Ad-hoc 네트워크에서 지연 ACK를 이용한 TCP 성능 향상에 관한 연구

박광채* · 나동건**

A Study on Delayed ACK Scheme for TCP Traffic in Ad-hoc Network

KwangChae Park* · Donggeon Na**

요 약

Ad-hoc 네트워크는 무선 링크들로 구성된 멀티-홉 네트워크이다. 그리고 무선 링크의 전송 특성은 유선에 비해 보다 불안정한 상태이다. 그러므로 Ad-hoc 네트워크에서는 패킷 손실이 자주 발생하고, 패킷 손실에 의한 TCP 연결 실패는 TCP 성능을 심각하게 저하시킨다. 또한, TCP 성능 저하는 무선채널에서 데이터 패킷과 ACK 패킷의 충돌에 의해서 발생한다. 본 논문에서는 ODA(Ordering-Delayed ACK)지연 알고리즘을 제안하고, ODA 알고리즘과 지연 ACK 알고리즘을 이용하여 무선 Ad-hoc 네트워크의 성능을 향상시키고자 한다. 제안된 ODA 알고리즘은 무선 Ad-hoc 네트워크에서의 수신 측에서 데이터 패킷의 수를 순서적으로 증가시킨다. 본 논문에서는 NS-2를 이용하여 컴퓨터 시뮬레이션을 수행하였다. 시뮬레이션 결과, 제안된 ODA 알고리즘은 기존의 지연 ACK 알고리즘에 비해서 무선 멀티-홉 Ad-hoc 네트워크 환경에서 채널 용량이 증가되고 네트워크의 성능이 향상됨을 확인할 수 있었다.

ABSTRACT

An ad hoc network is multi-hop network composed of radio links, and the transmission quality of a radio link is more unstable than that of a wired circuit. Packet loss thus occurs frequently in an ad hoc network, and the consequent connection failure results in a severe deterioration of TCP performance. TCP performance also deteriorates because of the collision of data packets and ACK packets in the radio channel. In this paper we study to improve the performance of the Mobile Ad-hoc network by using Delayed ACK algorithm with our proposed ODA(Ordering-Delayed ACK) method. The proposed ODA algorithm increases the number of the data packets orderly at the receiver side which is going to be applied for the Mobile Ad-hoc network. We accomplished a computer simulation using NS-2. From the simulation results, we find the proposed ODA algorithm obviously enlarge the channel capacity and improve the network performance at the situation of multi-hop of ad-hoc network than the existing Delayed ACK algorithm.

키워드

Ad-hoc, TCP, Delayed ACK

Ⅰ. 서 론

Mobile Ad-hoc 네트워크는 BS(Base Station) 혹은 AP(Access Point)에 의한 중앙 집중화된 관리나 표준화된 지원 서비스 없이 임시 망을 구성하는 무선 이동단말기들의 집합으로, 그 특성상 임시 구성 망이나 재해, 재난 지역

* 광운대학교 대학원 전자통신공학과

** 전북대학교 전자공학과

접수일자 : 2006. 7. 13

이나 전쟁터와 같이 기존의 기반 시설을 이용할 수 없는 환경에 적응하는 것으로 인식되어 왔다. 그 대표적인 예로 미국의 DARPA에서 추진해온 GloMo 프로그램을 들 수 있다.

IEEE 802.11을 사용하는 Ad-hoc 네트워크는 CSMA/CA 방식을 사용하며[1], 전송을 원하는 노드는 RTS 메시지를 전송하기를 원하는 노드에게 전달하고, RTS를 받은 노드는 주변의 모든 노드에게 CTS를 보내어 주위의 다른 노드들의 전송을 막도록 한다. 이를 통하여 데이터의 충돌은 많은 부분 경감되지만 데이터 전송에 참여하지 않는, 즉 CTS를 받은 노드는 데이터를 보내지도 받지도 못하기 때문에 이것 또한 가용 대역폭을 낭비하게 된다. 이를 알려진 노드 문제(exposed node problem)이라 부른다[2].

Ad-hoc 네트워크와 같은 경쟁 기반의 환경에서는 전송 트래픽의 전송 효율을 최대화하기 위해서, 전송 트래픽의 오버헤드를 최소화하는 것이 중요하다. TCP 환경의 전송 프로토콜에서 트래픽중 작은 크기의 데이터를 갖는 ACK 패킷을 중요한 컨트롤 패킷이라 할 수 있다. 하지만 이러한 ACK 패킷도 Ad-hoc 환경에서는 전송을 위해 채널을 사용하여야 하고, 다른 트래픽과 동일하게 경쟁하여야 한다. 또한 유선환경의 TCP는 self- clocking을 지키기 위하여 가능한 한 많은 수의 ACK 패킷을 보내어 전송 데이터의 전송 페이스를 조절한다. 그러므로 Ad-hoc 네트워크 환경에서 ACK 패킷을 줄이는 것은 매우 중요한 일이다. 이와 같은 문제점을 해결하는 방법으로 Ad-hoc 네트워크에서의 ACK 패킷을 줄이는 알고리즘 연구가 현재 진행되고 있으며 이 중 Ad-hoc 네트워크에 가장 적합하다고 할 수 있는 Delayed ACK(RFC 2581) 알고리즘에 대해 많은 연구가 이루어지고 있다.

Delayed ACK는 TCP의 수신측에서 보내는 ACK 패킷의 컨트롤에 대한 알고리즘이다. 최대 사이즈의 세그먼트를 가진 두 번째의 데이터 패킷을 받을 때마다 하나의 ACK 패킷을 송신측에 보내고, 수신 측에서 수신한 세그먼트에 대한 ACK를 보낼 때 ACK 외에 수신측 자신이 보낼 데이터가 있을 경우 ACK+데이터를 한 패킷에 같이 실어서 보내기 위한 방법이다. 때문에 수신측에서는 세그먼트를 받으면 바로 ACK를 전송하지 않고 일정 시간(보통 200ms)을 기다린 후 그 시간 내에 전송할 데이터가 발생하면 ACK와 함께 보내고 그렇지 않고 타임아웃이 발생하면 ACK만 전송하게 된다. 하지만, delayed ACK 알고리즘의 단점은 타임아웃이 200ms의 배수마다 계속 발생한다는 점에 있다. 또한 일반적인 delayed ACK의 알고리즘은 데이터 패킷을 2개 받았거나 수신측의 타이머에 의해 하나의 ACK 패킷을 보내므로 이 또한 가용 CW (Congestion Window)를 최대한 활용하지 못하는 단점을 가지고 있다.

따라서 본 논문에서는 수신측에서 ACK 신호를 보내는 시점을 송신측으로부터 받은 데이터 패킷의 수를 순차적으로 증가시킴으로써 Ad-hoc 네트워크의 효율을 높이고자 한다. 이와 같이 무선 환경에 맞게 연구되어진 delayed ACK 알고리즘에서 수신 데이터 패킷의 수를 순차적으로 증가시켜 컨트롤 패킷인 ACK 패킷의 전송수를 줄임으로써 채널간의 경쟁이나 충돌을 감소시켜 Ad-hoc 네트워크에서의 TCP 전송률을 높이기 위한 Ordering-Delayed ACK(ODA) 알고리즘을 제안한다.

본 논문의 구성은 2장에서 Ad-hoc 네트워크에서의 TCP 성능향상을 위한 관련 알고리즘에 대하여 기술하고, 제 3장에서 제안된 알고리즘에 대해 설명하고, 4장에서는 컴퓨터 시뮬레이션을 통해 제안된 알고리즘의 성능을 평가 및 분석하고, 마지막으로 제 5장에서 결론을 맺는다.

Ⅱ. Ad-hoc Network에서의 TCP

Ad-hoc 네트워크가 IP 기반 프로토콜 상으로 운용된다고 할 때 대부분의 IP 어플리케이션들은 TCP상에서 수행된다. Ad-hoc 네트워크에서는 토폴로지의 동적 변화로 인하여 트래픽의 지연이나 유실이 유선 네트워크에 비해 크기 때문에 일반 유선 네트워크에 사용되는 TCP를 Ad-hoc 네트워크에 사용할 경우 서비스의 안정적인 제공 및 네트워크 전체 성능이 저하되는 문제점이 있다. 그러므로 Ad-hoc 네트워크에 적합한 TCP 의 설계가 주요 연구 대상의 하나가 되고 있다. 특히 무선 링크에서의 높은 BER과 이동성에 따른 잦은 연결 절단(dis-connection) 및 핸드오프(hand-off)등은 연결 기반 TCP 패킷의 손실과 재전송을 유발하여 네트워크의 부하로 작용하며 이와 동시에 TCP 성능을 떨어뜨린다. 이를 해결하기 위한 방법으로는 I-TCP(Indirect-TCP), Fast Retransmit, Snoop, M-TCP 등의 wireless TCP를 이용하여 TCP를 개선하는 방법이 연구되고 있으며, 무엇보다도 TCP 성능에 영향을 주는 링크 계층의 안정화와 좋은 라우팅 프로토콜의 개발이 선행되어야 한다.

단일 홉 무선 네트워크에 비하여 멀티 홉 Ad-hoc 네트워크에서의 TCP는 end-to-end 네트워크에서 infrastructureless, self-organizing 멀티-홉 네트워크, 중앙 네트워크 메니지먼트의 부족 등의 심각한 문제를 안고 있다. 그러나 많은 해결 메커니즘은 infrastructure- based 네트워크에서 혼잡에 의한 손실과 에러에 의한 손실을 구별할 수 있는 기지국 기반 하에 연구/개발되었으나, 모바일 Ad-hoc 네트워크가 infrastructure를 가지고 있지 않기 때문에 그 메커니즘이 모바일 Ad-hoc 네트워크에 직접적으로 사용되기에는 무리가 있다.

최근 모바일 Ad-hoc 네트워크 상에서 TCP의 성능을 향상시키기 위한 여러 가지 방법들이 제안되었으며, 네트워크 feedback 정보를 이용한 feedback scheme을 가지는 TCP[3,4]와 feedback scheme을 가지지 않는 TCP[5,6] 그리고 lower layer enhancement scheme을 가지는 TCP 등의 세 가지 그룹으로 분류할 수 있다.

2.1. Delayed ACK Algorithm

RFC 2581에서 지정된 delayed ACK 알고리즘은 유선환경에서 송·수신단의 프로세싱 부하를 줄이기 위한 방법으로 제안되었다. 이것은 ACK 패킷을 보내기 위해 최소한 매번 2개의 최대크기 데이터 패킷을 받았을 때와 그렇지 않더라도 데이터를 받은 지 500ms 이내에는 반드시 ACK 패킷을 보내야하는 것을 말하며, 수신 측의 delay timer에 의해 200ms 동안 데이터 패킷을 수신하지 못하는 경우와 데이터 패킷의 순서가 바뀌어서 도착하는 경우에는 패킷손실 복구를 빨리 할 수 있도록 매번 도착하는 패킷에 대한 ACK 패킷을 보내주어야 한다[7]. 하지만, 수신측에서 delayed ACK 방법에 따라 매번 오는 데이터 패킷에 대해 ACK 패킷을 보내주지 않는 경우에는 ACK 패킷을 보내주는 경우보다 송신측은 다음과 같은 면에서 항상 손해를 보게 된다.

1) Slow-Start 구간의 시작에서 항상 ACK time out이 발생하게 된다. 이것은 송신자의 CWnd가 1이어서 수신자는 2개의 데이터 패킷을 받을 때까지 ACK 패킷을 전송하지 않고 기다리기 때문에 타임아웃이 발생하게 되는 것이다[8]. 이 문제는 초기 CWnd 값을 1로 하지 않고 RFC 2414에서 제안한 것과 같이 설정해주면 해결할 수 있다[9].

2) Slow-Start 구간에서 윈도우의 크기가 증가하는 정도가 한 RTT 당 2배씩 증가하지 않고 1.5배씩 증가하게 되는 문제가 있다. 이것은 TCP 수신측에서 받은 ACK 패킷의 개수를 바탕으로 윈도우의 크기를 증가시키기 때문이다[10]. 이것은 M. Allman이 제안한 LBC나 ABC를 사용함으로써 네트워크에 bursty한 데이터 패킷들을 보내는 것을 방지하면서도 Slow-Start 구간에서 매번 ACK 패킷을 받는 것과 비슷하게 성능을 개선할 수 있다[11].

3) Congestion avoidance phase에서 1 RTT에 한 패킷 단위만큼 CWnd를 증가시킬 수 없다. 만일 2개의 데이터 패킷마다 1개의 ACK를 보내주는 경우라면 2 RTT가 걸려야 CWnd를 1만큼 증가시킬 수 있다. 이것은 데이터 송신자가 ACK 패킷을 한 개 받을 때마다 CWnd를 1/ CWnd 만큼씩 증가시키기 때문이다.

4) TCP 송신자의 데이터 전송을 bursty하게 만들어 엑세스 네트워크 라우터의 버퍼 용량이 작은 경우 버퍼 오버플로우를 발생시킬 수 있다[12].

유선환경에서의 delayed ACK 알고리즘은 채널을 공유하는 Ad-hoc 네트워크에서 ACK 패킷을 줄임으로써 TCP 성능 향상에 많은 도움이 될 것이다. Ad-hoc 네트워크의 무선채널에서 데이터 패킷과 ACK 패킷의 충돌은 TCP 성능을 감소시키는 원인이 된다. 이에 Ad-hoc 네트워크에서는 ACK 패킷을 선별적으로 전송하는 delayed ACK 알고리즘을 사용하여 ACK 패킷의 전송을 줄임으로써 ACK 패킷을 전송하기 위한 네트워크 자원의 낭비를 줄일 수 있다. 그림 2.1은 delayed ACK 알고리즘의 개념을 보여주고 있다.

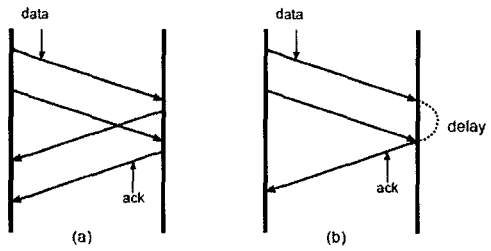

그림 2.1 Delayed ACK에서의 ACK 전송
Fig. 2.1 Transmitted ACK for delayed ACK.

Ⅲ. 제안된 Delayed ACK 알고리즘

Ad-hoc 네트워크에서 서로 공유하는 채널의 경쟁과 데이터 패킷과 ACK 패킷의 충돌을 최소화하여 TCP 전송률을 높이기 위한 방법으로 delayed ACK 알고리즘을 이용

하였다. 즉 TCP 수신측은 일정한 수의 데이터 패킷을 받았을 때 ACK 패킷을 송신 측에 보냄으로써 데이터 패킷에 비해 크기가 작은 제어 패킷인 ACK 패킷의 수를 줄여 채널간의 경쟁을 줄이고, ACK 패킷의 전송으로 인한 채널소모를 줄임으로써 TCP 성능을 향상시킬 수 있다. 하지만 유선환경에서 제안되어진 delayed ACK 알고리즘은 데이터 패킷의 수를 항상 2(d=2)로 고정해 놓았으며, 이러한 유선환경의 delayed ACK 알고리즘은 무선환경에 적용시키기에는 한계가 있다. 또한 Slow-Start 구간에서 항상 ACK time-out이 발생하게 되며 이것은 송신 측의 CWnd의 크기가 1이고 수신측은 2개의 데이터 패킷을 받을 때까지 ACK를 전송하지 않고 기다리기 때문에 발생하는 time-out을 회피할 수 없다. 매번 2개의 데이터 패킷을 받은 후 ACK 패킷을 전송해야 하는 delayed ACK 알고리즘은 데이터 패킷의 수가 증가함에 따라 채널에서의 충돌은 데이터 패킷의 수에 비례하여 발생함으로써 TCP 성능향상에 커다란 저해가 된다. 이러한 이유로 본 논문에서는 delayed ACK 알고리즘의 데이터 패킷의 수를 순차적으로 늘려감에 따라 Ad-hoc 네트워크에서의 채널 간 경쟁과 충돌을 줄이고 제어 패킷으로 인한 채널 부족현상과 더불어 전력 소비량을 줄일 수 있다.

본 논문에서는 Ad-hoc 네트워크에서 TCP 전송효율을 높이기 위해서 delayed ACK의 기본 알고리즘을 이용하여 수신 측에서 데이터 패킷의 수를 순차적으로 증가하여 ACK 패킷을 보내는 방법을 Ad-hoc 네트워크에 적용하는 ODA 알고리즘을 제안한다.

3.1. Ordering-Delayed ACK

앞에서 언급했던 바와 같이 delayed ACK에서는 데이터 패킷수를 고정 시켰을 때 발생하는 채널간의 경쟁과 충돌이 발생하여 해당 TCP 연결의 전송률이 급격히 감소된다. 따라서 제안된 알고리즘에서는 데이터 패킷의 완벽한 전송 후 수신측이 받아야 할 데이터의 수를 counter를 이용하여 순차적으로 증가시켜 설정함으로써 ACK 패킷의 전송을 줄이고 망 내에서 채널간의 경쟁과 충돌을 최소화하여 전송률을 높이도록 한다. 그림 3.1은 TCP 수신측에서의 Ordering-Delayed ACK 설정 흐름도를 보여준다.

아래의 의사코드는 제안된 알고리즘의 데이터 패킷의 수를 순차적으로 늘리는 설정에 대한 프로시져를 보여준다.

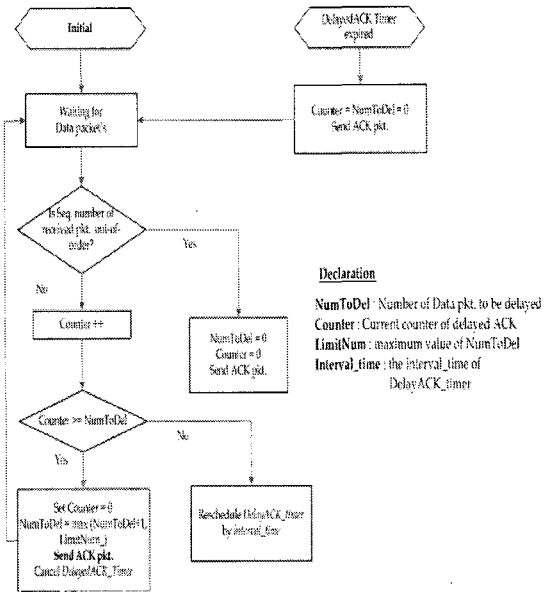

그림 3.1 제안된 ordering-dealyed ACK 알고리즘 순서도

Fig. 3.1 Flow chart of proposed ordering-delayed ACK

Case 1 : 수신 측에서 정상적인 순서에 맞추어 데이터 패킷을 수신하였을 경우, 수신 측에서는 count_ 변수를 하나씩 증가하여 수신측이 받아야할 데이터 패킷의 수를 설정한다. 또한 count_ 변수의 크기가 수신측에서 받아야 할 데이터 패킷 수의 크기보다 크거나 같다면 count_를 0으로 셋팅후 NumToDel_ 변수의 크기를 하나씩 증가시켜 최대 LimitNum_ 변수에 다다르게 되면 패킷수와 상관없이 LimitNum_ 변수값에 해당하는 데이터 패킷을 받았을 때 ACK 패킷을 전송하게 된다.

◆ 정상적으로 데이터 패킷을 수신했을 경우

```
if(th->seqno()==acker_->Seqno() ) {
    if (count_<NumToDel_) {
        save_ = pkt;
        delay_timer_.resched(interval_);
        count_ ++;
    } else {
        count_ = 0;
        NumToDel_ = max (NumToDel_ + 1,
LimitNum_ );
        ack(pkt);
}
```

declarations :

NumToDel_ : Number of Data pkt. to be delayed
Counter_ : Current counter of delayed ACK
LimitNum_ : maximum value of NumToDel
Interval_time_ : the interval_time of DelayACK_timer
MaxCWnd_ : limit increase maximum window size

Case 2 : 수신측에서 받은 ACK 패킷 sequence가 정상적이지 않을 경우 즉 수신된 데이터 패킷의 sequence 번호의 순서가 정상적이지 않을 경우, 전송중 패킷이 손실되었거나 패킷의 순서가 바뀐 것으로 판단하여 count_ 변수는 0으로 셋팅한 후 즉시 ACK 패킷을 보내어 송신측의 재전송 메커니즘을 이용하여 재전송하게 된다.

◆ Sequence 번호가 Out-of-Order 상황일 경우

```
if (th->seqno() != acker_->Seqno()) {
    count_ = NumToDel_ = 0;
    ack(pkt);
    if (delay_timer_.status() == TIMER_PENDING)
        delay_timer_.cancel();
}
```

Case 3 : 수신 측에서 데이터 패킷을 일정시간 받지 못했을 경우 delayed ACK 타이머를 작동시켜 200ms 동안 데이터 패킷을 수신하지 못하면 패킷 손실로 간주하여 송신측에 ACK 패킷을 보내며, count_변수와 NumToDel_ 변수를 0으로 셋팅하여 초기화한다.

◆ 수신측의 delayed ACK 타이머가 만료되었을 경우

```
void DelAckSink::timeout(int) {
    if ( save_ != NULL ) ack(pkt);
    count_ = NumToDel_ = 0;
}
```

Ⅳ. 시뮬레이션 결과 및 성능평가

4.1. 시뮬레이션 환경

본 논문에서 제안된 Ordering-Delayed ACK 알고리즘의 성능을 검증하기 위하여 캘리포니아 버클리 대학에서 개발된 NS(Network Simulator) 2.27을 사용하였다. ODA 알고리즘을 지원하기 위하여 C++로 작성된 TCP sink code에서 DelAckSink를 수정하였다.

라우터의 버퍼 관리 알고리즘은 AODV를 사용하였다. 비교를 위해서 표준의 TCP, 기존의 delayed ACK알고리즘, 변형된 delayed ACK 알고리즘 그리고 제안된 ODA 알고리즘의 성능 평가를 TCP Reno 환경에서 수행하였다. 성능 평가는 각각의 모델에서 홉 수의 증가에 따른 전송률, 시뮬레이션의 변수에 따른 전송률 그리고 CWnd 크기에 대해 성능평가를 하였다.

그림 4.1와 표4.1은 시뮬레이션에 사용된 Ad-hoc 네트워크 모델과 시뮬레이션 환경 파라미터를 보여준다. 네트워크 내의 노드와 노드 사이의 근접 거리는 200m, 각 노드의 전송 거리는 250m이며 각 노드의 간섭거리는 550m이다. 또한 하나의 FTP 연결로 첫 번째 노드에서 마지막 노드까지 전송되어 이어지는 chain-node 이다.

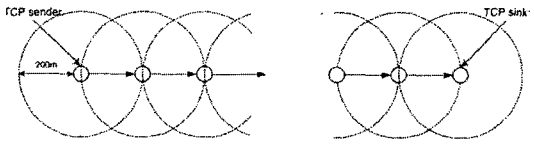

그림 4.1 시뮬레이션 네트워크 모델
Fig. 4.1 Network model for simulation.

표 4.1 시뮬레이션 파라미터
Table 4.1 Initial simulation parameter.

Parameter	Value
TCP Version	TCP Reno
Routing Protocol	AODV
MaxCWnd	7, 50
LimitNum	4, 6
Packet Size	1000 byte
Simulation Time	500s
DelayedAck Timer	200ms

4.2. 제안된 알고리즘의 성능평가

제안된 알고리즘의 성능평가를 위해 홉 수를 늘려가며 시뮬레이션을 수행하였다. 성능비교를 위해 Limit Num_ 변수값과 MaxCWnd의 크기를 변화시켜가며 시뮬레이션 시간 동안 홉 수에 따른 전송률에 대한 추이를 측정하였다. 전송률은 전송된 패킷의 크기 / 전송시간 * 전송 패킷 수로 구할 수 있다.

가. LimitNum 크기의 변화에 따른 전송률

그림 4.2와 4.3은 LimitNum 변수를 4와 6으로 변화시키며 시뮬레이션한 결과이다. 결과에서 보여주듯이 수신측의 ACK 패킷을 보내야 하는 최대 수신 패킷 수를 4와 6일 때로 변화시키며 시뮬레이션 한 결과, 적은 홉 수에서의 전송률의 차이는 LimitNum 가 6일때 성능이 좋지만 홉 수가 증가하면서 LimitNum 가 4일때 더 좋은 전송률을 나타낸다.

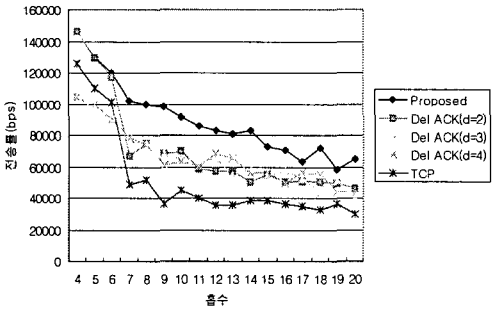

그림 4.2 LimitNum = 4일 때의 전송률
Fig. 4.2 Transmitted rates for LimitNum_= 4.

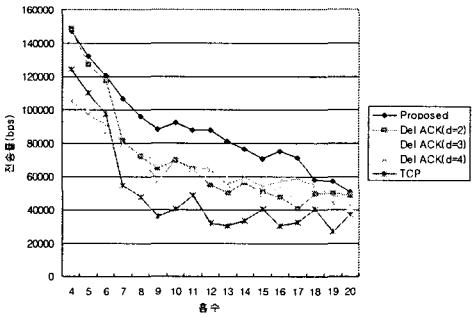

그림 4.3 LimitNum = 6일 때의 전송률
Fig. 4.3 Transmitted rates for LimitNum = 6.

이는 홉 수가 증가함에 따라 송신측의 데이터 패킷의 수도 증가하여 송신측에서의 RTO 발생 횟수와 수신측의 delayed timer에 의한 패킷 손실이 증가함을 알 수 있다. 또한 홉 수가 증가함에 따라 전송률의 감소는 노드들의 간섭 거리에 의한 간섭 영역을 벗어난 4홉 이상에서는 3홉 이하와는 달리 pipeline 효과가 발생하기 때문에 전송률이 높아짐을 알 수 있다. 하지만, 기존의 알고리즘은 수신측의 수신 패킷 수를 고정시킴으로써 홉 수가 늘어남에 따라 채널에서의 충돌이 증가하여 전송률이 낮아진다. 기존의 delayed ACK(d=2,3,4)알고리즘과 일반 TCP 알고리즘에 비하여 제안된 알고리즘의 전송률 차이는 홉 수에 따라 달라지지만, 20홉까지의 평균 전송률을 비교해 보았을 때 표 5.2와 같은 결과를 보여준다.

표 5.2 각 알고리즘의 평균 전송률(LimitNum)
Table 5.2 Average transmitted rate of each algorithm(LimitNum).

	LimitNum=4	LimitNum=6
TCP	101514.8	100579.8
D-ACK (d = 2)	123667.4	122193.6
D-ACK (d = 3)	125091.2	123071.8
D-ACK (d = 4)	125516.3	124595.9
Proposed	142909.5	139100.2

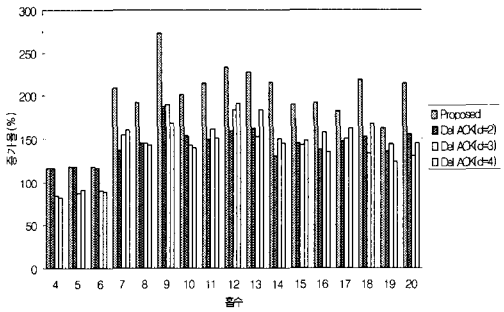

그림 4.4 LimitNum = 4일 때의 전송률 증가
Fig. 4.4 Transmitted rates enhancement for LimitNum = 4.

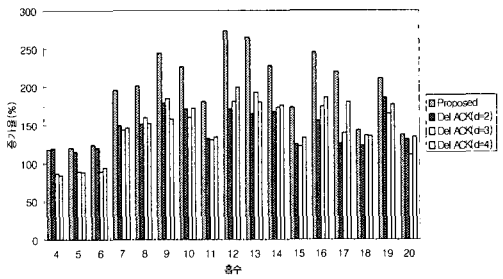

그림 4.5 LimitNum = 6일 때의 전송률 증가
Fig. 4.5 Transmitted rates enhancement for LimitNum = 6.

그림 4.4, 4.5는 표준의 TCP의 성능과 비교하여 보았을 때 각 알고리즘별 전송률의 증가율을 보여주며, LimitNum=4일때 기존의 delayed ACK (d=2,3,4)와 일반 TCP 알고리즘에 비하여 제안된 알고리즘의 전송률의 개선치는 각각 81%, 39%, 38%, 40%의 증가를 보여주고 있다. 또한 LimtiNum=6일때의 전송률의 개선치는 각각 76%, 41%, 41%, 46%의 성능향상을 보였다.

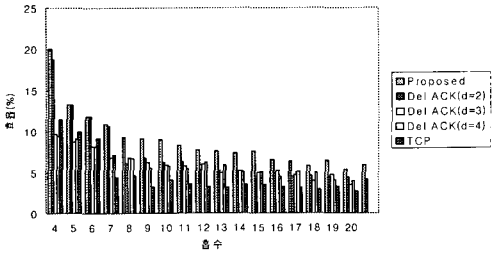

그림 4.6 LimitNum = 4일 때의 전송률
Fig. 4.6 Transmitted rates for LimitNu = 4.

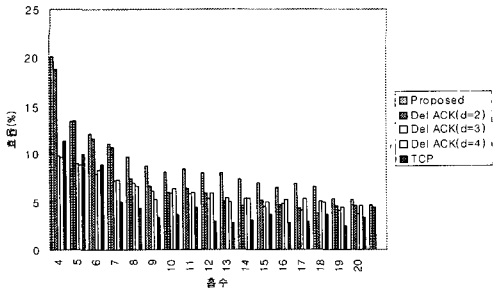

그림 4.7 LimitNum = 6일 때의 전송률
Fig. 4.7 Transmitted rates for LimitNum = 6.

그림 4.6, 4.7은 각 홉 수별 전송률을 보여주고 있다. LimtiNum=4일때 제안된 알고리즘과 기존의 알고리즘에서의 노드들의 평균 채널 사용량은 각각 13%, 11%, 11%, 11%를 보여주고 있으며, LimitNum=6인 경우에는 각각 13%, 11%, 12%, 11%의 채널 사용량을 보였다.

나. MaxCWnd 크기의 변화에 따른 전송률

Ad-hoc 네트워크에서의 CWnd의 무제한적인 증가는 네트워크상에서 혼잡을 야기시킬 수 있고, 또한 CWnd의 크기는 곧 송신측에서 보낼 수 있는 데이터 패킷의 수와 같다고 할 수 있다. 이러한 이유로 CWnd의 크기를 무제한적으로 올리게 된다면 네트워크상에서의 혼잡은 물론 Ad-hoc 네트워크 채널에서의 충돌 상황으로 인하여 패킷 손실을 야기함으로써 네트워크 성능을 저하시킬 수 있는 원인이 된다. 이에 최대한으로 늘릴 수 있는 CWnd의 크기를 제한함으로써 Ad-hoc 네트워크에서 TCP 성능을 향상시킬 수 있었다.

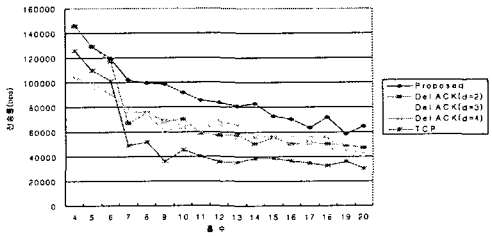

그림 4.8 MaxCWnd = 7일 때의 전송률
Fig. 4.8 Transmitted rates for MaxCWnd = 7.

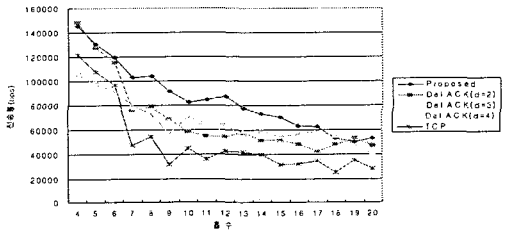

그림 4.9 MaxCWnd = 50일 때의 전송률
Fig. 4.9 Transmitted rates for MaxCWnd = 50.

그림 4.8과 4.9는 MaxCWnd 변수를 7과 50으로 변화시키면서 시뮬레이션한 결과이다. 앞에서 언급한 바와 같이 CWnd의 크기가 무제한적으로 늘어나게 되면 채널에서의 충돌로 인한 패킷 손실이 많아짐을 알 수 있다. 이 또한 TCP 성능을 저하시키는 요인으로서 LimitNum 변수의 결과와 비슷한 결과를 보이고 있다. MaxCWnd의 크기의 제한을 바탕으로 한 결과, 기존의 delayed ACK(d=2,3,4)알고리즘과 일반 TCP 알고리즘에 비하여 제안된 알고리즘의 전송률 차이는 홉 수에 따라 달라지지만, 20홉까지의 평균 전송률을 비교해 보았을 때 표4.3과 같은 결과를 보여준다. MaxCWnd=7일 때 28%, 30%, 29%, 81%의 성능 향상을 보이고 있고, MaxCWnd=50일 때는 27%, 29%, 25%, 82%의 증가를 나타냈다.

표 4.3 각 알고리즘에서 평균 전송률(MaxCWnd)
Table 4.3 Average transmitted rate of each algorithm(MaxCWnd).

	MaxCWnd=7	MaxCWnd=50
TCP	101514.8	100995.8
D-ACK (d = 2)	123667.4	123488.9
D-ACK (d = 3)	125091.2	123071.8
D-ACK (d = 4)	125516.3	124595.9
Proposed	142909.5	141901

그림 4.10, 4.11은 표준의 TCP의 성능과 비교하여 보았을 때 각 알고리즘별 전송률의 증가율을 보여준다. MaxCWnd=7일때 기존의 delayed ACK(d=2,3,4)와 일반 TCP 알고리즘에 비하여 제안된 알고리즘의 전송률의 개선치는 각각 81%, 39%, 38%, 40%이고, 또한 MaxCwnd=50일때의 결과는 각각 82%, 41%, 39%, 44%의 성능 향상을 보였다.

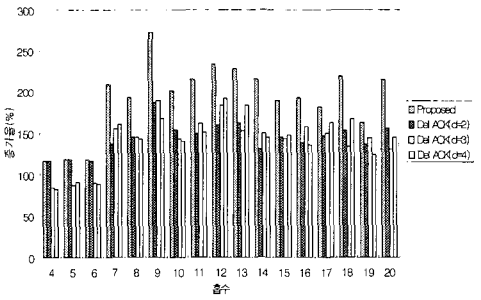

그림 4.10 MaxCWnd = 7일 때의 전송률 증가
Fig. 4.10 Transmitted rates enhancement for MaxCWnd = 7.

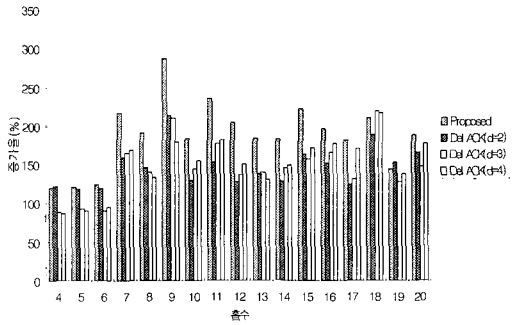

그림 4.11 MaxCWnd = 50일 때의 전송률 증가
Fig. 4.11 Transmitted rates enhancement for MaxCWnd = 50.

그림 4.12, 4.13은 각 홉 수별 전송률을 보여주고 있다. MaxCWnd=7일때 제안된 알고리즘과 기존의 알고리즘에서의 노드들의 평균 채널 사용량은 각각 13%, 11%, 11%, 11%이고, MaxCWnd=50인 경우에는 각각 13%, 11%, 12%, 11%의 채널 사용량을 보였다.

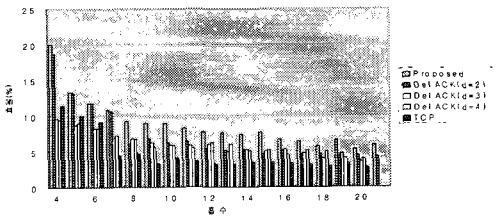

그림 4.12 MaxCWnd = 7일 때의 전송률
Fig. 4.12 Transmitted rates for MaxCWnd = 7.

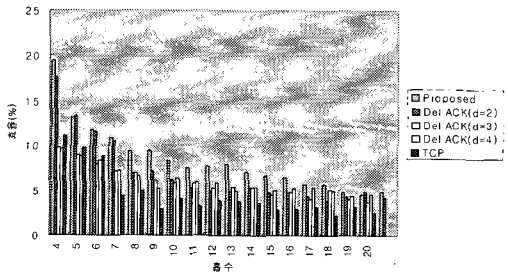

그림 4.13 MaxCWnd = 50일 때의 전송률
Fig. 4.13 Transmitted rates for MaxCWnd = 50.

V. 결 론

최근 Mobile Ad-hoc의 발전과 그 이용에 많은 관심을 갖게 되면서, 자연스럽게 네트워크 성능 향상을 위한 연구가 이루어지고 있다. 하지만 현재까지 MAC 계층의 연구와 라우팅 알고리즘에 편중되었던 반면 TCP에서의 성능향상을 위한 연구는 많지 않았다.

IEEE 802.11을 사용하고 있는 Ad-hoc 네트워크의 경우 채널의 공유로 인한 채널간의 경쟁과 채널사이에서의 충돌문제로 패킷 손실이 많아지고 이로 인해 네트워크 성능도 당연하게 저하되었다.

또한 Ad-hoc 네트워크에서의 손실은 간섭, 페이딩, 노이즈 등의 링크계층에서의 손실이 많지만, Ad-hoc 네트워크 상에서 이러한 손실들은 혼잡에 의한 손실로 간주하며

혼잡제어를 통한 네트워크의 성능이 감소되고 있다. 하지만 현재 Ad-hoc 네트워크에서의 TCP 성능 향상에 관한 연구가 활발히 진행되고 있으며, 이는 기존의 연구들과의 시너지 효과를 통하여 Ad-hoc 네트워크의 성능을 향상 시킬 수 있을 것으로 보인다.

Ad-hoc 네트워크에서 채널을 공유함으로써 발생하는 채널간의 경쟁과 충돌로 인해 네트워크의 성능은 현저히 감소하고 있다. 이를 개선하기 위하여 데이터 패킷보다 크기가 작은 컨트롤 패킷인 ACK 패킷의 수를 줄임으로써 채널간의 경쟁과 충돌을 줄일 수 있다. 이러한 이유로 Ad-hoc 네트워크의 TCP 성능향상을 위하여 delayed ACK 알고리즘이 널리 사용되고 있으나 이 알고리즘 또한 유선 환경에서 연구되었으므로 Ad-hoc 네트워크에 적용하기에는 무리가 따른다.

본 논문에서는 TCP 성능 향상을 위하여 지연 데이터 패킷의 수를 고정시켰던 기존의 delayed ACK에서의 지연 데이터 패킷 수를 순차적으로 증가시킴으로써 Ad-hoc 네트워크에서 데이터 패킷과 ACK 패킷의 채널 경쟁 및 충돌을 감소시켜 TCP의 성능을 향상시켰다. 또한 제안된 알고리즘은 시뮬레이션을 통해 성능을 평가한 결과, Ad-hoc 네트워크에서 채널간의 경쟁 및 충돌 감소, ACK 패킷에 의한 채널 선점으로 전송하지 못하였던 시간에 데이터 패킷을 전송할 수 있어서 기존의 delayed ACK 알고리즘과 일반적인 TCP 알고리즘보다 높은 성능을 보였다.

참고문헌

[1] Toh. C.K., 2002. Ad hoc Mobile Wireless Networks Protocols and Systems. Prentice Hall Inc.

[2] Anastasi, G., M. Conti and E. Gregori, 2003. IEEE 802.11 ad hoc networks: protocols, performance and open issues. IEEE Press and John Wiley and Sons, Inc., New York , USA.

[3] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash, "A Feedback-based Scheme for Improving TCP Performance in Ad Hoc Wireless Networks," IEEE Personal communications, 8 (1):34-39, Feb. 2001.

[4] G. Holland and N. H. Vaidya, "Analysis of TCP Performance over Mobile Ad Hoc Networks," *MOBICOM'99*, Seattle, Aug. 1999.

[5] K. Chen, Y. Xue, K. Nahrstedt, "On Setting TCP's Congestion Window Limit in Mobile Ad Hoc Networks," *IEEE ICC'03,* Anchorage, Alaska, May 2003.

[6] F. Wang and Y. Zhang, "Improving TCP Performance over Mobile Ad-Hoc Networks with Out-of-Order Detection and Response," *MobiHoc'02*, pp. 217-225, Lausanne, Switzerland, June 2002.

[7] M. Al lman, V. Paxson, W. St evens, TCP Congestion Control, RFC 2581, Apr. 1999.

[8] J. Heidemann, Performance Interactions Between P-HTTP and TCP Implementation, *ACMCCR27(2),* Apr. 1997.

[9] M. Allman, S. Floyd, C. Partridge, Increasing TCP's Initial Window size, *RFC 2414*, Sep. 1998.

[10] M. Allman, S.Floyd, C.Partridge, Increasing TCP's Initial Window size, *RFC 2414*, Sep. 1998.

[11] M. Allman, TCP Byte Counting Refinements, ACM CCR 29(3), July 1999.

[12] K. Fall, S. Floyd, Simulation-based Comparisons of Tahoe, Reno, and SACK TCP, CCR 26(3), July 1996.

저자소개

박 광 채(Kwang-Chae Park)

1975년 2월 조선대학교 전자공학과 (공학사)
1980년 2월 조선대학교 대학원 전자공학과(공학석사)
1994년 8월 광운대학교 대학원 전자통신공학과(공학박사)
※관심분야 : 데이터 통신및 프로토콜, 디지털 교환기, Ad-hoc Networks, 광대역 정보통신

나 동 건(Dong-Geon Na)

2002년 2월 전북대학교 전자공학과 (공학사)
2005년 8월 전북대학교 전자공학과 (공학석사)
※관심분야 : 데이터통신 및 네트워크, wireless, TCP-IP