Introduction to Computer Networks

Assignment 4: Solving NAT traversal problems

1. Goal

- Develop two programs; Registration Server and Client for a simple chatting service
- Develop the solution which let clients, under NAT environments, communicate each other.

2. System Overview

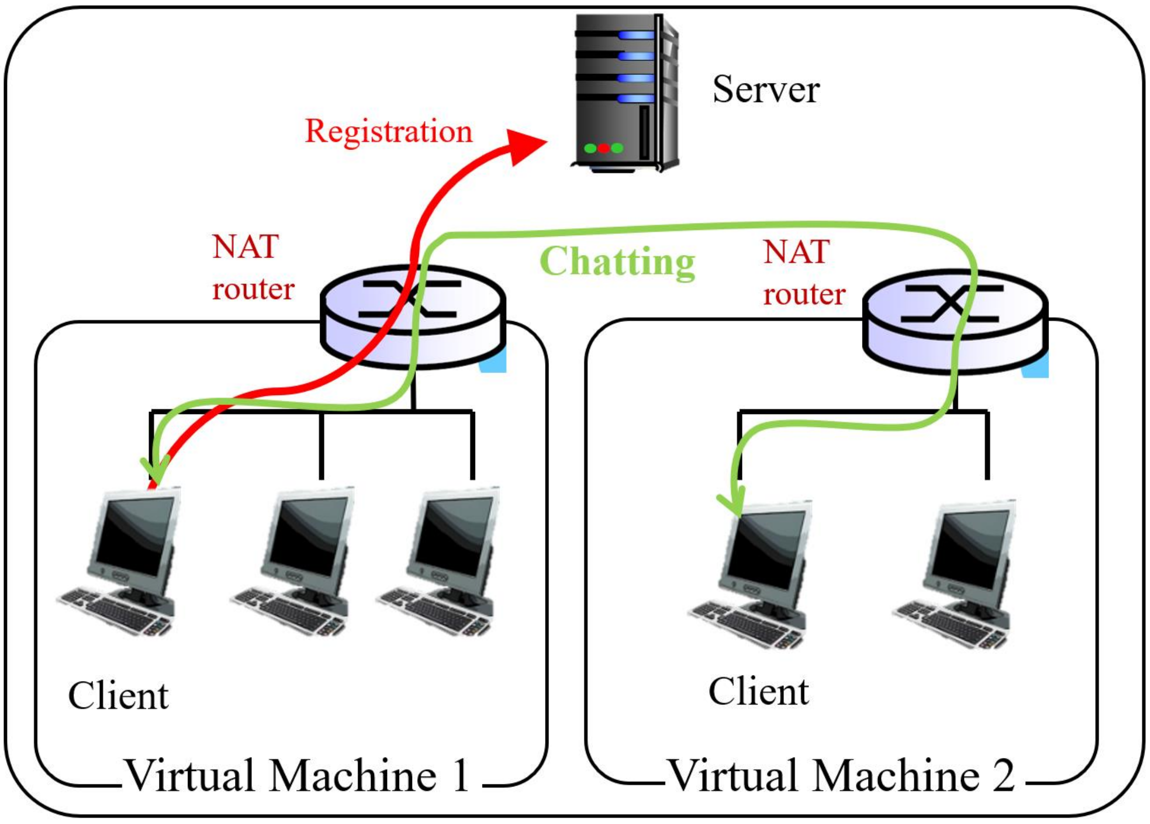

Figure 1. System configuration on a single computer

A Virtual Machine (VM) such as VirtualBox, provides a NAT service. For this assignment, you must install multiple (at least two) VMs on a single computer and also run multiple client programs on each VM.

Client programs make a TCP connection with the Registration Server. Using the TCP channel, clients can register their client IDs into the server and request current registered client information (including myself) from the Server.

The final goal is that clients communicate with other clients directly via UDP.

3. Development environments

- TA will evaluate your results on Linux. Therefore, we recommend you use C or C++ on Linux. You can also use Python (version 3.6+) on Linux(Ubuntu 16.04 LTS) or Windows 10.
- You have to describe your development environment information in detail in the report. If not, TA cannot evaluate your program and you will get zero points.
- For this assignment you need to use virtual machine. We recommend using **VirtualBox** for the virtual machine. Guideline is as follow links.
    - Install Virtual Box: https://kkensu.tistory.com/41
    - Install Ubuntu Virtual Machine: https://medium.com/@tushar0618/install-ubuntu-16-04-lts-on-virtual-box-desktop-version-30dc6f1958d0
    - How to Drag and Drop : https://www.manualfactory.net/11071
    - How to check my private IP address in Linux : use ‘ifconfig’ in terminal

4. Functionalities to implement

- Client program
  - When starting the client program, enters the client ID, server's IP address. Assume each client ID is unique.
  - After starting,

    1) The client program binds a **UDP** socket in any unused one. This is for chatting with other clients directly.

    2) The client connects to the registration server with **TCP** and sends a registration request including its client ID to the server. The client is registered as an "Asleep" status.

    3) The client receives the current registered client information from the server by **TCP**

    4) The client receives a new event of other clients (new registration or status change) from the server by **TCP**
  - The command '@show_list' displays the current registered client list as follows:

    client1 Asleep
    client2 Asleep
  - The client can change its status from 'Asleep' to 'Awake' using the command '@awake'. The command sends a **UDP** packet including its client ID to the same IP & port number of the Registration Server.
  - After sending the **UDP** packet, the client program can receive incoming messages from other clients. If the chat message arrives, display which client sent what message as follows:

    From client2 [I love Network!!!]

- If the client wants to send a chat message to other 'Awake' clients by **UDP**, use a command '@chat' with the opponent client ID as following;

  @chat　client2　I love Network too!!!

  If my status is still 'Asleep', first send the 'Awake' request to the Registration Server by **UDP** before sending the chat message.

- Of course, clients can send and receive messages multiple times by **UDP**.

- You can use the '@exit' command to send the unregistration request to the server by **TCP** and terminate the client program.

● Registration Server Program

- Run the server program on your computer (not VM private networks). Bind the port number 10080 for each **TCP** and **UDP**.
- When receiving registration request via a **TCP** connection, display it.

  client1　Asleep

  And the server has to **notify** this registration event to all registered clients by **TCP**.

- When receiving 'Awake' requests via **UDP**, change the status of the client to 'Awake', and display as following;

  client1　Awake　115.145.179.193:50484

  And the server has to **notify** this awake event to all registered clients by **TCP**.

- When receiving the 'unregistration request' by **TCP**, display the request and delete the client in the registration list

  client2 is unregistered

  Sometimes a client (**TCP**) is disconnected without the 'unregistration' request. Display the 'disconnection'. And delete the client in the registration list.

  client2 is disconnected

  And the server has to **notify** this event to all registered clients by **TCP**.

- Miscellaneous
  - Once clients become awake, the clients can chat each other even if the server is down.
  - Unique ID consists of alphabet and number without any white spaces. And its length does not exceed 32 bytes.
  - The size of a chatting message is limited in 200 letters.

4. Sample Results

Client 1	Client 2	Registration Server
# client1 starts		client1 Asleep
	# client 2 starts	client2 Asleep
@show_list client1 Asleep client2 Asleep		
@chat client2 hello	# message does not arrive	
@awake		client1 Awake 7.7.7.7:77
	@show_list client1 Awake 7.7.7.7:77 client2 Asleep	
From client2 [hello]	**@chat** client1 hello	client2 Awake 8.8.8.8:88
@show_list client1 Awake 7.7.7.7:77 client2 Awake 8.8.8.8:88		
@chat client2 nice to meet you	From client1 [nice to meet you]	
@exit # client1 terminates		client1 is unregistered
	# stop the program by Ctrl-C	client2 is disconnected

5. Submission

- The deadline is **12.15 (Sun) 23:59**.
  - For delayed submissions, a penalty of -15 points applies every 24 hours. After 72 hours, you get zero points.
  - In the case of plagiarism, you will receive 0 points for the first time and ***F*** for the second.
- Submit a zip file including a **report** and two (sender and receiver) program sources to iCampus
  - The report file format should be PDF.
  - Name the Report file as follows ***StudentID_Name.pdf*** (ex: 2019001_홍길동.pdf)
  - The report has to include the following things;
    1) Describe your development environment information in detail
       (versions of operating systems, languages, compilers/interpreter versions, compile options)
    2) Present how to design your assignment such as data structures and algorithms.
    3) Explain how to run both sender and receiver programs including the screen capture.

6. Scoring

- Total 100 points
  - 10 points: Report
    - 10 points for the well-written documentation.
  - 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 message, and display it
  - If my (client) status is Awake, display the message
- 10 points: The client send a chat message to other clients
  - 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
- 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.
- 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.

7. Q&A

- Leave your questions on the google sheet