**CSCI 353: Programming Assignment 3**

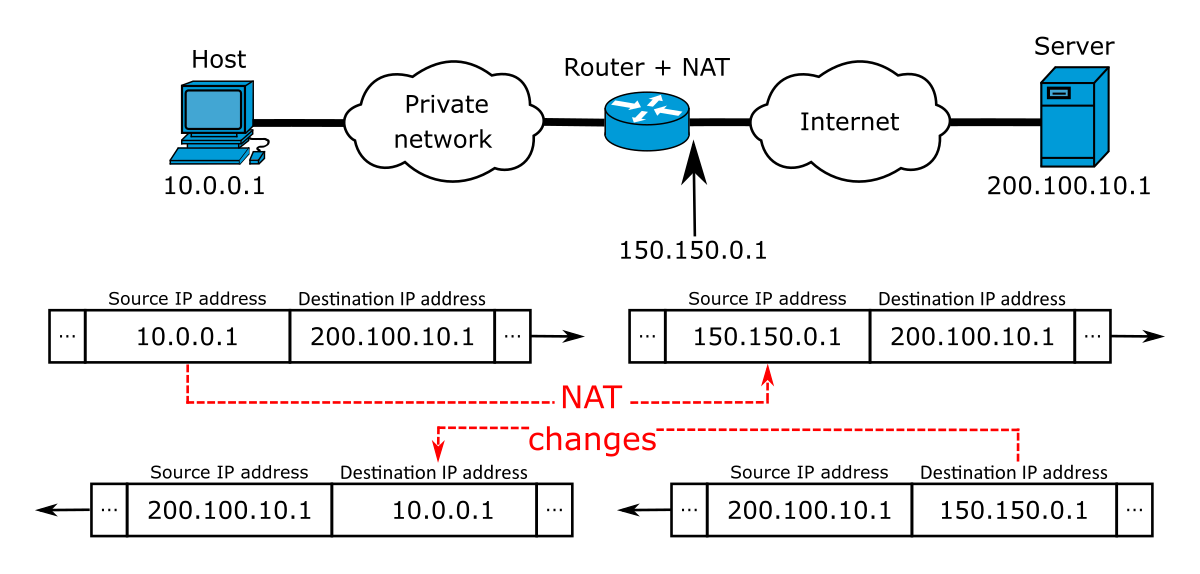
**THIS PROJECT MUST BE DEVELOPED IN PYTHON 3.5+**

**(Python 2 submissions will receive a 0)**

**Due on October 23rd, 11:59pm**

**Introduction**

As discussed in the lecture, network address translation (NAT) is a method to map multiple private hosts to one public IP address. In this assignment, you will be simulating the NAT process through writing a client, server, and translator program. Please name your files “client.py”, “server.py”, and “nat.py”. ReadMe files must be in .txt format and programs should execute directly from the command line. For all 3 programs, if nothing is specified on the command line, print usage instructions and quit.



The Network Address Translation Process

**Part 1 - Client and Server**

In this part of the assignment, you will create a simple echo client and server.

**Server** - Implement a server like in Assignment 1. It must run on a port specified on the command line. Print the same output as Assignment 1 but into a logfile. The command line options will be as follows:

**>>Python3 server.py -p portno -l logfile**

-p <portno> port number *of the echo server*

-l <logfile> name of logfile output should be saved to

The server should run until terminated. While running, if it receives a message, it should send that message back to the sender via a thread and log “recvfrom <senderIP, port> <message>”. Logged messages should also write to the terminal.

**Client** - Implement a client like in Assignment 1. Print the same output as Assignment 1 into a logfile. The client will connect to a destination IP and port. It will then send a register message with its name to the destination as “register <name>”. The command line options will be as follows:

**>>Python3 client.py -s <**serverAddress> **-p <**serverPort> **-m <**clientPort**> -n** clientName **-l** logfile

-s <serverAddress> echo server IP address

-p <serverPort> port number of the echo server

**-**m <clientPort> port number the client should run on

-n <clientName> name of the client (e.g. Tommy Trojan)

-l <logfile> name of logfile output should be saved to

Once the client has received a confirmation of registration with the destination server, it should run and accept user input. Input beginning with “sendto” should be sent to the server as a new message. A separate thread should listen for incoming messages and log them as “recvfrom <senderIP, port> <message>”. Logged messages should also write to the terminal.

**Note:**  The terminal output and log file format should exactly match the sample execution in Assignment 1, deviating from the format will cause the testing script to fail.

**Part 2 - NAT**

In this part of the assignment, you will create the translator program. Clients should connect to the NAT program rather than directly to the server. (LAN) The NAT program will then send client messages to the server and forward any echo messages from the server back to the clients. (WAN).

NAT - the translator program must run on a port specified at the command line and target a destination IP and port. It should also take an input IPv4 address to substitute client IPs with.

The command line options will be as follows:

**>>Python3 nat.py -m myPort -d destinationIP -p destinationPort -l logfile**

-m <myPor**t** > port the NAT runs on

-d < destinationIP > server IP address

-p < destinationPort > port the server is running on

-l <logfile> name of logfile output should be saved to

When the NAT receives a register message from a client, it should write their address and port to a table and generate an address-port pair for the client that is their substitute address to the outside world. The substitute address should be the IP of the NAT. This should be written to the logfile as “<ClientName> | <Client IP, port> | <routerIP, newPort>”. The NAT will then return a message telling the client it is registered. Afterward, when the NAT receives a sendto message from the client, it should prepend it with <routerIP, newPort> before sending it to the server. The server will return it to the NAT which will check the IP/Port against its table and determine which client to send the message back to. The final log file should look like a table of clients, their original addresses, and their new addresses.

**Note:** The routerIP should be the same for all clients (IP of the NAT), newPort is a randomly assigned port number by the NAT to the client. The client should be connecting to the nat and not the server.

**Program Execution:** Client(Sends Msg) -> NAT(Translates internal address) -> Server(Echo) -> NAT(Translates address) -> Client(Receives Message)

**Required Log Files:**

The following lines of output MUST be present in the logfiles. You can also print additional debugging information in the log files, but prepend debugging information with the keyword “DEBUG” so we can ignore it during grading.

The text in the angled brackets below should be replaced with the results from the communication between the client and the server. Do not include the angled brackets in the output.

**NAT Log File (nat.txt)**

<ClientName1> | <Client1 IP, port> | <NewIP, newPort>

<ClientName2> | <Client2 IP, port> | <NewIP, newPort>

<ClientName3> | <Client3 IP, port> | <NewIP, newPort>

**Server Log File (server.txt):**

server started on <1.2.3.4> at port <12345>…

recvfrom <senderIP, port> <message>

**Client Log File (client.txt)**

connecting to the server <1.2.3.4> at port <12345>

sending register message <client1>

sending <serverIP, port> <message>

**Grading**

You will be graded on the logfile. Your client should clearly display the echoed message it receives back with its new IP/Port prepended with a corresponding logfile. Your NAT should clearly display when a new client registers and the logfile should contain a table of clients, their original IP/Port, and their replacement IP/Port. Your server should clearly display received messages and log them. Please make sure your respective log files are formatted correctly.

**Test Case 1**

Text

Description automatically generated

**Test Case 2**

**Text

Description automatically generated**

**Test Case 3**

**Text

Description automatically generated**

**Code and Collaboration Policy**

You can discuss the assignment and coding strategies with your classmates. However, your solution must be coded and written by yourself. Please refer to the plagiarism policy in the course syllabus.

The submissions will be run through code similarity tests. Any flagged submissions will result in a failing score. Keeping your code private is your responsibility.

You are strongly encouraged to pair and test your code implementations with your peers in class.

**Submission**

* Nat.py
* Client.py
* Server.py
* Txt files corresponding to the provided test cases

You should develop and test your code on your own machines. However, we encourage you to commit your changes to your assigned GitHub repo as frequently as possible. For grading purposes, we will only look at your most recent version.

The README must contain USCID, email address and/or additional notes on usage if needed.

We will then run your programs using a suite of test inputs. After running the program, we will grade your work based on the output written to the corresponding text files.