Introduction to Computer Networks

Assignment 3: Pipelined Reliable Data Transfer over UDP

소프트웨어학과 김승현

2014310279

1. Development environments

```
(base) ➜  asmt1 sw_vers
ProductName:    Mac OS X
ProductVersion: 10.14.6
BuildVersion:   18G87
(base) ➜  asmt1 python -V
Python 3.7.1
```

2. How to Run

a. ‘sender.py’, ‘receiver.py’ 압축 해제 후 다른 머신으로 receiver.py 이동

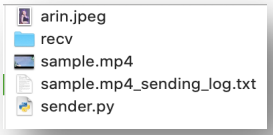
b. ‘receiver.py’ 실행 후 loss probability 입력

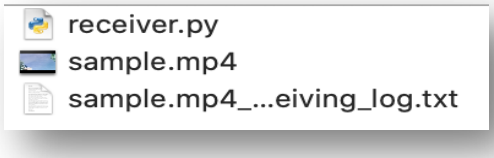
c. ‘sender.py’ 실행 후 IP address(receiver), window size, timeout, file_name 입력

```
(base) ➜  recv python3 receiver.py
packet loss probability: 0.02
socket recv buffer size:  9216
socket recv buffer size updated:  1000000

receiver program starts...
```

```
(base) ➜  asmt3 python3 sender.py
Receiver IP address : 127.0.0.1
widnow size: 16
timeout (sec): 0.05

file_name: sample.mp4
```

d. 결과 확인





sample.mp4_sending_log.txt

```
24.303 pkt: 20524       | Sent
24.303 pkt: 20525       | Sent
24.303 ACK: 20511       | received
24.303 pkt: 20526       | Sent
24.303 ACK: 20511       | received
24.303 pkt: 20527       | Sent
24.303 ACK: 20511       | received
24.303 pkt: 20511       | 3 duplicated ACKs
24.303 pkt: 20512       | retransmitted
24.333 ACK: 20511       | received
24.350 pkt: 20513       | timeout since 24.300
24.350 ACK: 20511       | received
24.351 ACK: 20511       | received
24.350 pkt: 20513       | retransmitted
24.351 ACK: 20511       | received
24.351 ACK: 20511       | received
```

sample.mp4_receiving_log.txt

```
0.065 ACK: 113  | sent
0.065 pkt: 114  | received
0.065 ACK: 114  | sent
0.066 pkt: 115  | arrived but dropped
0.066 pkt: 116  | received
0.066 ACK: 114  | sent
0.066 pkt: 117  | received
0.066 ACK: 114  | sent
0.066 pkt: 118  | received
0.066 ACK: 114  | sent
0.066 pkt: 119  | arrived but dropped
0.067 pkt: 120  | received
0.067 ACK: 114  | sent
0.067 pkt: 121  | received
0.067 ACK: 114  | sent
0.067 pkt: 122  | received
```

3. Design – Sender.py

Receiver에 0| 1| 2| 가 header로 포함된 msg를 보내며 0|은 data 전송의 시작을 나타내며 파일명(file_name), 패킷 총 수(packet_num), 시작 시간(start_t)의 정보가 포함되어있다.

1|은 packet의 data를 실제로 보내는 message이며 packet의 시퀀스 넘버(seq_num)를 포함하고 있다

2|는 packet 전송이 완료 되었음을 나타낸다.

a. Main()

```
packet_num = len(packet)
packet_timer = [0] * packet_num
start_t = time.time()
msg = "0¦" + file_name + "¦" + str(packet_num) + "¦" + str(start_t)
send_socket.sendto(msg.encode(), receiver_addr)

log_name = file_name + '_sending_log.txt'
flog = open(log_name, 'w+')
send_thread = threading.Thread(target = send_packet, args = (file_name, packet_num,
recv_thread = threading.Thread(target = recv_ack, args = (file_name, packet_num, rec
```

Parameter를 입력받은 후 file을 읽어 packet으로 나누어 저장한다. 입력 받은 정보를 0| 헤더를 포함하여 파일에 대한 정보를 receiver에게 보내고 Thread로 send_packet, recv_ack함수를 실행한다.

b. Send_packet()

```
global packet, packet_timer, start_t, done, nextseqnum, base, log_name
while True:
    with threading.Lock():
        now = time.time() - start_t
        tmp_window_size = (base + window_size) if (base + window_size) <= packet_num e
        if (nextseqnum < tmp_window_size) :
            msg = msg_preprocess(nextseqnum)
            packet_timer[nextseqnum] = now
            logger("{:.3f} pkt: {}\t¦ Sent\n".format(now, nextseqnum), log_name)
            send_socket.sendto(msg + packet[nextseqnum], receiver_addr)
            nextseqnum += 1

        for i in range(base, nextseqnum):
            time_after_send = now - packet_timer[i]
            if (packet_timer[i] != 0 ) and (time_after_send > time_out):
                msg = msg_preprocess(i)
                logger("{:.3f} pkt: {}\t¦ timeout since {:.3f}\n".format(now, i, packe
                logger("{:.3f} pkt: {}\t¦ retransmitted\n".format(now, i), log_name)
                send_socket.sendto(msg + packet[i], receiver_addr)
                packet_timer[i] = now
    if done:
        return
```

Base를 기준으로 window size내의 packet을 시퀀스 순서대로 receiver에 보내며, 전송시간 packet_timer에 기록한다, packet_timer를 체크하여 timeout 여부를 확인하여 timeout 발생 시 해당 packet을 재전송한다. 전송이 완료 되었는지는 recv_ack함수로부터 done 변수가

변경되었는지를 통해 확인하여 전송이 완료 되었을시 함수를 종료한다.

c. Recv_ack()

```
def recv_ack(file_name, packet_num, receiver_addr, window_size):
    global packet, packet_timer, done, nextseqnum, base, ack, log_name

    while True:
        with threading.Lock():
            msg, receiver_addr = send_socket.recvfrom(4096)
            now = time.time()-start_t
            header = msg[:6].decode()
            ack = int(header.split("¦")[0])
            logger("{:.3f} ACK: {}\t¦ received\n".format(now, ack), log_name)

            if base == ack:
                packet_timer[ack] = 0
```

receiver로부터 받은 ack를 확인하여 알맞은 ack가 들어온 경우 해당 packet의 timer를 정지시킨다.

```
if ack+1 == packet_num:
    msg = "2¦"
    send_socket.sendto(msg.encode(), receiver_addr)
    done = True
    now = time.time() - start_t
    logger('\n', log_name)
    logger("File transfer is finished\n", log_name)
    logger("Throughput: {:.2f} pkts / sec\n".format(packet_num/now), log_name)
    return

if base <= ack:
    diff = ack - base
    for i in range(diff+1):
        packet_timer[base] = 0
        base +=1
    dup_cnt = 0

elif ack < base:
    dup_cnt += 1
    if dup_cnt == 3:
        msg = msg_preprocess(ack+1)
        logger("{:.3f} pkt: {}\t¦ 3 duplicated ACKs\n".format(now, ack), log_name)
        logger("{:.3f} pkt: {}\t¦ retransmitted\n".format(now, ack+1), log_name)
        send_socket.sendto(msg + packet[ack+1], receiver_addr)
        packet_timer[ack+1] = now
```

마지막 ack를 받으면 전송이 끝났다는 message를 receiver에 전송하고 함수를 종료한다. 그 외 ack가 base이상인 경우(ack이하의 모든 패킷이 전송된 정상적인 경우) base를 ack에 맞춰 update해주며, ack가 base보다 작은 경우 (packet loss가 발생한 경우) dup_cnt를 +1씩 해주어 dup_cnt가 3이 되면 ack+1에 해당하는 packet을 재전송 한다.

4. Design – Receiver.py

Packet, buffer를 list에 저장하며 (index = packet_num) ack, cum_ack 두가지 변수를 이용하여 ack를 관리한다. Ack는 sender에게 전송할 ack를 나타내며, cum_ack는 현재 까지 받은 가장 큰 seq넘버를 기록한다. Message를 받으면 header를 조사하여 0이면 packet과 buffer를 초기화하고 log를 기록할 파일을 열며, 2면 파일을 열어 지금까지 받은 pakcet의 data를 기록하여 파일 복사를 완료한다.

```
if (p < packet_loss):
    logger("{:.3f} pkt: {:d}\t¦ arrived but dropped \n".format(now, seq), log_name)
else:
    now = time.time() - start_t
    if seq == ack+1 :
        packet[seq] = data
        if ack == cum_ack:
            cum_ack +=1
        ack += 1
        if ack < cum_ack:
            if None not in recv_buffer[ack+1: cum_ack+1]:
                for i in range(ack+1, cum_ack+1):
                    packet[i] = recv_buffer[i]
                ack = cum_ack
        logger("{:.3f} pkt: {:d}\t¦ received\n".format(now, seq), log_name)
        logger("{:.3f} ACK: {:d}\t¦ sent\n".format(now, ack), log_name)
        msg = b'%5d'%(ack) + "¦".encode()
        receiver_socket.sendto(msg, sender_addr)
    elif seq > ack+1:
        recv_buffer[seq] = data
        cum_ack = seq
        logger("{:.3f} pkt: {:d}\t¦ received\n".format(now, seq), log_name)
        logger("{:.3f} ACK: {:d}\t¦ sent\n".format(now, ack), log_name)
        msg = b'%5d'%(ack) + "¦".encode()
        receiver_socket.sendto(msg, sender_addr)
```

P는 0과 1사이의 랜덤한 수로 packet을 받을 때마다 초기화 되며, 이를 통해 일정한 확률로 packet loss가 발생한다. 이후 받는 packet의 시퀀스 넘버를 확인하는데

1) seq가 ack+1 과 같을 경우 (정상적인 경우)

packet에 데이터를 저장하고 ack와 cum_ack를 저장한다.

a. Cum_ack이 ack보다 클 경우 (중간에 loss가 발생해 data가 버퍼에 들어가 있는 경우)

Ack를 cum_ack에 맞게 업데이트 해주면서 버퍼의 데이터를 packet으로 옮긴다

2) Seq가 ack+1보다 큰 경우 (loss가 발생하는 경우)

Cum_ack만 seq에 맞게 업데이트 해주고, 데이터는 버퍼에 입력한다.

위와 같은 알고리즘을 통해 cum_ack는 항상 최신 seq_num과 동일하게 유지되며, loss가 발

생한 packet은 ack를 통해 확인할 수 있다. 따라서 ack와 cum_ack을 비교하여 발생한 loss에 대해 buffer를 확인하여 data를 복구할 수 있다.

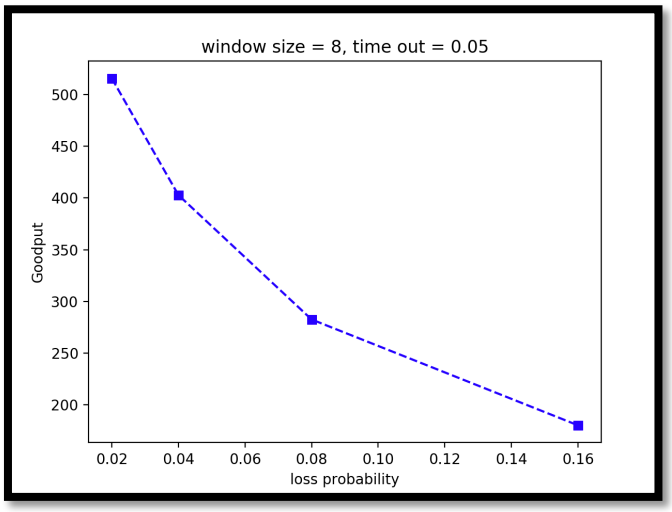
5. Concurrent file transfer

Concurrent file tranfer는 구현하지 못하였다.

6. Experimentation results(file size = 21.1MB)

a. window size = 8, time out = 0.05

Loss proabability	0.02	0.04	0.08	0.16
Goodput (pkts / sec)	515.72	402.85	282.43	180.02



b. loss probability = 0.02, time out = 0.05

Window size	2	4	8	16
Goodput (pkts / sec)	83.36	262.73	531.65	819.84