

Assignment #4

(Introduction to Computer Networks)

학번: 2013311659

이름: 곽 창 근

1. 개발 환경

OS : Windows10

사용 언어 : Python 3.7.0 64bit

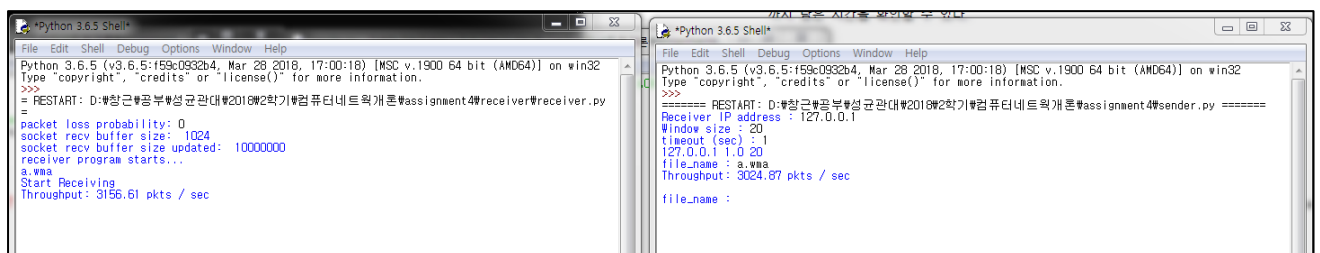
사용 프로그램 : Python 3.7.0 Shell

사용 라이브러리 : socket, threading, time, random, os

2. 설명 및 작동 방법

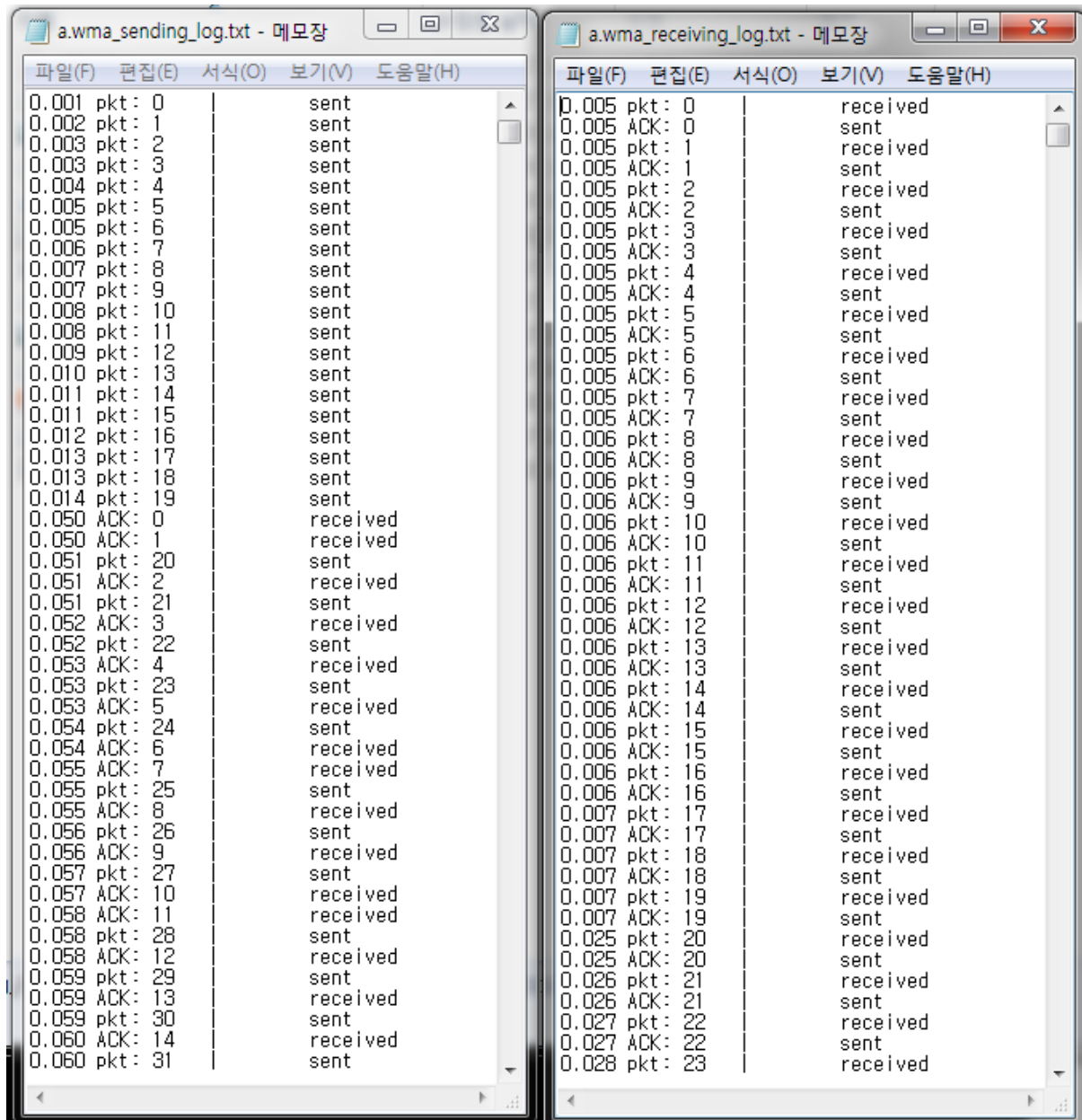
채점 기준 중, Allow concurrent file transfer 를 제외하고 모두 구현하였다. 채점 기준들을 중점으로 설명하겠다.

- 1) Packet loss 없을 때 window size 만큼 패킷 주고받고, ACK 받으면 window 이동해가며 보내기. 그리고 log file 기록



```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\참고문헌\공부\성균관대\2018\2학기\컴퓨터네트워크개론\Assignment4\receiver.py =====
packet loss probability: 0
socket recv buffer size: 1024
socket recv buffer size updated: 10000000
receiver program starts...
a.wna
Start Receiving
Throughput: 3156.61 pkts / sec
```

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\참고문헌\공부\성균관대\2018\2학기\컴퓨터네트워크개론\Assignment4\sender.py =====
Receiver IP address : 127.0.0.1
Window size : 20
timeout (sec) : 1
127.0.0.1 1.0 20
file_name : a.wna
Throughput: 3024.87 pkts / sec
file_name :
```



위 사진에서 보듯이 sender, receiver 파일 모두 문제 없이 잘 작동하고, 전송 결과로 sending_log, receiving_log 파일도 정상적으로 생성된 것을 볼 수 있다.

2) Goodput 계산하기

0.596 ACK: 1803	received	0.571 ACK: 1803	sent
0.597 pkt: 1804	sent	0.571 pkt: 1804	received
0.597 ACK: 1804	received	0.571 ACK: 1804	sent
0.597 pkt: 1805	sent	0.572 pkt: 1805	received
0.597 ACK: 1805	received	0.572 ACK: 1805	sent
0.597 pkt: 1806	sent	0.572 pkt: 1806	received
0.597 ACK: 1806	received	0.572 ACK: 1806	sent
0.597 pkt: 1807	sent	0.572 pkt: 1807	received
0.598 ACK: 1807	received	0.572 ACK: 1807	sent
0.598 pkt: 1808	sent	0.573 pkt: 1808	received
0.598 ACK: 1808	received	0.573 ACK: 1808	sent
0.598 pkt: 1809	sent	0.573 pkt: 1809	received
0.598 ACK: 1809	received	0.573 ACK: 1809	sent
0.598 pkt: 1810	sent	0.573 pkt: 1810	received
0.598 ACK: 1810	received	0.573 ACK: 1810	sent
0.599 pkt: 1811	sent	0.574 pkt: 1811	received
0.599 ACK: 1811	received	0.574 ACK: 1811	sent
File transfer is finished.		File transfer is finished.	
Throughput: 3024.87 pkts / sec		Throughput: 3156.61 pkts / sec	

위 사진에서 보듯이 로그 파일의 마지막에 goodput 계산한 값을 적게 하였다. Goodput 계산은 sender 와 receiver 에서 패킷을 주고받을 때 sequence 저장 용도로 사용한 변수 seq 을 이용해서 전송 끝난 시간에서 전송 시작 시간을 뺀 값을 나눠줘서 구하였다.

3) 주어진 Probability 에 따라 패킷 drop 시키기

0.058 ACK: 109	received	0.051 ACK: 108	sent
0.058 pkt: 110	sent	0.052 pkt: 109	received
0.058 ACK: 110	received	0.052 ACK: 109	sent
0.058 pkt: 111	sent	0.052 pkt: 110	received
0.059 pkt: 112	sent	0.052 ACK: 110	sent
0.059 ACK: 110	received	0.052 pkt: 111	received
0.059 pkt: 113	sent	0.052 pkt: 111	dropped
0.059 ACK: 110	received	0.053 pkt: 112	received
0.059 pkt: 114	sent	0.053 ACK: 110	sent
0.059 ACK: 110	received	0.053 pkt: 113	received
0.059 pkt: 110	3 duplicated ACKs	0.053 ACK: 110	sent
0.059 pkt: 111	sent	0.053 pkt: 114	received
0.060 ACK: 114	received	0.053 ACK: 110	sent
0.060 pkt: 115	sent	0.053 pkt: 111	received
0.060 ACK: 115	received	0.053 ACK: 114	sent

위의 사진은 드랍 probability 를 0.01 로 줬을 때 패킷 드랍이 일어난 부분을 스크린샷으로 찍은 것이다. 왼쪽이 sending_log, 오른쪽이 receiving_log 파일의 일부이다. 사진을 보면, sender 가 pkt 111 을 보냈으나 receiver 에서 이를 drop 하였기 때문에 그 후 다른 패킷을 받을 때에도 계속 ACK 110 을 보낸 결과 3 duplicated ACK 이 발생한 모습이다. 이는 아래의 코드를 통해 구현하였다.

```
#prob 값에 따른 패킷 드랍
if random.random() < prob :
    s = ( "%0.3f" % (cur_time - start_time) )
    print(s) #확인용
    f_log.write(s)
    continue
```

random.random()을 통해 [0, 1) 사이의 랜덤 값을 생성하고, 이와 비교를 통해서 입력해준 prob 확률만큼 drop 하였다. 그리고 continue 를 통해 정상적으로 패킷을 받았을 때의 과정을 진행하지 않고 넘어감으로써 drop 을 구현하였다.

4) Timeout, retransmit 구현하기

The screenshot shows a Python 3.6.5 Shell window with the following output:

```
>>>
===== RESTART: D:\창근\공부\성균관대\2018\2학기\컴퓨터네트워크개론\assignment4\sender.py =====
Receiver IP address : 127.0.0.1
Window size : 30
timeout (sec) : 0.05
127.0.0.1 0.05 30
file_name : a.wma
Throughput: 1301.65 pkts / sec
file_name : |
```

Below the shell window are two log files:

- a.wma_sending_log.txt - 메모장**

파일(F)	편집(E)	서식(O)	보기(V)	도움말(H)
0.002 pkt: 0	sent			
0.003 pkt: 1	sent			
0.003 pkt: 2	sent			
0.004 pkt: 3	sent			
0.005 pkt: 4	sent			
0.006 pkt: 5	sent			
0.006 pkt: 6	sent			
0.007 pkt: 7	sent			
0.008 pkt: 8	sent			
0.008 pkt: 9	sent			
0.009 pkt: 10	sent			
0.010 pkt: 11	sent			
0.011 pkt: 12	sent			
0.011 pkt: 13	sent			
0.012 pkt: 14	sent			
0.013 pkt: 15	sent			
0.014 pkt: 16	sent			
0.015 pkt: 17	sent			
0.015 pkt: 18	sent			
0.016 pkt: 19	sent			
0.017 pkt: 20	sent			
0.018 pkt: 21	sent			
0.019 pkt: 22	sent			
0.019 pkt: 23	sent			
0.020 pkt: 24	sent			
0.021 pkt: 25	sent			
0.022 pkt: 26	sent			
0.022 pkt: 27	sent			
0.023 pkt: 28	sent			
0.024 pkt: 29	sent			
0.052 ACK: 0	received			
0.052 pkt: 0	timeout since 0.002			
0.052 pkt: 0	retransmitted			
0.052 ACK: 1	received			
0.053 pkt: 1	sent			
0.053 ACK: 2	received			
0.053 pkt: 3	sent			
0.053 ACK: 3	received			
0.054 ACK: 4	received			
- a.wma_receiving_log.txt - 메모장**

파일(F)	편집(E)	서식(O)	보기(V)	도움말(H)
0.003 pkt: 0	received			
0.003 ACK: 0	sent			
0.004 pkt: 1	received			
0.004 ACK: 1	sent			
0.004 pkt: 2	received			
0.004 ACK: 2	sent			
0.004 pkt: 3	received			
0.004 ACK: 3	sent			
0.004 pkt: 4	received			
0.004 ACK: 4	sent			
0.004 pkt: 5	received			
0.004 ACK: 5	sent			
0.004 pkt: 6	received			
0.004 ACK: 6	sent			
0.004 pkt: 7	received			
0.004 ACK: 7	sent			
0.004 pkt: 8	received			
0.004 ACK: 8	sent			
0.004 pkt: 9	received			
0.004 ACK: 9	sent			
0.004 pkt: 10	received			
0.004 ACK: 10	sent			
0.004 pkt: 11	received			
0.004 ACK: 11	sent			
0.004 pkt: 12	received			
0.004 ACK: 12	sent			
0.004 pkt: 13	received			
0.004 ACK: 13	sent			
0.004 pkt: 14	received			
0.004 ACK: 14	sent			
0.005 pkt: 15	received			
0.005 ACK: 15	sent			
0.005 pkt: 16	received			
0.005 ACK: 16	sent			
0.005 pkt: 17	received			
0.005 ACK: 17	sent			
0.005 pkt: 18	received			
0.005 ACK: 18	sent			
0.005 pkt: 19	received			

위의 사진은 timeout 이 발생, 그 결과 retransmit 해준 것을 기록한 사진이다. sender 에서 timeout = 0.05 로 설정하였기 때문에 0.05 초가 지났음에도 ACK 0 이 오지 않자 pkt 0 에 대해서 timeout 이 언제를 기준으로 발생했는지 기록, 재전송 해주고 있다. 이는 아래의 코드를 통해 구현하였다.

```
# timeout일 때
if cur_time - check_time > timeout :
    #커서 위치 옮김 + 패킷 전송
    f_send.seek(bufsize*(check_ack+1), os.SEEK_SET)
    chunk = f_send.read(bufsize)
    seq = check_ack+1 # seq위치도 옮겨줌
    if len(chunk) < 1 :
        if check_ack == seq-1 :
            break
        else :
            continue
    # 전송
    s = "file=%s;seq=%d" % (file_name, seq)
    s_socket.sendto( s.encode(), (receiverIP, receiverPort) )
    s_socket.sendto( chunk, (receiverIP, receiverPort) )
    # log 기록
    f_log = open(log_name, 'a')
    s = ( "%0.3f" % (cur_time - start_time) ) + " pkt: " + str(
    print(s)#확인용
    f_log.write(s)
    s = ( "%0.3f" % (cur_time - start_time) ) + " pkt: " + str(
    print(s)#확인용
    f_log.write(s)
    f_log.close()
    # time 초기화
    timeline.clear()
    check_time = time.time()
    #timeline에 해당 패킷 보낸시간 저장
    timeline[seq] = cur_time
    seq += 1
    dup_ack = 0
    wanted_ack = check_ack+1
    continue
```

while 문 처음에 cur_time = time.time()을 통해 시간을 기록하였고, check_time 은 해당 패킷의 전송시간이다. 이 둘을 뺀 값이 timeout 보다 크면 timeout 발생, 해당 if 문을 실행한다. timer 는 1 개를 가지고 체크하며, timeline 이라는 dict 를 만들어서 패킷을 보낼 때마다 해당 패킷과 보낸 시간을 기록한다. 이렇게 해서 timer 1 개를 가지고도 구현을 할 수 있었다.

5) Duplicated ACK 구현하기

4.090 pkt: 13	sent	4.084 ACK: 13	sent
4.090 ACK: 13	received	4.084 pkt: 25	received
4.090 pkt: 13	3 duplicated ACKs	4.084 ACK: 13	sent
4.090 pkt: 14	sent	4.084 pkt: 14	received
4.091 ACK: 14	received	4.084 ACK: 14	sent
4.091 pkt: 15	sent	4.085 pkt: 15	received
4.091 pkt: 16	sent	4.085 pkt: 15	dropped
4.091 pkt: 17	sent	4.085 pkt: 16	received
4.092 pkt: 18	sent	4.085 pkt: 16	dropped
4.092 pkt: 19	sent	4.085 pkt: 17	received
4.092 pkt: 20	sent	4.085 pkt: 17	dropped
4.092 ACK: 14	received	4.086 pkt: 18	received
4.093 pkt: 21	sent	4.086 pkt: 18	dropped
4.093 ACK: 14	received	4.086 pkt: 19	received
4.093 pkt: 22	sent	4.086 pkt: 19	dropped
4.093 pkt: 23	sent	4.086 pkt: 20	received
4.094 pkt: 24	sent	4.086 ACK: 14	sent
4.094 ACK: 14	received	4.086 pkt: 21	received
4.094 pkt: 14	3 duplicated ACKs	4.086 ACK: 14	sent
4.094 pkt: 15	sent	4.087 pkt: 22	received
4.094 ACK: 15	received	4.087 pkt: 22	dropped
4.094 pkt: 16	sent	4.087 pkt: 23	received
4.094 ACK: 16	received	4.087 pkt: 23	dropped

위 사진을 보면 ACK 14 를 받고, 그 후에도 ACK 14 를 연속으로 3 번 받은 결과 3 duplicated ACK 이 발생한 것을 볼 수 있다. 이는 아래의 코드를 통해 구현하였다.

```
# 3 duplicated ack
if dup_ack >= 3 :
    #커서 위치 옮김 + 패킷 전송
    f_send.seek(bufsize*(check_ack+1), os.SEEK_SET)
    chunk = f_send.read(bufsize)
    seq = check_ack+1 # seq위치도 옮겨 줌
    if len(chunk) < 1 :
        if check_ack == seq-1 :
            break
        else :
            continue
    # 전송
    s = "file=%s;seq=%d" % (file_name, seq)
    s_socket.sendto( s.encode(), (receiverIP, receiverPort) )
    s_socket.sendto( chunk, (receiverIP, receiverPort) )
    # log 기록
    f_log = open(log_name, 'a')
    s = ( "%0.3f" % (cur_time - start_time) ) + " pkt: " + str(s)
    print(s)#확인용
    f_log.write(s)
    s = ( "%0.3f" % (cur_time - start_time) ) + " pkt: " + str(s)
    print(s)#확인용
    f_log.write(s)
    f_log.close()
    # time 초기화
    timeline.clear()
    check_time = time.time()
    #timeline에 해당 패킷 보낸시간 저장
    timeline[seq] = cur_time
    seq += 1
    dup_ack = 0
    wanted_ack = check_ack+1
    continue
```

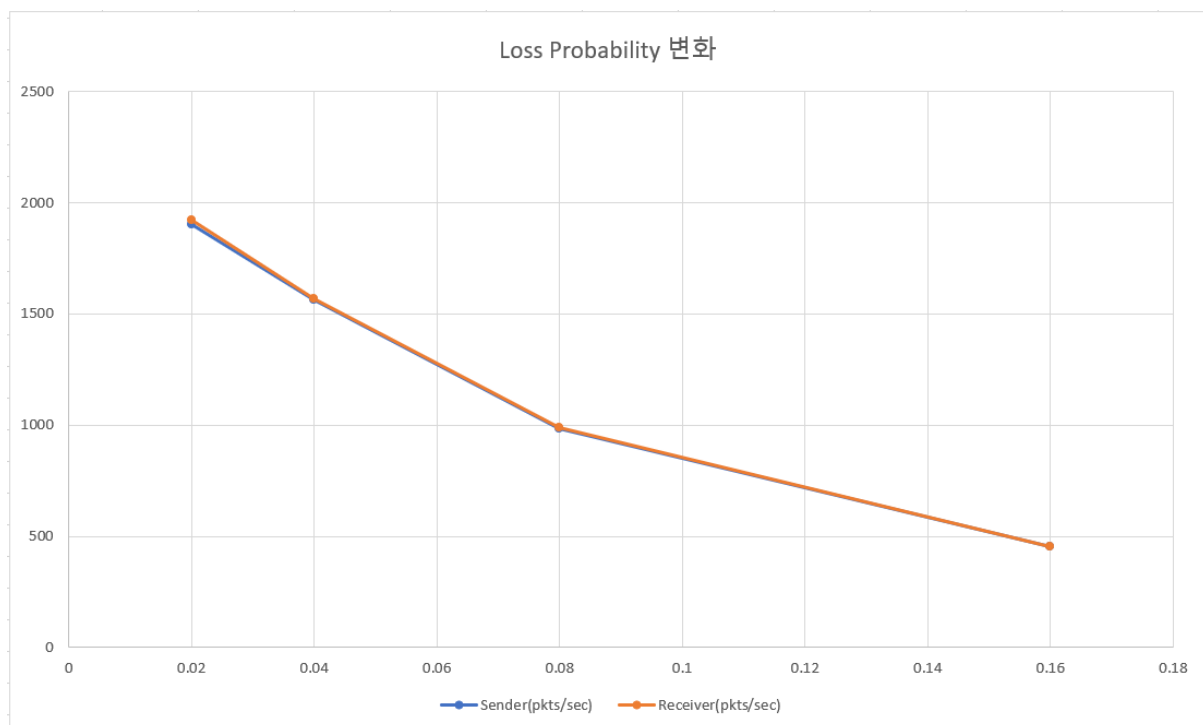
dup_ack 을 통해 ack 이 몇번 중복되서 왔는지를 체크하고, 이 값이 3 보다 크면 if 문을 실행해서 duplicated ACK 이벤트를 발생시킨다. ACK 을 확인해서 dup_ack 값을 증가시키는 것은 ACK 을 받는 스레드에서 중복 ACK 을 감지할 때마다 값을 1 씩 증가시켜준다.

6) Window size, probability 에 따라 달라지는 goodput 그래프 그리기

Goodput 측정은 10MB 파일을 갖고서 5 번 측정하여 평균을 내었다.

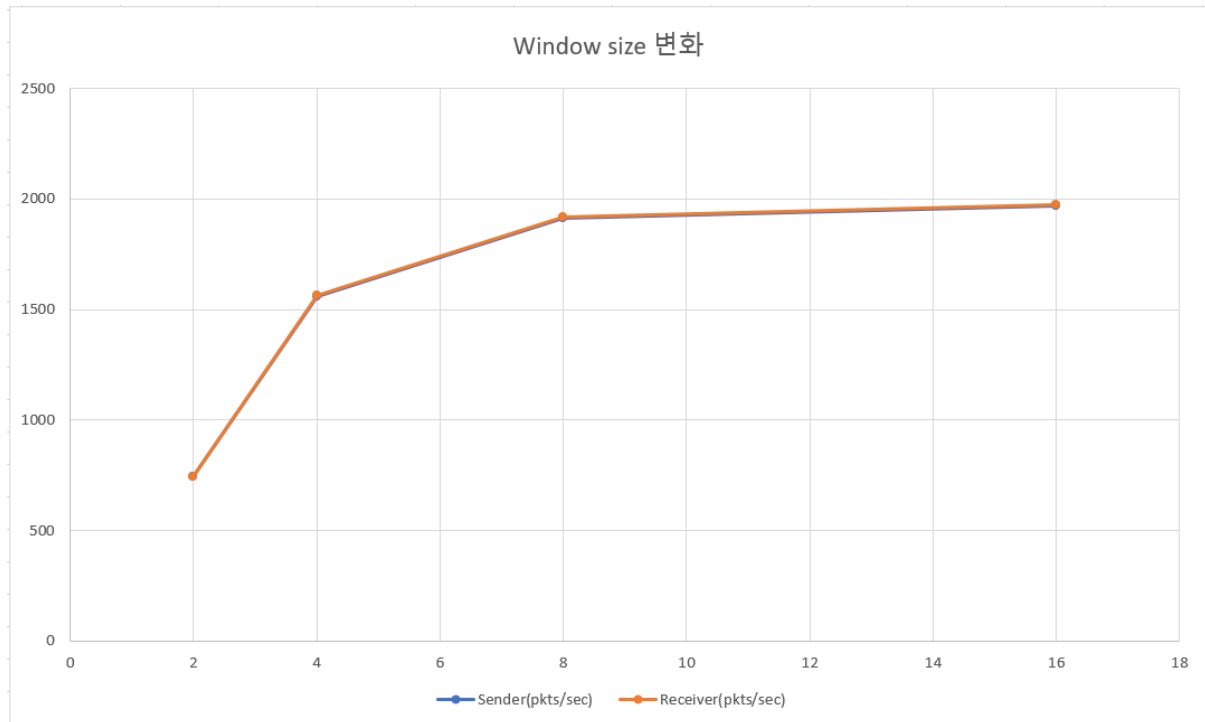
i) window size = 8, timeout = 0.05 일 때 loss probability 변화에 따른 그래프

Loss probability	0.02	0.04	0.08	0.16
Sender(pkts/sec)	1901.75	1561.11	981.66	453.14
Receiver(pkts/sec)	1920.72	1567.51	988.59	454.26



ii) loss probability = 0.02, timeout = 0.05 일 때 window size 변화에 따른 그래프

Loss probability	2	4	8	16
Sender(pkts/sec)	741.94	1560.98	1914.81	1970.45
Receiver(pkts/sec)	742.65	1562.66	1919.95	1975.57



7) 추가 설명

추가적으로 코드 이해를 위해 필요하다 생각하는 내용을 설명하겠다.

- Receiver

하나의 while 문을 계속 돌면서 filename 에 따른 fileReceive 함수를 실행하여 파일을 받는다.

tempbuf 는 drop 발생 시 나중 파일들을 미리 저장해놓기 위한 가변 배열이다. 배열에 값 저장은 addTempbuf 함수를 통해 이뤄지며, (패킷 넘버, 패킷)의 튜플 형태로 저장한다. 저장한 패킷들은 아래 코드를 통해 사용된다. 이미 저장된 패킷은 버리고, drop 된 패킷 받게되면 이어서 file write 할 수 있는 패킷들은 tempbuf 에서 꺼내어 file write 한다.


```

#ACK 보내주기
if seq == r_seq :
    f_rec.write(message)
    seq += 1
# f_rec.write한 패킷들은 버리기
while tempbuf != [] and tempbuf[0][0] < seq :
    print("del %d, len=%d" % (seq, len(tempbuf)))
    del tempbuf[0]
# tempbuf에 저장된 패킷 저장
while tempbuf != [] and tempbuf[0][0] == seq :
    print("write %d, len=%d" % (seq, len(tempbuf)))
    f_rec.write(tempbuf[0][1])
    del tempbuf[0]
    seq += 1

```

sender 와 통신하는 것은 2 개씩 주고받으면서 하도록 구현하였는데, 첫번째는 파일이름, seq 정보, 마지막 패킷 정보 등을 주고받고, 두번째는 파일 저장을 위한 패킷을 주고 받는다. 마지막 패킷 정보는 seq = -2 가 오면 전송이 끝난다는 뜻이다.

ACK 은 현재 받아야하는 seq 에 맞는 패킷을 받을 경우 보내준다.

- Sender

Receiver 와 마찬가지로 주된 하나의 while 문을 돌면서 파일 이름을 입력해주면 파일 전송을 시작한다. 프로그램을 끝내고 싶을 때는 'quit' 또는 'exit'를 입력하면 된다. 파일 이름이 입력되면 fileSend 함수를 실행하여 파일을 전송한다.

global 변수 ack, dup_ack 을 사용하는데, 이는 fileSend 내에서 실행한 스레드 함수와 변수를 공유하기 위함이다. 초기화를 제외하고 이 값들은 스레드로 실행한 함수 getACK 에서 변경하며, 이를 보고 패킷을 정상적으로 받았는지, duplicated ACK 이 발생했는지 등등을 판단한다.

fileSend 함수 내에서 while 문을 돌면서 하나의 파일을 모두 전송하는데, 이 while 문을 돌 때마다 cur_time 을 업데이트 해주어서 timeout 을 판별한다.

while 문 초반부에 check_ack 을 업데이트 해주는 것은 스레드에 의해서 ack 값이 while 문을 한 번 처음부터 끝까지 실행하는 도중에 변경되면 안되기 때문에 check_ack 에 저장하고, 이를 ack 대신 이용하여 확인하는 것이다. check_ack 을

이용하는 이유가 하나가 더 있는데, check_ack 과 ack 값이 달라졌다는 것은 리시버가 패킷을 받았다는 뜻이기에 timer 를 업데이트 하는 용도로도 사용된다.

wanted_ack 을 이용하는 것은 duplicated ACK 이나 timeout 발생 시, 불필요하게 중복적으로 이 이벤트가 발생하는 것을 막기 위해서이다.

```
# 현재 seq보다 ack이 크면 seq를 ack+1로 옮겨 줌
if check_ack >= seq :
    f_send.seek(bufsize*(check_ack+1), os.SEEK_SET)
    seq = check_ack + 1
```

위 코드는 receiver 에서 tempbuf 에 저장해놔던 패킷을 이용하여 ACK 갑자기 증가할 경우 파일 커서의 위치를 보정해주기 위함이다.

```
# seq가 window 넘어가면 멈춤
if seq - check_ack > window :
    #print("Pause for window.")
    continue
```

위 코드를 통해 window size 를 구현하였다. 윈도우 사이즈를 넘어가면 continue 를 통해 업데이트 하는 코드 부분을 실행하지 않고 있다.

```
# 파일 마지막까지 다 보냈을 경우
if len(chunk) < 1 :
    if check_ack == seq-1 :
        break
    else :
        continue
```

위 코드를 통해 파일을 다 보냈을 경우 receiver 로부터 마지막 패킷의 ACK 까지 받아야 비로소 while 문을 빠져나온다.

스레드에서 사용되는 getACK 함수는 따로 소켓을 열어서 receiver 의 메시지를 받는다. ACK 을 받기 위한 용도로 사용된다.