ÉCOLE NATIONALE SUPÉIEURE DE TÉLÉCOMMUNICATIONS

POLITECNICO DI TORINO
Faculty of information engineering
Specialisation in informatic engineering

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
RABRET Laurent

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. [23]

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 have no 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 important, 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 translation 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 external 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 [17]. This chapter attempts to describe the operations of NAT devices and to define the terminology used to identify various kind of NAT and to work with them.

3.1 NAT Binding

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 this chapter these functionalities will be analyzed and a list of the NAT types will be presented.

3.1.1 Address and Port Mapping

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 [20].

Discovering, exchanging, and using these temporary public endpoints requires that the two host collaborate through a well-known node on the public Internet with a protocol that allow applications to obtain external communication through the NAT.

When an internal open an outgoing session through the NAT, the NAT assigns an external public IP address and port number for this session. Every subsequent incoming packet from the extern endpoint can be received by the NAT, translated and forwarded to the internal endpoint. This mapping between the internal IP address and port tuple and the IP address and port is valid for all the session's duration.

For many applications, and in particular for peer-to-peer, know the NAT behavior with multiple simultaneous session binding is fundamental to allow an high connectivity. The behavior depends by the criteria used in the address mapping for a new session. Like showed in Figure 3.1 two session are established between the same internal address X:x and two different addresses in the external network. When the internal endpoint sends from address and port X:x to the external endpoint Y1:y1 to establish a connection, the NAT creates a mapping for the session identified by the tuple (X, x, Y1, y1) using the public address and port N1:n1. If the internal endpoint creates a second session with Y2:y2, this session is mapped on the NAT with the public addres and port N2:n2 [19]. The relationship between N1:n1 and N2:n2 is critical and determines the NAT behavior. The next paragraph exposes this behaviors and does a classification of the different NAT type.

NAT types for TCP communication There are four different ways NAT boxes implement the mapping and filtering, so, using the classification in [18], there are four NAT types:

- *Independent*: All requests from the same internal IP address and port are mapped to the same external IP address and port. So the NAT reuses the mapping for subsequent session initiated from the same IP address and port (X:x). Specifically, for the mapping showed in Figure 3.1, N1:n1 equals N2:n2 for all values of subsequent endpoint Yi:yi.

- *Address Dependent*: All requests from the same internal IP address and port are mapped to the same external IP address and port only when the destination

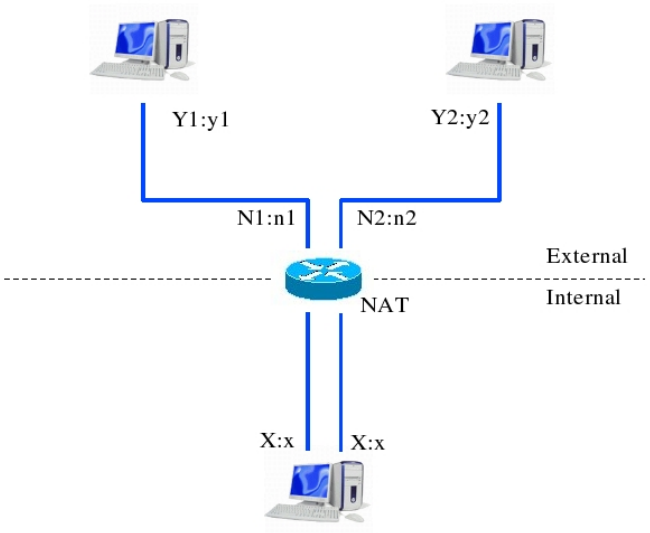

Figure 3.1: NAT address and port mapping

IP is the same, regardless of the external port. Specifically, N1:n1 equals N2:n2 if, and only if, Y2 equals Y1 for any port y.

- *Address and Port Dependent*: It is like an Address Dependent NAT, but the restriction includes port numbers. The NAT reuses the port mapping for subsequent sessions initiated from the same internal transport address (X:x) only for sessions to the same external IP address and port. Specifically, N1:n1 equals N2:n2 if, and only if, Y2:y2 equals Y1:y1.
- *Session Dependent*: A symmetric NAT is one where all sessions from the same internal IP address and port, to the same destination IP address and port, are mapped with a different external IP address and port. Specifically, N1:n1 not equals N2:n2 even if Y2:y2 equals Y1:y1. Thus, with this mechanism, two subsequent sessions have not the same 4-tuple.

This terminology is not globally accepted, some other definitions can be found and it is valid only for TCP protocol.

NAT types for UDP communication For UDP, as there are not session establishment, the mapping is not extremely different but the terminology is not the same. The four treatments observed, like reported in STUN documentation [13], are:

- *Full Cone*: All requests from the same internal IP address and port are mapped to the same external IP address and port and any external host can send a packet to the internal host, by sending a packet to the mapped external address.

- *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 Y) can send a packet to the internal host only if the internal host had previously sent a packet to IP address Y.
- *Port Restricted Cone*: The implementation is like in a restricted cone NAT, but the restriction includes port numbers. Specifically, an external host can send a packet, with source network address Y:y, to the internal host only if the internal host had previously sent a packet to IP address Y and port y.
- *Symmetric*: 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.

3.1.2 Port Assignment

The *port preservation* is the attempt of the NAT to preserve the port number used internally when does a mapping to an external IP address and port. For the mapping showed in Figure 3.2, if the port n is already in use and no other external IP addresses are available, the NAT switch to the Network Address and Port Translator (NAPT) mode. The *port preservation* behavior does not guarantee that the external port n will alway be the same as the internal port x but only that the NAT will preserve the port if possible. The difference in allocation (n2 - n1) between a fresh external binding (N2:n2) when the last port allocation was (N1:n1) is called NAT's *Binding Delta*. It may be constant or random and the tests have showed that the delta value is usually 1 or 2 and rarely random. If the NAT is an Independent type the Binding Delta value is 0. Some NATs assign the same external IP and port to two different internal transport addresses even in the case of collision and rely the mapping from the external endpoint (Y1:y1, Y2:y2). This *port overloading* fails if the two internal endpoints are establishing sessions to the same external destination (e.g., a SIP proxy, a web server, etc). If a NAT preserves the parity of the port, i.e., an even port will be mapped to an even port, and an odd port will be mapped to an odd port. Specifically, for the Figure 3.2, if the external port n assigned for the internal port x is even if and only if x is even then the NAT preserves even-odd port parity. If the NAT assign address N1:n1 and N1:(n1+1) for two connection initiated from internal ports X1:x1 and X1:(x1+1) then the NAT preserves *next-higher port parity*, referred like *port contiguity preservation* too. The *port contiguity preservation* can be usefull for the RTCP=RTP+1 [1] rule in the RTP protocol but this port reservation is problematic since it is wasteful and a NAT cannot reliable distinguish between UDP packets where there is or not contiguity rule. The last

[1] Real Time Protocol (RTP) and Real Time Control Protocol (RTCP)

point for the port assignment behavior is the binding range allocated by the NAT. The IANA defined three ranges: *well-known* from 0 to 1023, *registered* from 1024 to 49151, and *dynamic/private* from 49152 to 65515. Use an already registered port can cause bad effects. The NATs' behavior for range port allocation is variable. Some NATs allocate an external port binding that is in the same port range as internal port, other NAT devices may prefer to use the dynamics range first or possibly avoid the actual registered ports in the registered range. Other NATs preserve the port range if is in the well-known range. [18] [19]

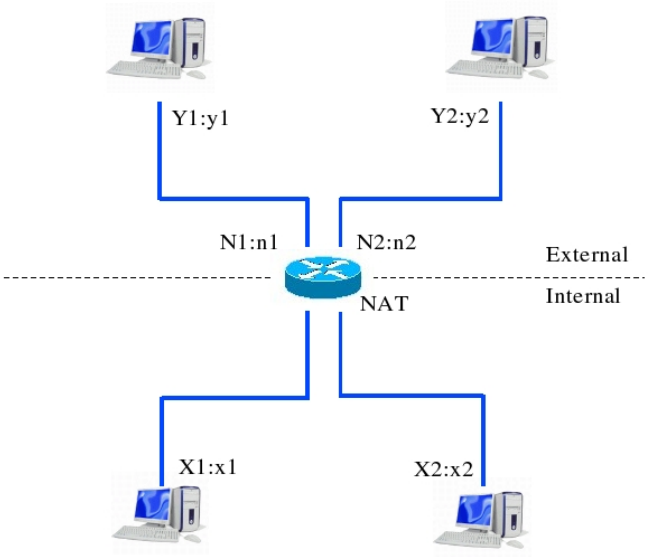

Figure 3.2: The mapping in a Network Address and Port Translator (NAPT) mode

3.1.3 Binding Lifetime

A NAT keeps the mapping alive for an idle period. After this period the allocated binding address is released. The length of this idle period and the way the mapping timer is refreshed vary and could depend on the connection state and the NAT implementation. For a TCP connection the idle timer can be short when the connection is in the SYN_SENT or LAST_ACK state and long in the ESTABLISHED state.

The binding may be released upon receiving certain packets such as TCP RST packets or ICMP errors for a connection.

Some NATs do a mapping N:n for several sessions (X1:x1; Yi:yi). If this mapping is refreshed for all session on that mapping by an outbound traffic, the NAT is said to have a NAT *Mapping Refresh Scope* of *Per mapping*. Otherwise we have a *Per session* Mapping Refresh Scope and the mapping is refreshed only on a specific

session on that particular mapping by an outbound traffic. For some connectionless protocol, like UDP, the NAT can have different *refresh direction* behavior. If the mapping is refreshed for an outcoming packet the NAT has an *Outbound Refresh* behavior of *true*. If the mapping is refreshed for an outcoming packet the NAT has an *Inbound Refresh* behavior of *true*. For NAT that refresh the mapping for incoming and outcoming packets both properties are *true*.

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, and the NAT cannot know the destination host in the private network. Every TCP connection must be often previously initiated by a node in the private network. This is not a guarantee that the NAT will accept the inbound packet. More specifically, using the mapping showed in Figure 3.1, we can classify the NAT behavior in four different criteria for endpoint filtering, like exposed in [18] and [19]:

- *Open*: The NAT does not filter any packets.
- *Endpoint Independent*: The NAT filters out only packets not destined to the internal address and port X:x, regardless of the external IP address and port source. Specifically, for mapping in Figure 3.1, if there is a connection between X:x and Y1:y1, mapped on NAT as N1:n1, the NAT routes the incoming SYN packets with destination N1:n1 to the internal address X:x for all values of source address Yi:yi.
- *Endpoint Address Dependent*: In this case, the NAT will filter out packets from external IP Y destined for the internal endpoint X:x if X:x has not sent packets to Y previously, regardless of the external port. In other words, as showed in Figure 3.1, if a connection exists between internal address X:x and external address Y1:y1, NAT routes incoming SYN packets to X:x, only for the Y1 IP source address but for any value of the source port y.
- *Endpoint Address and Port Dependent*: This behavior is similar to the previous, except that the filtering depends of the external port too. The NAT will filter out packets from external endpoint Yi:yi destined for the internal endpoint X:x if X:x has not sent packets to Yi:yi previously. NAT will route incoming SYN packets to X:x, only for the Y1:y1 source address

Moreover, a NAT may filter out some TCP packets viewed as insolecited traffic even if they come from the IP and port to which a packet was sent recently. For

example, if a SYN was sent to the external address Y1:y1 through the NAT that maps this connection with the address N1:n1, an incoming SYN from endpoint Y1:y1 to the destination N1:n1 may be filtered out. Like SYN, all other TCP packets from Y1:y1 to N1:n1 are filtered out, except for a TCP SYNACK or TCP RSTACK.

There are different NAT's reactions to an unsolicited packet. The NAT may generate an ICMP error or TCP RST packet to inform the sender, or simpler, drop the packet silently. The behavior in front of an unsolicited packet is important for the NAT traversal, especially if we attempt to establish a TCP connection between endpoint behind two different NATs. An ICMP error or a TCP RST packet may release the mapping for a particular tuple (X:x, Y:y) as explained in section 3.1.3.

3.3 Hairpinning Behavior

If two host are behind the same NAT and want to exchanging traffic the NAT that supports hairpinning can route the traffic between the two endpoints without apply the filtering rules, even if the endpoints use each other's external IP addresses and ports. Following the mapping in Figure 3.2, if the internal host X1:x1 sends packets to the external address N2:n2, that is an active mapping for the internal address X2:x2, the NAT routes the packets to X2:x2 with source address either the original internal source address X1:x1, or translates it to the external mapped address N1:n1. A non-translation of the internal source address to the external mapped address may cause problems to applications that expecting an external IP address and port.

3.4 Packet Mangling

A NAT may mangle packets by modifying bytes that need not be changed for successful operation or by not changing bytes that should be changed. In particular, a NAT may change the IP header of the packet traversing the NAT: modifying the TCP sequence number adding a random offset, and subtracting the same offset from the acknowledgment numbers in incoming packets; changing the time-to-live value to something different of its value decremented by one. It may even mangle payloads by translating bytes that look like the internal or external transport address encoded in the message payload. It may not correctly translate IP packets in the payload of ICMP packets.

3.5 NAT Behavioral Requirements

3.6 NAT: security behaviors or not?

3.7 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.
- Multi Operative System context: ...
- 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

TCP over UDP: ...
...

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.

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, and require no special NAT behavior and no changes to NATs. 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 [13]

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. 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 compares 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 NAT boxes. In the case of a mismatching one or more NATs are present.

If the NAT is a full-cone, the 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 continue the discover procedure. It sends a *Binding Request* with flags that tell 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 another port. If a responce is received, the client is just behind a Restricted NAT. The Figure 5.1 shows the flow for the type discovery process. The different types are be explained in section 3.1.1.

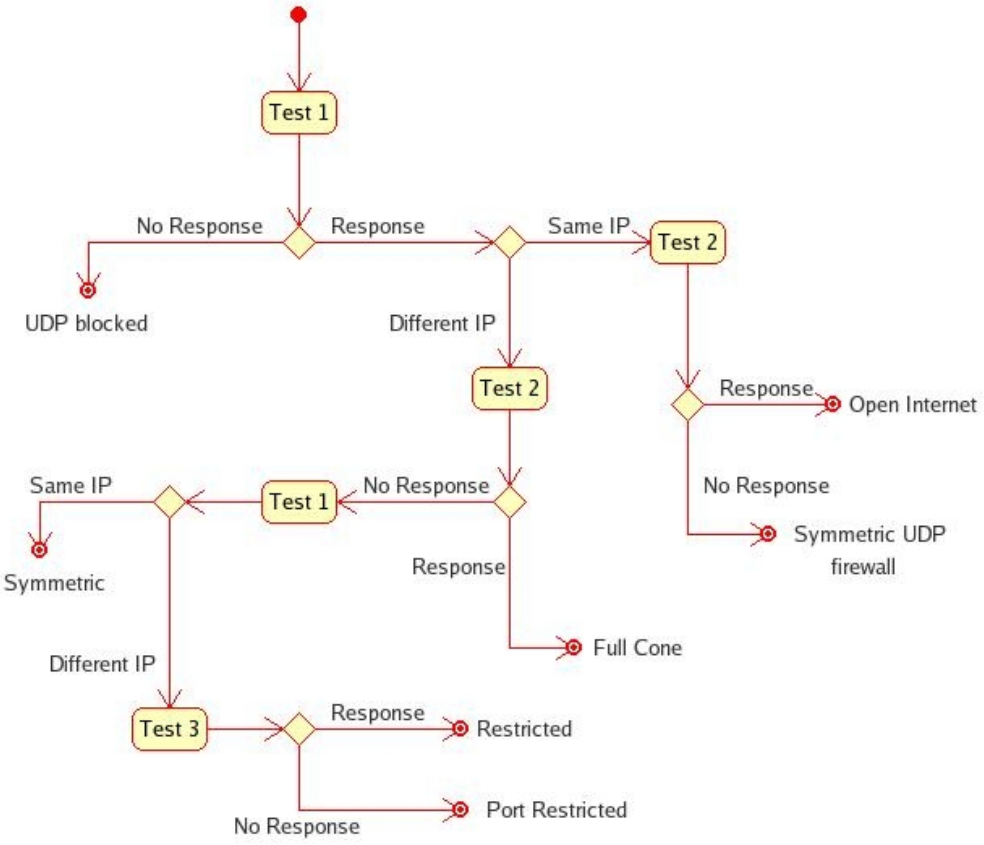

Figure 5.1: STUN - Flow for for type discovery process

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

The STUN protocol 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.

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 behavior 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, tunneling, 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 architecture.

5.2.1 Overview of operations

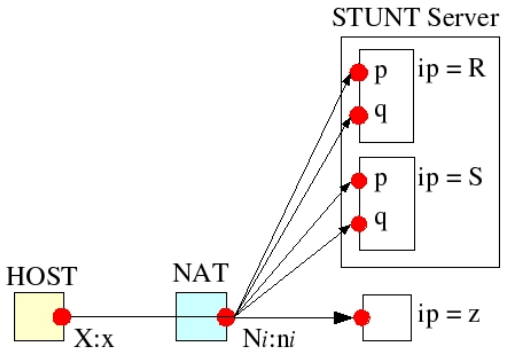

Figure 5.2: STUNT - binding behavior

A STUNT client contacts one STUNT server multiple times to check if the binding changes. If it does not then the client checks if the binding changes for a different server at the same IP address but different port, and finally whether it changes for a third server at a different IP address and port. Once the NATs binding behaviour is known the client can find or predict the binding for new connections through a simlpe algorithm.

A client typically also learns the filtering behaviour of the NATs to decide between TCP Traversal techniques. It does so by testing the NATs response to incoming TCP

SYN packets in various situations.

The Figures 5.3 and 5.4 show how the client can accomplish useful tasks to discover the binding and endpoint filtering behaviors.

In the Figure 5.3 there are the operations to discover the NAT's binding behavior. The client initiates Test 1, where it establish a TCP connection to the server R:p, bound to local address X:x (Figure 5.2). It sends a *Binding Request*. The server puts the source address N1:n1 in a *Binding Response* message and sends it back to the source IP address and port of the request. The client compares the local IP and port X:x with these send by the server. If the addresses are the same, between the client and the open Internet there are not NAT boxes.

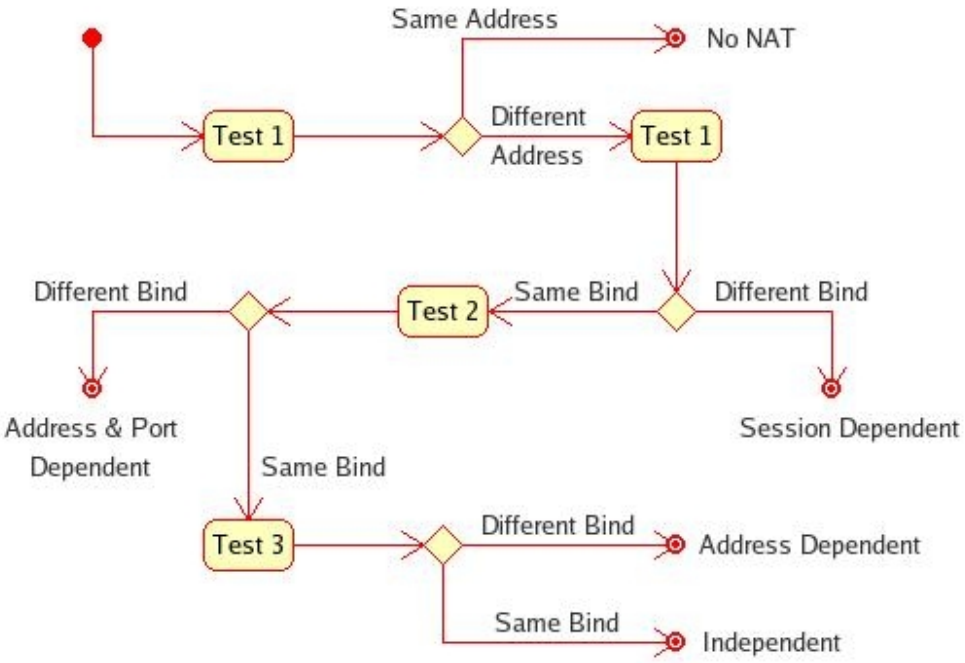

Figure 5.3: STUNT - binding behavior

In the case of a mismatching one or more NATs are present. The client then repeats Test 1 and compares the address N2:n2 sent by server with N1:n1, if they are different the binding is session dependent. Otherwise the client start Test 2 that is similar to Test 1 but connects to server R:q. It learns its binding, call it N3:n3. If N3:n3 is not the same as N1:n1 the binding behavior is Address and Port Dependent. If the mapping did not change Test 3 is executed, which is like Test 1 with S:q as the server address. If the new binding is different from N1:n1 then the binding

behavior is Address Dependent, otherwise it is independent of the destination.

To know the endpoint filtering behavior the client executes the tasks in Figure 5.4. It establish a TCP connection with STUNT server and initiates Test 1 where the server sends an initial SYN from an alternate IP and port to the client's IP but to a random port. If the SYN is received by the client then the NAT has a Open filtering behavior. Otherwise Test 2 is done, where the client establish a TCP connection with the server, send a *Packet Request* message. Then the server replyes sending a SYN packet from a different address and port. If the SYN is received then NAT allows any host to contact the client once a mapping is established, thus the NAT has an Endpoint Independent behavior. If the test fail, the client initiates Test 3 which is identical to test Test 2 but the SYN is sent by the server from the same address and a different port. If the client receives the packet then the NAT filter is Address Dependent, otherwise Address and Port Dependent.

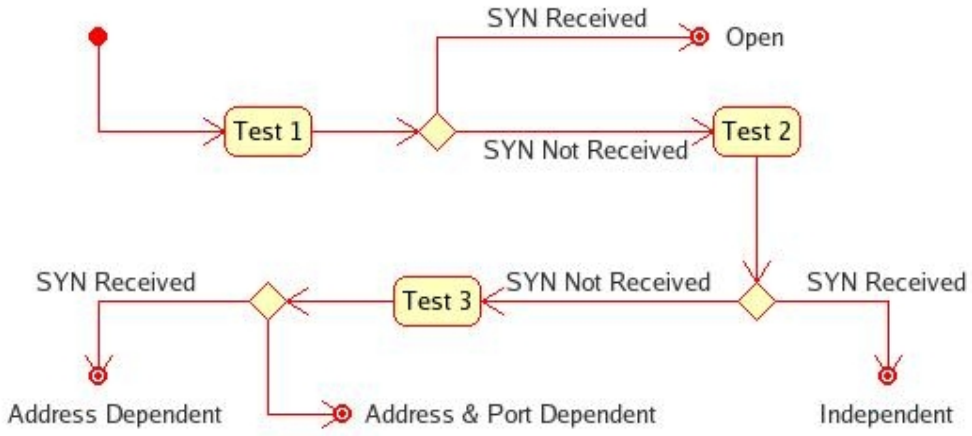

Figure 5.4: STUNT - endpoint filtering behavior

STUNT improvment: These tasks, however, don't allow to discover the configuration where the NAT doesn't accept incoming SYN at all, even if a previous SYN has been sent to the destination address and port. It's for this reason that we suggest a STUNT variation, adding a fourth test, where the client establish a TCP connection with server to communicate with it and send a SYN packet to an alternate port k on STUNT server. The packet has a low time-to-live and doesn't reach the server. An ICMP error occurs. Now the client ask to the server to send a SYN packet from port k. If the client doesn't receive the packet then the NAT filter is Address and Port Dependent, otherwise the NAT doesn't accept incoming SYN.

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. In this section, we will see how Hole Punching works, and it will be usefull to implements the entire architecture for TCP communication across NATs.

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 ...

...

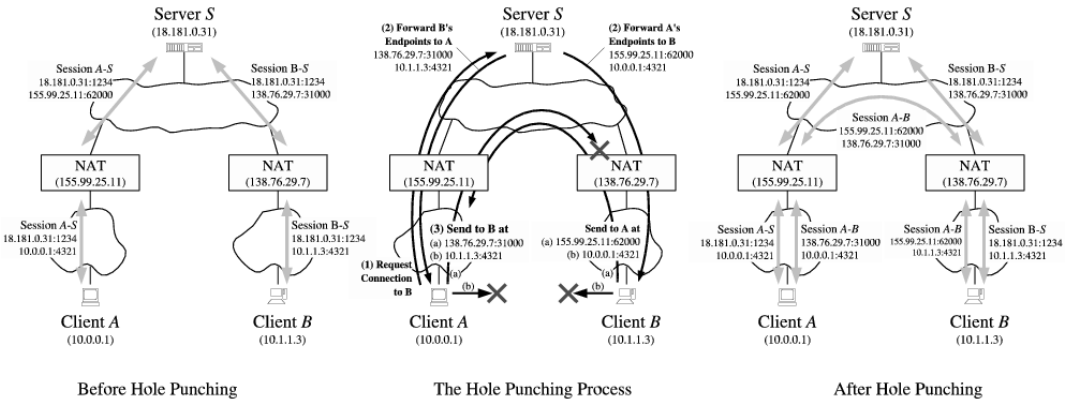

Figure 5.5: UDP Hole Punching: peers behind different NATs

Peers behind multiple levels of NAT ...

...

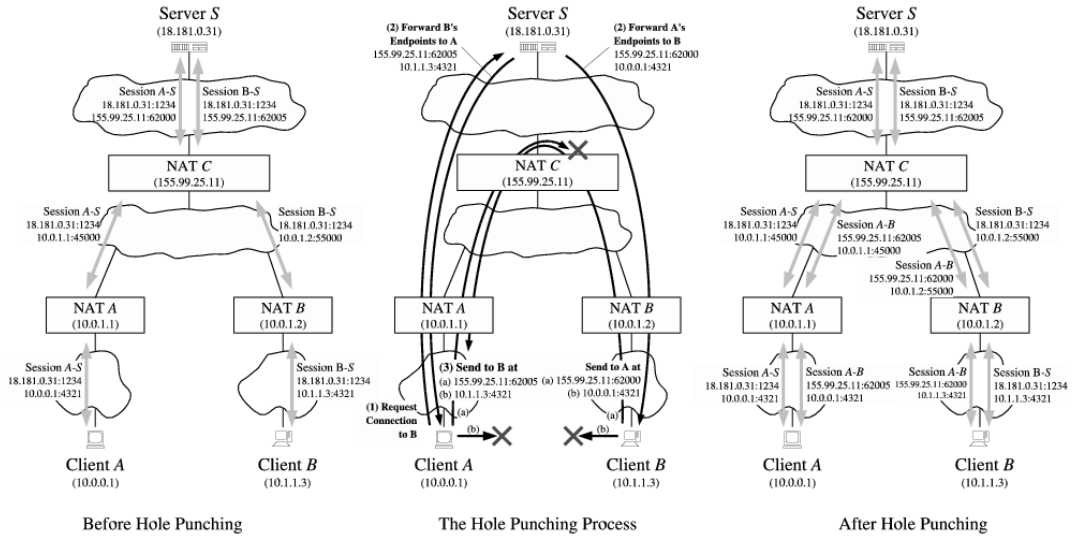

Figure 5.6: UDP Hole Punching: peers behind multiple levels of NAT

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

...

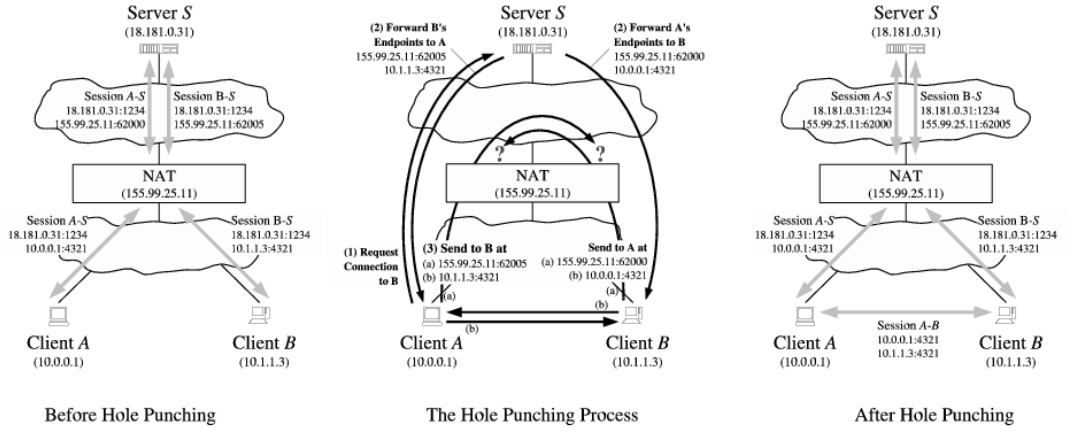

Figure 5.7: UDP Hole Punching: peers behind the same NAT - hairpin translation

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. In a traditional TCP connection one party must be the initiator (send the initial SYN packet) while the other listen for the initiation. The listening peer will be prevented from seeing the initial SYN because the SYNs will dropped at the listening peer's NAT. So, in order to establish a TCP connection between two hosts behind NATs, each NAT must believe that the connection is initiated from the endpoint in its private network. To obtein this, some approaches are proposed in the next chapter.

Chapter 6

Our Approach

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.

Independently other authors worked on TCP connectivity through NAT and had similar results to ours. Severals approaches developed by other authors are exposed in chapter 5, and we have centralized our attemption on four particular solutions: the NUTSS framework [5], a SIP-based approach to UDP and TCP network connectivity, the NatTrav solution [7], based on standard communication interfaces and stacks, the NATBlaster method [8], a TCP connectivity technique avoiding spoofing, and the P2PNAT [3], a peer-to-peer communication across NAT that takes advantage of the simultaneous open scenario defined in the TCP specifications [14].

6.1 Network Architecture

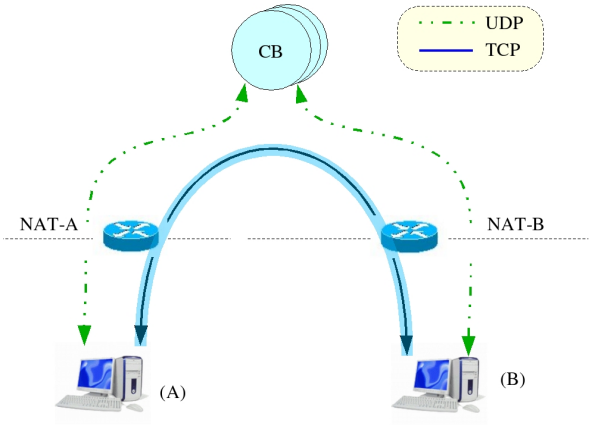

Figure 6.1: Network Architecture

The Figure 6.1 shows the net architecture implemented in the different approaches. Like explained in chapter 2, the problem comes up when two endpoints want to establish a TCP connection but both are behind a NAT box. As we will see in this chapter we could use a third-party globally reachable in the public Internet, the Connection Broker (CB). In the CB we will concentrate functionalities like (i) discovering NAT configuration and mapping through the STUN/STUNT protocols, (ii) allowing an UDP connectivity between endpoint using the Hole Punching protocol and (iii) facilitating the TCP connection establishment through the NATs. The Connection Broker can be replicated to allow availability and scalability. We can have, thus, more than one CB and every endpoint involved in a connection could be connected to a different CB. A CB connectivity and information integrity is required in the solution implementation.

In the SOLIPSIS environment, such as in every peer-to-peer world, each node in the peer-to-peer network that has valid public IP address can be a potential CB, or, more generally, a Super Node to help other NATed endpoint to establish a NAT traversal connection.

6.1.1 Puncher Protocol

All the methods presented in this chapter need a previously UDP data exchange to recuperate all the required endpoints' information. To establish an UDP traversal data communication, we have implemented the Hole Punching approach, exposed in the section 5.3.1. The Figures 6.2 and 6.3 show the message sequences to allow UDP communication between endpoints and Connection Broker. An endpoint A that wants to establish TCP connection with another endpoint B, cannot communicate directly with it but it have to ask help to a third-party in the Internet network, the Connection Broker. Specifically, the Figure 6.2

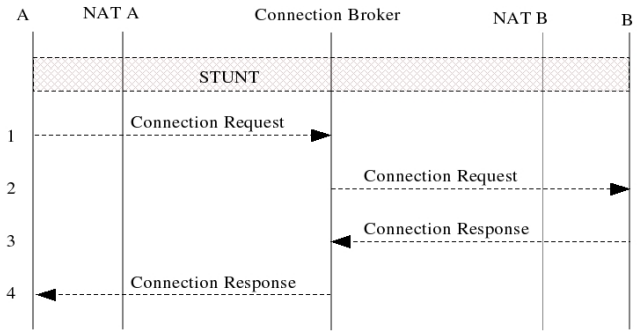

Figure 6.2: Hole Punching Protocol through Connection Broker

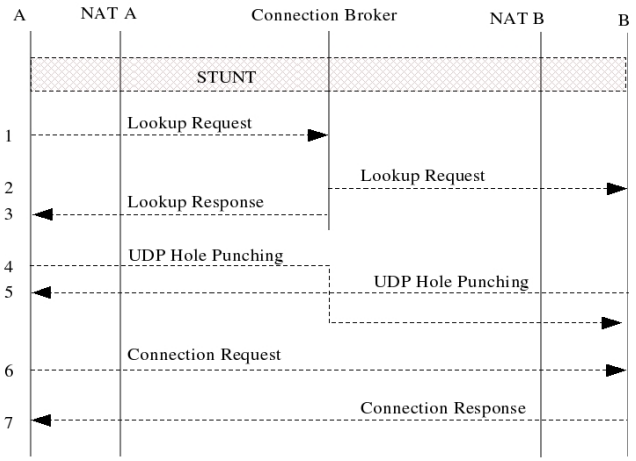

Figure 6.3: Hole Punching Protocol for directly UDP communication between endpoints

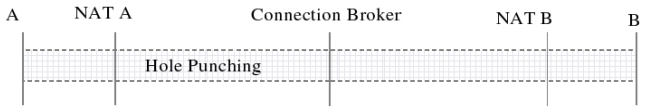

Figure 6.4: Hole Punching Protocol

6.2 TCP NAT Traversal Methods

...

6.2.1 Behaviors Observed by NAT and Application

What the NAT observes and how it reacts with the different approaches during TCP connection attempts depends on the timing, TCP implementation and NAT configuration. The NAT can see different inbound and outbound packet sequences and, for every one of them, it can reacts in different modes. The different sequences observed for the tests are tied to the endpoint filtering behaviors of the NAT exposed in paragraph 3.2.

- *Inbound SYN*: a SYN packet used to initialize a TCP connection is consired by the major number of NAT an unsolicited traffic and it isn't forwarded to the internal endpoint, that usually has no globally valid public address. In accord with to the NAT classification in paragraph 3.2, only the *Open* type allows to a SYN packet to traverse the NAT and enter in the local area network (LAN). More important, in this case, is to consider the NAT reaction to an unsolicited SYN packet, because, like showed in Figure 6.1, NAT-B message reply to a SYN packet generated by endpoint A, will be an incoming packet for NAT-A. NAT-B may reply with a TCP reset response (RSTACK), or an ICMP unreachable error, or may drop the packet silently and a timeout will be raised.

- *Low time-to-live value*: The initial SYN packet may have a low time-to-live to avoid the destination's NAT response. In this case an ICMP error is sent by network between NAT-A and NAT-B. If the time-to-live is not reduced, we will be in the previous situation where the destination's NAT will see an inbound SYN. Several tests have showed that the majority of the NAT, to an inbound SYN, replies dropping the packet silently. It for this reason that an approach that leaves the time-to-live unchanged is more effective [1].

- *Inbound ICMP error and reset response RSTACK*: An inbound packet, like ICMP error or RSTACK, may cause the release of the mapping on the NAT or reducing the binding lifetime.

[1] In our implementation, we chose to leave the time-to-live unchanged to obtain better results. A future improvement can make the method's chosen adaptable with the NATs to be crossed

- *A not normally packet sequences*: Various non-standard sequences, as an outbound SYN followed by an inbound SYN, or an outbound SYN followed by outbound SYNACK, may be seen as not normally. The NAT filtering behavior may filter out this messages and, additionally, release the mapping for the addressee endpoint.

The various packet sequences and the percentage of NATs not accepting them is showed Table 6.2 [6]

Sequence	**Filtered**
outbound SYN - inbound SYNACK	0%
outbound SYN - inbound SYN	16.1%
outbound SYN - inbound ICMP - inbound SYNACK	4.5%
outbound SYN - inbound ICMP - inbound SYN	22.2%
outbound SYN - inbound RST - inbound SYNACK	17.1%
outbound SYN - inbound RST - inbound SYN	27.9%
outbound SYN - outbound SYNACK[11]	-

Table 6.1: Packet sequences and percentage of NATs not accepting them

[11] There are no statistical results for this sequence at the moment.

6.2.2 TCP Traversal: STUNT and P2PNAT Methods

This method is the simplest to implement, but, as showed in the [statistical chapter!!!] it is the most effective and doesn't require many user issues.

The message sequence in Figure 6.5 (a) shows this approach proposed in [18] and [7]. An endhost A sends out a SYN packet (message 1); this packet is blocked on the other side by the NAT-B that sees it as an unsolicited traffic. The sender A than abort the connection attempt and creates a passive TCP socket on the same address and port. The endpoint B then initiates a regular TCP connection sending a SYN packet (messages 2, 3 and 4).

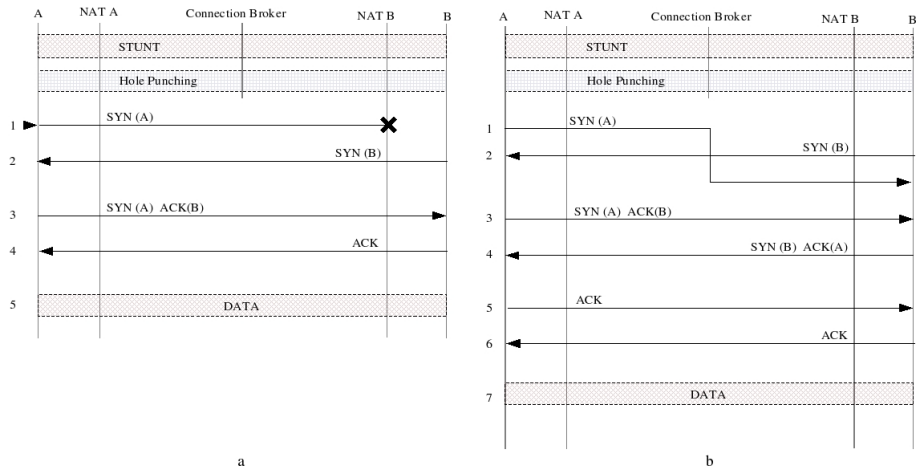

Figure 6.5: The (a) STUNT-2 and (b) P2PNAT methods

Method's issues. The NAT must accept the sequence outbound SYN followed by an inbound SYN that is not a normal sequence seen in a TCP client/server connection, so the NAT may block it. Moreover, NAT-B can reply with an ICMP unreachable error or with an RSTACK packet that may bring NAT-A to not accept the following incoming SYN or, definitively release the session's mapping.

The second method presented in this paragraph is a variation of the first approach to reduce the NAT rejection of the sequence of an inbound SYN following an outbound SYN. It takes advantage of the simultaneous TCP connection specified in [14]. As the Figure **??** (b) shows both endpoints, A and B, send a SYN packet, initiating a TCP connection. Normally, if the two endpoints are well synchronized, the outcoming SYN packet cross the NAT before that the incoming SYN packet reaches the same NAT. Specifically, like showed in Figure 6.5 (b), the SYN(A) crosses the NAT-A before that the SYN(B) reaches the NAT-A, because the distance between A

and NAT-A is smaller than distance between NAT-A and NAT-B. When one end's SYN arrives at the other end's NAT it can be dropped for two reasons. The first can be because it is arrived before the other end's SYN leaves that NAT. The second reason is that NAT not accept at all an incoming SYN, after an outcoming SYN too. The first endpoint's stack ends up followinf TCP simultaneous open while the other side follows a normal open. In this case the message sequence is like the first method presented before, the STUNT-2 method.

Method's issues. This approach requires that at least one NAT accepts the sequence of an inbound SYN after an outbound SYN. Moreover, it requires a tight synchronization between endpoints and no drastic actions if RST packets or ICMP errors are received coming from destination's NAT. In addition, not all the operative systems support the TCP simultaneous open.

Implementation's issues. ...

6.2.3 TCP Traversal: STUNT Method with Spoofing

With this method, proposed in [5] and [18] too, we try to establish a TCP connection traversing NATs that not accept the sequence outbound SYN - inbound SYN. Both endpoints, A and B, send an initial SYN (message 1 and 3 in Figure 6.6), that is blocked on the other side as an unsolicited traffic. The endpoint learn the initial TCP sequence number by listening for the outbound SYN over a RAW socket. Both endpoints communicate these numbers to a globally reachable Connection Broker, CB, (messages 2 and 4). The CB can create and send the SYNACK response for A and B, spoofing the source address (messages 5 and 6). The ACK messages complete the TCP three way handshake (messages 7 and 8).

Method's issues. The NAT may change the TCP sequence number of the initial SYN such that the spoofed SYNACK based on the original sequence number appears as an out-of-window packet when it arrives at the NAT. Moreover, it requires a third-party to spoof a packet from an arbitrary address, that can be dropped by network filters. Statistics show that more than 70% spoofed traffic is detected by network controls. Both the NATs may drop the inbound SYN silently, or reply with an ICMP error or a RSTACK packet that may compromise the connection establishment. It needs, in addition, to open a RAW socket on endpoints' machine, which requires superuser privileges on that machine.

Implementation's issues. ...

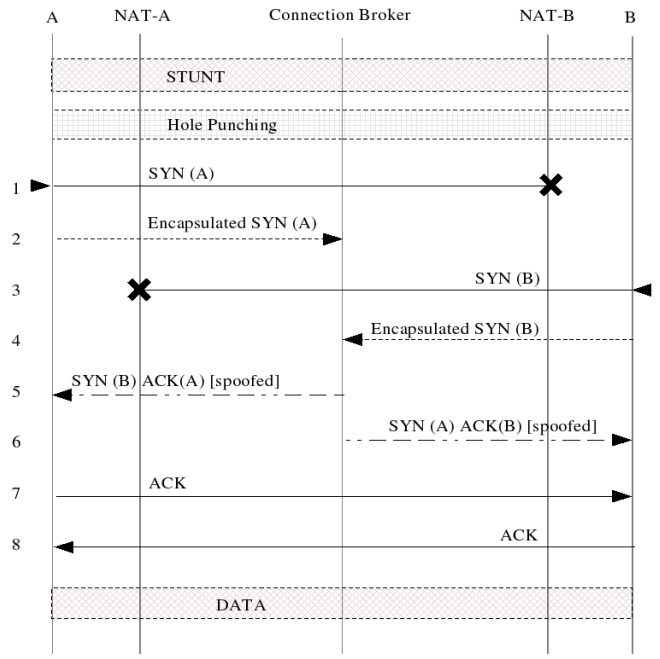

Figure 6.6: The STUNT-1 method's sequence message

6.2.3.1 TCP Traversal: STUNT Method with Spoofing from Well-Known Machine

To limited the spoofing detection problem, this method comes as a variation of the previous method that requires spoofed packets to be injected from Connection Broker to any endpoints. The message sequence is similar to that showed in the STUNT-1 approach, with an adjunctive step to chose the good machine that can spoof packet without network detection. In this approach, the Hole Punching session allows a directly UDP session betwen endpoints [2], like showed in Figure 6.3.

Before to start the connection attempt, each endpoint starts a *test spoofing procedure* to find a Super Node that can spoof packets to the sender of the request forcing the packet's source address to the remote endpoint (messages 1 and 2). Both endpoints send the initial SYN packet that is blocked on the destination's NAT (messages 3 and 5), learn the initial sequence number through a RAW socket and communicate these numbers to the other endpoint using the directly UDP session (messages 4 and 6). With messages 7/8 each endpoint communicates via UDP his initial sequence number, the remote sequence number and address to the Connection Broker choose that craft a SYNACK packet and spoofs it to the endpoint (messages 9/10). The ACK messages complete the TCP three way handshake (messages 11 and 12).

[2] The same result can be obtained using the Connection Broker to exchange the UDP messages between endpoints. For more clarity and simplicity we have preferred a directly UDP session.

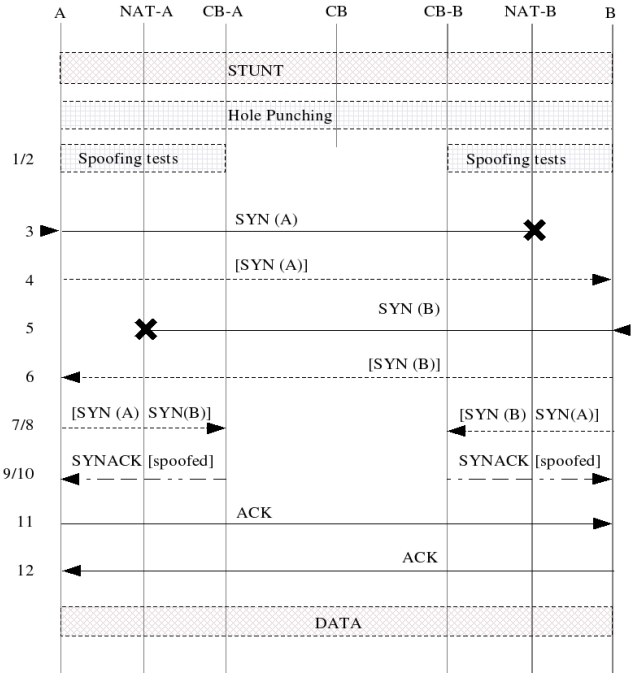

Figure 6.7:

Method's issues. Like in the previous approach, the NAT may change the TCP sequence number of the initial SYN such that the spoofed SYNACK appears as an out-of-window packet. It requires a third-party to spoof a packet and, even if we have solved the network detection problem, is not certain that each endpoint can find a Super Node allowed to spoof a packet from the address requested. NAT's reply to an inbound SYN with an ICMP error or a RSTACK packet may compromise the connection establishment. A RAW socket must be opened on endpoints' machine, which requires superuser privileges.

Implementation's issues. ...

6.2.3.2 Some considerations on spoofing approach

Spoofing is not good!!! ...

6.2.4 TCP Traversal: NATBlaster Method

To solve the spoofing problem, here is exposed a possible solution, presented in [8] and called NATBlaster. We suppose that both the NATs involve in the communication don't accept incoming SYN packet, after an outcoming SYN packet, thus the

method P2PNAT, presented in paragraph 6.2.2, cannot be applied. Each endpoint sends out the initial SYN packet listening the sequence number through a RAW socket. Both the SYN packet are dropped by the NATs. The two endpoints exchange the sequence number (messages 2 and 4) through an UDP connection and each sends a manipulated SYNACK packet the other expects to receive through a RAW socket. Each endpoint manipulates the SYNACK packet just setting the ACK sequence without modify the source address, and thus, without spoofing, and sends it to the other endpoint (messages 5 and 6). Once the SYNACK packets are received, the connection setup is completed with the ACKs exchange.

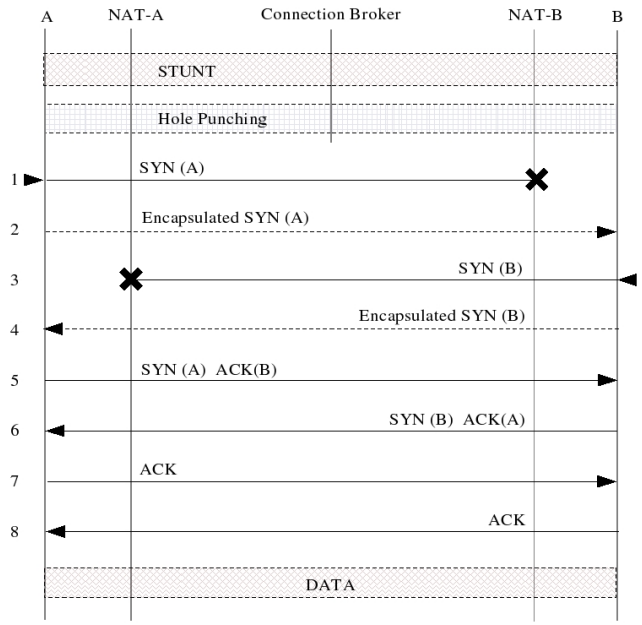

Figure 6.8:

Method's issues. As seen in the STUNT approach with spoofing, this method fails if the NAT changes the sequence number of the initial SYN packet and if the NAT doesn't allow an outbound SYNACK packet immediately after an outbound SYN, a not normally sequence of packets. This approach requires to open a RAW socket on endpoints' machine, which needs of superuser privileges.

Implementation's issues. ...

6.2.5 TCP Traversal: Low Time-To-Live Approaches

All the approaches previous presented can be slightly modified setting a low value of the IP time-to-live field, to avoid possible negative effects for incoming SYN packet on public NAT side. The SYN packet sent will be dropped in the middle of the network and an ICMP unreachable error will be returned. As explained in paragraph 6.2.1, an ICMP packet may cause problem on NAT, which may releases the mapping for the session, or not accept following packets. The figure **??**, for each methods previous exposed, shows the version with a low time-to-live value. These versions present the same problems seen before, except that the response to the first outbound SYN is always an inbound ICMP message. Normally, the solution with no low time-to-live value is more effective because the majority of the NATs drop the incoming SYN silently. We can thus take advantage from this approach when we want to traverse the addresse's NAT that reply with a RSTACK packet not accepted by sender's NAT.

Implementation's issues. ...

6.2.6 TCP Traversal: How to Choose the Rigth Method

...

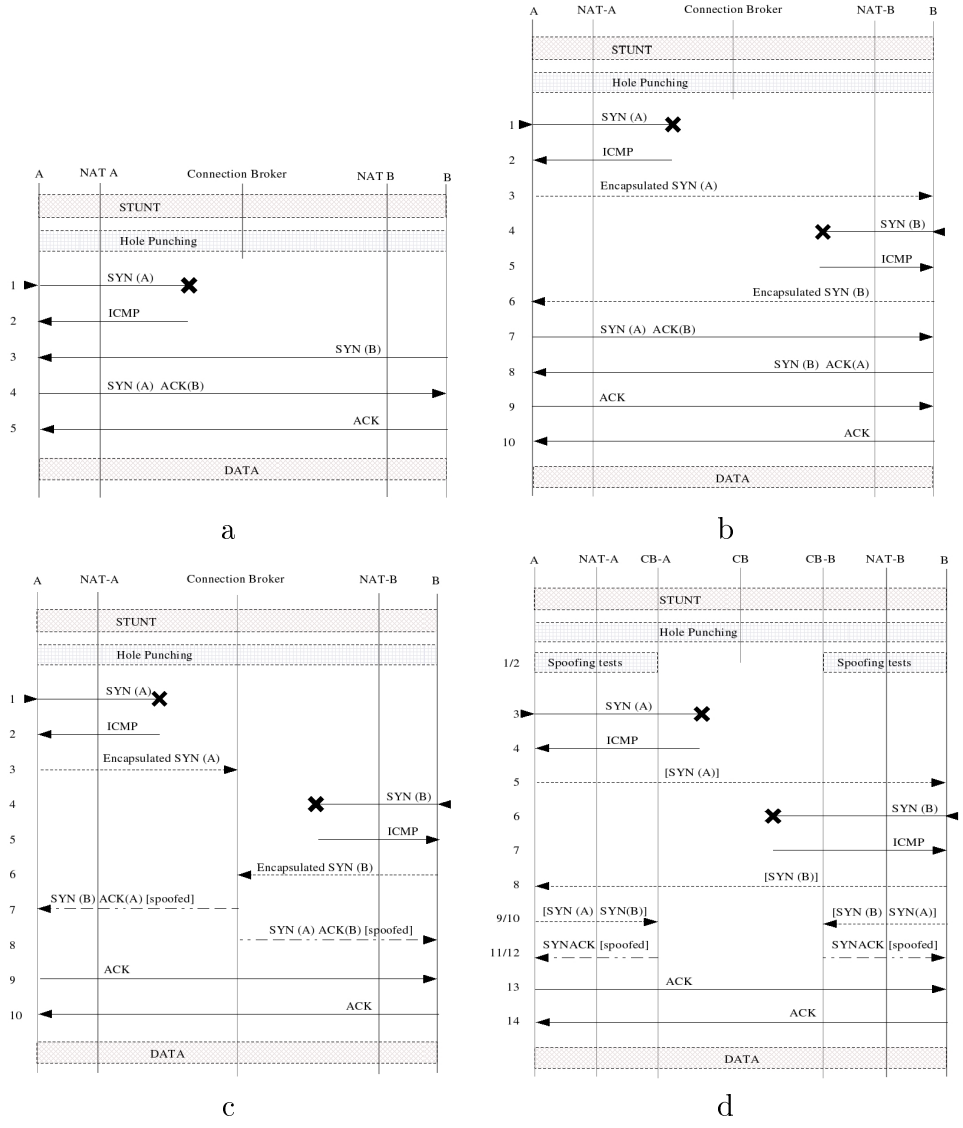

Table 6.2: Low Time-To-Live Approaches: (a) STUNT-2, (b) NatTrav, (c) STUNT-1, (d) STUNT-1 spoofing from well-known machine

Chapter 7

NTCP: NAT Traversal TCP Implementation

7.1 Uml Diagramas

7.2 Principal functions

Bibliography

[1] Solipsis
http://solipsis.netofpeers.net/wiki2/index.php 7

[2] Almost TCP over UDP (atou), Tom Dunigan, 2004-01-12
http://www.csm.ornl.gov/~dunigan/net100/atou.html

[3] Peer-to-Peer Communication Across Network Address Translators, Bryan Ford, Pyda Srisuresh, Dan Kegel, 2005-02-17
http://www.brynosaurus.com/pub/net/p2pnat/ 31

[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, Saikat Guha, Yutaka Takeda, Paul Francis, Cornell University and Panasonic Communication, USA
http://nutss.gforge.cis.cornell.edu/pub/fdna-nutss.pdf 31, 37

[6] Characterization and Mesurement of TCP Traversal Through NATs and Firewalls, Saikat Guha, Paul Francis, Department of Computer Science, Cornell University, USA
https://www.guha.cc/saikat/pub/draft-imc05-stunt.pdf 35

[7] TCP Connections for P2P Apps: A Software Approach to Solving the NAT Problem, Jeffrey L. Eppinger, Institute for Software Research International School fo Computer Science, Cornegie Mellon University, January 2005, CMU-ISRI-05-104
http://reports-archive.adm.cs.cmu.edu/anon/isri2005/CMU-ISRI-05-104.pdf 31, 36

[8] NATBLASTER: Establishing TCP Connections Between Hosts Behind NATs, Andrew Biggadike, Daniel Ferullo, Geoffrey Wilson, Adrian Perrig, Information Networking Institute, Carnegie Mellon University, Pittsburgh, USA
http://www.andrew.cmu.edu/user/ggw/natblaster.pdf 31, 39

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

[10] 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

[11] Dive Into Python
http://diveintopython.org

[12] Twisted Documentation
http://twistedmatrix.com/projects/core/documentation/howto/index.html

IETF - Request for Comments

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

[14] rfc 793 : TCP - Transmission Control Protocol

[15] rfc 768 : UDP - User Datagram Protocol

[16] rfc 1928: SOCKS Protocol Version 5

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

IETF - Internet Draft

[18] STUNT: Simple Traversal of UDP Through NATs and TCP too, Saikat Guha, Cornell University, 11 December 2004
https://www.guha.cc/saikat/pub/draft-imc05-stunt.pdf 13, 22 31, 36 11 12, 15, 16, 36, 37

[19] NAT Behavioral Requirements for Unicast UDP, C. Jennings, Cisco System, 11 April 2005 12, 15, 16

[20] Application Design Guidelines for Traversal of Network Address Trankators, B. Ford, P. Srisuresh, D. Kegel, February 2005 12

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

[22] Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for Offer/Answer Protocols, J. Rosenberg, Cisco Systems, 17 July 2005.

[23] SOLIPSIS Node Protocol

Tool

[24] Python
http://www.python.org/

[25] Twisted
http://twistedmatrix.com/products/twisted

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

[27] Subversion
http://subversion.tigris.org/

[28] RapidSVN
http://rapidsvn.tigris.org/
Mailing list

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

[30] p2p-hackers mailing list
http://zgp.org/mailman/listinfo/p2p-hackers 7