Group19 Project 3: Network Address Translation

Nicholas van Huyssteen, Lehann Smith
20869118@sun.ac.za, 19027176@sun.ac.za

21 August 2019

1 Introduction

The purpose of this project was to implement a simulation of a Network Address Translation box (hereafter referred to as a NAT-box). Simulating a NAT-box included a component dedicated to the construction, routing and modification of packets. Further components include a DHCP implementation and dynamic address translation form.

2 File(s) Description

NAT_box.java: The NAT_box file contains the communication protocols. It provides the functionality to process and route messages from clients. It also contains the DHCP implementation, ARP communication, error and status message functionality. A private static class called ClientConnection is contained inside the NAT_box file. For every client that connects to the server, an instance of this class is created on a thread that mirrors an instance of the Client file. Important methods in the NAT_box class include recvMessage(), sendToClient(), the various packet processors and the DHCP methods. recvMessage() listens for a packet from any connected client and classifies it as a message or status packet. sendToClient() determines what modifications need to be made to the packet, processes it and forwards it to the correct client.

Client.java: The Client file contains all the important information on the connected client, including IP-addresses as assigned by the NAT-box if internal. The Client has a listener thread that continually waits for incoming

packets and sends them to the relevant processing methods. Significant methods include the DHCP client-side implementation, various processing and extraction methods that act on incoming/outgoing packets and send/receive methods. sendMessage validates the input, processes it into NAT-box usable packets and sends it through. recvMessage classifies an incoming packet as a status packet or a message. If the latter, the message is printed to the output.

NatTable.java: NatTable contains the network address translation table and methods to enter or lookup IP-addresses, ports and the relevant Client-Connection instances that belong to the entries. It also contains a threaded timer that test NatTable-entry timestamps every predetermined amount of seconds and removes them if necessary.

Helpers.java: The Helpers class is a collection of static methods that are used universally throughout the other classes to do things such as generate IP and MAC addresses.

3 Program Description

A General Overview: The implementation is composed of initial communication methods between newly connected clients and the NAT-box (DHCP), managing entries in the NAT-table and packet-routing/modification during communication between clients and the NAT-box.

Setting up the NAT-box When the NAT-box is launched it generates a number of internal IP-addresses (depending on the parameter the NAT-box is launched with at runtime) available to be assigned to internal clients and a NatTable class. The NAT-box waits for connections.

Setting up the Client When the client is launched, it connects as either an internal or an external client based on user input. It receives DHCP communication as below from the NAT-box.

The DHCP and ARP implementation: Initial communication between the client and NAT-box is based on the Dynamic Host Configuration Protocol. An internal Client executes the method DHCPDiscover*, which broadcasts a discovery message to all entities on the network. Any entity, with the exception of the NAT-box, drops this Message. The NAT-box receives

the discovery message and in turn executes DHCPOffer, which checks the pool of available IP's to assign for an IP that isn't in use. If such an IP is found, it gets sent to the client, otherwise it sends a message containing "FULL_CAPACITY". upon receiving one of these messages, the client executes DHCPRequest. If the message reads "FULL_CAPACITY" it prints an appropriate message to the commandline and disconnects the client. If an IP is detected in the message, this IP is assigned to a "workingIP" variable and a message is sent back to the NAT-box to indicate that the IP has been accepted. When the NAT-box receives this message it executes DHCPAck, which alters the list of NAT-box IP's to show that the IP offered to the client is now in use.

After the DHCP process has concluded, the client executes ARPRequest which sends a specific request message to the NAT-box. Upon receiving this message, the NAT-box executes ARPReply, which sends the NAT-box's IP address as well its MAC address. The client then receives this message, extracts the addresses and stores them in variables, to be used for future communication with the NAT-box.

*Actually, all clients execute DHCPDiscover, but the message gets dropped if the NAT-box detects an external IP.

Communication after Client Connection The NAT-box creates an instance of the ClientConnection private class (that contains the sockets and stream that each client will use to communicate with the box) for every client that makes a connection. The NAT-box waits for incoming packets, then uses the address pool and NatTable to classify, modify and route the packets or return error packets if need be. THe NAT-box creates, maintains and removes entries in the NAT-table based on communications. A convention is used in packet construction. A string is constructed

The client continually listens for incoming message/error packets and displays them to the user. When input to the client is received, it modifies the input into a packet usable by the NAT-box for further processing. This packet is sent to the NAT-box for routing.

Maintaining the NAT-table Every time an internal client attempts to communicate with an external client, the NAT-table is checked for an existing entry. If it exists, the timestamp is updated else an entry is created that binds the source IP/port to the destination IP/port along with a timestamp. When an external client attempts to communicate with an internal client, if an entry for that external client to the attempted port exists then a translated destination IP is returned and the timestamp is updated. A threaded timer

continually checks for outdated (elapsed time since last refresh exceeds the timeout parameter) entries to remove.

Disconnecting Clients When a client disconnects the NAt-table is checked for entries corresponding to the IP-address of the client. If any are found, they are removed. If an internal client the NAT-box releases it's assigned IP-address into the available address pool.

4 Experiments

4.1 Experiment 1: Connecting more internal clients than available IP-addresses

Hypothesis: The NAT-box will return an error packet to the extraneous client. Further communication will continue as normal.

Method: Create a NAT-box that generates 3 internal addresses. Create 6 clients and attempt to connect them all as internal clients. Attempt to communicate normally between successfully connected clients after all clients have attempted connection. Print out the address pool after every change made to the data structure.

Results: IP addresses are successfully assigned and the address pool is correctly modified to reflect active/inactive addresses. The NAT-box would respond correctly to the first extraneous client, and every odd-numbered client attempting to connect thereafter. However, the second and every even-numbered client would drop the status packets. The NAT-box blocks communication until an odd-numbered client attempts to connect, whereupon communication is possible again.

Conclusions: NAT-box status packets block communication on the NAT-box if the full DHCP and ARP execution sequence is not completed. A socket timeout could be implemented to resolve blocking in case a client is not able to complete the full DHCP/ARP.

4.2 Experiment 2: Send message from an internal client to an internal client

Hypothesis: Packets between internal clients will be correctly routed without modification.

Method: Connect three internal clients and attempt to send messages between them. Print out full packets both clientside and NAT-box-side.

Results: Packets are routed correctly without modification beyond the setting up of the convention forms. The client sends an acknowledge packet upon receiving the packet and the NAT-box prints out that the packet has been routed.

Conclusions: Internal to Internal communication is fully functional.

4.3 Experiment 3: Send message from an internal client to an external client

Hypothesis: Packets from an internal client to an external client will be routed correctly and an entry will be created in the NAT-table binding the internal IP/port to the external IP/port.

Method: Connect an internal and external client and attempt to send a message to the external client using its IP value as the destination address. Print out full packets both clientside and NAT-box-side. Print out any updates to the NAT-table.

Results: Packets are routed correctly. An entry is created and timestamped in the NAT-table corresponding to the correct addresses and ports. The packet is correctly set-up, transmitted and received. The client sends an acknowledge packet upon receiving the packet and the NAT-box prints out that the packet has been routed.

Conclusions: Internal to External communication is fully functional.

4.4 Experiment 4: Send message from an external client to an internal client

Hypothesis: If a prior message has been received by the external client from the internal client at an elapsed time that is less than the specified timeout parameter, the packet will be routed. Else, the packet will be dropped.

Method: Connect an internal and external client and attempt to send a message to the internal client using the IP of the NAT-box as destination address. Attempt to send a message from the internal client to the external client and subsequently retry sending a message from external to internal using the IP of the NAT-box as destination address in conjunction with the port obtained from the NAT-table. Print out full packets both clientside and NAT-box-side. Print out any updates to the NAT-table.

Results: The packet is dropped and an error packet is returned to the external client. "Packet not routed" is printed by the external client. The second attempt results in a successful NAT-table entry being created. The message is routed correctly and the NAT-table timestamp is refreshed. The client sends an acknowledge packet upon receiving the packet and the NAT-box prints out that the packet has been routed.

Conclusions: External to internal communication is in fully working condition if the NAT-box IP address is used by the external client and a NAT-table entry exists.

4.5 Experiment 5: Send message from an external client to an external client

Hypothesis: The packet will be dropped and an error packet returned.

Method: Connect two external clients and attempt to send a message using their IP-addresses.

Results: The packet is dropped and the sending client prints out the error message "packet not routed" received from the NAT-box.

Conclusions: External to external communication is not routed by the NAT-box.

4.6 Experiment 6: Attempt to send message from external to raw internal IP address

Hypothesis: The packet will be dropped and an error packet returned.

Method: Connect an internal and external client and attempt to send a message to the internal client using the assigned IP of the internal client as the destination address. Attempt to send a message from the internal client to the external client and subsequently retry sending a message from external to internal using the assigned IP of the internal client as the destination address in conjunction with the port obtained from the NAT-table. Print out full packets both clientside and NAT-box-side. Print out any updates to the NAT-table.

Results: The first attempted packet is dropped. The second attempt results in the packet being routed to the internal client.

Conclusions: The NAT-table implementation does not hide the internal IP's successfully. Routing using the internal IP should not be possible and the packet should be dropped. The NAT-table translates to internal IP's when using its address and when using internal addresses. This issue could be fixed by checking for any attempted destination IP from an external client. If the destination IP is not the IP of the NAT-box, the packet should be dropped and an error packet returned.

4.7 Experiment 7: Attempt to send message from external to internal client after the NAT-table entry has expired

Hypothesis: The packet will be dropped and an error packet returned.

Method: Create a NAT-table with a low timeout parameter. Connect an internal and external client. Attempt to send a message from the internal client to the external client. Wait for the NAT-table to print that it has removed the entry and is now empty. Subsequently, try sending a message from external to internal using the NAT-box IP as destination address in conjunction with the port obtained from the NAT-table Print out full packets both clientside and NAT-box-side. Print out any updates to the NAT-table.

Results: The packet is dropped and an error packet is returned to the external client. The external client prints out "packet not routed".

Conclusions: The timed NAT-table functions as expected.

5 Issues Encountered

We encountered a challenge routing packets in the event that one external client had open NAT entries to multiple internal clients. Before implementation based on port differentiation to distinguish between different internal clients, the external client would write to all clients that it had an open NAT table entry to. We encountered an issue when attempting to issue an internal IP-address when all available addresses were already assigned. The NAT-box would respond correctly to the first extraneous client, and every odd-numbered client attempting to connect thereafter. However, the second and every even-numbered client would drop the status packets and have communication blocked indefinitely.

6 Significant Data Structures

Significant use was made of ArrayLists in the implementation of the NAT-Box. The address pool that stores IP-addresses available to internal clients was constructed as an ArrayList of ArrayLists of Strings. The inner ArrayList held each generated IP-address, it's corresponding port and their status as in- or out of use. An ArrayList that stores every created instance of the Client-Connection class (a thread on the NAT-box side that holds the counterpart sockets and streams to the client instance) was central to communication.

7 Design

Making use of a mirrored ClientConnection class as a counterpart to every instance of client that contains the sockets and stream for that specific client and it's own self-contained packet processing methods.

8 Compilation

The project is compiled with maven. In the repository directory open a terminal and run the script ./addNB.sh (timeoutseconds) (addresspool_size) to compile and run a NAT-box with NATTable entries being removed after timeoutseconds and addresspool_size internal IP's available for assignment. Run ./addClient.sh to connect a client to the NAT-Box, and type "in" or "ex" when prompted to connect as an internal or an external Client.

9 Execution

The project is compiled with maven. In the repository directory open a terminal and run the script ./addNB.sh (timeoutseconds) (addresspool_size) to compile and run a NAT-box with NATTable entries being removed after timeoutseconds and addresspool_size internal IP's available for assignment. Run ./addClient.sh to connect a client to the NAT-Box, and type "in" or "ex" when prompted to connect as an internal or an external Client.

10 Libraries

No external libraries were used.