Implementation of a NAT and Firewall Traversal Library

Damien Auroux

Supervisors: Prof. Karl Aberer
Nicolas Bonvin

Distributed Systems Laboratory

January 14, 2009
I- Motivations and goals of the project
II- Description of the implementation
III- Evaluation and results
Motivations

• NATs are more and more widely used, but they block incoming connections
• P2P and VoIP software require incoming connections to work properly
• NAT configuration is possible but not acceptable

We need a way to get through NATs without any user action!

• NAT traversal techniques are difficult to implement
• Several techniques are needed for a proper result

We need a simple framework for software developers!
Goals and requirements

• A few frameworks already exist, with their advantages and drawbacks
• The designer of the NATaWare project improved its design and wanted a new, better implementation from scratch

Main goals:
• Implement multiple NAT traversal techniques into an open-source Java-based framework
• Efficiency
• Easy to use (black box)

We want software developers to use our framework without hesitation and without any knowledge of NATs
Application

NATaware

NAT Discovery Service

Connection Service

Rendezvous Service

Socket Service

Node

Network

Communicates with other peers

Discovers if the node is behind a NAT

Establishes connections with other nodes

Manages a list of public nodes and relays

Receives all incoming connections
Node P registers to the directory and will periodically exchange lists of public nodes with the directory and other public nodes.

Node A registers to the directory and learns about node P.

Node A asks Node P for public nodes; Node A asks Node P for NAT Discovery and learns that it is NATed; Node A asks Node P to become a rendezvous, Nodes A and P establish a tunnel for incoming connections for Node A.

Node B registers to the directory and learns about node P; node B asks the directory for node A, the directory gives the address of the tunnel.

Node B establishes a connection towards A via P; if possible, Nodes A and B try to establish a direct communication.
1. Opens a connection (red) towards a public node and sends a NAT Discovery Request
2. Forwards the request to the Server Message Handler
3. Tries to open a new connection (green) towards the client to check if it is reachable
4. Forwards the request to the Server Message Handler (only if the Client is reachable from outside)
5. Replies that the Client’s Socket Service is reachable (only if the Client is reachable from outside)
6. The Server Message Handler checks if the Client has replied on the second connection
7. Gives the NAT Discovery Result to the Client on the first connection
**Rendez-vous Service**

**Node A**
- Public Nodes
  - Node B
  - Node C
  - Node D
  - Node E
- Rendezvous Nodes
- Server Message Handler
- Client Message Handler

**Node B (non-NATed node)**
- Public Nodes
  - Node C
  - Node D
  - Node E
- Server Message Handler
- Client Message Handler

1. Opens a connection towards a public node (the list initially contains a few hard-coded nodes). If Node B is unreachable, it is marked as dead and we try another one.

2. Forwards the request to the Server Message Handler

3. Node B adds the nodes sent by Node A to its list

4. Replies with the list of other known Public nodes and their current status

5. Node A updates its list of Public nodes and will periodically check if they are alive and ask for updates
Rendez-vous Service (2)

Node A (NATed)

- Opens a connection towards a public node and asks it to become a rendez-vous
- Forwards the request to the Server Message Handler
- Node B creates a tunnel forwarding incoming traffic for Node A to a chosen port

Node B (non-NATed node)

- Replies to A with the address of the Tunnel Server
- A creates a tunnel client which connects to the tunnel server and receives relayed incoming connections, and adds B to its list of rendez-vous
NAT Traversal: Node A wants to establish a connection with Node B

1. Node A asks the directory D for contact information for Node B
2. Case 1: B is public, Node A receives its address from D and tries to establish direct communication to node B
3. Case 2: B is not public but has a rendez-vous C, Node A receives from D the address of tunnel on node C and connects to B through it
4. Case 2: B receives a connection attempt from A through the tunnel
5. Case 2.1: Node A is public; B creates a reverse direct connection to A
6. Case 2.2: Node A is NATed too, the tunnel will be used for communication
Evaluation

Qualitative evaluation from
• Simulations
• Real tests with several hosts (both locally and on the Internet) was successful.

A few advanced tests could not be performed:
• Large-scale tests
• Various types of NAT devices
• Unusual operating systems
Conclusions

Can software developers use this framework? YES

• Connection between two hosts is always successfully established without any configuration, as long as both hosts and the directory are physically connected and running
• Initialization and connection establishment take no more time than in commercial software
• Reasonably resilient to network outages and malicious behaviour
• Works as a black box, very easy to use for the developer (around 15 generic lines of code)
Conclusions

Will software developers use this framework? PROBABLY NOT YET

- The use of relays between two NATed hosts is inefficient, the framework needs more sophisticated NAT traversal techniques (like TCP hole punching)
- Scalability can be improved by distributing the directory service among multiple nodes
- Security features should be added
- Requires more advanced testing (real-world large-scale tests)