NETWORK ADDRESS TRANSLATION

Project 3

SEPTEMBER 1, 2022

Markus Sass (22548890@sun.ac.za)
Chris Langeveldt (23632135@sun.ac.za)

Table of Contents

Overview 2
Required Features Implemented 2
  Packet Construction: 2
  Simulator Design 2
  Correct Routing 2
Unimplemented Features 2
Additional Features Implemented 2
Description of files 3
  NatBox.java 3
  ClientHandler.java 3
  NatBoxListenerThreadListener.java 3
  TableRow.java 3
  Client.java 3
  ClientListenerThread.java 4
  Packet.java 4
  Timeout.java 4
Program Description 4
Experiments 5
  Echo Reply 5
  Internal -> Internal: 5
  Internal -> External: 6
  External -> Internal: 7
  External -> External: 8
  Client -> Unknown: 8
  Conclusion 8
Issues Encountered 9
Significant Data Structures 9
Design 9
Compilation 10
  NAT-box 10
  Client 10
Execution 10
  NAT-box 10
  Client 10
Libraries and Packages 10

Overview

The aim of this project was to implement a simulated router environment that has NAT functionality. The core purpose of the NAT-Box was to be able to translate incoming and outgoing packet, to maintain the appearance of a private locally connected network. The implementation was intended to have functionality for sending multiple packet types (UDP/TCP, ICMP, DHCP), and be able to translate them accordingly. In addition to this, a minimal DHCP implementation was to be present which allowed the allocation of internal IP addresses to internal clients.

Required Features Implemented

Packet Construction:

1. Ethernet frame elements included
2. IP packet elements included
3. Payload included

Simulator Design

4. DHCP implemented
5. IP addresses and hardware addresses implemented
6. Addresses distributed implemented

Correct Routing

7. Changing of packet information implemented
8. Internal routing (no changes to packets made)
9. Internal/External routing implemented

Unimplemented Features

All project required features are implemented, however, some thread-safety principles failed and internal entries to the NAT-table were handled somewhat incorrectly (discussed in issues encountered).

Additional Features Implemented

Apart from doing the simplistic traditional NAT implementation, for the rest of the project, we implemented a few additional features. These include:

- Multiple Command Client Shell:
  The client can specify any command that can be run at any time, making the client-side implementation more seamless. The commands available are (for more details, see the /help command in the client):

  - /help - Provides a list of commands as well as a short description of each of them.

  - /send - Sends a packet (encapsulated inside a network packet using objects), to the given destination using the supplied protocol.

  - /whoami - Displays all relevant information of the client. This includes it's IP address, MAC address, local IP, port number and type of connection (internal/external).

  - /exit - Safely close all open sockets and streams and exits the program.

- Command NatBox Shell:
  - /table - Displays the current NAT-table

- ICMP error handling messages

- Dynamic timeout for inactive client user

- Extensive command line output/logging

Description of files

NatBox.java

The NAT-Box handles all functionality of the NAT box implementation, including the handling of packet routing and communication between clients. It keeps track of the connected clients and assigns internal IP addresses to them.

ClientHandler.java

This class function as the threads of the NatBox class that handles each client connection. The class is responsible for all communication to and from the NatBox class with the different clients.

NatBoxListenerThreadListener.java

This file acts similarly to the NAT-Box's Listener, allowing for simultaneous communication. The NAT-table can be view through the command '/table'.

TableRow.java

Each TableRow object is an entry in the NAT-table. It stores the relevant information needed in the table.

Client.java

The Client allows for connections on the network, either connected to a NAT-Box (in this case an internal client) or an external Client which simply connects directly to the network.

ClientListenerThread.java
This class allows for simultaneous receiving and sending communication, creating a thread for the receiving of packets.

Packet.java
This is the object used for all communications. The object contain element of the ethernet frame, IP packet and payload. It also contains error messages and their corresponding codes.

Timeout.java
This class allows for keeping track of client inactivity and proceeds to gracefully disconnect once their time has exceeded.

Program Description

When the NAT-Box is started, it requests it's public IP address used for the simulated environment. When a client connects (specifying the client type in prompt), the NAT-Box's IP address needs to be entered to accept the connection. However, the setup for an internal client includes a (minimal) DHCP request and response interaction with the NAT-Box to retrieve its' internally assigned IP address. The connection and the client's information are displayed both sides, however, can be upon request using the shell commands '/whoami'.

When sending a packet internally, the packet is sent to directly to the specified internal IP entered, which upon arrival of the received client is 'echo replied' back to the sender's client. Both sent and received packets are printed fully (displaying all information), as well as the ECHO_REPLY packet is printed.

When sending a packet from internal to external, the destination of the external's IP address is entered. This is then sent to the NAT-Box, which connects to the external client and relays that packet to the external client, changing its appropriate source of the packet information.

When sending a packet from external to internal, the external client must fill the port of the internal client addition to the IP address on the NAT-Box. This information is used to connect to the NAT-Box and sent the packet to the NAT-Box, which then identifies the internal device using port forwarding within the NAT-table to send the corresponding packet to that internal client. During the translation, the packet's destination and source information is adjusted accordingly.

When sending from external to external or when the recipient client does not exist, the packet is dropped and notified accordingly.

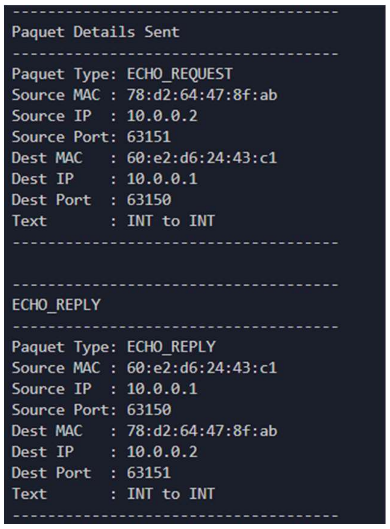
Experiments

Echo Reply

- **Hypothesis:** The packet sent, and the Echo Reply will match up but the source MAC address, source IP address and source port number, will be switched with the destination MAC address, destination IP address and destination port number.
- **Variables used:** The information for the packet sent (independent) and the information for the packet received (dependent).
- **Method:** In the following three routing experiments (internal -> internal, internal -> external, external -> internal) we check that the hypothesis holds when sending a message.
- **Results:** The results can be seen in the results section of the following three experiments.
- **Conclusion:** The hypothesis holds in all these routing cases. The packet sent, and the Echo Reply matches up but with the source MAC address, source IP address and source port number, switched with the destination MAC address, destination IP address and destination port number.

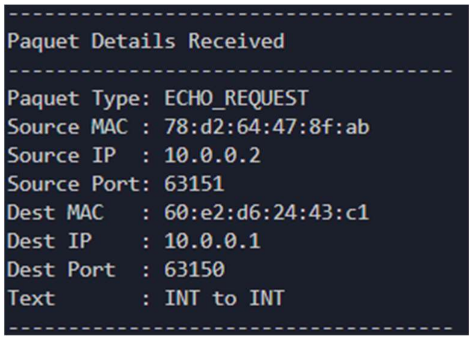
Internal -> Internal:

- **Hypothesis:** For internal-to-internal communication, the packet sent, and the packet received contain the exact same information.
- **Variables used:** The sender being internal(independent), the receiver being internal(independent), the information for the packet sent (independent) and the information for the packet received (dependent).
- **Method:** Sending a packet with a message from one internal client to another internal client.
- **Results:**



```
-----------------------------------
Paquet Details Sent
-----------------------------------
Paquet Type: ECHO_REQUEST
Source MAC : 78:d2:64:47:8f:ab
Source IP  : 10.0.0.2
Source Port: 63151
Dest MAC   : 60:e2:d6:24:43:c1
Dest IP    : 10.0.0.1
Dest Port  : 63150
Text       : INT to INT
-----------------------------------

-----------------------------------
ECHO_REPLY
-----------------------------------
Paquet Type: ECHO_REPLY
Source MAC : 60:e2:d6:24:43:c1
Source IP  : 10.0.0.1
Source Port: 63150
Dest MAC   : 78:d2:64:47:8f:ab
Dest IP    : 10.0.0.2
Dest Port  : 63151
Text       : INT to INT
-----------------------------------
```

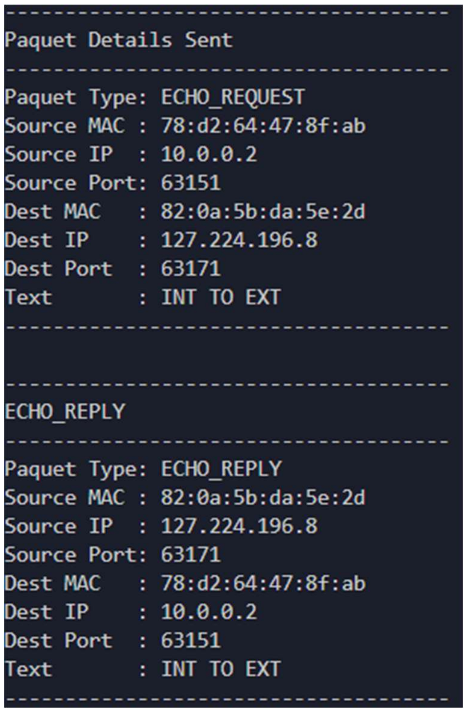


```
-----------------------------------
Paquet Details Received
-----------------------------------
Paquet Type: ECHO_REQUEST
Source MAC : 78:d2:64:47:8f:ab
Source IP  : 10.0.0.2
Source Port: 63151
Dest MAC   : 60:e2:d6:24:43:c1
Dest IP    : 10.0.0.1
Dest Port  : 63150
Text       : INT to INT
-----------------------------------
```

- **Conclusion:** The hypothesis holds. For internal-to-internal communication, the packet sent, and the packet received indeed contain the exact same information.

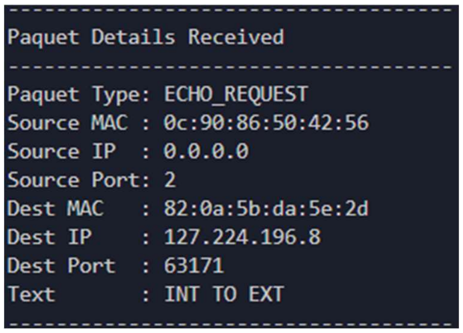
Internal -> External:

- **Hypothesis:** For internal-to-external communication, the packet sent, and the packet received match up but now the packet received by the external client has the NAT-box's MAC address, IP address and port number as the source MAC address, source IP address and source port number respectively instead of the internal client's MAC address, IP address and port number.
- **Variables used:** The sender being internal(independent), the receiver being external(independent), the information for the packet sent (independent) and the information for the packet received (dependent).
- **Method:** Sending a packet with a message from an internal client to an external client.
- **Results:**



```
-----------------------------------
Paquet Details Sent
-----------------------------------
Paquet Type: ECHO_REQUEST
Source MAC : 78:d2:64:47:8f:ab
Source IP  : 10.0.0.2
Source Port: 63151
Dest MAC   : 82:0a:5b:da:5e:2d
Dest IP    : 127.224.196.8
Dest Port  : 63171
Text       : INT TO EXT
-----------------------------------


-----------------------------------
ECHO_REPLY
-----------------------------------
Paquet Type: ECHO_REPLY
Source MAC : 82:0a:5b:da:5e:2d
Source IP  : 127.224.196.8
Source Port: 63171
Dest MAC   : 78:d2:64:47:8f:ab
Dest IP    : 10.0.0.2
Dest Port  : 63151
Text       : INT TO EXT
-----------------------------------
```

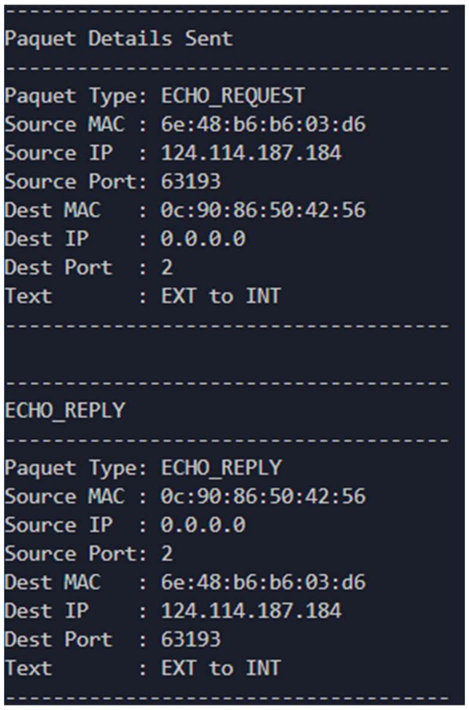


```
-----------------------------------
Paquet Details Received
-----------------------------------
Paquet Type: ECHO_REQUEST
Source MAC : 0c:90:86:50:42:56
Source IP  : 0.0.0.0
Source Port: 2
Dest MAC   : 82:0a:5b:da:5e:2d
Dest IP    : 127.224.196.8
Dest Port  : 63171
Text       : INT TO EXT
-----------------------------------
```

- **Conclusion:** The hypothesis holds. For internal-to-external communication, the packet sent, and the packet received match up, but the packet received by the external client has the NAT-box's MAC address, IP address and port number as the source MAC address, source IP address and source port number respectively.

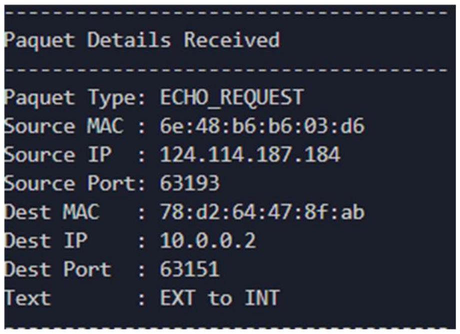
External -> Internal:

- **Hypothesis:** For external-to-internal communication, the packet sent, and the packet received match up but now the packet received by the internal client has its own local MAC address, IP address and port number as the destination MAC address, destination IP address and destination port number respectively instead of the NAT-box's MAC address, IP address and port number.
- **Variables used:** The sender being external(independent), the receiver being internal(independent), the information for the packet sent (independent) and the information for the packet received (dependent).
- **Method:** Sending a packet with a message from an external client to an internal client.
- **Results:**



```
------------------------------------
Paquet Details Sent
------------------------------------
Paquet Type: ECHO_REQUEST
Source MAC : 6e:48:b6:b6:03:d6
Source IP  : 124.114.187.184
Source Port: 63193
Dest MAC   : 0c:90:86:50:42:56
Dest IP    : 0.0.0.0
Dest Port  : 2
Text       : EXT to INT
------------------------------------


------------------------------------
ECHO_REPLY
------------------------------------
Paquet Type: ECHO_REPLY
Source MAC : 0c:90:86:50:42:56
Source IP  : 0.0.0.0
Source Port: 2
Dest MAC   : 6e:48:b6:b6:03:d6
Dest IP    : 124.114.187.184
Dest Port  : 63193
Text       : EXT to INT
------------------------------------
```

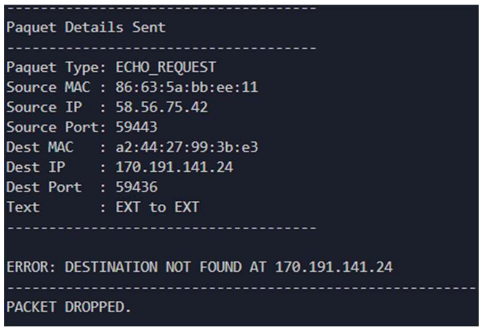


```
------------------------------------
Paquet Details Received
------------------------------------
Paquet Type: ECHO_REQUEST
Source MAC : 6e:48:b6:b6:03:d6
Source IP  : 124.114.187.184
Source Port: 63193
Dest MAC   : 78:d2:64:47:8f:ab
Dest IP    : 10.0.0.2
Dest Port  : 63151
Text       : EXT to INT
------------------------------------
```

- **Conclusion:** The hypothesis holds. For external-to-internal communication, the packet sent, and the packet received match up but now the packet received by the internal client has its own local MAC address, IP address and port number as the destination MAC address, destination IP address and destination port number respectively.

External -> External:

- **Hypothesis:** For external-to-external communication, the packet sent, should be dropped as this routing should not be handled in this NAT-box.
- **Variables used:** The sender being external(independent), the receiver being external(independent), the packet sent (independent) and if the packet is dropped or not(dependent).
- **Method:** Sending a packet from an external client to an external client.
- **Results:**



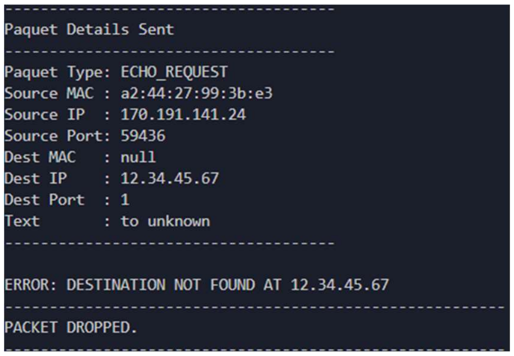
```
------------------------------------
Paquet Details Sent
------------------------------------
Paquet Type: ECHO_REQUEST
Source MAC : 86:63:5a:bb:ee:11
Source IP  : 58.56.75.42
Source Port: 59443
Dest MAC   : a2:44:27:99:3b:e3
Dest IP    : 170.191.141.24
Dest Port  : 59436
Text       : EXT to EXT
------------------------------------

ERROR: DESTINATION NOT FOUND AT 170.191.141.24
------------------------------------------------------
PACKET DROPPED.
```

- **Conclusion:** The hypothesis holds. For external-to-external communication, the packet sent is dropped as this routing should not be handled in this NAT-box.

Client -> Unknown:

- **Hypothesis:** For any client, the packet sent to an unknown client, should be dropped.
- **Variables used:** The type of sender(independent), the receiver being unknown(independent), the packet sent (independent) and if the packet is dropped or not(dependent).
- **Method:** Sending a packet from a client to an unknown client.
- **Results:**



```
------------------------------------
Paquet Details Sent
------------------------------------
Paquet Type: ECHO_REQUEST
Source MAC : a2:44:27:99:3b:e3
Source IP  : 170.191.141.24
Source Port: 59436
Dest MAC   : null
Dest IP    : 12.34.45.67
Dest Port  : 1
Text       : to unknown
------------------------------------

ERROR: DESTINATION NOT FOUND AT 12.34.45.67
------------------------------------------------------
PACKET DROPPED.
------------------------------------------------------
```

- **Conclusion:** The hypothesis holds. For any client, the packet sent to an unknown client is dropped.

Conclusion

From these tests we concluded that our implementation acts as we expected it to act with regards to routing. We conclude that these experiments were successful in that all hypotheses hold.

Issues Encountered

Most issues we encountered were of conceptual nature. We had difficulties in understanding what was expected and what needed to be done. For example, we assumed that when an internal client connects to the NAT-box, it should immediately be entered into the NAT-table. Of course, in hindsight we now know that an internal client should only be added to the NAT-table once it attempts to communicate with an external client.

We encountered issues with packet construction and the data structures we needed to use. After some clarification from the demi we realised that a packet need not be a byte array. We ended up using a newly made class 'Paquet.java' for our packet objects.

We had trouble figuring out what the source and destination information of the packet should be in each of the different routing scenarios. After spending some time on this problem, we came to the solution.

We had some trouble implementing ICMP messaging as, from our research, ICMP messaging is not supported in Java. We ended up adding similar functionality to our 'Paquet' class.

We spent some time figuring out how we could refresh the NAT-table dynamically. We decided to make a class 'Timeout.java' that serves as a thread to each internal client and removes inactive clients.

Significant Data Structures

- table - This is an array list in 'NatBox.java' that contains all the entries in the NAT-table. These entries are 'TableRow.java' objects.
- pool - This is an array list in 'NatBox.java' that contains all the available IP addresses that the NAT-box can assign to its internal clients. The elements are string.
- clientHandlers - This is an array list in 'NatBox.java' that contains all the thread objects that handle the communication with each client. These elements are 'ClientHandler.java' objects.

Design

The 'NatBox.java' acted as our server and all clients made a TCP connection to this server. We decided to only use TCP connections in our implementation for coherence purposes.

We decided to treat all clients similarly with basically only a Boolean 'internal' to distinguish between internal and external clients.

We also tried to simplify ARP requests by keeping storing the MAC and port information of each client in their respective client handlers when they connect to server. This simplified the process of requesting this information before sending the actual message.

We also designed our implementation in such a way that allows for simultaneous sending, receiving and other command typing. We did this by having a separate thread for listening/waiting for message to arrive.

We also improved usability by adding command features. We also included extensive and detailed output/logging.

Compilation

NAT-box

javac NatBox.java

Client

javac Client.java

Execution

NAT-box

java NatBox

Client

java Client

*follow prompts displayed in the shell

Libraries and Packages

- java.io
- java.net
- java.util