = 1;
    state = CongestionAvoidance;
       Retransmit packet; /* 손실된 패킷을 재전송
한다 */
/* 3번 중복된 Ack 신호를 받을 경우 */
Three Duplicate Ack
    cong_RTT = max_RTT;  // cong_RTT 갱신
      ssthresh = ssthresh/2;
      cwnd = ssthresh;
      max_RTT = .0;
      state = SlowStart;
    Retransmit packet; // 손실된 패킷 재전송
```

그림 1. 제안된 TCP 폭주제어 알고리즘의사코드
Fig. 1 Pseudocode of the proposed TCP congestion control algorithm

III. 시뮬레이션 환경

본 논문의 성능평가에 사용된 시뮬레이터는 객체기반 이산이벤트 시뮬레이터(object based discrete event simulator)로 ANSI/C++ 로 작성되었으며 다양한 설정을 기본으로 ATM망에서 TCP 트래픽을 시뮬레이션 하여 TCP 성능을 평가 할 수 있도록 하였다. 시뮬레이션 프로그램에 사용된 TCP 계층은 연결제어와 slow-start, 흐름 및 폭주 제어, 재전송, 빠른 재전송/빠른 복구 등과 같은 실제 TCP의 동작을 시뮬레이션 하여 실제상황과 근접한 결과를 얻을 수 있도록 구현되었으며 사용된 TCP의 버전은 Reno이다. 본 논문에서는 제안한 TCP 폭주제어 알고리즘의 순수한 성능평가를 위하여 기본적으로 ATM의 UBR클래스를 사용하며 EPD(Early Packet Discard), FIFO 시스템에 기반을 두어 성능평가를 하였다.

본 논문에서의 시뮬레이션 구성은 그림 2에서와 같이 ATM 카드를 장착한 워크스테이션들이 ATM 스위치와 직접 연결된 pear-to-pear 모델로 두 대의 ATM 스위치에 6대의 워크스테이션이 결합된 네트워크로 구성되어있다. 각 워크스테이션은 3개의 TCP가 연결되어 있고 각 TCP는 greedy 소스로 무한 크기의 파일을 전송할 수 있도록 하였다.

TCP의 MTU(Maximum Transfer Unit)는 기본 960 바이트로 20개의 ATM셀을 이루며, TCP의 시뮬레이션 파라미터는 표 1과 같다.

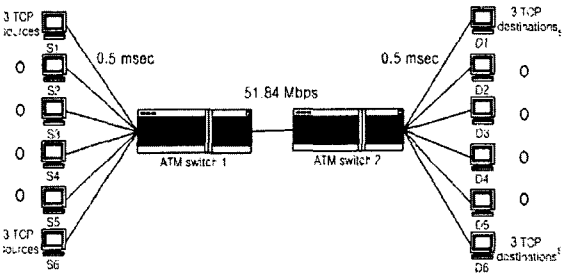

그림 2. 시뮬레이션 구성도
Fig. 2. Simulation Configuration

네트워크의 모든 링크는 SONET OC-1인 51.84Mbps이며 ATM 스위치간의 지연시간은 4/9/14/19msec이고, 워크스테이션과 ATM 스위치간의 지연시간은 0.5msec이다. 그림 2의 왼쪽에 위치한 워크스테이션들의 TCP 연결은 송신원이고, 오른쪽에 대응하는 워크스테이션의 TCP 연결은 수신원이고 각 워크스테이션 쌍은 하나의 UBR VC로 연결되어 있다.

표 1. 시뮬레이션 파라미터
Table 1. Simulation parameter

Parameter	Value
Cell discarding type in switch	EPD
EPD threshold	400
Buffer size	500
Window Size	65,535 bytes
Retransmission Timer	50 msec
Maximum Segment Size	960 bytes
File size	infinite(∞)

IV. 성능 평가

그림 3은 TCP 연결의 시간에 따른 기존 알고리즘의 폭주 윈도우의 추이를 보여준다. ATM 스위치에서 패킷 손실이 일어날 때 마다 TCP는 폭주제어 메커니즘을 수행하고, 폭주 윈도우의 크기를 줄이고 다시 폭주 윈도우의 크기를 점진적으로 늘리는 과정을 반복하므로 폭주 윈도우는 톱니 모양의 진행 패턴을 보인다.

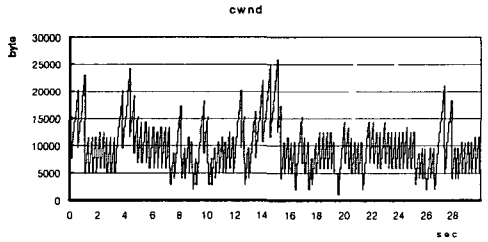

그림 3. 기존 알고리즘의 cwnd 추이
Fig. 3 The cwnd transition of existing algorithm

반면 그림 4의 제안된 알고리즘이 보여주는 폭주 윈도우의 추이를 보면 네트워크에서 급격한 전송량 증가에 따른 폭주상황이 발생한 경우, 이때의 RTT를 기억하여 다음 폭주 상태 직전에 폭주 윈도우의 증가를 방지하므로 폭주윈도우가 증가함으로서 급격히 커질 수 있는 전송량을 사전에 방지하여 재 폭주 상황을 감소시키므로 시간의 흐름에 따라 안정적인 폭주 윈도우의 상태를 찾아감을 확인 할 수 있다. 예로 보여진 폭주 윈도우의 추이는 양측에서 평균치에 근사한 결과를 보여준 소스 중 선정하였다.

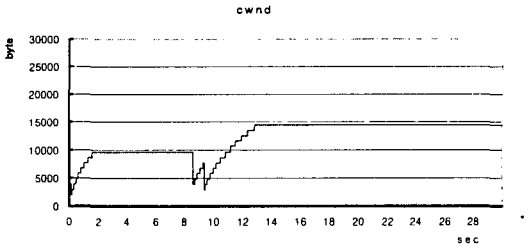

그림 4. 제안된 알고리즘의 cwnd 추이
Fig. 4 The cwnd transition of the proposed algorithm

그림 5와 그림 6은 ATM 스위치 버퍼에서의 추이를 나타낸 그림으로 15ms이상의 스위치 간 지연시간에서 버퍼의 점유 상태는 데이터의 전송과 처리시간의 전체적인 지연시간의 상승에 따른 결과로 매우 조밀하지 못한 모습을 보여주는데 제안된 알고리즘은 이 경우에 폭주상태 재 발생 방지 기능이 동작함으로서, 기존의 알고리즘에 비해 복구시간이 길어짐에 따른 비효율적인 버퍼 낭비 현상을 어느 정도 완화시켜 주고 있음을 알 수 있다.

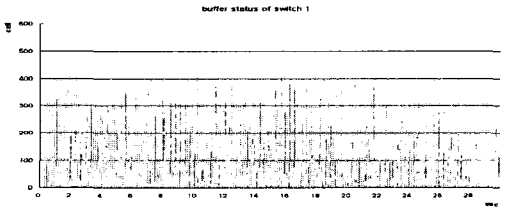

그림 5. 기존 알고리즘의 스위치 버퍼의 추이(delay = 15ms)
Fig. 5 The transition of switch buffer of existing algorithm

표 2와 표 3은 버퍼의 효율을 나타낸 것으로 시뮬레이션에서 사용된 ATM스위치의 버퍼는 500 cell의 용량을 갖고 있다. 표 2는 1ms 간격마다 측정한 버퍼의 점유 추이를 기록하여 이를 단위시간으로 나눈 값을 표시한 것이다. 모든 스위치 간 지연시간에서 제안된 알고리즘은 기존의 알고리즘보다 우수한 평균 버퍼 점유량을 보여주는데 평균값의 경우, 기존의 알고리즘이 매초마다 500의 버퍼 용량 중 169.4481의 점유량을 보이는데 반해 제안된 알고리즘은 50%를 상회하는 점유량을 보여주고 있다.

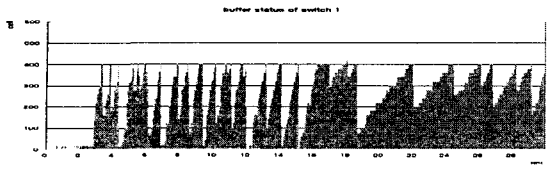

그림 6. 제안된 알고리즘의 스위치 버퍼의 추이(delay = 15ms)

Fig. 6 The transition of switch buffer of the proposed algorithm

표 2. 평균 버퍼 점유량

Table 2. Average buffer occupancy

delay	기 존	제 안
5ms	254.6249	363.2307
10ms	182.7859	320.4337
15ms	137.4517	254.1773
20ms	102.9264	191.2083
평 균	169.4481	282.2625

표 3은 단위시간당 버퍼의 점유량을 백분율의 이용효율로 나타내어 제안된 알고리즘과 기존 알고리즘의 버퍼 효율의 차이를 보여준다. 제안된 알고리즘은 평가된 모든 스위치 간 지연시간의 실험에서 기존의 알고리즘보다 평균값에서 22.56%의 높은 버퍼 이용율을 보여준다.

표 3. 버퍼 이용 효율

Table 3. Buffer utilization efficiency

delay	기 존	제 안	백분율 차
5ms	50.92%	72.65%	21.73%
10ms	36.56%	64.09%	27.53%
15ms	27.49%	50.84%	22.99%
20ms	20.59%	38.24%	17.65%
평 균	33.89%	56.45%	22.56%

표 4는 기존 알고리즘과 제안된 알고리즘의 패킷 드롭율을 보여준다. 스위치 간 지연시간별로 종단의 각 VC의 링크마다 측정한 값을 종합하여 백분율로 표현하였는데 기존의 알고리즘은 모든 지연시간별 성능평가에서 대략 0.1%의 매우 적은 드롭율을 보인다.

표 5는 제안된 알고리즘과 기존 알고리즘의 스위치 간 지연시간별 TCP goodput을 나타낸다. 5ms의 지연시간에서 1.06Mbps, 10ms의 지연시간에서는 3Mbps, 15ms와 20ms의 지연시간에서는 4Mbps에 달하는 큰 차이를 보이고 있다. 지연시간이 길어짐에 따라 네트워크 스위치의 처리 시간도 상승하게 되는데 이는 전체 네트워크의 응답 지연을 야기하게 된다. 따라서 성능 평가에서 보이는 바와 같이 지연시간이 연장됨에 따라 네트워크의 전체 수율 역시 점차적으로 감소하게 되는데 제안된 알고리즘은 지연시간이 길어져 전체 네트워크 성능이 저하되는 경우에도 효과적으로 이를 완화하여 주므로 기존 알고리즘에 비해 효율의 감소가 적은 모습을 보여준다.

표 4. 패킷 드롭율

Table 4. Packet drop rate

delay	기 존	제 안
5ms	5.08739%	0.11568%
10ms	2.22520%	0.10509%
15ms	1.69941%	0.09212%
20ms	1.12751%	0.07981%

표 5. 전체 수율

Table. 5 Total throughput

delay	기 존	제 안
5ms	4.57505E+07	4.68096E+07
10ms	4.33674E+07	4.63727E+07
15ms	4.13545E+07	4.54574E+07
20ms	3.94122E+07	4.35044E+07

V. 결 론

본 논문에서는 TCP의 폭주처리 상황에서 RTT 파라미터를 이용해 폭주윈도우의 크기를 조절함으로서 기존의 알고리즘에서 단순히 ACK가 도착할 때마다 지수적으로 증가하였던 폭주윈도우의 크기를 폭주발생시 기억된 RTT에 따른 제어기법으로, 폭주의 재 발생을 방지하여 폭주발생과 복구를 위해 소비되는 전송지연과 스위치 버퍼이용률의 감소를 완화시켰다. 시뮬레이션 결과에서 보

여 진 바와 같이 기존의 알고리즘이 계속적인 폭주발생에 따른 제어에 의해 톱니모양의 폭주윈도우 변화를 보여주었던 것에 비해 제안된 알고리즘은 시간이 흐름에 따라 점차 안정적인 폭주 윈도우 크기를 유지함을 볼 수 있었으며, 스위치에서의 각 지연시간별 버퍼 추이와 이용효율에서도 효율에서 평균 22.56% 향상되는 등 기존의 알고리즘보다 효율적인 결과를 보여주었다.

참고 문헌

[1] M. Hassan, M. Atiquzzaman, Performance of TCP/IP over ATM networks. Artech House, 2000.

[2] The ATM Forum Technical Committee, "Draft TM 4.1 Traffic Management Specification," ATM Forum/BTD-TM-02.02, Dec. 1998.

[3] S. Savage et al., "TCP Congestion Control with a Misbehaving Receiver," ACM Comp. Commun Rev, vol. 29, no. 5, pp. 71-78, Oct. 1999.

[4] M. Allman, H. Balakrishnan, S. Floyd, "Enhancing TCP's Loss Recovery using Limited Transmit," RFC 3042, Jan. 2001.

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

[6] Goyal, R.; Jain, R.; Fahmy, S.; Vandalore, B. Kalyanaraman, S. "Design issues for providing minimum rate guarantees to the ATM unspecified bit rate service," ATM Workshop Proceedings, IEEE, pp. 169-175, 1998.

[7] S. Sathaye, "ATM Forum Traffic Management Specification, Version 4.0," ATM Forum Technical Committee, Mar. 1996.

[8] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno and SACK TCP," ACM Computer Communication Review, pp 5-21, July 1996.

[9] W. Stevens, "TCP slow start, congestion avoidance, fast retrasmit and fast recovery algorithms," Internet RFC 2001, Jan. 1997.

[10] 최지현, "패킷 네트워크의 효율향상을 위한 개선된 TCP 흐름제어 알고리즘", 석사학위논문, 2003.

저자 소개

최지현(Jihyoun Choi)

2001년 2월 : 전북대학교 전자공학사
2003년 2월 : 전북대학교 전자정보공학부 석사
※ 관심분야 : 교환 및 전송, 무선 LAN

김변곤(Byungon Kim)

1990년 2월 : 항공대학교 전자공학사
1997년 2월 : 전북대학교 전자공학석사
2001년 8월 : 전북대학교 전자공학박사
2001년 8월 - 현재 : 군산대학교 전자정보공학부 계약교수
※ 관심분야 : 교환 및 전송, ATM 트래픽제어

김남희(Namhee Kim)

1992년 2월 : 군산대학교 정보통신공학사
1994년 2월 : 전북대학교 전자공학석사
1997년 8월 : 전북대학교 전자공학박사
2002.2 - 현재 : 군산대학교 컴퓨터정보과학과 전임강사
※ 관심분야 : 컴퓨터네트워크, ATM 트래픽제어

전용일(Yongil Jun)

1981년 - 현재 : 한국전자통신연구원, 책임연구원
※ 관심분야 : 홈 네트워크, 무선 LAN, 임베디드시스템

정경택(Kyungtaek Chung)

1982년 : 전북대학교 전자공학사
1984년 : 전북대학교 전자공학석사
1992년 : 전북대학교 전자공학박사
1990년 - 현재 : 군산대학교 전자정보공학부 교수
※ 관심분야 : 홈 네트워크, 무선 LAN, 임베디드시스템

전병실(Byoungsil Chon)

1967년 2월 : 전북대학교 전자공학사
1970년 8월 : 전북대학교 전자공학석사
1976년 8월 : 전북대학교 전자공학박사
1971년 - 현재 : 전북대학교 전자정보공학부 교수
1998년 - 2000년 : 전북대학교 학생처장
※ 관심분야 : 교환 및 전송, 무선 LAN, ATM