논문 15-40-08-11 The Journal of Korean Institute of Communications and Information Sciences '15-08 Vol.40 No.08
http://dx.doi.org/10.7840/kics.2015.40.8.1542

고속 통신 구현을 위한 Delayed ACK Timeout 값의 유동적인 적용 연구

이 준 엽◆, 이 웅 희*, 김 황 남°

Regulating Delayed ACK Timeout to Construct High Speed Transmission

Joon Yeop Lee◆, Woonghee Lee*, Hwangnam Kim°

요 약

Delayed ACK은 TCP에서 기본적으로 사용되고 있는 기술로써 회선의 혼잡도를 완화하고 데이터 전송 시에 ACK을 처리하는데 들어가는 부하를 줄이는 효과를 가지고 있다. 다만 이러한 Delayed ACK은 다음 ACK이 생성 될 때까지 Delayed ACK timeout 값만큼 ACK전송을 지연시키게 되는데 이 값이 너무 커지면 추가적인 ACK의 발생을 너무 오래 기다림으로써 통신 속도의 저하를 발생시키게 된다. 본 논문에서는 Delayed ACK Timeout 값이 일반적인 ACK의 발생 빈도보다 지나치게 큰 값을 가짐으로써 발생하는 통신 속도 저하를 방지하기 위해, Delayed ACK Timeout 값을 조절하여 불필요한 지연을 줄여 전송 속도를 개선하는 알고리즘을 제안한다.

Key Words : Data transmission, TCP, Delayed ACK, CWND, ATO

ABSTRACT

Delayed ACK is an algorithm implemented to decrease the number of ACK transmissions by delaying an ACK transmission and by waiting for additional ACK instead of transmitting the ACK immediately. By using Delayed ACK in TCP, the congestion of network and the overhead of handling ACKs can be reduced. Waiting time of Delayed ACK is defined as the Delayed ACK timeout, and it is fixed in Window OS basically. However, the fixed value of Delayed ACK timeout is not suitable for dynamic network circumstance, and it may cause unnecessary delay. This paper proposes a regulating Delayed ACK timeout algorithm to reduce the aforementioned unnecessary delay caused by the lengthy default value of the Delayed ACK timeout. We confirm that TCP transmission performance in dynamic network circumstance can be improved using the proposed algorithm.

Ⅰ. 서 론

TCP(Transmission Control Protocol)는 데이터 전달의 신뢰성이 보장되는 통신 프로토콜로써 현대에는 모바일 기기를 포함하여 거의 대부분의 프로그램과 응용 프로그램계층의 프로토콜이 사용하는 전송계층

※ 이 논문은 2015년도 BK21 플러스 사업과 2013년도 정부(교육부)의 재원으로 한국 연구재단의 지원을 받아 수행된 기초연구사업임 (No. NRF-2013R1A1A2010388)
◆ First Author : School of Electrical Engineering, Korea University, charon7@korea.ac.kr, 학생회원
° Corresponding Author : School of Electrical Engineering, Korea University, hnkim@korea.ac.kr, 종신회원
* Second Author : School of Electrical Engineering, Korea University, tgorevenge@korea.ac.kr, 학생회원
논문번호 : KICS2015-05-156, Received May 27, 2015; Revised July 8, 2015; Accepted August 4, 2015

프로토콜이다[1]. TCP 통신은 같은 전송계층의 프로토콜인 UDP(User Datagram Protocol)와는 달리 데이터의 신뢰성을 보장하기 위해 수신자가 segment 단위의 데이터를 전송 받을 때 마다 성공적으로 받았다는 정보인 ACK(Acknowledgement)을 송신자에게 보낸다. 송신자는 ACK을 받으면 데이터 전송이 성공한 것으로 간주하며, 만일 대기시간동안 받지 못하면 데이터 전송이 실패한 것으로 간주, 재전송을 하게 된다. 이 ACK은 segment를 받을 때 마다 생성되므로, segment를 보내는 만큼 ACK을 수신자가 송신자에게 전송하게 되므로 이 전송이 회선의 혼잡도와 이를 처리하는 단말기에 영향을 주게 된다. 따라서 TCP에서는 과도한 ACK 전송이 주는 부담을 완화하기 위해 Delayed ACK(Delayed Acknowledgement)기술을 적용하고 있다. Delayed ACK은 TCP 통신에서 기본적으로 사용하게 설정되어 있을 정도로 그 효과를 이미 입증한 기술이다[2]. 다만 Delayed ACK은 ACK의 전송을 지연시키게 되는데 이 지연시간은 최대 Delayed ACK timeout 값만큼 지연될 수도 있으며, 이 지연 가능성이 매 ACK을 보낼 때 마다 존재하게 된다. 이러한 상황에서 기기의 성능이 향상되고, 통신회선이 점점 빨라지는 현재 통신환경을 고려하지 않고 Delayed ACK timeout 값이 고정되어 있는 것은 비효율적인 지연을 발생시킬 수 있다. 현재 Delayed ACK Timeout값은 Windows OS의 경우에는 0.2초로 고정되어 있으며[3] 이는 데이터의 전송 간격을 고려하면 매우 큰 값이다. 과거 통신환경이 느리고 불안정 하였을 경우에는 0.2초의 값이 적절 하였을지 모르나, 충분히 회선이 빠르지만 일시적인 오류로 데이터가 소실됐거나, 송신자의 데이터 처리속도가 빨라서 ACK 처리가 크게 부담되지 않는 상황에서도 Delayed ACK timeout값을 0.2초로 고정시키는 건 사용가능한 통신 자원에 비해 지나치게 오랜 지연을 유발할 수 있다. 이는 결과적으로 통신자원이 충분함에도 불구하고 전송이 불필요하게 지연되면서 상대적으로 통신 속도가 저하되는 결과를 가져올 수 있다.

이러한 통신환경이 변하게 된 원인으로는 전반적인 통신품질의 향상과 무선통신 빈도의 증가가 있다. 현대에는, 특히 우리나라는 LTE의 빠른 보급과 회선망의 증축을 통해 통신환경이 나날이 좋아지고 있다. 그리고 과거에는 다수의 ACK을 생성하고 처리하면서 발생하는 기기의 부담이 컸으나, 스마트폰의 보급을 포함하여 통신 기기들의 성능이 발달하면서 더 이상 ACK을 관리하는 정도의 일은 기기에 크게 부담이 되지 않는다. 한편으로 모바일 단말기가 보급화 되고, RFID를 활용한 IoT의 연구가 진행되면서 무선통신의 비중이 점점 높아지고 있다. 이러한 무선통신환경에서는 더 이상 데이터 손실의 원인이 다수의 데이터에 의한 회선 혼잡만 존재하는 것이 아니다[4][5]. 무선통신에서는 기지국간의 연결을 재설정하는 hand-off, Wi-Fi 존을 벗어남으로써 LTE로의 통신방식의 전환, 불안정한 무선 통신환경, 전파간섭 등 다양한 데이터 전송을 방해하는 요소를 가지고 있으며, 그만큼 데이터 손실 확률도 높다. 즉, 회선과 기기의 처리속도는 높아지고 있지만, 정작 받아야 할 데이터의 손실률은 증가했다고 볼 수 있다. 이러한 데이터 손실은 수신자의 ACK이 빠른 주기로 생성되지 못하여 지연시간이 Delayed ACK timeout을 초과할 만큼 오래 지연되는 경우를 유발할 수 있다. 따라서 Delayed ACK timeout값이 크면 클수록 통신 속도측면에서 불필요한 지연이 많이 발생하게 된다.

따라서, 본 논문에서는 이러한 고속 통신이 가능한 통신환경과 무선통신의 특성을 고려하여, 효율적인 통신을 위해 유동적인 Delayed ACK tiemout 값을 사용할 것을 제안한다. 이를 구현하기 위해 Delayed ACK의 파라미터인 Delayed ACK timeout의 값을 데이터를 전달받는 빈도와 비슷한 값으로 설정함으로써, ACK의 전송 빈도를 높이고 불필요한 지연을 줄이는 알고리즘을 제시한다. 송신자 입장에는 ACK을 받을 때 마다 전송하는 데이터의 양을 결정하는 CWND (Congestion Window)를 증가시키므로, 빠르게 ACK을 받게 되면 그만큼 CWND가 빠르게 증가하게 된다. 즉, 송신자가 높은 빈도로 ACK을 전송 받게 되면 이는 전반적인 전송속도의 증가로 연결된다[6]. 따라서 본 논문에서 제시하는 적절한 Delayed ACK Timeout 값을 도출하여 통신에 적용한다면, Delayed ACK 의 효과를 일부 유지하면서도, 빠른 전송속도 증가를 기대할 수 있다.

Ⅱ. 관련 이론

본 논문에서는 Delayed ACK의 일부 파라미터를 조정하여 효과적이고 빠른 통신을 구현하려 한다. 이 알고리즘에 활용되는 관련 이론들에 대하여 간략하게 다룬다.

2.1 TCP 통신환경 개선을 위한 Delayed ACK의 작동방식과 문제점

TCP 통신에서 송신자는 segment 단위로 데이터를 전송하고, 수신자는 각각의 segment가 수신될 때 마

다 해당 segment에 대한 ACK을 전송하게 된다. 만일 다량의 segment를 연속적으로 수신하면 다수의 ACK이 빠르게 생성되는데, 이때 ACK을 일일이 따로 전송하게 되면 서버에 큰 부하를 주게 되고, 이를 처리하는 프로세스에도 부하가 걸리게 된다[7]. 이러한 현상을 완화하기 위해서 Delayed ACK을 사용한다. Delayed ACK은 수신자가 전송해야 하는 ACK이 생겼을 때, 이를 바로 보내지 않고 일단 지연시킨다. 이후에 일정 크기 이상의 데이터가 모이면 즉시 전송을 하거나, 혹은 일정 크기만큼의 데이터가 모이기를 Delayed ACK timeout 값만큼의 시간을 기다린 다음에 ACK을 전송한다. 일정크기의 데이터가 모이는 조건은 일반적으로 2개의 ACK이 생성되면 그 조건이 충족된다. 일반적인 경우라면 처음 생성된 ACK은 두 번째 ACK이 생성될 때 까지 약간 지연되었다가 한꺼번에 전송되게 되며, 이때 지연되는 시간은 무제한으로 기다리지 않고 Delayed ACK timeout 값만큼의 시간까지 기다리게 된다. 이 시간이 초과한다면 더 이상 지연 시키지 않고 모인 데이터를 즉시 전송한다. 원활한 데이터 통신이 이루어진다면, Delayed ACK timeout값을 초과할 정도로 오래 기다리는 경우는 잘 발생하지 않지만, 만약 회선혼잡이 발생하거나, 데이터를 처리해주는 단말기의 처리지연 등의 예외적인 문제가 발생하여 추가적인 ACK 생성이 늦어지는 경우가 자주 발생하면, ACK의 전송이 Delayed ACK timeout 값만큼 자주 지연되어서 결과적으로 통신 속도가 저하되는 문제가 발생한다.

2.2 전송속도를 결정하는 CWND의 원리

CWND (congestion window)는 TCP에서 데이터 전송 시에 송신자가 얼마만큼의 데이터를 한 번에 전송할지를 결정한다. 일반적으로 송신자는 ACK을 받으면 다음 순서의 segment를 전송하지만, CWND의 크기만큼의 데이터는 이전 segment의 전송 성공 여부에 관계없이 보낸 다음, 각 segment들은 각자의 ACK을 기다리게 된다. 이 CWND의 크기는 segment를 전송한 후에 송신자가 ACK을 받을 때 마다 늘어나며, 이와 동시에 CWND를 움직여서 바로 다음 순번의 segment를 전송할 준비를 하게 된다. CWND크기의 증가량은 현재 TCP의 통신 상태에 따라 다르게 적용되는데, 크게 slow start가 적용될 때와 congestion avoidance가 적용될 때의 두 가지 경우로 나눌 수 있다[8]. 일단 TCP 연결이 성사된 직후나, 재전송 시간을 초과한 경우에는 CWND의 크기는 1부터 시작한다. 여기서 1은 한 개의 segment의 크기를 뜻한다.

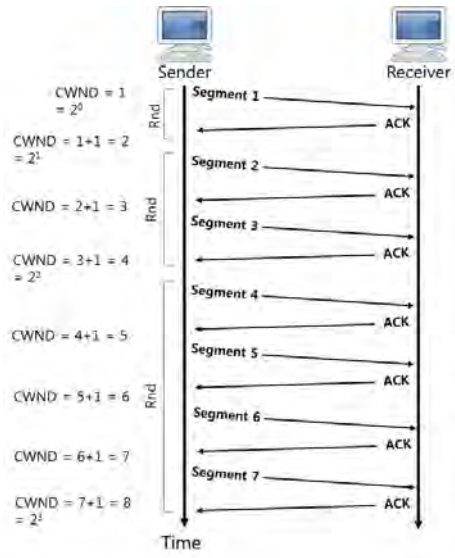

그림 1. slow start 알고리즘이 적용되었을 때의 CWND 변화량
Fig. 1. CWND variation in Slow start algorithm

CWND가 1부터 시작할 때는 slow start알고리즘에 따라서 ACK을 받을 때 마다 CWND의 크기가 1씩 증가하게 된다. 따라서 한 번의 전송주기인 Rnd (Round of transmission) 만큼의 전송을 하고나면 CWND의 크기는 2배씩 커지게 된다. 그림 1을 통해서 slow start 알고리즘에 따른 CWND 변화량을 확인할 수 있다.

Slow start를 통해서 CWND의 크기를 빠르게 키우고, 만일 한번이라도 재전송이 일어나게 되면 추가적인 회선 혼잡을 피하기 위해 CWND값을 1/2로 줄이게 된다. 후로는 재전송 시간을 초과하기 전 까지는 congestion avoidance 알고리즘이 적용된다. congestion avoidance 알고리즘이 적용되면 이전 CWND의 크기가 n만큼의 크기였을 경우, ACK을 받을 때 마다 1/n씩 CWND의 크기가 증가한다. 즉, 자신의 이전 CWND의 크기만큼의 ACK을 받으면 비로소 CWND의 크기가 1 증가하게 된다. 이 과정을 나타내면 그림 2와 같다. 이와 같이 TCP 프로토콜에서는 송신자가 ACK을 받을 때 통신 속도를 결정하는 CWND의 증가가 이루어지므로, 빠르게 ACK을 받는 것은 통신 속도 증가 효과로 이어지게 된다.

2.3 Delayed ACK의 문제해결을 위한 리눅스의 Quick ACK

Delayed ACK은 TCP통신에서 이루어져야 할 ACK 응답을 오히려 지연시킴으로써 장기적으로는

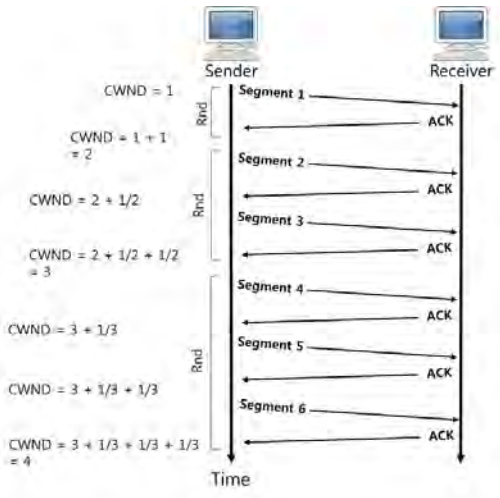

그림 2. congestion avoidance 알고리즘이 적용되었을 때의 CWND 변화량
Fig. 2. CWND variation in congestion avoidance algorithm

기존의 통신 방식보다 유익한 효과를 얻지만, 통신 시작 직후나, 재전송이 발생한 직후에는 CWND의 크기 증가 속도가 기존보다 느려진다는 단점이 있다[9]. 이는 송신자가 처음에 전송할 때에는 1개의 segment만 보내는데, 한 개의 segment만이 수신되었으므로, 두 번째 ACK이 생성되지 않아서 첫 데이터 전송시에 발생한 ACK은 항상 Delayed ACK timeout값만큼 지연되었다가 전송되는 문제가 발생하였기 때문이다. 이렇게 느리게 ACK을 받게 되면, 송신자가 ACK를 받아야 다음 데이터를 전송하면서, 더 많은 데이터를 보내기 위해 CWND의 크기를 늘린다는 점을 생각하면, 결과적으로 통신 속도의 증가 속도가 늦어지게 된다. 즉, 송신자의 CWND의 증가속도를 올려 빠른 데이터 전송을 유도하기 위해서는 빠르게 ACK을 보내야 한다. Delayed ACK은 이러한 측면에서는 ACK전송을 지연시키므로, 불리하다고 볼 수 있다. 리눅스에서는 통신 시작 직후의 느린 CWND 증가속도 문제 해결을 위해서, TCP연결이 성립된 직후에는 2 회의 segment에 한해서 ACK을 지연 시키지 않고 즉시 ACK을 전송하도록 하는 Quick ACK을 사용하고 있다. 이 Quick ACK을 통해서 Delayed ACK이 적용되지 않은, 기존의 빠른 ACK전송의 이점을 볼 수 있다[10].

Ⅲ. 알고리즘 설명

본 논문에서 제시하는 알고리즘의 구조와 알고리즘에서 사용할 Delayed ACK timeout 값의 결정 방식에 대해 설명한다.

3.1 알고리즘의 설명

Delayed ACK timeout값이 커지면 불필요한 기다림을 유발하지만, 반대로 지나치게 작은 값을 가지게 되면 Delayed ACK 자체가 작동되지 않아서 Delayed ACK의 장점을 살릴 수 없게 된다. Delayed ACK은 결국 ACK의 생성 빈도에 따라 전송시간이 결정된다고 볼 수 있는데, ACK의 생성 속도는 데이터를 수신받는 시간 간격과 비슷하다는 점을 이용, Delayed ACK Timeout값을 최근에 데이터를 수신 받는 시간 간격 값들의 평균에 가까운 값으로 설정하였다. 이를 통해 예외적으로 데이터의 전송이 늦는 경우에는 더 이상 ACK을 지연시키지 않고 즉시 전송하게 되고, 평균적인 속도로 데이터가 전송되어 온다면 지연시킨 ACK을 같이 전송하게 되므로 일반적인 상황에서는 Delayed ACK 효과가 나타나게 된다. 결과적으로 ACK은 항상 ACK이 생성되는 속도와 비슷한 빈도로 전송되게 된다. 이는 송신자에게 지연 없는 ACK을 받을 수 있도록 하며, 이를 통해 빠른 CWND 증가효과를 기대할 수 있다. 이 알고리즘을 적용하면 수신자의 지나친 지연시간의 방지효과와, 송신자의 빠른 CWND 증가를 유도하여 송신자가 더 빠르게 데이터를 전송하도록 하는 효과를 얻을 수 있다.

3.2 유동적인 Delayed ACK timeout 값의 설정

Delayed ACK timeout 값을 결정할 때 최근에 수신 받은 데이터 전송간격에 a만큼의 비중을 주어서 계산에 반영하면, 통신 회선이 느려져서 데이터의 도착 간격이 길어지는 경우에는 길어진 시간이 Timeout값에 반영되므로 적절한 Delayed ACK timeout값을 가질 수 있으며, 만일 timeout값이 너무 작게 설정되어 있어서 Delayed ACK이 적용되지 않고 ACK이 무조건적으로 즉시 전송되는 현상이 일어날 경우에도 데이터 전송간격을 토대로 알맞은 시간을 가지게 된다. 새로운 Delayed ACK timeout값을 *Tn*, 현재 Delayed ACK timeout값을 *To*, 가장 최근에 측정된 데이터 전송간격을 *M*, 최근에 측정된 전송간격의 반영 비율을 *a* 라고 한다면, 다음 수식을 segment를 전송받을 때마다 적용하여 적절한 Delayed ACK timeout값을 도출해낼 수 있다.

$$Tn=(1-a)To+aM \quad (1)$$

통신환경의 변화에 더 빠르게 맞춰나가고 싶다면 *a*

값을 크게 설정하고, 회선의 변동이 너무 불규칙하여 Delayed ACK timeout값을 비교적 안정적으로 유지하고 싶다면 *a* 값을 작게 설정하면 된다. 다만 *a* 값의 크기가 너무 커져서 최근 데이터 값의 비중이 커지게 설정하면, 통신 시에 일시적인 끊김 현상이나 급격한 패킷 몰림 현상 등의 패킷 간격이 급격하게 변하는 상황이 발생 하였을 때 이 통신간격이 거의 그대로 Delayed ACK timeout값에 반영되게 된다. 무선 통신에서 이러한 상황이 자주 발생하며, 이런 경우에는 새로운 *Tn* 값이 0.0002 ~ 5초 사이의 매우 편차가 큰 값이 매번 바뀌어 적용되게 된다. 이렇게 되면 일정시간 동안 Delayed ACK이 전혀 적용되지 않거나, 혹은 ACK을 너무 오랫동안 전송하지 않게 되어서 오히려 통신 품질을 떨어뜨릴 수 있다. 따라서 무선 통신의 경우, 기존의 통신으로 얻어진 평균값에 비중이 더 두어야 문제없이 통신이 가능하다. 유선통신은 무선 통신보다 비교적으로 안정적인 통신간격을 보이므로 *a* 값이 큰 값으로 설정되어도 안정적으로 통신환경의 변화에 맞출 수 있으나, TCP 통신에서는 유/무선을 판단할 수 없으므로, 대부분의 통신환경을 고려하였을 때 안정적인 통신이 가능하였던 0.1값을 *a* 에 적용하여 실험을 하였다.

3.3 알고리즘의 적용

TCP segment를 전송받을 때 본 알고리즘이 적용된다. *pretime* 에는 이전에 데이터를 받은 시간이 저장되어 있으며, 이를 현재 시간인 *time*값과의 차이를 통해 데이터 전송간격인 *M*을 계산하게 된다. 이후 *time*값, 즉 현재시간은 다음번 segment를 받았을 때의 전송간격 계산을 위해 *pretime*에 저장된다. 만일 SYN 메시지를 받으면 이는 TCP 통신이 최초로 시작된다는 뜻 이므로, SYN을 받은 시각을 *pretime*에 저장한 다음 SYN+ACK 전송 함수를 실행한다. 전송받은 segment가 SYN이 아니라면 일반적인 데이터 전송이므로, 새로운 Delayed ACK timeout값인 *ATO*를 설정한 다음, TCP의 Delayed ACK 알고리즘에 따라 데이터를 전송하는 함수인 *send_ACK()*을 작동시킨다. 이후 send_ACK() 함수에서 Delayed ACK 규칙에 따라 ACK을 전송하게 되며, 이때 여기서 결정된 ATO 값이 Delayed ACK timeout값으로 반영되게 된다. 본 알고리즘의 의사코드는 다음과 같다.

```
Algorithm 1 Delayed ACK Adjustment
1:  if received segment is SYN then
2:      pretime = time
3:      operate send_SYN+ACK() function
4:  else
5:      M = time - pretime
6:      pretime = time
7:      new ATO = (1-a)*ATO + a*M
8:      operate send_ACK() function
9:  end
```

Ⅳ. 시뮬레이터 실험

본 논문에서 제시한 알고리즘을 적용하여 실험을 진행하였다. 먼저 실험환경 설정이 용이한 시뮬레이터를 통해 알고리즘이 적용되었을 때의 결과를 알아보았다.

4.1 실험환경 구축

시뮬레이터 실험은 ns3를 사용하여 진행하였다. 실험 환경은 그림 3에서 볼 수 있듯, AP에 무선으로 연결된 Node 2가 AP에 P2P로 연결된 Node 1에게로 TCP 통신을 한다. AP를 경유하며, 유선 회선의 상태는 5Mbps를 지원하고, 회선 자체의 통신지연 값은 2ms가 되도록 설정을 하였다. Node 2에서 1000개의 데이터를 1Mbps속도로 Node 1에게 보내도록 설정하였으며, 이 실험을 10초간 진행한 다음 실험 시간 동안 송신자의 CWND의 변화를 출력하도록 하였다. 본 실험에서는 현실성을 고려하여 불규칙한 빈도로 데이터 손실이 발생하는 ns3 오류모델중 하나인 RateErrorModel을 사용하였으며, 오류 빈도 값은 0.00001로 설정하였다.

그림 3. TCP 통신 실험환경
Fig. 3. TCP communication experiment environment

4.2 Segment 전송간격 측정

먼저 Delayed ACK timeout값을 설정하기 위해 위에서 구성한 통신 환경에서 일정량의 TCP통신을 진행한 다음 해당 결과를 pcap파일로 추출하여 분석하였다. 결과는 그림 4와 같다. 해당 결과에서 볼 수 있듯, segment의 수신 간격은 매우 짧으며, 수신자에게 segment가 0.0006~0.0011초 간격으로 들어오는 것을 확인하였다. 해당 데이터 수신간격을 토대로 새로운 Delayed ACK timeout값을 설정하게 된다.

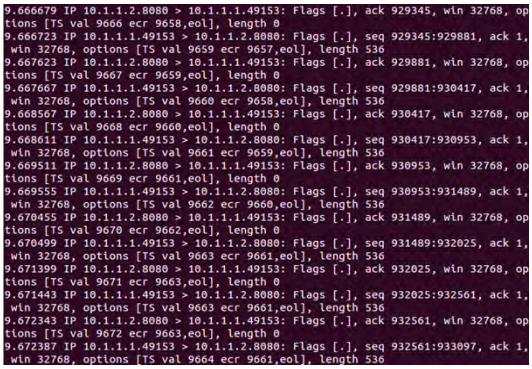

그림 4. pcap 파일에서 추출한 TCP 통신 결과화면
Fig. 4. result page of TCP communication from pcap

4.3 새로운 Delayed ACK timeout값을 적용 후 CWND 측정

위에서 실험한 결과에서 최근 10개의 데이터 수신 간격 값을 사용하여서 새로운 Delayed ACK timeout 값을 0.0008초와 0.0009초로 설정하여 각각 실험하였다. 이후 전송 환경을 바꾸지 않고, 오직 Delayed ACK timeout 값만 새로 얻은 값을 적용함으로써, 본 실험의 목적인 빠른 간격으로 segment를 수신 받으면 Delayed ACK 효과를 얻고, 큰 간격으로 segment를 수신 받으면 즉시 ACK을 전송할 수 있도록 하였다. 이렇게 변경된 Delayed ACK timeout 값을 적용하여 두 번째 실험을 진행하였으며 이때의 송신자의 CWND 크기 값의 변화량을 기존의 Delayed ACK timeout값이 0.2초일 때 통신시의 CWND크기 변화량이랑 비교하였다.

4.4 실험결과 분석

그림 5, 그림 6 에 나타나 있듯이 Delayed ACK timeout의 값을 기본 값인 0.2초로 설정했을 경우와, 앞서 제시한 값으로 변경하였을 때의 송신자의 CWND 크기 변화량을 비교하였다. 데이터 전송 시작 시, 재전송시에 CWND의 크기 증가 속도가 확연히 다른 것을 볼 수 있으며, 기존의 0.2초 설정보다 더 빠르게 CWND가 증가하고 있음을 알 수 있다. 또한 이렇게 커진 CWND는 재전송이 발생할 때 CWND의 크기가 1/2로 줄어들어도 기존보다는 큰 값을 유지하게 된다. 따라서 통신 시작 이후로 재전송이나 회선혼잡이 발생하여도 항상 기존의 Delayed ACK timeout 값을 적용한 경우 보다 더 큰 CWND 값을 유지할 수 있으며, 이는 본 알고리즘의 적용을 통해 기존보다 더 빠른 전송속도의 증가효과를 얻을 수 있음을 알 수 있다. Delayed ACK timeout값을 0.0009초로 설정하면

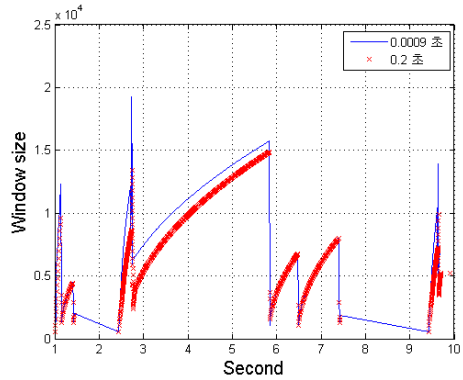

그림 5. 0.0009초 설정시의 CWND변화량 측정 결과
Fig. 5. Result of CWND measurement with 0.0009sec

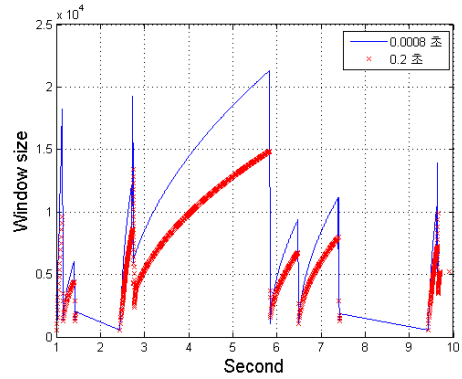

그림 6. 0.0008초 설정시의 CWND변화량 측정 결과
Fig. 6. Result of CWND measurement with 0.0008sec

CWND의 증가량은 약간만 상승하지만 그만큼 Delayed ACK효과가 자주 나타나서 회선 상태가 원활해지는 효과를 얻을 수 있고, 0.0008초로 설정하면 Delayed ACK효과는 줄어들지만 그만큼 빠른 ACK전송의 이점인 CWND의 빠른 회복을 확인할 수 있다.

V. 기기 적용 실험

본 논문에서 제시한 알고리즘을 실제 사람들이 사용하는 기기인 안드로이드 스마트폰에도 적용하여 실험을 진행하였다.

5.1 실험환경 구축

본 실험은 안드로이드의 커널 수정을 통해 실제 데이터 통신 실험을 진행하였다. 먼저 커널 수정 없이 데이터 통신을 진행하여서 shell 메시지를 통해 segment가 수신될 때 마다 메세지를 표시도록 하였다. 그림 7에서 보이듯이 0.0003초 ~ 0.0009초 정도의 간

격으로 데이터를 수신 받고 있음을 알 수 있다. 해당 결과를 토대로 한 개의 안드로이드 스마트폰에는 새로운 Delayed ACK timeout값인 0.0004초로 설정하였으며, 다른 한 개의 안드로이드 스마트폰에는 기존의 0.2초를 적용하였다. 데이터 전송과 결과화면을 볼 수 있는 iperf 어플리케이션을 통해 무선 데이터 통신을 30초간 실행하였으며, 이때의 데이터의 전송량을 비교하였다.

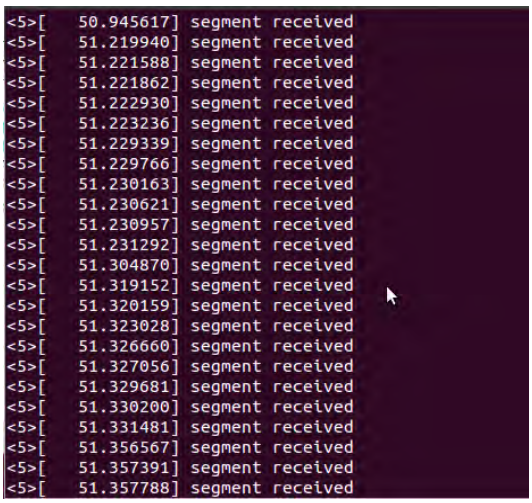

그림 7. 안드로이드의 shell 메시지 출력화면
Fig. 7. Output of android shell message

5.2 실험결과 분석

실험 결과는 그림 8과 같다. 두 개의 안드로이드 스마트폰에서의 데이터 전송량을 비교하였으며, 매 초마다 전송속도를 출력하도록 하여서 그래프를 그렸다. 안드로이드를 통한 실험에서도 데이터 재전송이 일어나는 부분에서 빠른 CWND회복을 통해 기존보다 더 빠르게 데이터 전송속도가 증가하는 것을 볼 수 있으며, 결과적으로 더 높은 전송속도를 보여주고 있다. 이러한 무선 통신을 총 5번 실험을 진행하였으며, 실험

표 1. 평균 전송속도 비교표
Table 1. A comparative table of Average throughput

Experiment	Original (Mbit/sec)	Adjusted (Mbit/sec)
①	7.04	8.43
②	8.01	8.29
③	7.19	8.40
④	7.33	8.45
⑤	7.00	7.85

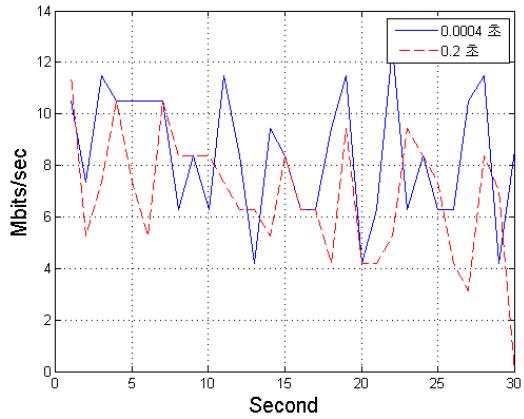

그림 8. 전송속도 변화량 측정 결과
Fig. 8. Result of throughput variation measurement

때 측정된 전송 속도의 평균값을 구하였다. 해당 결과값은 표 1에서 보이듯이 조정된 Delayed ACK timeout 값을 가질 때 더 빠른 전송속도를 보여주고 있다.

5.3 회선이 복잡한 환경에서의 실험

본 실험에서는 3개의 AP에 2대의 스마트폰으로 iperf를 사용한 통신을 진행하여 통신회선이 복잡한 상황을 만들었다. 이러한 환경에서 제 3번째 스마트폰을 연결하여 실험을 진행하였으며, 처음에는 기존의 스마트폰으로 통신을 하였고, 두 번째로는 환경에 맞춰서 조정된 새로운 Delayed ACK timeout값인 0.0006초가 적용된 스마트 폰으로 통신을 하여서 평균 전송속도를 측정하였다. 총 30초씩 4번 측정하였으며, 결과는 표 2와 같다. 다만 통신시의 전송 속도가 최소 1.73Mbit/sec에서 최대 12Mbit/sec까지 전송 속도의 변동이 있었으며, 이는 본 논문에서 제안한 알고리즘이 빠르게 ACK을 전송하는 특성이 있어서 그만큼 회선 복잡도가 높아진 것이 원인이라 생각한다. 하지만 이러한 통신회선이 복잡한 환경에서도 빠른 ACK 전송과 불필요한 지연을 없앰으로 써 더 높은 전송속도를 확보할 수 있었으며, 결과적으로 안정성을

표 2. 회선이 복잡한 환경에서의 평균 전송속도 비교표
Table 2. A comparative table of Average throughput in congestion environment

Experiment	Original (Mbit/sec)	Adjusted (Mbit/sec)
①	6.34	6.74
②	5.65	5.69
③	5.84	6.27
④	5.81	6.10

다소 희생하였어도 더 높은 전송속도를 얻을 수 있었다.

Ⅵ. 결 론

본 논문에서는 Delayed ACK timeout 수치를 조정하여서 전송속도 증가 효과를 가져 올 수 있음을 보였다. Delayed ACK에서 문제시 되었던 TCP 연결 직후와 재전송시의 CWND 회복속도 문제에 대해 본 알고리즘을 통해 빠른 CWND 회복으로 대응할 수 있었고, Delayed ACK timeout으로 인한 지연시간도 줄어들어서 장기적으로도 빠른 통신 속도를 얻을 수 있었다. 현재 Windows OS의 경우 Delayed ACK timeout 값은 0.2초로 고정되어 있지만, 각 segment의 간격에 따른 Delayed ACK Timeout의 변동 알고리즘이 추가된다면 데이터 통신이 발생할 때의 통신환경이 Delayed ACK 알고리즘에 반영되어 통신 속도의 개선이 가능할 것으로 예상된다.

References

[1] K. Chae, T. H. Nguyen, M. Park, and S. Jung, “A study on advanced TCP snoop algorithm considering the feature of network layer,” in *Proc. KICS Int. Conf. Commun.*, pp. 581-582, 2013.

[2] Y. Chen, et al., “Understanding TCP incast throughput collapse in datacenter networks,” in *Proc. 1st ACM Workshop on Research on Enterprise Networking(WREN ’09),* pp. 73-82, 2009.

[3] Microsoft, Design issues – Sending small data segments over TCP with Winsock(2014), Retrieved June., 29, 2014, from https://support.microsoft.com/en-us/kb/214397

[4] M. Lee, “One-way queuing delay mechanism for detecting the ACK losses in heterogeneous networks,” in *Proc. KICS Int. Conf. Commun.*, pp. 181-182, 2013.

[5] B.-H. Oh, S. Kim, and J. Lee, “Retransmission persistence management with ARQ in multi-hop wireless network,” in *Proc. KICS Int. Conf. Commun.*, pp. 674-683, 2014.

[6] M. Allman, S. Flayd, and C. Partidge, *Increasing TCP’s initial window*, RFC3390, Sept. 1998.

[7] M. Allman, *TCP congestion control with appropriate byte counting (ABC)*, RFC 3465, 2003.

[8] W. R. Stevens, *TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms*, RFC2001, Jan. 1997.

[9] J. Chen, et al., “TCP with delayed ack for wireless networks,” *Ad Hoc Networks,* vol. 6, no. 7, pp. 1098-1116, 2008.

[10] S. Ha and I. Rhee, “Hybrid slow start for high-bandwidth and long-distance networks,” in *Proc. PFLDnet,* 2008.

이 준 엽 (Joon Yeop Lee)

2014년 8월 : 고려대학교 전기전자전파공학부(공학사)

2014년 9월~현재 : 고려대학교 전기전자공학과 석사과정

<관심분야> 통신공학, 네트워크공학

이 웅 희 (Woonghee Lee)

2013년 2월 : 고려대학교 방사선학과(보건학사)

2013년 2월 : 고려대학교 전기전자전파공학부(공학사)

2013년 3월~현재 : 고려대학교 전기전자공학과 박사과정

<관심분야> 통신공학, 네트워크공학

김 황 남 (Hwangnam Kim)

1992년 3월 : 부산대학교 컴퓨터공학과(공학사)
1994년 2월 : 서울대학교 컴퓨터공학과(공학석사)
2004년 2월 : 미국 Urbana-Champaign 소재 Illinois 주립대학 컴퓨터과학과(공학박사)
1994~1999년 : LG 전자 주임연구원
2000~2001년 : Bytemobile 소프트웨어 엔지니어
2004~2005년 : 미국 Urbana-Champaign 소재 Illinois 주립대학 Post Doctorate Fellow
2005~2006년 : 삼성전자 책임연구원
2006~현재 : 고려대학교 전기전자전파공학부 교수
<관심분야> 통신공학, 네트워크공학, 융합IT, CPS