TECHNICAL REPORT SCE-12-04

DEPARTMENT OF SYSTEMS AND COMPUTER ENGINEERING

CARLETON UNIVERSITY

NAT TRAVERSAL IN PEER-TO-PEER ARCHITECTURE

Marc-André Poulin[1], Lucas Rioux Maldague[1],
Alexandre Daigle[1], François Gagnon[2]

1 Cégep de Sainte-Foy, Canada
marc-andre.poulin@sipoulin.ca, lucasr.mal@gmail.com, daigle_alexandre@hotmail.com

2 Carleton University, Canada
fgagnon@sce.carleton.ca

Abstract. Peer-to-peer networks are well known for file sharing between multiple computers. They establish virtual tunnels between computers to transfer data, but NATs makes it harder. A NAT, *Network Address Translation*, is a process which transforms private IP addresses, such as 192.168.2.1, into public addresses, such as 203.0.113.40. The idea is that multiple private addresses can hide behind a single public address and thus virtually enlarge the number of allocable public IP addresses. When an application in the local network establishes a connection to Internet, the packet passes through the NAT which adjusts the IP header and maps an external port to the computer which sent the request. When packets are received from the Internet by the NAT, they are forwarded to the internal host which is mapped to the port on which the packet was received, or dropped if no mapping exists. In this paper, we will introduce you to NAT and P2P, we will discuss the numerous ways NATs use to translate private IP addresses into public ones, we will discuss known techniques used to fix the problem and we will also present how popular peer-to-peer programs bypass NATs. This paper is written so anybody with a reasonable knowledge of networking would grasp the essentials. It is important to keep in mind that the traversal methods presented in this document work for UDP and TCP and require no manual configuration of the *Network Address Translator* itself.

1. INTRODUCTION TO NAT

Back in the middle of the '90s, NAT, which stands for *Network Address Translation*[1], became a popular way to decrease the IPv4 exhaustion rate [1]. They provide a way to create private subnets while using only a single public IP address. To achieve this goal, the gateway, which acts as a NAT, will change the IP address and port of all packets it forwards or receives. To do so, the NAT will bind a port, according to some rules defined by the manufacturer, to the computer within the subnet which sends a packet. Then, the NAT will change the port field of the UDP or TCP header to match the port he just opened and will also change the source IP of the IP header to match its own IP address. By doing this, the NAT will act like a proxy, so when the final destination will answer to the request (coming from the NAT), the NAT will look in its own dynamic table of port / private IP pair[2] and forward the packet to the original computer which made the request. Also, a NAT acts as a firewall, so if no outgoing connections have been made, no ports are opened except those specified in the NAT settings. Because different computers from the private network can't use the same external port in the NAT, since the port identifies the private computer to send data to, the NAT will bind a different port for each computer and for each different connection from the same computer.

2. INTRODUCTION TO P2P

Peer-to-peer (P2P) networks aim to create a decentralized and distributed communication mean between peers [2]. This allows a fail-proof architecture in which every peer has the same importance as others. There is ideally no peer with more importance than any other. Peer-to-peer is the complete opposite of a centralized server. Unfortunately, most peer-to-peer networks use a tracker system in order to maintain a list of shared files[3], therefore requiring the use of a central server.

In P2P, the peers (or nodes), are the main components. They directly share their data with the other peers which request it. Examples of peer-to-peer architectures can be found in the fields of VoIP technology, file sharing and grid computing.

[1] Further references of the acronym NAT will stand for *Network Address Translator*

[2] The table might contain additional information, depending on the type of NAT, which will be described in section 4

[3] Three real-world peer-to-peer networks are described in section 7, each with a different architecture. More details about the tracker method can be found there.

3. THE PROBLEM

In a traditional network, every host is identified by a single address and is therefore accessible by any other host. But, with NATs, this situation has changed: all the hosts behind one NAT are accessible with a single address [1]. It then makes connections attempts, from a client which is not behind a NAT to one which is, more difficult since the NAT will have no idea to which host the packets must be redirected [3].

One solution to this problem is to manually configure the NAT device to forward external connections on specified ports to local hosts. This technique is called *port forwarding*. It works, but it's hard to manage, especially if there are multiple hosts behind the NAT. Additionally, every application on every single host requires one port forwarding rule. Furthermore, if the computer's IP address changes, the rules must be modified accordingly.

Other than port forwarding, few other techniques exist to traverse NATs [4]. They all require a server or a peer not behind a NAT to initiate or relay a connection, since direct communication between two NATed hosts is nearly impossible as it would require guessing the NAT's dynamically bounded ports, which are sometimes randomly chosen.

The rise of peer-to-peer applications such as VoIP and file sharing these days make it a concerning subject, especially since almost 70 % of Internet users are behind a NAT [5]. This number is also expected to increase with the current shortage of IPv4 addresses. Moreover, the arrival of IPv6 is most probably not going to solve the problem, since NATs are also used as security devices because of their property to "hide" local hosts.

4. TYPES OF NAT

Network Address Translation is implemented in various ways, each of those affecting protocols differently. The four main types of NATs are *Full Cone*, *Address-Restricted Cone*, *Port-Restricted Cone* and *Symmetric*. A short description of each follows.

4.1 FULL CONE

Full Cone is the simplest type of NAT [5]. It is used in old devices and in some entry-level home routers. Due to its low capacities and simplicity, there is virtually no full cone NATs in corporate and large-scale networks. Here is how they work (numbers refer to Figure 4.1.1):

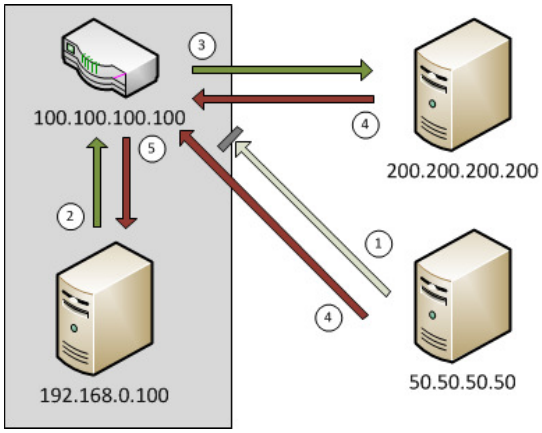

Figure 4.1.1

1. It's important to note that external connection attempts to a closed port are refused. For example, assume 50.50.50.50 tries to connect to the NAT (100.100.100.100) using the destination port 2000, the connection would then be refused.

2. An internal host, 192.168.0.100, sends a packet from port 2000 to 200.200.200.200:80. The packet is first transmitted to the NAT (192.168.0.1), which is the default gateway.

3. The NAT sends the packet to 200.200.200.200:80, keeping the source port 2000 since it’s currently unused. Otherwise, another free port would have been used. The NAT also saves the mapping between external port 2000 and host 192.168.0.100:2000. This binding is done the same way as a port-forwarding rule except that it's only temporary and automated: traffic coming on this external port is redirected to the internal host on the original port. Also, any further attempt by the host to contact any server using this port will be done using the previously set binding.

4. 200.200.200.200 responds from port 80 to the NAT (100.100.100.100:2000). Also, 50.50.50.50 resends a packet to 100.100.100.100:2000.

5. The NAT checks if a mapping exists for port 2000. There is one, so both packets are transmitted to the host specified in this mapping, 192.168.0.100:2000. No verification is done on the packet to check if the source IP address is the same as the one to which the first packet was sent, since this information was not stored. In other words, any packet, from any source address/port combination, coming to the NAT on this specific port will be forwarded to the internal host.

It's a very simple and easy to implement method since minor processing is required. This is why it is widely used in entry-level devices. But, there are many concerns with full-cone devices. First, since the maximum number of bindings is limited to the number of ports on the device (65535 for TCP, the same for UDP, minus the port-forwarding rules), the number of users on the local network is limited. There cannot be 65535 users, since every application on every computer that accesses the Internet uses at least one binding. This is very limitative, especially for medium to large networks. Another concern is about the security of those devices. Since no verification is done on the source of a packet when it is forwarded to an internal host, one could send fake replies to a request very easily.

4.2 Address-Restricted Cone

In this type of NAT, once a port is opened by a computer from the internal network, the port is considered open, but only to the external host the internal computer sent data to. If this condition is not met, for instance if any outside computer tries to communicate with the internal computer without having previously been contacted by it, the NAT will reject the connection [5].

For example, if we look at figure 4.2.1, the computer behind the NAT has the private IP address 192.168.0.100 and the computer on Internet has the public address 200.200.200.200. In order to start a connection between the internal and the external computer through the NAT's firewall, the following steps must occur:

1. 192.168.0.100:2020 sends a packet to 200.200.200.200:80.
2. The NAT creates a mapping between 192.168.0.100:2020 and external port 4040 for 200.200.200.200 and sends the packet, after modifying its IP header, to 200.200.200.200:80.
3. 200.200.200.200 receives the packet on port 80 and sends a reply to 100.100.100.100:4040.
4. Once the NAT receives the packet, the first thing to check is if a dynamic port forwarding rule exists for the destination port 4040. If there is, then the source IP of the packet (200.200.200.200) is compared against the IP of the port forwarding rule (200.200.200.200). If they match, the NAT forwards the packet to the internal host and port (192.168.0.100:2020) which is associated to the external port on which the packet was received. Otherwise, it simply drops it. No further validation is done on the origin of the packet (such as the *source* port number, which is 80).

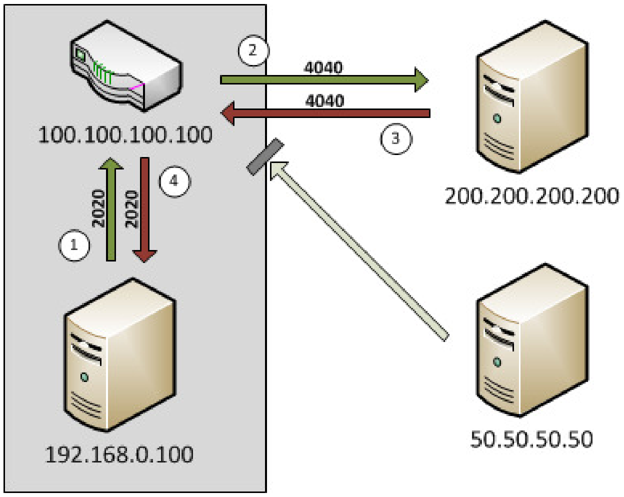

Figure 4.2.1

This behaviour suggests that the NAT stores a dictionary of source/destination mappings which contains the source's IP address and port, the external port which was bound and the destination's IP address. This also means the destination doesn't have to receive the packet to create the *hole* in the NAT since the packet could get lost between the moment it goes through the NAT and the moment it's received by the destination.

From a security point of view, this type of NAT is much better that *Full Cone* since the NAT acts as a firewall. But, this design still has some flaws because once a binding is made between an internal and an external host, any packet coming to the specific bounded port, with a source address which is in the list of allowed addresses for this port, is allowed to pass through and will be forwarded to the internal host.

4.3 Port-Restricted Cone

This type of NAT is very similar to *Address-Restricted Cone*. It works the same way, except that the NAT stores the destination port along with the destination IP address [5]. So, when the NAT receives a packet on his external interface, it will not only check if a binding exists for the packet's source IP address, it will also check the packet's source port to be sure that it is not a connection from another application on the external host.

Taking the example presented in *Address-restricted Cone*, the mapping would have included the destination port of the packet (80). So, if 200.200.200.200 sends a packet from port 3030 to 100.100.100.100:4040, the NAT will check the port forwarding rule for port 4040, which says that packets from 200.200.200.200:80 on port 4040 are to be redirected to 192.168.0.100:2020. Here, the source port (3030) does not match to the one specified in the rule (80) so the packet is dropped. If the source port would have been 80, the packet would have been transmitted to the local host.

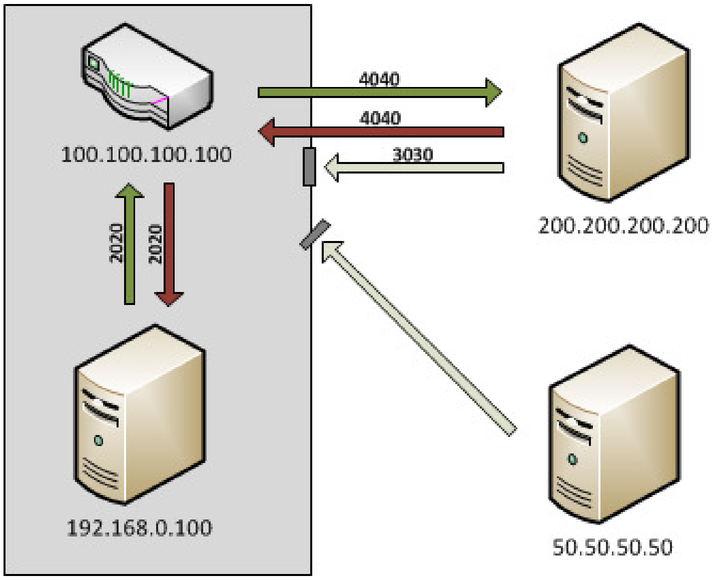

Figure 4.3.1

4.4 SYMMETRIC NATS

Symmetric NATs, as STUN [6] describes it, are NATs mostly owned by enterprises with a better security filter. Based on 2009 data collected by Delft University of Technology in the Netherlands [7], on 3500 peers located in 8 different countries, about 17% of users were using a symmetric NAT, the second most common type after *Port restricted cone*.

Symmetric NATs use a unique mapping for every outgoing connection made by an internal host's IP address and port to an external host's IP and port. Hence, only previously contacted hosts respecting the IP and port mapping previously attributed can send back packets to the internal host. Any other outgoing connection for the same internal host, with the same source port, to a different external host or even to a different destination port on the same external host would result in a *different* mapping. As a result, most NAT traversal attempts that do not involve modifying the NAT configuration will fail. However, according to [8], a custom technique discovered in 2008 by researchers at Waseda University in Japan, based on port prediction and short TTL values, has proven to be successful about 99% of the time in symmetric NAT traversal.

NATs of this type also have no standard rule on source port assignation when forwarding local packets to external hosts. Some try to preserve the original source port while others change it every time, using various methods. They also have no specified rules about which range of port to use for source port assignment. In fact, the port assignation method or range ruling depends on the device manufacturer.

Represented by Figure 4.4.1, the symmetric NAT works as follow:

1. 192.168.0.100:2020 wants to send a packet to 200.200.200.200:80. Since this address is not located on its network, it first sends it to its gateway, 100.100.100.100:2020. Another packet is also sent to 50.50.50.50:80 from the same host (192.168.0.100:2020).

2. The NAT chooses to use external port 4040 for this connection to 200.200.200.200. This means every packet received from 200.200.200.200:80 on port 4040 will be forwarded to 192.168.0.100:2020. It then sends the packet with a modified IP header to 200.200.200.200:80. For the packet sent to 50.50.50.50:80, external port 6060 is chosen.

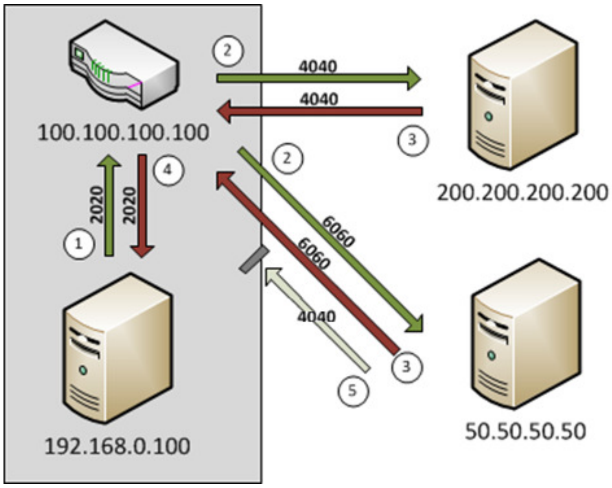

Figure 4.4.1

3. 200.200.200.200:80 sends a reply to 100.100.100.100:4040. Since a mapping exists for packets coming from this address/port on external port 4040 (created in step 2), the NAT forwards the reply to 192.168.0.100:2020. Also, 50.50.50.50:80 replies to 100.100.100.100:6060 and since a mapping for packets coming from 50.50.50.50:80 on port 6060 exists, the packet is forwarded to 192.168.0.100:2020.

4. Both packets are transmitted to 192.168.0.100:2020.

5. 50.50.50.50:80 sends a packet to 100.100.100.100:4040. Even tough a mapping exists for this port, the source address of the packet (50.50.50.50) does not match the source address of the mapping (200.200.200.200). Therefore, the packet is dropped.

5. HAIRPINNING

Hairpinning is a technology that allows peers behind the same NAT to communicate using their public IP address [6].

For instance, take two hosts, X1 and X2. They are on the same local network, behind a common NAT, and they want to initiate a connection with each other. Since they are not aware that they are on the same network (they do not know the private or public address of each other, only a profile name or any similar identifier), they will first contact a rendezvous server to begin a hole-punching process (see the *UDP Hole Punching* section for more details on this). During the process, each host will obtain from the server the public end point of each other, consisting of an IP address and a UDP port. Let's say X1's private end point is 192.168.1.101:1001 and X2's is 192.168.1.102:1002. The NAT's public address is 200.200.200.200, and there is a mapping between 192.168.1.101:1001/public port 1001 and another between 192.168.1.102:1002/public port 1002 (these mappings were created when X1 and X2 contacted the rendezvous server). So, their *public* end points will be respectively 200.200.200.200:1001 and 200.200.200.200:2002. Referring to Figure 5.1, here is how hairpinning works:

1. To initiate a connection, X1 will send a packet to 200.200.200.200:1002;
2. The packet will be send to the NAT (default gateway), since it's not an address of the local network;
3. If the NAT supports hairpinning, it will forward the packet to X2 (192.168.1.102:1002), since there is a dynamic port forwarding rule between its *public* interface's port 1002 to X2 (192.168.1.102:1002);
4. A connection is then successfully established between X1 and X2.

But, NATs which do not support hairpinning would have caused the connection attempt to fail. When a NAT which does not support hairpinning would have received the packet from X1 to X2 (200.200.200.200:1002), it would have discarded it since, even though it would have known to whom the packet is destined to, no port forwarding rule for port 1002 exists on its *private* interface. The port-forwarding rules on the public interface are not checked for packets coming on the private interface.

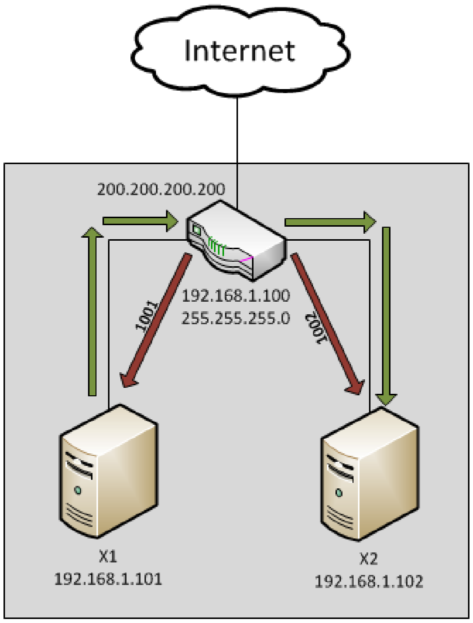

Figure 5.1

6. NAT Traversal Techniques

NAT traversal is a serious concern for decentralized architectures because peers cannot communicate directly to nodes behind NATs. On the security stand point, this is good news, but on the ease-of-use one, it is not. For now, there is no known technique that does not involve the participation of the peer behind the NAT, therefore making NATs more secure than required. Here is a description of the most popular techniques used to traverse NATs, as defined in RFC 5128 [5].

6.1 Relaying

The easiest, always working but least efficient solution for peer-to-peer communication between NATs is relaying [9]. It requires a server that can handle high loads of data since all the traffic between peers will transit through it. For example, if one peer (peer A) wants to communicate with another peer (peer B), they must both open and maintain a connection with a relaying server that is not behind a NAT (step 1 in Figure 6.1.1). Once this is done, peer A sends data to the relaying server (step 2) which will transmit it to B (step 3). The reverse operation is done when B sends data to A. In fact, all data transmitted by the two peers will pass through the relaying server.

Relaying is supported by NATs of all type (Full Cone, Address-Restricted Cone, Port-Restricted Cone and Symmetric), in any network architecture (no NAT, single-level NAT, multiple-level NAT, etc.), since it's basically a client-server communication. Only, relaying has a major drawback: all the data is transmitted through a server so it uses a lot of bandwidth and has little privacy, since the owner of the server can see all the transmitted data and even filter it. Furthermore, no connection can be established between peers if the server is down.

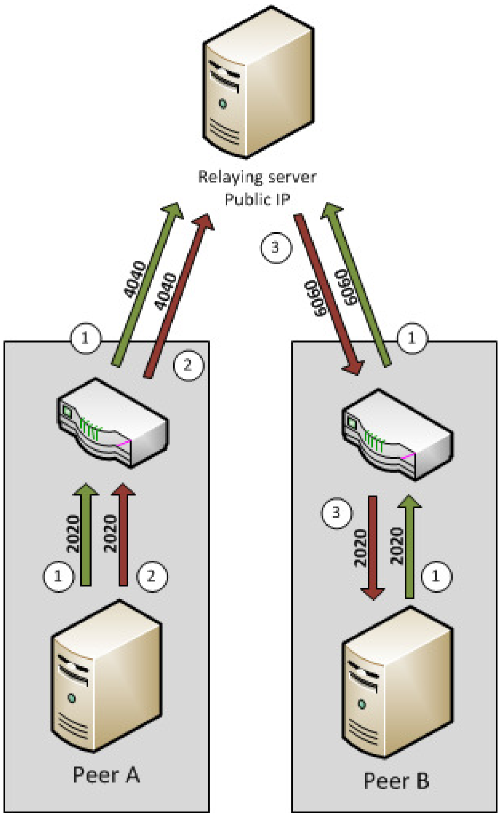

Figure 6.1.1

6.2 Connection Reversal

This technique is a simplification of the UDP/TCP hole punching in that it works only when a peer is not behind a NAT while the other is [10]. Its principle is to let the peer behind the NAT (peer A) send a packet to the peer which is not behind a NAT (peer B). Doing this will open a port in peer A's NAT and peer B will be able to get through the hole that has just been punched. But, this raises a deeper problem: peer A needs to know peer B's IP address and TCP/UDP port to open the connection. To resolve this problem, one could simply use an external server on which the peers would share their addresses so that peer A could open the hole of his NAT. Another solution would be to create a "*friend list*" [4] in every peer; those would connect to their friends and then everyone would share their own friend list with all of their friends so that the list is constantly maintained up-to-date and shared. This technique works on all types of NATs because the process simulates a client-server connection (i.e. the client, behind the NAT, makes all the connections.). It is notably used by Skype for users not behind a symmetric NAT [11].

Here is how connection reversal works:

1. Peer A (200.200.200.200), sends a packet from its port 2020 to the rendezvous server 50.50.50.50:2020. 192.168.0.100:2020, peer B, sends the same packet to its gateway 100.100.100.100:2020 which routes it from 100.100.100:6060 to 50.50.50.50:2020.

2. The rendezvous server now knows peer B is behind a NAT because the source port and IP are not the same as said in the packet (this data was included in the packet sent in step 1).

3. The rendezvous server sends a packet to peer B, 100.100.100.100:6060 which routes it to 192.168.0.100:2020 according to its mapping, with inside the external IP and external port to reach peer A.

4. Peer B, 100.100.100.100:2020, sends a packet to peer A, 200.200.200.200:2020, punching a hole in the NAT for peer A to answer. The connection is then possible between peers A and B, and the rendezvous server is not required anymore.

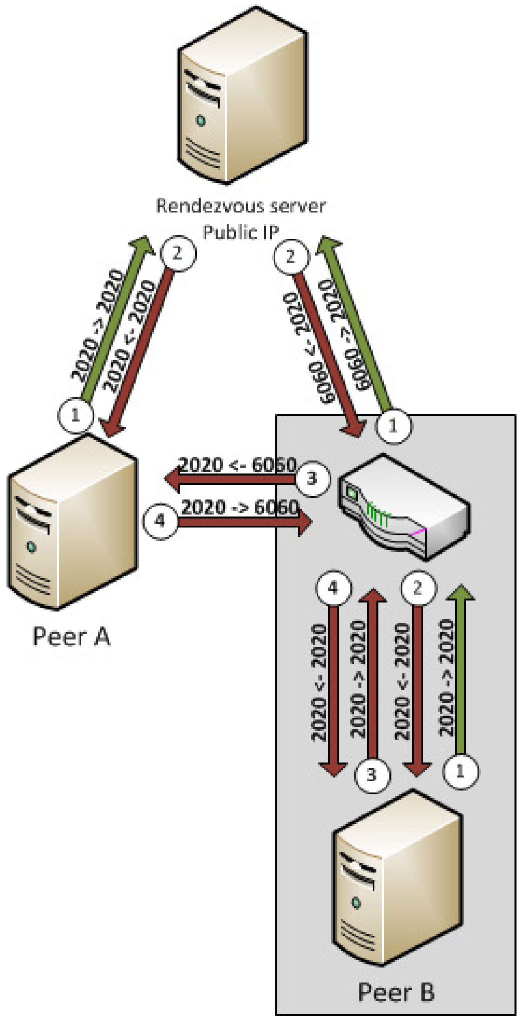

Figure 6.2.1

[4] This implementation is used by Tribler, a BitTorrent client developed at the Delft University of Technology and the Vrije Universiteit Amsterdam. Their approach is detailed further in the *Existing Technologies* section.

6.3 UDP Hole Punching

This technique makes it possible for two hosts which are both behind NATs to initiate a connection to each other, with the help of a rendezvous server. This server is only required for the exchange of peer's addresses. In fact, any other peer who is not behind a NAT could play this role.

As its name says it, the technique uses the UDP protocol to communicate data. A drawback of this technique is that it does not work with symmetric NATs, because it takes for granted the NAT will keep the same public port for any host contacted from a same local source port and IP [12].

To explain how UDP hole punching works, let's take the example of peers A and B, which are both behind a different NAT, and public rendezvous server S which is not behind a NAT (Figure 6.1.1). A's private address is 192.168.0.100, and its NAT's public address is 100.100.100.100. B's private address is 10.10.10.200, and its NAT's public address is 100.100.100.200. Server S's address is 200.200.200.200.

1. The first step in establishing a connection between two clients, A and B, is for each of them to initiate a connection with the rendezvous server, S [13]. So, A sends a packet from port 2020 to 200.200.200.200:2020. The packet goes to the NAT, the default gateway. The same thing happens for B.

2. A's NAT creates a mapping between external port 4040 and 192.168.0.100:2020, and forwards the packet to 200.200.200.200:2020. B's NAT creates a mapping between external port 6060 and 10.10.10.200:2020, and forwards the packet to 200.200.200.200:2020.

3. S receives both packets, and sends a packet containing the public end-point of B to A (from 200.200.200.200:2020 to 100.100.100.100:4040) and another containing the public end-point of A to B (from 200.200.200.200:2020 to 100.100.100.200:6060).

4. A's NAT receives the packet from 200.200.200.200:2020 on port 4040. It transmits it to A (192.168.0.100:2020) because of the existing binding created at step 2. The same thing happens on B's side: its NAT receives the packet from 200.200.200.200:2020 on port 6060. It transmits it to B 10.10.10.200:2020) because of the existing binding created at step 2.

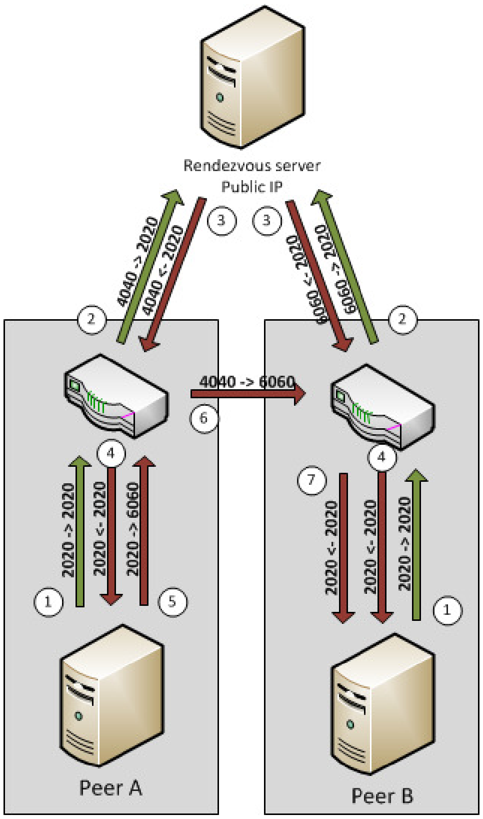

Figure 6.1.1

5. Having B's public end point, A sends a packet to it, from port 2020 to 100.100.100.200:6060. B does the same: it sends a packet from port 2020 to 100.100.100.100:4040. They both send their packets to their respective NATs since the packets are destined to addresses outside the local network.

6. A's NAT uses the same mapping created in 2 since the sender's IP and source port is already bound to external port 4040. If the NAT is address or port-restricted, B's public end point is added to the list of authorized addresses and ports for this mapping. The packet is then transmitted from port 4040 to 100.100.100.200:6060. The same thing happens on B's side, and its packet is transmitted to 100.100.100.100:4040 from port 6060. Depending on the timing and the type of A and B's NATs, the first

packet transmitted may be blocked by either NAT because it arrives before the mapping is made. Sending it continuously until an acknowledgement is received from the destination solves this problem.

7. B receives the packet from A (100.100.100.100:4040) on port 6060. Because of the mapping created at step 6, it is transmitted to B (10.10.10.200:2020). The same thing happens on A’s side. A connection is then successfully established between A and B.

UDP hole punching works flawlessly with hosts behind different NATs [14]. If the clients are behind the same NAT, it will only work if their NAT supports hairpinning. To overcome that, it's useful to send connection attempts from A to B not only to their public end-points, but also to their private end-points.

There is a potential problem here. Let’s take the example of two peers located in different private networks. A sends a packet to B in its private network and inadvertently reaches C, another node in A’s local network which uses the same local end-point (IP and destination port) than B. The packet was then transmitted to the wrong peer. A solution to that is to include a unique identifier, other than the IP/port mapping, in the communication protocol. This way, packets received by any host can be checked and dropped if they happen to have reached the wrong destination. This way, communication will always be possible.

UDP hole punching also works very well on hosts behind multiple levels of NATs, since the punching is done a single time for all levels of NATs. For instance, let’s see what happened if peer A is behind a domestic NAT which itself is behind an ISP-wide NAT, while peer B is simply behind a single domestic NAT. First, both peers contact the rendezvous server. For B, this is exactly the same as described before. For A, is not very different, just longer. First, A sends a packet to its NAT, which then transmits it to the ISP NAT, creating a mapping between A and the ISP NAT. Second, the ISP NAT receives the packet and transmits it to the rendezvous server, creating a mapping between A’s NAT and the public server. Thus, the public end-point of A is the one of its ISP NAT, and a hole is punched in *both* NATs. The rest of the procedure is exactly the same, B contacting A on the ISP NAT, which will transmit the packet to the domestic NAT which, seeing packets from it (the mapping for A/ISP NAT exists), will forward them to A.

A special situation arises if the hosts are each behind a different domestic NAT but are behind the same ISP NAT. For UDP hole punching to work, the ISP NAT must support hairpinning, since it must redirect packets from A’s NAT to B’s NAT and vice-versa. Otherwise, packets won't be routed correctly and communication will be impossible.

6.4 TCP Hole Punching

As with UDP hole punching, TCP hole punching requires a rendezvous server for the exchange of peer addresses, timing and flooding the network [15]. Normally, when two hosts establish a TCP session, one sends the other a *SYN* packet and the other responds with a *SYN/ACK* packet. However, it is also possible to establish a TCP session with two hosts sending each other a *SYN* packet at the same time and then an *ACK* packet. This technique is known as “Simultaneous TCP Open”.

With TCP hole punching, the idea is to continuously send *SYN* packets to the other NATed host until it does the same on its end. The majority of NATs (more than 50%) do not respond to unsolicited *SYN* requests, which is the desired behaviour. As a result, the simultaneous TCP open technique does work on most NATs. Once both ends sent their *ACK* packet, the hole punching is completed and the TCP session is now functional.

This technique may not always work. If the other NAT device responds with a *RST/ACK* to the first incoming *SYN* packet, this would make the local NAT device to automatically close the session (it will ignore following requests from this IP). Another possible scenario of failure would be a NAT device which forbids simultaneous TCP open sessions.

6.5 UPnP

Universal Plug-n-Play (UPnP) is a protocol developed by Microsoft to bring the concept of *plug-n- play* to desktop computers [16]. The idea is to plug to the network any kind of device, whether it is a printer, scanner, desktop computer, etc. and make it work automatically. The entire configuration is hidden to the user and the devices configure themselves based on the existing network topology.

The *Internet Gateway Device Protocol* (IGDP) implemented in UPnP makes NAT traversal easier [17]. The goal of this protocol is to automate port forwarding on gateways.

IGDP has four main features:

- Fetch the public (external) IP address;
- Enumerate existing port mappings;
- Add or remove port mappings;
- Assign lease times to mappings.

Even if IGDP works with any kind of NAT, it has two major downsides. First, its default security measures are virtually inexistent. Any malicious program could set port-forwarding rules using IGDP by simply sending “AddPortMapping” packets with SOAP to the NAT and thus opening the host to the Internet. The second downside is the limited number of devices implementing it, therefore limiting its usage. Users could also disable it on their devices.

6.6 Summary

Every NAT traversal technique can be used under certain conditions. Of course, the ultimate technique is UPnP but it is not supported by all NATs and it's not all users or system administrators who want to use a protocol as intrusive. Therefore, other techniques can be used in most of the cases, even though some tend to be rather complex to implement.

Those techniques are summarized in Table 6.6.1, which is inspired from [9]. The techniques presented in bold are the suggested ones. The leftmost columns represent the initiating peer and the topmost the contacted peer.

	None	Full Cone	Address-Restricted Cone	Port-Restricted Cone	Symmetric
None	**Direct** Relaying	**Direct** Relaying Con. Reversal Hole Punching UPnP[1]	Relaying **Con. Reversal** Hole Punching UPnP[1]	Relaying **Con. Reversal** Hole Punching UPnP[1]	Relaying **Con. Reversal** Hole Punching UPnP[1]
Full cone	**Direct** Relaying Con. Reversal Hole Punching UPnP[1]	**Direct** Relaying Con. Reversal Hole Punching UPnP[1]	Relaying **Con. Reversal** Hole Punching UPnP[1]	Relaying **Con. Reversal** Hole Punching UPnP[1]	Relaying **Con. Reversal** Hole Punching UPnP[1]
Address-restricted cone	**Direct** Relaying Con. Reversal Hole Punching UPnP[1]	**Direct** Relaying Con. Reversal Hole Punching UPnP[1]	Relaying **Hole Punching** **UPnP[1]**	Relaying **Hole Punching** **UPnP[1]**	Relaying **Hole Punching** **UPnP[1]**
Port-restricted cone	**Direct** Relaying Con. Reversal Hole Punching UPnP[1]	**Direct** Relaying Con. Reversal Hole Punching UPnP[1]	Relaying **Hole Punching** **UPnP[1]**	Relaying **Hole Punching** **UPnP[1]**	**Relaying** **UPnP[1]**
Symmetric	**Direct** Relaying Con. Reversal Hole Punching UPnP[1]	**Direct** Relaying Con. Reversal Hole Punching UPnP[1]	Relaying **Hole Punching** **UPnP[1]**	**Relaying** **UPnP[1]**	**Relaying** **UPnP[1]**

Table 6.6.1

1: UPnP only works if the router who act as a NAT support this technology

7. EXISTING TECHNOLOGIES

As said earlier, P2P architectures need a way to communicate from peer to peer. Here are some well-known P2P-based programs and the techniques they use.

7.1 TRIBLER

Tribler is a peer-to-peer client which does not require a central server to support the communication [18]. This open-source project is developed at the Delft University of Technology and Vrije Universiteit. The European Union also uses this project as a base for video-on-demand television in their P2P-Next project.

First and foremost, Tribler is not a traditional peer-to-peer client. It uses socials phenomena to connect peers and find content. It uses a small relational database which contains the discovered content. It also handles friendship between peers, so that friends can easily share data. Technically, this increases the download and upload speeds between these privileged peers. Tribler also figures out the tastes of the user and can regroup users with the same tastes in groups (called *taste buddies*). This whole social approach allows searching content directly from other peers, with a preference to friends and peers with the same tastes. To find the first peers, the software connects with a pre-known "super-peer" to get the information [19].

In order to maintain the friend list, the peer cache, metadata cache and preference cache are shared to known peers. All of this data, called *megacaches* by Tribler's team, is compressed and sent over the network to allow a friend-to-friend connection.

Tribler's friend system replaces the central tracker. Therefore, a rendezvous server does not have to be used if peers which want to share data have a common friend which is not behind a NAT. However, Tribler mostly uses UPnP to get through NATs because of its simplicity of use. A rendezvous server is then not necessary at all. Also, even though Tribler does not use a lot of hole punching techniques, their whole approach would still allow true decentralized architecture without UPnP.

7.2 SKYPE

The Skype VoIP software uses a protocol based on a peer-to-peer architecture to connect users. Even though their protocol is closed-source, reverse-engineering attempts have been successful in determining how it works [20].

The protocol requires a minimal centralized infrastructure, since it uses a distributed network of peers to relay communications. There are three kinds of entities in the network: "super-nodes", ordinary nodes and the login server. Each client (node) maintains a list (host cache) of reachable super-nodes [11].

Any client which is not behind a NAT, has a good bandwidth and has enough processing power can become a super-node. The super-nodes help nodes behind NATs to communicate with each-other, by relaying data or acting as rendezvous servers.

The advantage of using peers to relay communications is that it does not require many Skype-owned resources. Also, the network has a high reliability, since every peer keeps a list of many super-nodes. If one of them goes down, there are still many others which can be used. Additionally, the host-cache is constantly updated by the peers themselves, so no interaction of Skype servers is required.

7.3 GNUTELLA

Gnutella is well known peer to peer network made by GNU [21]. It is completely decentralized and was the first of its kind. It celebrated a decade of existence on March 14th 2010. It has an estimated market share of 40%, with over a million peer-to-peer users. A lot of known P2P programs are built on it, such as *LimeWire*, *Morpheus* or *Shareasa*. Each of these programs uses at least one of the techniques described earlier to traverse NATs, the most popular being UDP and TCP hole punching.

8. CONCLUSION

The problem caused by NATs is that they make hosts in their internal network unreachable from hosts outside of their network [1]. There are many techniques which can be used to bypass this problem, many of which have been presented in this analysis. But, with the exception of UPnP, which is not very secure and is disabled on many devices [16], they all need a peer which is not behind a NAT either to share public end-points (rendezvous server), or to relay traffic.

Relaying is not very efficient because it requires a public server to relay all the traffic between two hosts. Connection reversal only works if one of the peers is not behind a NAT and it also requires a rendezvous server. For UDP and TCP hole-punching, a public rendezvous server is also involved at the beginning of a connection. The fact that they all need a public server causes many problems: no connection is possible if the server is down, or if the peers are not allowed to use it. It also presents privacy concerns, especially for relaying since all the traffic can be logged and inspected. For rendezvous servers, its owner knows exactly who contacted who, particularly if the peers must identify themselves to the server with a login before attempting to connect to another peer. But, those servers are particularly useful to keep a peer list. Since a lot of peer's public IP addresses change frequently, it would be very hard for peers to try to contact other peers knowing only the other's IP address, even if a public server is involved. Also, a peer list can be useful in situations where peers have data to share. In this case, the server would also have a list of files shared by peers. This list could then be transmitted to the peers and they would contact the peers which have the files they want.

But, the peer list could also be decentralized to all the peers. In this situation, every peer maintains its own list and it's updated by contacting other peers. Since all the peers contact themselves, the list can be fairly up-to-date. This technique is used in Tribler for example [19]. With this method, there is no need for a central server. A method that does not require one is used, such as UPnP. Hole-punching and connection reversal techniques can still be used; they will simply use a peer which is not behind a NAT to act as the rendezvous server. An example of this is Skype [11].

9. REFERENCES

[1] Wikipedia contributors *Network address translation* http://en.wikipedia.org/wiki/Network_address_translation [accessed February 8th 2011].

[2] Wikipedia contributors *Peer-to-peer* http://en.wikipedia.org/wiki/Peer-to-peer [accessed February 8th 2011].

[3] Venkatachalam G. (2006) Developing P2P Protocols across NAT, Linux Journal [Internet] June 30th 2006. Available at http://www.linuxjournal.com/article/9004 [accessed January 25th 2011].

[4] Wikipedia contributors *NAT traversal* http://en.wikipedia.org/wiki/NAT_traversal [accessed February 1st 2011].

[5] Srisuresh, P. et al. (2008) State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs) http://www.ietf.org/rfc/rfc5128.txt [accessed February 1st 2011].

[6] Rosenberg, J. et al. (2003) STUN – Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs) http://www.ietf.org/rfc/rfc3489.txt [accessed February 1st 2011].

[7] L. D'Acunto and J.A. Pouwelse and H.J. Sips. A Measurement of NAT & Firewall Characteristics in Peer-to-Peer Systems. In Proceedings of the ASCI Conference, Zeewolde, the Netherlands, June, 2009.

[8] Yuan Wei et al. (2008) A New Method for Symmetric NAT Traversal in UDP and TCP http://www.goto.info.waseda.ac.jp/~wei/file/wei-apan-v10.pdf [accessed January 25th 2011]

[9] Wacker, A. et al. (2008) A NAT Traversal Mechanism for Peer-To-Peer Networks http://www.pap.vs.uni-due.de/files/wacker-nat-traversal.pdf [accessed January 25th 2011].

[10] Wikipedia contributors *Connection reversal* http://en.wikipedia.org/wiki/Reverse_connection [accessed February 8th 2011]

[11] Wikipedia contributors *Skype protocol* http://en.wikipedia.org/wiki/Skype_protocol [accessed February 8th 2011].

[12] Wikipedia contributors *UDP hole punching* http://en.wikipedia.org/wiki/UDP_hole_punching [accessed February 1st 2011].

[13] Eichhorn, D. (2006) A Peer-to-Peer Network Framework with Network Address Translation Traversal. Ph.D, University of Zurich.

[14] Müller, A. et al. (2010) Autonomous NAT Traversal http://grothoff.org/christian/pwnat.pdf [accessed January 25th 2011].

[15] Wikipedia contributors *TCP hole punching* http://en.wikipedia.org/wiki/TCP_hole_punching [accessed February 1st 2011].

[16] Hemel, A. (2006), *Universal Plug and Play: Dead simple or simply deadly?* http://www.sane.nl/sane2006/program/final-papers/R6.pdf [accessed February 8th 2011].

[17] Wikipedia contributors *Internet Gateway Device Protocol* http://en.wikipedia.org/wiki/Internet_Gateway_Device_Protocol [accessed February 8th 2011].

[18] Pouwelse, J. A. et al., (2007) *Tribler : a social-based peer-to-peer system* http://www.pds.ewi.tudelft.nl/~pouwelse/CPE-Tribler-final.pdf [accessed February 8th 2011].

[19] Ernesto (2010) *Truly Decentralized BitTorrent Downloading Has Finally Arrived,* TorrentFreak [Internet] December 8th 2010 Available at http://torrentfreak.com/truly-decentralized-bittorrent-downloading-has-finally-arrived-101208/ [Accessed February 8th 2011].

[20] Baset, A. et al. (2004) An analysis of the Skype Peer-to-Peer Internet Telephony Protocol http://www.cs.columbia.edu/~library/TR-repository/reports/reports-2004/cucs-039-04.pdf [accessed March 22nd 2011]

[21] Wikipedia contributors *Gnutella* http://en.wikipedia.org/wiki/Gnutella [accessed February 8th 2011].