ÉCOLE NATIONALE SUPÉRIEURE DE TÉLÉCOMMUNICATIONS

POLITECNICO DI TORINO
Facoltà di Ingegneria dell'informazione
Corso di Laurea specialistica in ingegneria informatica

Final project report

Peer-to-Peer Reliable Connectivity Across Network Address Translator

A Software Approach to Solving NAT Traversal Problem

Tutors:

AJMONE Marco
DAX Philippe
GORGES Didier

Student:
GABALLO Luca

YEAR 2004-2005

Abstract Peer-to-peer applications need direct connection between nodes, and Network Address Translation (NAT) cause well-knows difficulties since the peers have no globally valid IP address. Some NAT traversal approaches are emerging, but they suffer from several problems. This work analyzes the NAT functionalities, the UDP/TCP communication through the NATs, documents the different approaches and try to find a good solution for the NAT traversal problem.

Contents

Chapter 1

Solipsis

A shared virtual world is a computer-generated space used as a metaphor for interactions. Entities, driven by users or by computer, enter and leave the world, move from one virtual place to another and interact in real-time. The SOLIPSIS system is a network of peers, where peers are connected entities sharing a virtual space. It intends to be scalable to an unlimited number of users and accessible by any computer connected to the Internet. It does not make use of any server and is solely based on a network of peers. [1]

In the SOLIPSIS system, each participating computer runs a specifc software that holds and controls one or several peers. These peers implement the entities of the virtual world and "perceive" their surroundings. Peers can be lite pieces of software and Solipsis aims to be accessible to low end computers at 56kbs and to mobile wireless devices and not only to full featured broadband connected engines. Connected peers may exchange data like video, audio, avatars movements or any kind of events affecting the representation of the virtual world.

SOLIPSIS is a system intended to create a massively shared computer- generated space. This virtual world may welcome entities, lite pieces of software running on any computer connected to Internet. The entities, driven by users or by computer, enter and leave the world, move from one virtual place to another and interact in real- time.

In comparison with other existing systems for shared virtual world, SOLIPSIS is a fully *scalable* system. That is, the behavior of SOLIPSIS is not altered when the number of active entities varies several orders of scale. Moreover, SOLIPSIS ensures the coherency of the virtual world, in the sense that it guaranties that a virtual scene is identically perceived by all entities observing it.

The virtual world is the surface of a torus. Each entity has a position on the torus. The SOLIPSIS protocol provides all the material for communication between entities. Especially, entity characteristics and virtual events can be easily transmitted from one entity to another. The SOLIPSIS protocol also ensures that the virtual world is still unique by guarantying the connexity of the communication graph modeling the network of participating nodes. [21]

Chapter 2

The Problem

If we are building some peer-to-peer (P2P) applications, we require TCP connections between two machines that can connect to the Internet from various places (so they don't have a fixed network address) and, often, they are behind a Network Address Translation (NAT) box. The problem is that computers attached to the Internet via NAT boxes can make outbound connection to the public Internet but cannot receive inbound connections.

NAT boxes allow several computers, in a private network, to share one public IP address and their communication to the public Internet pass through the NAT box, that maintains a table of mappings and use port translation in order to determine which computer should receive the response. This cause many difficulties for a peer-to-peer communication, since the peers cannot be reachable at any global valid network address. The new Internet address architecture, where there is a global address realm and many private address realms interconnected by NATs, make it difficult for two nodes on different private networks to communicate each other directly.

To make the new services based on P2P, like teleconferencing or on-line gaming, completely accessible, it's very import, and urgent, to implement a technique that consent to the different protocols to traverse the NATs. The objective of this work is to study different approaches to pass the NATs and in particular, for each of these, the scalability, the mobility, the standard interfaces, and the security. The work must give us a possible approach to solve the problem.

The next section describes the NAT functionalities and the different types of NAT behaviors .

Chapter 3

Network Address Translation (NAT)

Network Address Translation is a method by which IP addresses are mapped from one address realm to another, providing transparent routing to end hosts. The need for address traslation arises when a network's address cannot be used outside the network either it is a not valid public address, or because the address must be kept private from the externel network. Address translation allows transparent routing beetween two host in different network by modifying end node addresses en-route and maintaining state for these updates [16]. This chapter attempts to describe the operations of NAT devices and to define the terminology used to identify various kind of NAT and work with them.

3.1 NAT Binding

3.1.1 Address and Port Mapping

3.1.2 Port Assignment

3.1.3 Binding Lifetime

3.2 Filtering of unsolicited traffic

The filtering function allows the NATs to drop incoming session that are deemed unsolicited. The most common filtering policy is to permit a communication session to cross the NAT if and only if it was initiated from the private network. All packets arriving from the public Internet, like an attempt by another peer to establish a connection, are dropped because they are not part of an existing communication session, that must be previously initiated by a node in the private network.

3.3 Hairpinning Behaviour

3.4 Packet Mangling

3.5 NAT Behavioral Requirements

To understand a NAT traversal connection the concept of *session* is fundamental. A *session endpoint* for a TCP or UDP session is a pair (IP address, port number) and a session is identified by two session endpoints. So we can define a session like a 4-tuple (source IP, source port, target IP, target port). Address translations performed by NAT are session based and would include translation of incoming as well as outgoing packets belonging to that session. Session direction is identified by the direction of the first packet of that session.

There are two principal NAT functions that interferes with a peer-to-peer application and thus requires NAT transversal support: address translation and the filtering of the unsolicited traffic. In the next sections these functionalities will be analyzed and a list of the NAT types will be presented.

3.6 Address Translation

In order to enable many private hosts behind the NAT to share the use of a smaller number of public IP addresses (often just one) the NAT modifies the IP-level and often the transport-level header information in packets flowing across. So the users behind a NAT have no unique, permanently IP address to use in the public Internet, and can communicate only with the temporary public address (IP and port) that the NAT assign them dynamically. Establish a communication between two peers that reside behind two different NAT (in two different private network) is very hard, because neither of them has a permanent and valid public IP address that the other can reach at any time, so in order to establish a peer-to-peer connection the hosts must rely on the temporary public address their NAT assigns them as a result of a prior outgoing client/server style communication sessions.

Discovering, exchanging, and using these temporary public endpoints requires that the two host collaborate through a well-known node on the public Internet. There are several protocols that allow applications to obtain external communication through the NAT. We will analyze them with the objective to find a good technique and implement it.

3.7 NAT types

There are four different ways NAT boxes implement the mapping and filtering, so, using the classification in [12], there are four NAT types:

1. *Full cone*: all requests from the same internal IP address and port are mapped to the same external IP address and port. Furthermore, any external host can send a packet to the internal host, by sending a packet to the mapped external address.

2. *Restricted cone*: all requests from the same internal IP address and port are mapped to the same external IP address and port. Unlike a full cone NAT, an external host (with IP address X) can send a packet to the internal host only if the internal host had previously sent a packet to IP address X.

3. *Port Restricted cone*: a port restricted cone NAT is like a restricted cone NAT, but the restriction includes port numbers. Specifically, an external host can send a packet, with source IP address X and source port P, to the internal host only if the internal host had previously sent a packet to IP address X and port P.

4. *Symmetric*: A symmetric NAT is one where all requests from the same internal IP address and port, to a specific destination IP address and port, are mapped to the same external IP address and port. If the same host sends a packet with the same source address and port, but to a different destination, a different mapping is used. Furthermore, only the external host that receives a packet can send a UDP packet back to the internal host.

3.8 NAT: security behaviors or not?

3.9 NAT: IETF guidelines

Chapter 4

Solution requirements

- Scalability: a peer-to-peer application is intended open to a theoretically infinite community. A completely scalable system allows the number of instances to grow several orders of scale without any behavior problem. The solution must satisfy this important requirement.
- Standard interface: use standard interfaces for network (e.g. Sockets) allows a simpler implementation and the use of many network packages/libraries.
- Security: ...
- Mobility: communication between peers that move from network to network. SOLIPSIS, like much other P2P applications, is conceived to be implemented on mobile terminals too, like mobile phones or PDA, and the solution must be independent from a particular NAT configuration. We need of an implementation that need for any NAT reconfiguration like the additions of port mappings, allow UDP or turn on UPnP.

Chapter 5

NAT transversal: some approaches

- STUN (Simple Traversal of UDP protocol Through NATs): a UDP-based protocol to determine if the machine is NATed and if so, via what type of NAT. It isn't a cure-all, it allows incoming UDP packets through a subset of existing NAT types (it doesn't enable incoming UDP packets through symmetric NATs) [12]

- STUNT (Simple Traversal of UDP Through NATs and TCP too): extends STUN to include TCP functionality. It is a lightweight protocol that allows applications running behind a NAT to determine external IP and port-binding properties, packet filtering rules and various timeouts associated with TCP connections through the NAT. Knowing these parameters allows applications to establish TCP sessions between two NAT'ed hosts.

- UPnP: The UPnP architecture offers pervasive peer-to-peer network connectivity. The UPnP architecture is a distributed, open networking architecture that leverages TCP/IP and the Web to enable seamless proximity networking in addition to control and data transfer among networked devices. Unfortunately, there are high security problems and many NAT boxes turned off it. ...

- IPv6: the new standard that specifies 128 bit IP addresses. It is possible that the new longer IPv6 addresses will reduce the need for a NAT box. However, as the adoption of the IPv6 addresses is slow, the NAT boxes are used to bridge between IPv4 and IPv6 networks.

- NUTSS: a SIP-based approach to UDP and TCP network connectivity that proposes to combine NAT traversal, URIs, Tunneling over UDP, SIP and STUN. It is a research project at Cornell [5]

5.1 STUN: Simple Traversal of UDP protocol Through NATs

STUN is a protocol that allows users to discover if there is a NAT between them and the public Internet and, in this case, the type of the NAT. It allows to discover if there is an UDP firewall too. With STUN, a peer-to-peer application can determine the public network address allocated to it by the NAT, without require any special NAT behavior. The STUN's objective is to provide a mechanism for NATs traversal and allows applications to work through the existing NAT infrastructure. This section resumes the STUN protocol defined in RFC 3489 [12]

5.1.1 Terminology

In the STUN protocol there are two principal entities: a STUN client, that generates the STUN requests, and the STUN server that receives the STUN requests, and send STUN responses.

5.1.2 Overview of operations

To know behind what type of NAT the client is and to discover his public network address, it starts the STUN discovery procedure sending the initial *Binding Request* to server using UDP [mattere nota per TCP]. A *Binding Request* that arrives to the server may have traversed one or multiple NAT levels, and it source address will be the mapped address created by the NAT closest to the server. The server puts this address in a *Binding Response* message and sends it back to the source IP address and port of the request. The client compare the local IP and port with these send by the server. If the addresses are the same, between the client and the open Internet there are not NATs boxes. In the case of a mismatching one or more NATS are present.

If the NAT is a full-cone, the public IP and port mapped by the NAT are public and any host can use them to communicate with the user behind the NAT. But the user is not sure that the NAT is a full-cone and if everybody can join it on this address. For this reason, it's necessary to follow in the STUN discovery procedure.

The client sends a second *Binding Request* from the same IP and port address but to a different IP address, to a second STUN server that reply with the public IP and port address that NAT mapped for this communication. If they are different from those in the first *Binding Response*, the client knows it is behind a symmetric NAT. Otherwise, the client just knows that the NAT is not a symmetric type and it must to continues the discover procedure. It sends a *Binding Request* with flags that tell to the server to send a response from a different IP address and port than the request was received on. If the client receives this response, it knows it is behind a full cone NAT. Otherwise the NAT can be a port restricted cone NAT or just a restricted cone NAT. To discover it, the client ask to STUN server, to sending a *Binding Response* from the same IP address than the request was received on, but from onother port. If a responce is received, the client is behind a just restricted NAT.

5.2 STUNT: Simple Traversal of UDP Through NATs and TCP too

Simple Traversal of UDP Through NATs (STUN) has enabled a new generation of peer-to-peer applications that can function in the presence of Network Address and Port Translators (NATs) devices. It allows applications to discover the presence and type of NAT and learn the binding allocated by the NAT without requiring any changes to the NATs themselves. However, STUN is limited to UDP and does not work for TCP. The behavior of NATs when it comes to TCP is more complex and cannot be captured with the mechanisms provided by STUN.

Lacking TCP support, many peer-to-peer applications fall back onto transferring data over UDP. As a result applications forfeit congestion avoidance in the form of Explicit Congestion Notification and instead each application must re-engineer TCP friendliness into its UDP based protocol. TCP friendliness often deviates from the application's primary goals and is subsequently ignored causing congestion problems later. Another approach is to run an entire TCP/IP stack encapsulated inside UDP datagrams. This approach incurs overheads introduced by encapsulation, decapsulation and transmission of extra header data.

The protocol described here, Simple Traversal of UDP Through NAT and TCP too (STUNT), extends STUN allowing applications behind a NAT to discover the NATs behaviour for TCP packets and to learn the transport address binding allocated by the NAT. Using STUNT, applications can establish raw TCP sessions with other NAT'ed hosts without using encapsulation, tunelling, relaying or a userspace stack, and without forfeiting any of benefits of using TCP. Like STUN, STUNT requires no changes to NATs and works with a large majority of NATs currently present in the network architechture.

5.3 Hole Punching

5.3.1 UDP Hole Punching

This approach enables to establish a direct UDP session between two nodes, even when both may lie behind different NATs, with the help of a well-known third machine, the *connection broker.* The connection broker must have a public address to allow the NAT'ed peers to contact it and helps them to discover the other peer's public address. UDP hole punching relies on the properties of common firewalls and NATs to allow peer-to-peer applications to *punch holes* and establish direct connection with each other. This technique works good with UDP, but we will see that, with some modifications and attentions, it can works with TCP too.

We will consider three specific scenarios, and how applications can be designed to handle both of them gracefully. In the first situation, representing the common case, two nodes desiring direct peer-to-peer communication reside behind two different NATs. In the second, the two peers reside behind multiple levels of NAT and the last scenarios shows two clients that reside behind the same NAT, but do not necessarily know that they do.

Peers behind different NATs ...

Peers behind multiple levels of NAT ...

Peers behind the same NAT: the hairpin translation ...

Create a peer-to-peer UDP session ...

5.3.2 TCP Hole Punching

To establish a peer-to-peer TCP session we have to use a more refined approach, but this approach is very similar to the UDP hole punching. It is more complicated because from a TCP packet the NAT can extracts much more informations than from an UDP packet.

Create a peer-to-peer TCP session ...

Chapter 6

Our Approach

6.1 TCP NAT Traversal Protocol

Various techniques have been developed for traversing NAT/firewall with UDP, because of the difficulty to establish a TCP connection. This difficulty is due to the asymmetric nature of the TCP protocol. In this chapter we will explain different approaches that we have formulated, implemented and tested to arrive to the final solution.

6.1.1 TCP Traversal with Spoofing (NUTSS like)

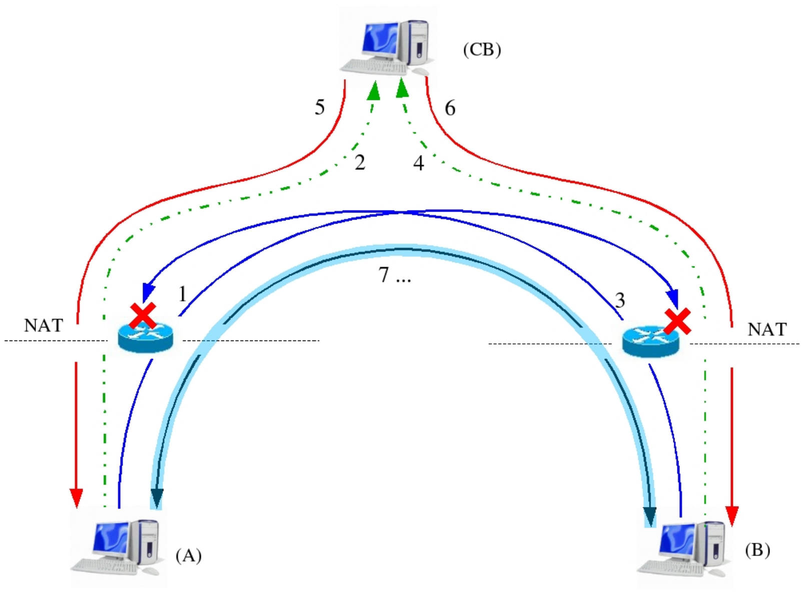

Figure 6.1:

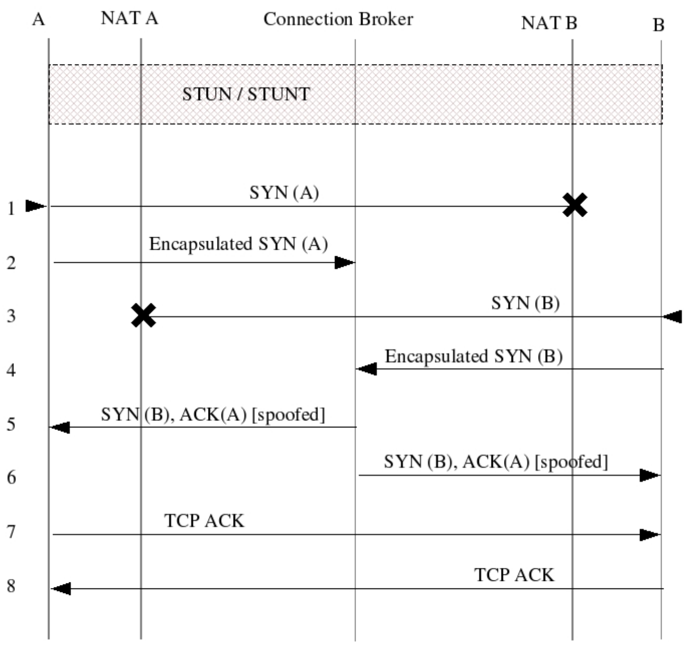

Figure 6.2:

6.1.2 TCP Traversal with Spoofing from well-known machine

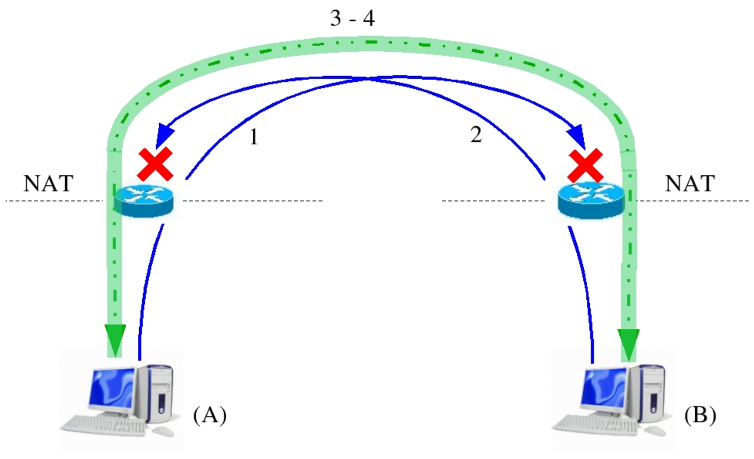

Figure 6.3:

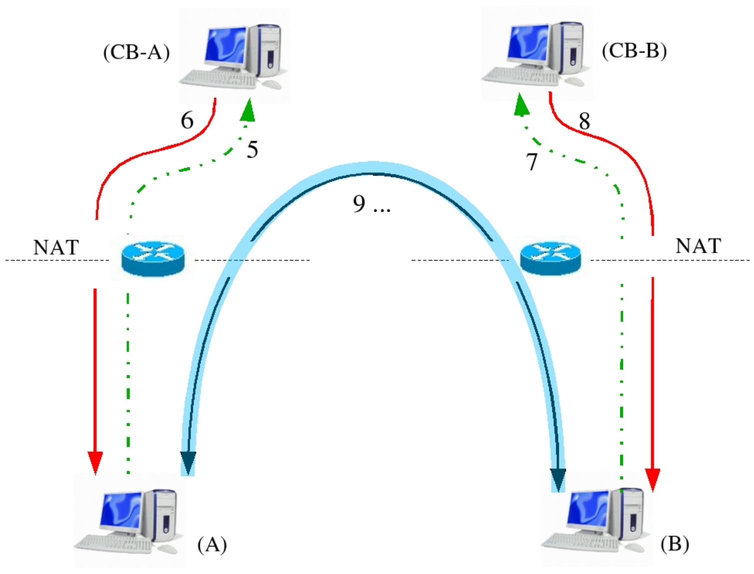

Figure 6.4:

6.1.3 TCP Traversal without Spoofing

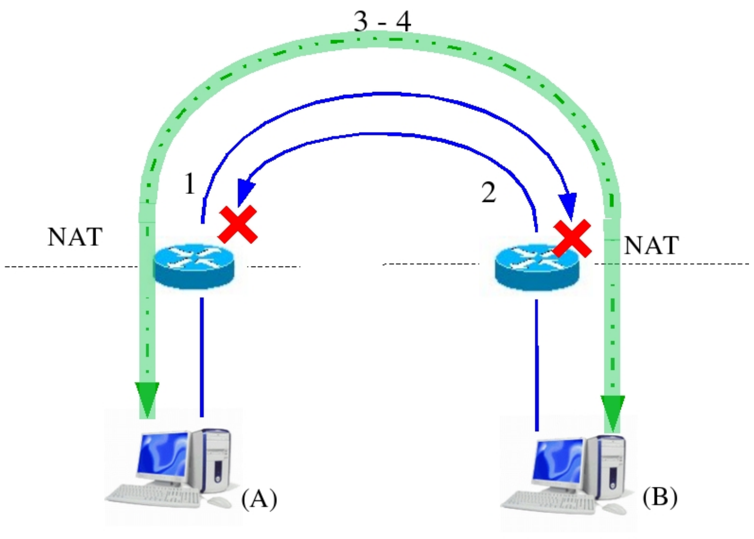

Figure 6.5:

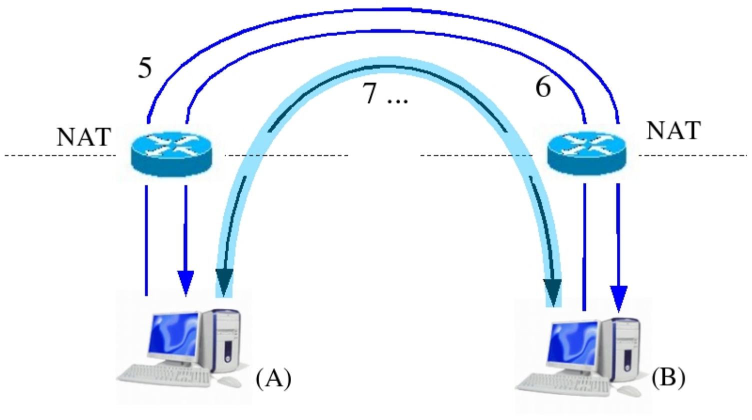

Figure 6.6:

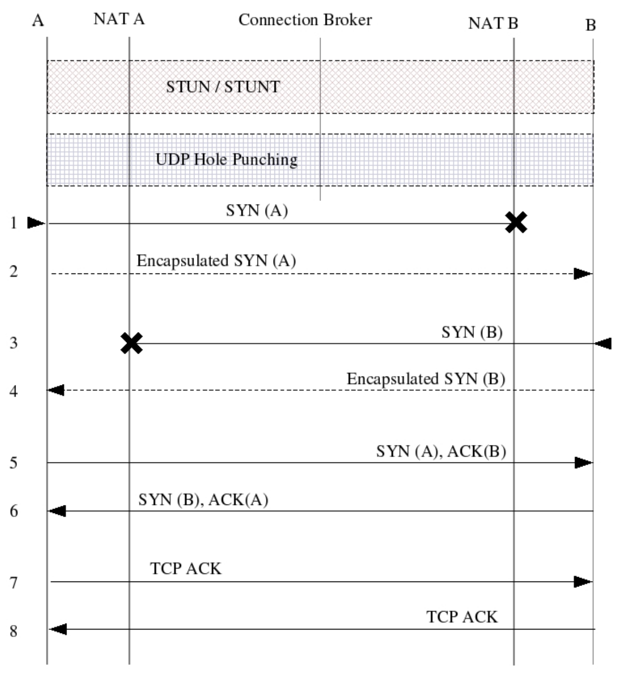
A
NAT A
Connection Broker
NAT B
B
STUN / STUNT
UDP Hole Punching
1
SYN (A)
2
Encapsulated SYN (A)
3
SYN (B)
4
Encapsulated SYN (B)
5
SYN (A), ACK(B)
6
SYN (B), ACK(A)
7
TCP ACK
8
TCP ACK

Figure 6.7:

Bibliography

[1] Solipsis
`http://solipsis.netofpeers.net/wiki2/index.php`

[2] Almost TCP over UDP (atou)
`http://www.csm.ornl.gov/~dunigan/net100/atou.html`

[3] Peer-to-Peer Communication Across Network Address Translators, Bryan Ford, Pyda Srisuresh, Dan Kegel
`http://www.brynosaurus.com/pub/net/p2pnat/`

[4] Connection à travers des pare-feux, Laurent Viennot
`http://gyroweb.inria.fr/~viennot/enseignement/dea/stages2004-2005/firewall.html`

[5] NUTSS: A SIP-based Approach to UDP and TCP Network Connectivity
`http://nutss.gforge.cis.cornell.edu/pub/fdna-nutss.pdf`

[6] Characterization and Mesurement of TCP Traversal Through NATs and Firewalls
`https://www.guha.cc/saikat/pub/draft-imc05-stunt.pdf`

[7] TCP Connections for P2P Apps: A Software Approach to Solving the NAT Problem
`http://reports-archive.adm.cs.cmu.edu/anon/isri2005/CMU-ISRI-05-104.pdf`

[8] Transfert de fichier, TFTP, Laurent Viennot
`http://www.enseignement.polytechnique.fr/profs/informatique/Laurent.Viennot/majReseau/2004-2005/td4/index.html`

[9] Architecture de gestion de traversé de protocoles au travers des pare-feu et des routeurs NAT, JOEL BQO-Lqn TRAN, Université de Sherbrooke, September 2003

[10] Dive Into Python
`http://diveintopython.org`

[11] Twisted Documentation
`http://twistedmatrix.com/projects/core/documentation/howto/`

`index.html`

IETF - Request for Comments

[12] rfc 3489: STUN - Simple Traversal of UDP Through NATs

[13] rfc 793 : TCP - Transmission Control Protocol

[14] rfc 768 : UDP - User Datagram Protocol

[15] rfc 1928: SOCKS Protocol Version 5

[16] rfc 2663: IP Network Address Translator (NAT) Terminology and Considerations

IETF - Internet Draft

[17] NAT Classification Results using STUN, C. Jennings, Cisco Systems, October 24, 2004

[18] STUNT: Simple Traversal of UDP Through NATs and TCP too
`https://www.guha.cc/saikat/pub/draft-imc05-stunt.pdf`

[19] NAT Behavioral Requirements for Unicast UDP

[20] Application Design Guidelines for Traversal of Network Address Trankators

[21] SOLIPSIS Node Protocol

Tool

[22] Python
`http://www.python.org/`

[23] Twisted
`http://twistedmatrix.com/products/twisted`

[24] Emacs
`http://www.gnu.org/software/emacs/`

[25] Subversion
`http://subversion.tigris.org/`

[26] RapidSVN
`http://rapidsvn.tigris.org/`

Mailing list

[27] Twisted-Python mailing list
`http://twistedmatrix.com/cgi-bin/mailman/listinfo/twisted-python`

[28] p2p-hackers mailing list
`http://zgp.org/mailman/listinfo/p2p-hackers`