Assignment 3:
Solving NAT traversal problems

2017313008 김태은

1. Development environment

- Version of operating system: **Windows 10, Ubuntu 18.04.4 LTS**
- Programming language: **Python**
- Interpreter version: **Python 3.6.9**

2. How to design

① server.py

```
from socket import *
from threading import Timer
import time

registerList={}  # registration list
privateList={}  # private IP address of client
recvTime={}  # keep-alive message receiving time
```

server에서는 python dictionary를 사용하여
registration list, private ip address list, keep-alive receiving time list를 생성한다.
registerList의 key값은 client ID, value는 (public ip address, port number)이다.
privateList의 key값은 clientID, value는 (public ip address, private ip address)이다.
recvTime의 key값은 clientID, value는 keep-alive message receiving time이다.

```
if __name__ == '__main__':

    serverSocket=socket(AF_INET, SOCK_DGRAM)
    serverSocket.bind(('', 10080))
    print("server program starts....")

    timer=Timer(10, keepTimer)
    timer.start()

    clientID=""
    sendMsg=""
    privateAddr=""
```

먼저 통신에 이용할 소켓을 생성하고, 포트(10080)에 바인딩한다.
timer는 keep-alive 메시지를 10초마다 체크하는 타이머로, 이후에 설명할 것이다.

```
while True:
    message, clientAddr = serverSocket.recvfrom(2048)
    #print(message.decode())
    if "@keepalive" in message.decode():
        clientID=message.decode().split("@")[0]
        recvTime[clientID]=time.time()  # update receiving time
        continue

    elif "@exit" in message.decode():
        clientID=message.decode().split(" ")[0]
        sendMsg=clientID+" is unregistered "
        sendMsg+=registerList.get(clientID)[0]+":"+str(registerList.get(clientID)[1])

    else:
        clientID = message.decode().split(" ")[0]
        privateAddr = message.decode().split(" ")[1]
        sendMsg = clientID + " " + clientAddr[0] + ":" + str(clientAddr[1])


    print(sendMsg)
```

client로부터 메시지를 받아들인다.
만약 그 메시지에 “@keepalive”가 들어있다면 이것은 NAT entry를 유지하기 위한 keep-alive message이므로 recvTime을 업데이트하고 다음 메시지가 오는 것을 기다리게 된다. (continue)

만약 그 메시지에 “@exit”가 들어있다면 해당 client가 연결을 종료하는 request를 보낸 것이다. 밑에서 registration을 삭제하고 그 삭제 정보를 client들에게 보내야 하므로 미리 sendMsg를 만들어놓는다.

만약 두 경우가 아닌 경우에는 client가 처음 등록 메시지를 보냈을 때이다. 이 때 private IP address도 같이 보내오므로 그것을 privateAddr에 저장하고 미리 sendMsg를 만들어놓는다.

```
if "@exit" not in message.decode():

    for i in range(0, len(registerList)):
        registerMsg=list(registerList.keys())[i]+" "+list(registerList.values())[i][0]+":"+str(list(registerList.values())[i][1])
        serverSocket.sendto(registerMsg.encode(), clientAddr)

    for i in range(0, len(privateList)):
        if clientAddr[0]==list(privateList.values())[i][0]:
            cmp1 = list(privateList.values())[i][1].split(".")
            cmp2 = privateAddr.split(".")
            if cmp1[0] + "." + cmp1[1] + "." + cmp1[2] == cmp2[0] + "." + cmp2[1] + "." + cmp2[2]:
                privateMsg = list(privateList.keys())[i] + "@private:" + list(privateList.values())[i][1]
                serverSocket.sendto(privateMsg.encode(), clientAddr)

    registerList[clientID]=clientAddr
    recvTime[clientID]=time.time()
    privateList[clientID] = (clientAddr[0], privateAddr)
    #print(privateList)

for i in range(0, len(registerList)):
    serverSocket.sendto(sendMsg.encode(), (list(registerList.values())[i][0], list(registerList.values())[i][1]))
    if "@exit" not in message.decode():
        if list(registerList.values())[i][0] == privateList.get(clientID)[0]:
            cmp1 = privateList.get(clientID)[1].split(".")
            cmp2 = privateList.get(list(registerList.keys())[i])[1].split(".")
            if cmp1[0] + "." + cmp1[1] + "." + cmp1[2] == cmp2[0] + "." + cmp2[1] + "." + cmp2[2]:
                privateMsg = clientID + "@private:" + privateList.get(clientID)[1]
                serverSocket.sendto(privateMsg.encode(), (list(registerList.values())[i][0], list(registerList.values())[i][1]))

if "@exit" in message.decode():
    del registerList[clientID]
    del recvTime[clientID]
    del privateList[clientID]
```

메시지에 “@exit”가 없는 경우, 즉 client가 처음 등록하는 메시지인 경우 기존의 client registration list를 그 client에게 먼저 보내주어야 한다.
그래서 registerList를 for문으로 돌며 registration list들을 clientAddr로 보낸다.

또한 이 registration list들 중 해당 client와 같은 NAT에 있는 client들의 private ip address를 추가적으로 보내준다. 이 때 비교 방법은 먼저 같은 public ip address를 가지고 있는지 보고(같은 NAT이라면 같은 public ip address로 변환될 것이다.), 그 후에 private ip address의 prefix가 동일한지 본다.
서로 다른 NAT이라도 같은 private ip address를 가질 수 있지만, 여기서는 먼저 public ip address를 비교한 후이므로, 같은 public ip address를 가지면서 private ip address의 prefix가 같다면 같은 NAT이라고 판단할 수 있다. (서로 다른 NAT이라면 public ip address가 다를 것이다.)

따라서 같은 NAT이라고 판단된 경우에는 private ip address를 추가적으로 보내준다.

그리고 registerList, recvTime, piravetList에 해당 client 정보들을 업데이트한다.

다음으로 해당 client의 registration이나 unregistration 등의 정보들을 다른 registered client들에게도 보내주어야 한다. 따라서 registerList를 for문으로 돌며 해당 정보(sendMsg)를 보내준다. 이 때 새로 들어온 해당 client가 기존에 존재하던 client와 같은 NAT에 있다면, private ip address도 추가적으로 보내주어야 한다. 따라서 아까와 마찬가지로 public ip address가 서로 같은지 보고 같다면, private ip address의 prefix를 비교하여 같은 NAT인지 아닌지를 판단한다.

만약 메시지에 “@exit”가 있다면 unregistered request이므로 registerList, recvTime, privateList에서 해당 client를 삭제한다.

```python
def keepTimer():

    for i in range(0, len(recvTime)):

        if i>=len(recvTime): break

        # detect client termination without receiving an unregistration request
        if time.time()-list(recvTime.values())[i] > 30:
            clientID=list(recvTime.keys())[i]
            sendMsg = clientID + " is off-line "
            sendMsg+=registerList.get(clientID)[0]+":"+str(registerList.get(clientID)[1])
            print(sendMsg)

            del registerList[clientID]
            del recvTime[clientID]
            del privateList[clientID]

            for j in range(0, len(registerList)):
                serverSocket.sendto(sendMsg.encode(), (list(registerList.values())[j][0], list(registerList.values())[j][1]))

    timer=Timer(10, keepTimer)
    timer.start()
```

keepTimer에서는 keep-alive message의 유효시간을 10초마다 체크한다. recvTime 을 for문으로 돌며 만약 (현재시간-keepalive receiving time)이 30초를 초과한다면, 해당 client가 Ctrl+c 등으로 강제 종료되었음을 의미하고, 따라서 registration list에서 해당 client를 삭제할 것이다.
registerList, recvTime, privateList에서 각각 해당 client를 삭제하고, 삭제 정보를 registered client들에게 전송한다.

② client.py

```python
from socket import *
from threading import Thread, Timer


registerList = {}
privateList = {}
stop = False
```

client에서는 python dictionary를 사용하여
registration list, private ip address list (같은 NAT의 client)를 생성한다.
registerList의 key값은 client ID, value는 (public ip address:port number)이다.
privateList의 key값은 clientID, value는 (private ip address)이다.

```python
def main():

    recvThread = Thread(target=recvExecute, args=(clientSocket, clientID))
    cmdThread = Thread(target=cmdExecute, args=(clientSocket, clientID))

    recvThread.start()
    cmdThread.start()


if __name__ == '__main__':
    clientID=input("Enter client ID: ")
    serverAddr=input("Enter server IP address: ")

    clientSocket = socket(AF_INET, SOCK_DGRAM)
    clientSocket.bind(("", 10081))

    # get private IP address
    ipSocket = socket(AF_INET, SOCK_DGRAM)
    ipSocket.connect((serverAddr, 80))
    privateIP = ipSocket.getsockname()[0]
    #print(privateIP)

    clientSocket.sendto((clientID + " " + privateIP).encode(), (serverAddr, 10080))

    timer = Timer(10, keepTimer)
    timer.start()

    main()
```

client 프로그램이 실행되면 client ID와 server IP address를 입력받는다.
그 다음, 통신에 이용할 소켓을 생성하고, 포트(10081)에 바인딩한다.

ipSocket은 private IP address를 알기 위해 생성한 소켓이며
serverAddress에 연결함으로써 알아낼 수 있다.

만약 client program이 처음 시작되었다면 server에 registration request를 보내야 한다. 따라서 clientID와 private IP address를 같이 server로 전송한다.

timer는 keep-alive message를 10초마다 전송하기 위해 만들었고, 이후에 설명할 것이다.

main 함수가 실행되면 메시지를 받는 recvThread와 명령어를 입력하는 cmdThread가 시작된다.

```
def recvExecute(clientSocket, clientID):
    global stop

    while True:
        if stop == True: return

        message, serverAddr = clientSocket.recvfrom(2048)

        #print(message.decode())

        if "unregistered" in message.decode() or "off-line" in message.decode():
            if clientID not in message.decode():
                userID=message.decode().split("is")[0][:-1]
                del registerList[userID]
                if userID in privateList:
                    del privateList[userID]

        elif "@chat" in message.decode():
            fromID = message.decode().split("@chat")[0]
            msg = message.decode().split("@chat ")[1]
            print("From " + fromID + " [" + msg + "]")

        elif "@private" in message.decode():
            userID = message.decode().split("@private:")[0]
            privateAddr = message.decode().split("@private:")[1]
            privateList[userID] = privateAddr

        else:
            msgID = message.decode().split(" ")[0]
            msgAddr = message.decode().split(" ")[1]
            registerList[msgID] = msgAddr
```

recvExecute 함수: recvThread에서 실행되는 함수이다. 메시지가 들어오는 것을 처리한다. 만약 다른 client의 “unregistered” 나 “off-line” 정보가 들어온다면 registerList에서 해당 client를 삭제한다. 만약 같은 NAT에 있는 client라면 privateList에서도 해당 client를 삭제한다.

만약 다른 client에서 온 @chat message라면 보낸 client와 메시지 내용을 출력한다.

만약 같은 NAT에 있는 client의 private ip address를 담은 메시지라면 privateList에 해당 정보를 저장한다.

다른 경우(else)는 다른 client가 새로 등록되었다는 정보이므로 registerList에 해당 정보를 저장한다.

여기서 stop==True일때는 @exit명령어를 입력했을 때라서, recvThread도 종료시키기 위해 설정하였다.

```python
def cmdExecute(clientSocket, clientID):
    global stop

    while True:
        command = input("")

        if command == '@exit':
            clientSocket.sendto((clientID + " " + command).encode(), (serverAddr, 10080))
            stop = True
            timer.cancel()
            return

        elif command == '@show_list':
            for i in range(0, len(registerList)):
                print(list(registerList.keys())[i] + " " + list(registerList.values())[i])

        elif "@chat" in command:
            msgID = command.split(" ")[1]
            content = command.split(msgID + " ")[1]

            # under the same NAT
            if msgID in privateList:
                addr = privateList.get(msgID)
                clientSocket.sendto((clientID + '@chat ' + content).encode(), (addr, 10081))

            else:
                if msgID in registerList:
                    addrport = registerList.get(msgID)
                    addr = addrport.split(':')[0]
                    port = int(addrport.split(':')[1])
                    clientSocket.sendto((clientID + '@chat ' + content).encode(), (addr, port))

                else:
                    print("There is no " + msgID + " in registration list.")
```

cmdExecute 함수: cmdThread에서 실행되는 함수이다. 명령어가 입력되는 것을 처리한다. 만약 명령어가 @exit이라면 server에 unregistration request를 전송한다. 그리고 timer를 종료하고 프로그램을 종료한다.

만약 명령어가 @show_list라면 지금 registration list에 저장된 client들을 출력한다.

만약 명령어가 @chat이라면 다른 client에게 메시지를 보내는 것이다. 만약 전송 대상 client가 private List에 존재한다면 같은 NAT안에 있다는 것을 의미한다. 따라서 private ip address 를 이용하여 메시지를 전송한다.

만약 전송 대상 client가 private List에 존재하지 않는다면 같은 NAT안에 있는 것이 아니다. 따라서 registerList에 저장되어있는 public ip address, port를 이용하여 메시지를 전송한다.

마지막으로 (else) registration list에 저장되어 있지 않는 client를 대상으로 했을 경우 그런 client는 registration list에 없다는 문구를 출력한다.

```python
def keepTimer():
    global timer

    clientSocket.sendto((clientID + "@keepalive").encode(), (serverAddr, 10080))
    timer = Timer(10, keepTimer)
    timer.start()
```

keepTimer에서는 keep-alive message를 server에게 10초마다 전송한다.

3. How to run

먼저 host computer에 위치한 server program을 실행한다.
(host computer의 OS가 Windows10이라서 pycharm program을 이용하여 실행했다,
만약 host computer가 리눅스 기반이라면 "python3 server.py" 를 통해 실행할 수 있다.)

Server가 실행되면 다음과 같은 메시지가 뜨고, client로부터 요청을 받을 준비가 된다.

```
server program starts....
```

그 후, host computer에서 client 1개, vmware에서 client 3개를 실행한다.

먼저, host computer에서 client 1개를 실행한다.

```
Enter client ID: client1
Enter server IP address: 192.168.56.1
```

<client1 화면>

client ID, server IP address를 입력하면 그 정보가 server로 전달된다.

```
server program starts....
client1 192.168.56.1:10081
```

<server화면>

제대로 registration 요청이 들어온 것을 확인할 수 있다.

다음으로, vmware에서 client 3개를 실행한다.

```
taeee@ubuntu:~$ python3 client.py
Enter client ID: client2
Enter server IP address: 192.168.56.1
```

<client2 화면>

```
taeeun@ubuntu:~$ python3 client.py
Enter client ID: client3
Enter server IP address: 192.168.56.1
```

<client3 화면>

```
client@ubuntu:~$ python3 client.py
Enter client ID: client4
Enter server IP address: 192.168.56.1
```

<client4 화면>

```
server program starts....
client1 192.168.56.1:10081
client2 192.168.56.1:57729
client3 192.168.56.1:57970
client4 192.168.56.1:57977
```

<server화면>

제대로 registration 요청이 들어온 것을 확인할 수 있다.

```
taeeun@ubuntu:~$ python3 client.py
Enter client ID: client3
Enter server IP address: 192.168.56.1
@show_list
client1 192.168.56.1:10081
client2 192.168.56.1:57729
client3 192.168.56.1:57970
client4 192.168.56.1:57977
@exit
taeeun@ubuntu:~$
```

@show_list

@exit

```
client3 is unregistered 192.168.56.1:57970
```

<server 화면>

```
taeee@ubuntu:~$ python3 client.py
Enter client ID: client2
Enter server IP address: 192.168.56.1
@chat client4 I love Network!!
```

@chat

```
client@ubuntu:~$ python3 client.py
Enter client ID: client4
Enter server IP address: 192.168.56.1
From client2 [I love Network!!]
```