Introduction to Computer Networks

Assignment 4: Solving NAT traversal problems

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. ‘server.py’, ‘client.py’ 압축 해제 후 다른 머신으로 client.py 이동

b. ‘server.py’ 파일을 열고 server_host에 server ip 입력

client.py server.py

asmt4 > 2014310279 > server.py > ...

```
import os
import datetime
import socket
import threading

####### server ip 입력 #######
server_host = '127.0.0.1'
##############################
```

c. ‘server.py’ 실행 시 나타나는 ip를 ‘client.py’실행 시 입력

```
network/asmt4/2014310279 via © base
➜ python3 server.py
server starts on ip : 192.168.0.85
elsa Asleep
anna Asleep
anna Awake 192.168.0.92:40270

elsa Awake 192.168.0.92:38248

elsa is unregistered
anna is disconnected
```

```
air@ubuntu:~/py$ python3 client.py
Input Your ID : elsa
input server's IP address : 192.168.0.85
elsa Asleep

@show_list
elsa Asleep
anna Awake 192.168.0.92:40270
```

3. Design – server.py

Server는 client와 tcp통신을 통해 데이터를 주고 받으며, thread를 통해 접속 중인 client를 관리한다. Threads 라는 dictionary안에 접속중인 client의 정보를 저장한다.

a. server()

```
def server():
    tcp_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcp_server.bind(server_addr)
    print("server starts on ip : {}".format(server_host))
    try:
        tcp_server.listen(200)
        while True:
            connection_socket, (ip, port) = tcp_server.accept()
            new_thread = Client_thread(ip, port, connection_socket)
            new_thread.start()

    except KeyboardInterrupt:
        tcp_server.close()
```

Tcp 소켓을 만든 후, client에서 접속 할 때마다 thread를 만들어 thread안에서 각 client와 통신한다.

b. Class Client_Thread()

```
def run(self):
    client_id = ""
    while True:
        try:
            # num ¦ client_id ¦ cmd ¦
            recv_msg = self.tcp_conn.recv(4096)
            if not recv_msg:
                print("{id} is disconnected".format(id=client_id))
                del clients[client_id]
                break
            recv_msg = recv_msg.decode()
            cmd = recv_msg.split("¦")
            client_id = cmd[1]
            if cmd[0] == '0':
                clients[client_id] = [client_status[0], self.ip, int(cmd[2])]
                print("{} {}".format(client_id, clients[client_id][0]))
                msg = self.get_list()
                self.tcp_conn.send(msg.encode())
```

처음 접속시 client로 부터 id, ip, 채팅용 udp port를 전달 받아 이를 clients에 저장한다. While문 내에서 tcp로 들어오는 메세지를 전달 받는데, 메세지가 오지 않을 시 연결이 끊긴 것으로 간주하여 client를 삭제하고 thread를 종료한다.

```
elif cmd[0] == '2':
    src_id = cmd[1]
    if src_id in clients:
        if clients[src_id][0] == client_status[0]:
            clients[src_id][0] = client_status[1]
            msg= "{id} {status} {ip}:{port}\n".format(
                id = src_id, status = clients[src_id][0],
                ip = clients[src_id][1], port = clients[src_id][2])
            print(msg)
    dst_id = cmd[2]
    msg_to_send = cmd[3]
    if dst_id in clients:
        if clients[dst_id][0] == 'Awake':
            msg = "#{ip}¦{port}¦{msg}".format(
                ip = clients[dst_id][1], port = clients[dst_id][2],
                msg = msg_to_send)
            self.tcp_conn.send(msg.encode())
```

@chat 명령어 입력시 2로 된 header 메세지를 받아 보내는 client가 asleep 상태일 경

우 awake시킨 후 받는 client가 awake인지를 확인하여 받는 client의 정보를 전달한다.

그 외에도 client의 command에 따라 달라지는 메세지를 체크하여 그에 따른 정보를 전달하는 역할을 수행한다.

4. Design – client.py

a. Client()

```
def client():
    client_ID = input("Input Your ID : ")
    server_host = input("input server's IP address : ")
    server_addr = (server_host, 10080)
    tcp_client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcp_client.connect(server_addr)
    done = False

    udp_client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    udp_client.bind(('', 0))
    udp_port = udp_client.getsockname()[1]

    msg = "0¦" + client_ID + "¦" + str(udp_port)
    tcp_client.send(msg.encode())
    recv_msg = tcp_client.recv(2048)
    print(recv_msg.decode())
    cmd_handler = Cmd_handler(tcp_client, udp_client, client_ID)

    try:
        #tcp_client.listen(200)
        while not done:
            cmd = input()
            cmd_handler.run(cmd)
            if cmd == '@exit':
                done = True
```

서버와 통신할 tcp, 채팅할 udp 소켓을 연 후 client의 정보를 server에 전달한다. 이후 입력 받는 command를 cmd_handler에서 처리한다.

b. Cmd_handler() – init

```
class Cmd_handler():

    def __init__(self, tcp_conn, udp_conn, cid):
        self.tcp_conn = tcp_conn
        self.udp_conn = udp_conn
        self.cid = cid
        self.tcp_msg = None
        self.upd_msg = None
        self.tcp_recv_thread = threading.Thread(target = self.tcp_receiver)
        self.tcp_recv_thread.start()
        self.udp_recv_thread = threading.Thread(target = self.ucp_receiver)
        self.udp_recv_thread.start()
```

Cmd_handler는 초기화 시 tcp, udp 메세지를 받는 tcp_receiver, udp_receiver함수를 thread로 실행하여 client로 오는 정보를 받는다.

c. Cmd-handler() – run

```
def run(self, cmd):
    self.cmd = cmd
    if self.cmd[0] != '@':
        return
    tmp_cmd = self.cmd[1:].split(" ")
    request = tmp_cmd[0]
    if request == "show_list":
        msg = "1¦" + self.cid + "¦" + request
        self.tcp_conn.send(msg.encode())
    elif request == 'awake':
        msg = "1¦" + self.cid + "¦" + request
        self.tcp_conn.send(msg.encode())
    elif request == 'exit':
        msg = "3¦" + self.cid
        self.tcp_conn.send(msg.encode())
        return
    elif request == 'chat':
        dst_id = tmp_cmd[1]
        msg_to_send = ' '.join(tmp_cmd[2:])
        chat_msg = "From {src_id} [{cmsg}]".format(src_id=self.cid, cmsg=msg_to_send)
        msg = "2¦" + self.cid + "¦" + dst_id + "¦" + chat_msg
        self.tcp_conn.send(msg.encode())
```

입력받은 커맨드에 따라 적절한 메세지를 만들어 server에 전송한다. 그 후 tcp_receiver, udp_ reciever를 통해 결과 값을 받는다.

d. Cmd-handler() – receiver

```
def tcp_receiver(self):
    while True:
        recv_msg, recv_addr = self.tcp_conn.recvfrom(4096)
        self.tcp_msg = recv_msg.decode()
        if(self.tcp_msg[0] == '#'):
            addr = self.tcp_msg.split("¦")
            dst_ip = addr[0][1:]
            dst_port = int(addr[1])
            chat_msg = addr[2]
            self.udp_conn.sendto(chat_msg.encode(), (dst_ip, dst_port))
        else:
            print(recv_msg.decode())

def ucp_receiver(self):
    while True:
        recv_msg, recv_addr = self.udp_conn.recvfrom(4096)
        self.upd_msg = recv_msg.decode()
        print(self.upd_msg)
```

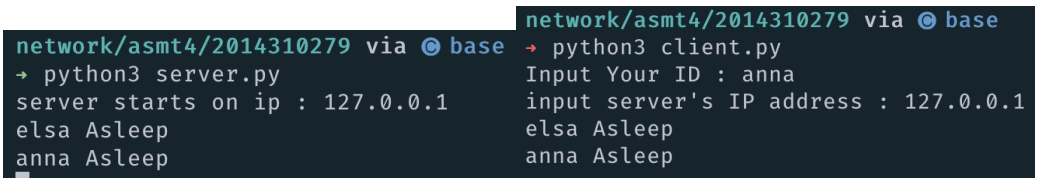
While문을 통해 실시간으로 들어오는 메세지를 처리하며, 채팅 시도 시 tcp로 서버로 부터 채팅할 클라이언트의 주소 정보를 받아 udp로 메세지를 보낸다.

5. Scoring (편의를 위해 Local에서 실행 후 캡쳐)

■ 20 points: The client registers to the registration server.

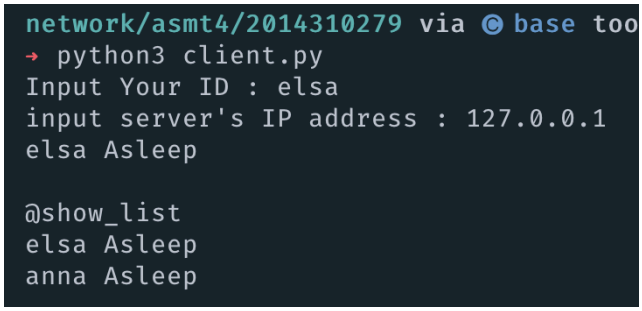
- The new client sends a registration request to the server by TCP

- The server displays the new client information, and send back the current registered client list to the new client.



■ 10 points: For the '@show_list' command.

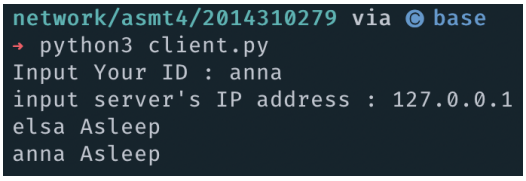
- The client program can show the current registered client list.



■ 10 points: The client maintain the current registered client list.

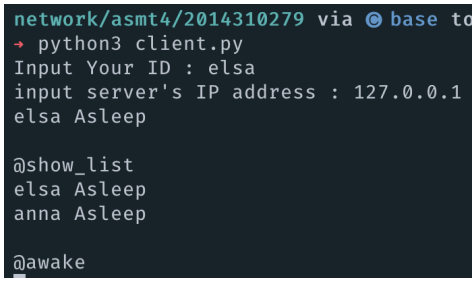
- When the client is first registered, the client can receive all registered client information form the server by TCP

- Whenever a new client registration, the server notify the new event to all clients, and the clients can update their list.



■ 10 points: 'Awake' request to the registration server.

- The client sends the request to server by UDP.
- The server displays the request, and notify the event to all clients by TCP.



■ 10 points: The client receives an incoming chat meesage, and display it

- If my (client) status is Awake, display the message

```
@show_list
elsa Asleep
anna Asleep

@awake
From anna [hi elsa build snowman?]
```

- 10 points:

- If my (client) status is ‘Asleep’, first send the Awake request to the server by UDP
- Send the chat message to the target client by UDP

```
@show_list
elsa Awake 127.0.0.1:64889
anna Asleep

@chat elsa hi elsa build snowman?
@show_list
elsa Awake 127.0.0.1:64889
anna Awake 127.0.0.1:50594
```

- 10 points: For ‘@exit’ command

- Clients send the unregistration request to the server, and terminate
- The server display the request, delete the client in the registration list, and notify the event to all clients by TCP.

```
@awake
From anna [hi elsa build snowman?]
@exit
Exception in thread Thread-1:
Traceback (most recent call last):
  File "/Users/air/anaconda3/lib/py
hreading.py", line 917, in _bootstr
```

```
→ python3 server.py
server starts on ip : 127.0.0.1
elsa Asleep
anna Asleep
elsa Awake 127.0.0.1:64889

anna Awake 127.0.0.1:50594

elsa is unregistered
```

- 10 points: For client disconnection without an unregistration request.

- The server detects the TCP disconnection, display the disconnection, delete the client in the registration list, and notify the event to all clients by TCP.

```
@chat elsa hi elsa build snowman?
@show_list
elsa Awake 127.0.0.1:64889
anna Awake 127.0.0.1:50594

^CException in thread Thread-1:
Traceback (most recent call last):
  File "/Users/air/anaconda3/lib/p
```

```
network/asmt4/2014310279 via © b
→ python3 server.py
server starts on ip : 127.0.0.1
elsa Asleep
anna Asleep
elsa Awake 127.0.0.1:64889

anna Awake 127.0.0.1:50594

elsa is unregistered
anna is disconnected
```