컴퓨터 네트워크 과제 3 report

2014313015 진승우

1. 작업 환경

- windows 7 professional K
- python language
- python(3.7.0)
- python(3.7.0) interpreter
- IE version 9

2. 소스 코드

(1) sender

```
from socket import *
import threading
from struct import *
import time
```

- 제가 사용한 라이브러리들은 다음과 같습니다.
socket, threading, time, 그리고 packet에 들어갈 헤더 정보들을 담기 위해 struct 라이브러리를 사용하였습니다.

```
if __name__ == '__main__':
    IP = input('Receiver IP address: ')
    window_size = int(input('window size: '))
    time_out = float(input('timeout (sec): '))
    #IP = '127.0.0.1'
    #window_size = 100
    #time_out = 0.5
    print('')

    thread_lock = threading.Lock()
    packet_data = dict()
    packet_state = []
    packet_timeout = dict()


    src_name = input('file name: ')
    #src_name = 'Wildlife.wmv'
    src_file = open(src_name, 'rb')
    chunk_size = 1024
    file_size = 0
    order = 0
    while True:
        file_size_indicator = src_file.read(1380)
        if not file_size_indicator:
            break
        file_size += len(file_size_indicator)
        packet_data[order] = file_size_indicator
        order += 1
    src_file.close()
    #print(order)
    log_file = open(src_name + '_sending_log.txt','w')
    log_file.close()
```

다음은 sender의 main 부분입니다.

IP, window_size, time_out을 input으로 받고 file_size_indicator라는 변수에 src_file로부터 데이터를 읽어와 저장하였습니다.

그리고 logfile을 초기화시켜주기위해 'w' 형태로 open하고 바로 닫도록 구현하였습니다.

```
serverPort = U
serverSocket = socket(AF_INET,SOCK_DGRAM)
serverSocket.bind(("",serverPort))

str_order = str(order)

data = bytearray(src_name, encoding = 'ascii') + bytearray('|', encoding = 'ascii') + bytearray(
data_packet = pack('>HHLLHHL',0, 10080, 0, 0, 0, window_size, 0)
data_packet = data_packet + data
serverSocket.sendto(data_packet, (IP, 10080))

t1 = threading.Thread(target=ack_receiver , args = (serverSocket, order, packet_data,window_size
t2 = threading.Thread(target=send, args = (serverSocket, IP, order, packet_data, window_size, sr
t3 = threading.Thread(target=timeout_check, args = (serverSocket, order, time_out, packet_data,

t1.start()
t2.start()
t3.start()
```

main의 나머지부분은 다음과 같습니다. 첫 패킷에 보낼 파일에 대한 이름과 window_size를 담아 receiver단으로 전송하였습니다. 아래의 각 thread 호출은 바로 다음 설명드리겠습니다.

```
def timeout_check(serverSock, order, timeout, packet_data, IP, window_size, src_name):
    global oldest, send, packet_state, packet_timeout, start
    while True:
        with thread_lock:
            if oldest == order :
                break
            elif oldest == send :
                continue
            elif (time.time() - float(packet_timeout[oldest])) >= float(timeout) :
                ta = threading.Thread(target = log_file_write, args = (src_name, oldes
                ta.start()
                ta.join()
                #log_file_write(src_name, oldest, time.time() - start, 4)
                packet_retransmit = pack('>HHLLHHL', 0, 10080, oldest, 0, 0, window_si
                packet_retransmit = packet_retransmit + packet_data[oldest]
                serverSocket.sendto(packet_retransmit, (IP, 10080))
                tb = threading.Thread(target = log_file_write, args = (src_name, oldes
                tb.start()
                tb.join()
                #log_file_write(src_name, oldest, time.time() - start, 3)
                packet_timeout[oldest] = time.time()
```

첫 번째로, timeout_check 쓰레드는 sender가 패킷을 보낼때마다, timeout이라는 array변수에 보낸 시간을 저장하게되고, 그것을 현재 시간에서 뺄셈을 하여 timeout 여부를 if문에서 판단한 뒤에 재전송 여부를 결정하였습니다.

```
def ack_receiver(serverSocket, order, packet_data, window_size, IP, src_
    global oldest, send, packet_state, packet_timeout, start
    ack = serverSocket.recvfrom(1400)
    while True:
        data = serverSocket.recv(1400)
        x, x, ack_num, x = unpack(">LLLH", data[:14])
        tc = threading.Thread(target = log_file_write, args = (src_name,
        tc.start()
        tc.join()
        #log_file_write(src_name, ack_num, time.time() - start, 0 )
        with thread_lock:
            packet_state[ack_num] += 1
            oldest = ack_num + 1
            if oldest > send :
                send = oldest
            if packet_state[ack_num] == 3 :
                packet_retransmit = pack('>HHLLHHL', 0, 10080, oldest, 0
                packet_retransmit = packet_retransmit + packet_data[olde
                serverSocket.sendto(packet_retransmit, (IP, 10080))
                packet_state[ack_num] = 0
                td = threading.Thread(target = log_file_write, args = (s
                td.start()
                td.join()
                #log_file_write(src_name, oldest, time.time() - start, 2

        if oldest == order :
            tk = threading.Thread(target = log_file_write, args = (src_n
            tk.start()
            tk.join()
            break
```

ack_receiver는 다음과 같습니다. ack만을 받기위한 소켓을 생성하여 만약 ack가 제대로 들어왔다고하면, oldest를 1 증가시킵니다. 여기서 oldest 변수는 보냈거나 보낼 패킷 중 가장 작은 sequence number를 뜻합니다. 그렇게 oldest 변수를 증가시켜 몇 번 패킷까지 보내졌는지 판단하고 window를 sliding 시킬 것입니다. 도중에, 만약 같은 패킷이 3번 오게되면 timeout과 무관하게 재전송하는 3 duplicate ACK 기법을 사용하였습니다.

```
def send(serverSocket, IP, order, packet_data, window_size, src_nam
    global oldest, send, packet_state, packet_timeout, start
    oldest = 0
    send = 0
    start = 0
    packet_state = []
    packet_timeout = dict()
    while True:
        while send - oldest < window_size and oldest != order :
            with thread_lock:
                packet = pack('>HHLLHHL', 0, 10080, send, 0, 0, win
                packet += packet_data[send]
                serverSocket.sendto(packet, (IP, 10080))
                if start == 0:
                    start = time.time()
                te = threading.Thread(target = log_file_write, args
                te.start()
                te.join()
                #log_file_write(src_name, send, time.time() - start

                packet_timeout[send] = time.time()
                #print(f"{time.time() - start:.3f} pkt: {send} Send
                packet_state.append(0)
                send += 1
                if send == order :
                    break
        if send == order:
            break
```

다음은 send 쓰레드입니다. send는 이름 그대로 window_size내에 포함되는 패킷을 receiver로 전송시키는 쓰레드입니다.

```
def log_file_write(src_name, num, time, sent) :
    #with thread_lock:
        #log_file = open(src_name + '_sending_log.txt','a')
        while True:
            try:
                log_file = open(src_name + '_sending_log.txt','a')
                break
            except PermissionError :
                continue
        delta = "%0.3f"% float(time)
        if sent == 0 :
            log_write = str(str(delta) + " ACK: " + str(num) +" | recei
        elif sent == 1:
            log_write = str(str(delta) + " pkt: " + str(num) +" | sent₩
        elif sent == 2 :
            log_write = str(str(delta) + " pkt: " + str(num) +" | 3 dup
        elif sent == 3:
            log_write = str(str(delta) + " pkt: " + str(num) +" | retra
        elif sent == 4:
            log_write = str(str(delta) + " pkt: " + str(num) +" | timeo
        elif sent == 5:
            goodput = "%0.3f"% (num/float(time))
            log_write = "₩n" + "File transfer is finished." + "₩n" + "T

        if not log_write == 'ed' :
            log_file.write(log_write)
        log_file.close()
```

그리고 중간중간 log_write를 하는 쓰레드가 보이셨을텐데, 이 부분은 sending_log_file에 받은 패킷이나 보낸 패킷을 쓰도록 하였습니다.

(2) receiver 부분

```
from socket import *
import threading
import time
from struct import *
import random
import queue
```

라이브러리는 socket, threading, time, struct, random, queue를 사용하였습니다.
queue같은 경우는 받은 패킷들을 잠시동안 맡아두는 버퍼역할로 사용하였습니다.

```
if __name__ == '__main__':
        probability = 10.0
        while probability < 0.0 or probability > 1.0:
            probability = float(input('Packet loss probability: '))
        buffer_size = int(input('Socket recv buffer size: '))
        if buffer_size <= 1000000:
            buffer_size = 1000000
        thread_lock = threading.Lock()
        print('socket recv buffer size updated: ' + str(buffer_size))
        print('')
        print('receiver program starts...')
    #while True:
        clientSocket = socket(AF_INET, SOCK_DGRAM)
        clientSocket.bind(("127.0.0.1",10080))
        data, clientAddress = clientSocket.recvfrom(buffer_size)
        window_size, x = unpack('>HL', data[14:20])
        window_size = int(window_size)
        receive_data = data[20:].decode('ascii')
        des_name = receive_data.split('|')[0]
        order = int(receive_data.split('|')[1])
        #print(order)
        log_file = open(des_name + '_receving_log.txt','w')
        log_file.close()
        ack_data = pack('>HHLLHHL', 10080, 0, 0, 0, 0, window_size, 0)
        clientSocket.sendto(ack_data, clientAddress)

        receive_queue = queue.Queue()
        queue_size = buffer_size

        t2 = threading.Thread(target = packet_process , args = (clientSocket, clientAddress , des_name,
        t1 = threading.Thread(target = packet_capture , args = (clientSocket, probability, order, buffe

        t1.start()
        t2.start()
```

receiver의 main함수부분도 sender와 다르지 않습니다. sender의 첫 패킷들을 받아서 초기화 해주고 log_file또한 초기화해주었습니다.

```
def packet_capture(clientSocket, probability, order, buffer_size, receive_queue, des_na
    global oldest, remain_buffer_size, start
    remain_buffer_size = buffer_size
    oldest = 0
    receive_oldest = 0
    start = 0
    while True:
            data = clientSocket.recv(1400)
            if oldest > receive_oldest :
                receive_oldest = oldest

            if random.random() < probability :
                seq_num, ack_num = unpack('>LL', data[4:12])
                seq_num = int(seq_num)
                if start == 0:
                    start = time.time()
                ta = threading.Thread(target = log_file_write, args = (des_name, seq_nu
                ta.start()
                ta.join()
                continue

            if remain_buffer_size >= 1400 :
                remain_buffer_size -= 1400
                seq_num, ack_num = unpack('>LL', data[4:12])
                seq_num = int(seq_num)
                receive_queue.put(data)
                if start == 0:
                    start = time.time()
                tb = threading.Thread(target = log_file_write, args = (des_name, seq_nu
                tb.start()
                tb.join()
```

다음은 receiver의 packet_capture부분입니다. packet을 캡처한 뒤, random함수를 이용하여 확률에 따라 이 패킷을 receive 할지, drop할 지에 대해 결정하였습니다. receive 했을 경우, 해당 queue에 넣어주었습니다.

```
def packet_process(clientSocket, clientAddress, des_name, order, window_size, receive_queue):
    global oldest, remain_buffer_size, start
    temp = dict()
    i = 0
    oldest = 0
    cumul_ack = 0
    des_file = open(des_name, 'wb')
    while True:
            data = receive_queue.get()
            remain_buffer_size += 1400
            seq_num, ack_num = unpack('>LL', data[4:12])
            seq_num = int(seq_num)
            ack_num = int(ack_num)
            temp[seq_num] = data
            if seq_num > oldest:
                if oldest != 0:
                    ack_data = pack('>HHLLHHL', 10080, 0, 0, oldest- 1, 0, window_size, 0)
                    clientSocket.sendto(ack_data, clientAddress)
                tc = threading.Thread(target = log_file_write, args = (des_name, oldest, time
                tc.start()
                tc.join()

            elif seq_num == oldest:
                des_file.write(data[20:])
                ack_data = pack('>HHLLHHL', 10080, 0, 0, seq_num, 0, window_size, 0)
                clientSocket.sendto(ack_data, clientAddress)
                oldest = oldest + 1
                td = threading.Thread(target = log_file_write, args = (des_name, oldest, time
                td.start()
                td.join()

                while  temp.get(oldest):
                    data = temp[oldest]
                    oldest = oldest + 1
                    des_file.write(data[20:])
                    cumul_ack = 1
```

다음 쓰레드는 받은 패킷을 처리하는 부분입니다. 만약 받은 패킷의 sequence number 순서대로이면, ACK 패킷을 보내주고 순서대로가 아니면 잠시 array에 저장하여 in order로 들어올때까지 sliding를 중지하였습니다.

3. 동작 과정

```
Packet loss probability: 0.08
Socket recv buffer size: 1
socket recv buffer size updated: 1000000

receiver program starts...
```

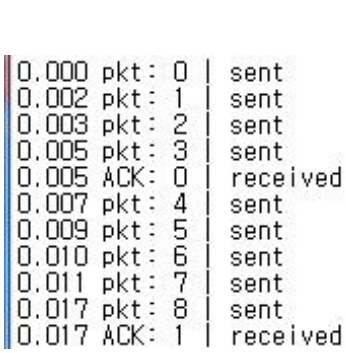
먼저, receiver 단에서 설정을 해주고,

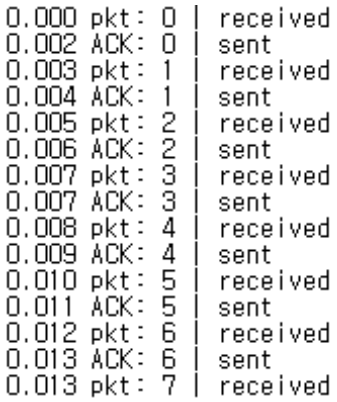
```
C:₩Windows₩py.exe
Receiver IP address: 127.0.0.1
window size: 8
timeout (sec): 0.05

file name: kk.zip
```

sender 단에서 설정을 해주면 파일 전송이 시작됩니다.

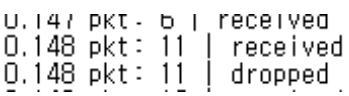
(동시에 파일 전송을 하는 것은 구현하지 않았습니다.)

4. 결과

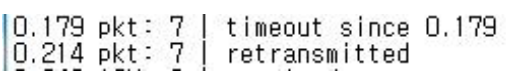




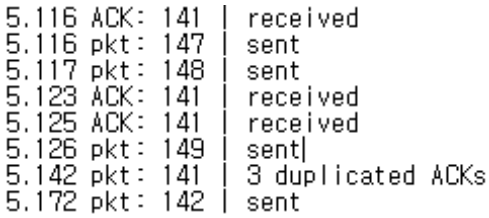
window_size를 8로 설정했을 때의 결과입니다. 다음과 같이 window_size를 적절하게 지키며 sent와 receive를 주고 받는 것을 볼 수 있습니다.



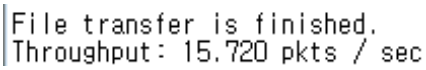
그리고 일정확률에 의해서 받은 packet을 drop하는 경우도 있습니다.



만약 timeout이 발생할 경우, sender에서는 packet을 재전송합니다.

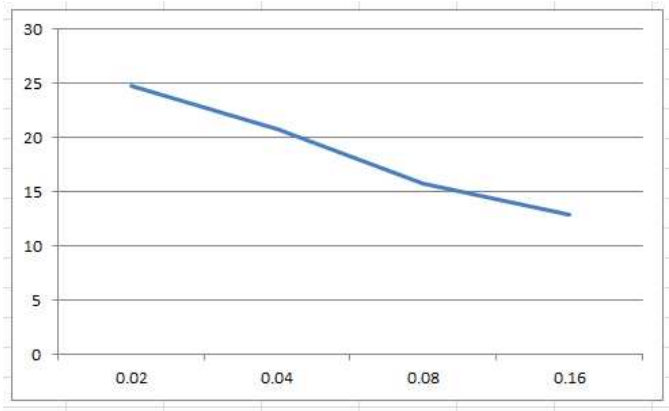


그리고 같은 ACK를 세 번 받으면, 패킷을 timeout과 관련없이 재전송해주었습니다.

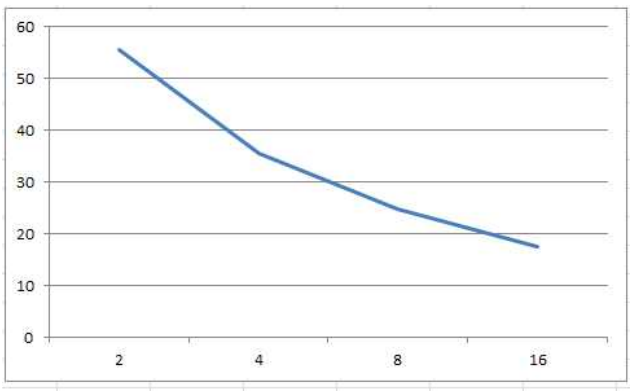


파일 전송이 완료되었습니다.

6. goodput graph



packet drop probability가 증가할수록 goodput은 감소하게 되었습니다.



window_size가 증가할수록 goodput은 감소하게 되었습니다.