Project 3 Report

M Shepherd, 19059020@sun.ac.za
M von Fintel, 20058837@sun.ac.za

21 September 2018

Network Address Translation

Contents

1 Introduction

For this project, we had to implement a NAT box in Java. Network Address Translation was thought up as a solution to the problem of there being too few IP addresses in IPv4 to be able to supply each computer connecting to the internet with a unique IP. NAT allows local computers, referred to as nodes from here on, to form a network using non-unique IP's. This allows the nodes to be able to communicate with each other inside the network without packet modification, and without each having to have unique IP addresses. The NAT box is allocated one or more unique IP's by the ISP. These unique IP's are mapped to internal nodes when they want to send nodes to packets outside the local network. The packet header from the internal node is modified so it looks as if it originated from the NAT box.

2 Project Overview

2.1 Client

Internal Client:

Our Internal client acts as a device that would like to be a part of the internal network. To connect to the NAT-Box, this device is allocated an Internal IP address via a DHCP implementation.

The client can then send messages to other internal clients and any external clients that have already interacted with the NAT-Box.

External Client:

Our External client acts as a device that would like to communicate with a device that is contained in the internal network. When an external client interacts with the NAT-Box, the NAT-Box remembers its IP and allows its internal clients to the communicate with it.

External clients are allowed to communicate with Internal clients, but the NAT-Box drops any packets that are deemed to be communications between two External clients, as this must be routed via an external router.

2.2 NAT-Box

The NAT-Box handles all client connections and communications, including all of the algorithms and implementations required.

The NAT-Box is constantly listening for attempted connections from internal and external clients and will execute the correct protocols for the connections of these clients until a predetermined limit has been reached.

Every client is added to the NAT-Table, with External clients being mapped to their own IP address and internal clients having their internal IP address (obtained via DHCP) mapped to their externally valid IP address.

For every client that connects, a `ClientInstance` object is created. This object handles the receiving, parsing, editing and routing of all packets that are sent through the NAT-Box.

Routing rules:

Internal $\longrightarrow$ Internal: Packet is forwarded without changing its data.

Internal $\longrightarrow$ External: The packet header is modified and an entry is written into the NAT table, or refreshed if it already exists, that binds the source IP to the destination address.

External $\longrightarrow$ Internal: The packet is routed according to the entry in the NAT table, if there is no corresponding entry in the table for the source, the packet is dropped and an error packet is returned.

External $\longrightarrow$ External: Packets are dropped as they should be routed by external networks.

3 Features

3.1 Extra Features

We implemented all features that were mentioned in the project specification, barring the Ethernet frames and port forwarding. It is unclear whether their exclusion from the marking rubric indicates that these are considered extra features. These features are:

- Minimal ICMP
- TCP Packet forwarding
- Internal IP releasing on lease timeout
- Byte array packet construction

3.2 Unimplemented Features

We did not implement Ethernet frames in our packet construction, nor did we implement port forwarding.

4 Description of Source Files

The source files are contained in the *networkaddresstranslation* package.
The files are:

- Client.java
- NATBox.java
- ClientInstance.java
- TableEntry.java

5 Program Flow Description

The program flow is explained in detail in sections 2.1 and 2.2.

6 Experiments and Testing

The experiments that we performed were used to test whether the packets were being routed and edited correctly. We also experimented with different manners of allocating the IP addresses in the DHCP implementation.

6.1 Routing tests

In these tests, we sent multiple packets through every possible permutation of Internal and External clients and compared the resultant packets with what was expected.

Internal $\longrightarrow$ Internal:

Expected:

The packet does not get changed and arrives successfully at the destination. The destination receives the packet correctly and the source receives an ACK that the packet was forwarded correctly.

Result:

The result was as expected.

Internal $\longrightarrow$ External:

Expected:

The Packet is edited to reflect the internal client's externally valid IP. The destination receives the packet correctly and the source receives an ACK that the packet was forwarded correctly.

Result:

The result was as expected.

External $\longrightarrow$ Internal:

Expected:

The Packet is edited to reflect the internal client's externally valid IP. The destination receives the packet correctly and the source receives an ACK that the packet was forwarded correctly.

Result:

The result was as expected.

External $\longrightarrow$ External:

Expected:

The packet is dropped and an error packet is returned to the source.

Result:

The result was as expected.

External $\longrightarrow$ Un-allocated Internal:

Expected:

The packet is dropped and an error packet is returned to the source.

Result:

The result was as expected.

Internal $\longrightarrow$ Unknown External:

Expected:

The packet is dropped and an error packet is returned to the source.

Result:

The result was as expected.

Internal $\longrightarrow$ Un-allocated Internal:

Expected:

The packet is dropped and an error packet is returned to the source.

Result:

The result was as expected.

6.2 Conclusions

From our tests, we can conclude that our implementation behaves as expected in all situations. This shows that all of our routing rules have been implemented correctly to our understanding.

7 Issues Encountered

We did not encounter any issues with the implementation of the project, but encountered many with understanding the project specification and the theory.

Most of the time spent on this project was doing research and as a group, we really struggled to understand what was expected of us.

8 Algorithms and Data Structures Used

8.1 Algorithms

DHCP

Dynamic Host Configuration Protocol (DHCP) is a client server protocol to allocate IP's to hosts in a local network. It enables the NAT protocol and also allows computers to be added and removed from networks with minimal manual configuration to be done.

In our implementation, when an internal client connects to the NAT box, it sends a discover message to all nodes in the network using `DatagramSockets` and `DatagramPackets`. When the NAT box notices the discover message, it reads the client's IP that was sent along with the discover message and sends an offer message back. The offer message contains the first unique IP that is availiable in the NAT table. The client then sends an accept message back saying that it would like that IP. The NAT box adds this mapping to the NAT table and then sends a acknowledge message back to the client. The client can then connect to the external internet using this allocated IP.

ICMP

Internet Control Message Protocol (ICMP) is an error-reporting protocol network devices like routers use to generate error messages to the source IP address when problems prevent delivery of IP packets. ICMP creates and sends messages to the source IP address indicating that a gateway to the Internet that a router, service or host cannot be reached for packet delivery. In our implementation, an extra error payload is added to the failed packet before it is returned.

8.2 Data Structures

For our NAT-table and DHCP pool, we used linked lists of TableExtry objects that we created. These objects contain an externally valid address, and an internal IP.

Each client connection to the NAT-Box is mapped to an entry in the NAT-Table, but is represented by a `ClientInstance` object. This object handles the routing functionality for that specific client.

Our Packets are byte array headers, that can be easily sent through sockets.

9 Compilation

From the commandline, go to the folder where the source code is located, as is indicated in the README. Compile all the files by using the `make` command.

10 Execution

After following the compilation instructions, The NAT-Box must be started with the command

```
java networkaddresstranslation.NATBox <Table Refresh Time>
```

Clients can then be started using the command

```
java networkaddresstranslation.Client <host> 8000 <Internal(0)/External(1)>
```

11 Conclusion

This was a fun project to work on, and both of us learnt a lot. The whole cycle, from building a basic DHCP implementation, to routing packets, to testing the final product and all the steps in between,was a big learning curve. It is safe to say that we are both more confident in doing development in a team and know how to optimise the process.

This project also really helped us understand how NAT, IP, DHCP, ICMP and packets work and it was awesome to implement it practically, instead of only learning the theory in class.

We achieved all our goals and were very happy with how our project turned out. All our testing confirmed that we had implemented it correctly.