NAT Boxes and Firewalls: A Look at VoD and Skype

Aurosish Mishra (06CS3027) Harshvardhan Agarwal(06CS3022)
Raymanpreet Singh Matharu (06CS3023)

1 Introduction

In the past decade, there has been a significant rise in the use and development of peer-to-peer (P2P) technology. Its scalability and robustness made this technology popular in a large spectrum of application domains, such as distributed file sharing, Internet telephony, live streaming and video on demand(VoD). A major obstacle encountered in P2P communication is the presence of Network Address Translators(NATs) and firewalls. They can cause some peers not to be reachable at any globally routable IP address. Such devices are, in fact, becoming a default setting among home users, who often do not have the knowledge on how to configure them to allow P2P traffic. So, it is of paramount importance to develop techniques that will enable P2P systems to transcend the NAT boundaries.

As the internet expands, and the amount of information and resources increases, it is becoming a requirement for even the smallest of businesses and homes to connect to the Internet. *Network Address Translation (NAT)* is a method of connecting multiple computers to the Internet (or any other IP network) using one IP address. This allows home users and small businesses to connect their network to the Internet cheaply and efficiently. The impetus towards increasing use of NAT comes from a number of factors:

- A world shortage of IP addresses
- Security needs
- Ease and flexibility of network administration

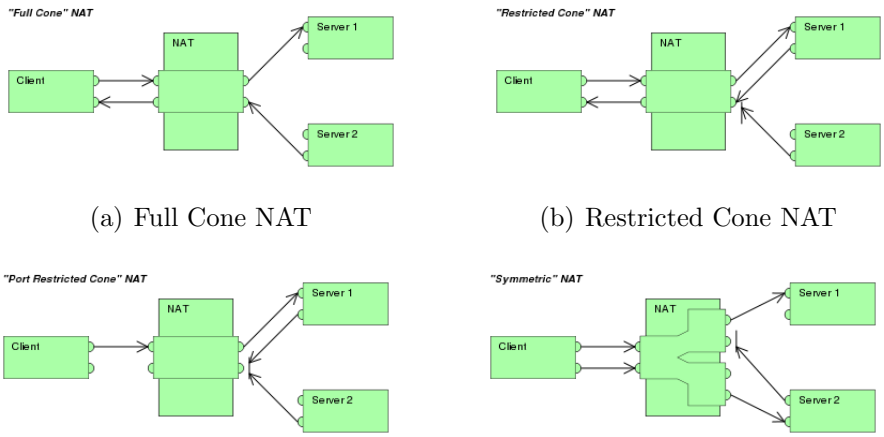
2 NAT

A Network Address Translator is a device that allows multiple machines on a private network to communicate with the Internet using a single globally unique IP address. This is accomplished by modifying the network information (IP address and port) in the packets that transit through it. The way a given internal pair <IP address,port> is translated to a given external pair <IP address,port> is called *address mapping policy*. The rapid reduction of the IPv4 address space has stimulated the widespread deployment of Network Address Translators (NATs) on the Internet. Furthermore, companies' network security policies include the utilization of NATs and/or firewalls in order to hide the network topology and control both inbound and outbound traffic. As a result, recent studies state that more than 73% of Internet end-hosts are located behind NATs or firewalls.Though initially created as a temporary solution for alleviating the IPv4 address shortage, the NAT technology will probably continue to be a long-term part of the Internet.

A firewall is a device (or set of devices) which inspects network traffic passing through it, and filters (i.e. denies or permits) the transmission based on a set of rules. NATs and firewalls break the end-to-end connectivity principle of the Internet, since they prevent the hosts they are protecting from receiving connections initiated from external hosts. This architecture is very suitable for client/server communication, in the typical case when the client is inside the private network and the server is outside and has a public address, but it is not suitable for P2P communication. This is because NATs do not allow inbound connections unless they are manually configured to do so.

To make things even more difficult, there is no standardized NAT/firewall behaviour. However, a classification of different kinds of NATs can be made according to a combination of address mapping policy and filtering behaviour for UDP traffic:

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



(a) Full Cone NAT

(b) Restricted Cone NAT

(c) Port Restricted Cone NAT

(d) Symmetric NAT

Figure 1: Types of NAT

- *Restricted Cone NAT.* 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 A) can send a packet to the internal host only if the internal host had previously sent a packet to IP address A.

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

- *Symmetric NAT.* 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 NAT Traversal

NAT traversal is a general term for techniques that establish and maintain TCP/IP network and/or UDP connections traversing network address translation (NAT) gateways. NAT traversal techniques are typically required for client-to-client networking applications, especially p2p and Voice-over-IP (VoIP) deployments. Many techniques exist, but no single method works in every situation since NAT behavior is not standardized. Many techniques require assistance from a computer server at a publicly-routable IP address. Some methods use the server only when establishing the connection (such as STUN), while others are based on relaying all data through it (such as TURN), which adds bandwidth costs and increases latency, detrimental to real-time voice and video communications.

3.1 The NAT Traversal Problem

NAT devices are installed to alleviate the exhaustion of the IPv4 address space by allowing the use of private IP addresses on home and corporate networks (internal networks) behind routers with a single public IP address facing the public Internet. The internal network devices are enabled to communicate with hosts on the external network by changing the source address of outgoing requests to that of the NAT device and relaying replies back to the originating device. This leaves the internal network ill-suited to host servers, as the NAT device has no automatic method of determining the internal host for which incoming packets are destined. This problem has not generally been relevant to home users behind NAT devices for general web access and e-mail. However, applications such as peer-to-peer file sharing (such as BitTorrent or Gnutella applications), VoIP services (such as Session Initiation Protocol) and the online services of current generation video game consoles (such as the Xbox 360's Xbox Live or the PS3's PlayStation Network) require clients to be servers as well, thereby posing a problem for users behind NAT devices, as incoming requests cannot be easily correlated to the proper internal host. Furthermore many of these types of services carry IP address and port number information in the application data, potentially requiring substitution or special traversal techniques for NAT traversal.

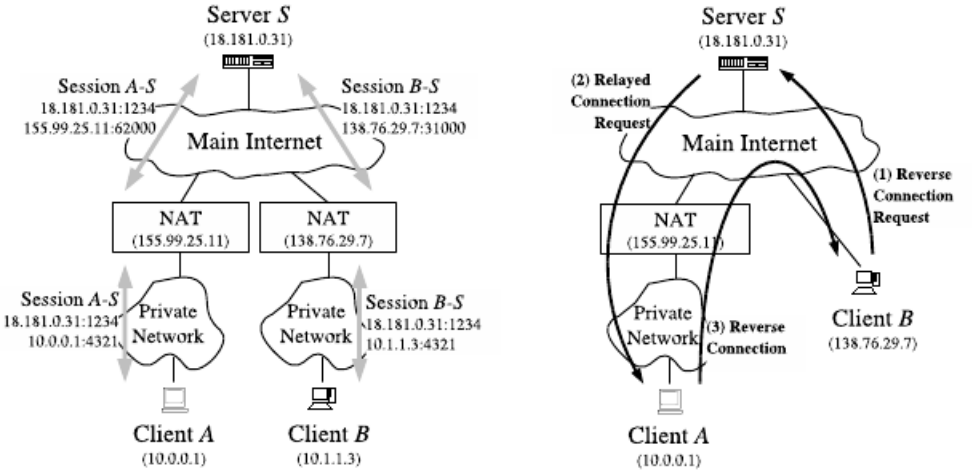
Several different protocols have been designed for NAT Traversal. Listed below are a few of the popular ones.

1. NAT Traversal based on NAT behaviour:
    - Session Traversal Utilities for NAT (STUN)
    - Traversal Using Relay NAT (TURN)
    - NAT-T Negotiation of NAT-Traversal in the IKE
    - Teredo tunneling (uses NAT traversal to provide IPv6 connectivity)
    - Session Border Controller (SBC)
    - UDP/TCP/ICMP hole punching
2. NAT traversal based on NAT control:
    - Realm-Specific IP (RSIP)
    - Middlebox Communications (MIDCOM)
    - SOCKS
    - NAT Port Mapping Protocol (NAT PMP)
    - Internet Gateway Device (IGD) Protocol, defined by the Universal Plug and Play (UPnP) Forum.
    - Application Layer Gateway (ALG)

3.2 Relaying

The most reliable -but the least efficient -method of P2P communication across NAT is simply to make the communication look to the network like standard client/server communication, through relaying. Suppose two client hosts A and B have each initiated TCP or UDP connections to a well-known server S, at S's global IP address 18.181.0.31 and port number 1234. As shown in Fig.2a,the clients reside on separate private networks, and their respective NATs prevent either client from directly initiating a connection to the other. Instead of attempting a direct connection, the two clients can simply use the server S to relay messages between them. For example, to send a message to client B, client A simply sends the message to server S along its already-established client/server connection, and server S forwards the message on to client B using its existing client/server connection with B.

Relaying always works as long as both clients can connect to the server. Its disadvantages are that it consumes the server's processing power and network bandwidth, and communication latency between the peering clients is likely increased even if the server is well-connected. Nevertheless, since there is no more efficient technique that works reliably on all existing NATs, relaying is a useful fall-back strategy if maximum robustness is desired. The TURN protocol defines a method of implementing relaying in a relatively secure fashion.



(a) NAT Traversal: Relaying (b) NAT Traversal:Connection Reversal

Figure 2: Popular NAT Traversal Techniques

3.3 Connection Reversal

Some P2P applications use a straightforward but limited technique, known as connection reversal, to enable communication when both hosts have connections to a well-known rendezvous server S and only one of the peers is behind a NAT, as shown in Fig.2b. If A wants to initiate a connection to B, then a direct connection attempt works automatically, because B is not behind a NAT and A's NAT interprets the connection as an outgoing session. If B wants to initiate a connection to A, however, any direct connection attempt to A is blocked by A's NAT. B can instead relay a connection request to A through a well-known server S, asking A to attempt a 'reverse' connection

back to B. Despite the obvious limitations of this technique, the central idea of using a well-known rendezvous server as an intermediary to help set up direct peer-to-peer connections is fundamental to designing the more general hole punching techniques.

4 UDP Hole Punching: The Way Skype Does It

Increasingly, computers are being positioned behind firewalls to protect systems from internet threats. Ideally, the firewall function will be performed by a router, which also translates the PC's local network address to the public IP address (Network Address Translation, or NAT). This means an attacker cannot directly adress the PC from the outside - connections have to be established from the inside. This is of course a problem when two computers behind NAT firewalls require to talk directly to each other - if, for example, their users want to call each other using Voice over IP (VoIP), which is exactly what Skype is used for. The dilemma is clear - whichever party calls the other, the recipient's firewall will decline the apparent attack and will simply discard the data packets. The telephone call doesn't happen. Or at least that's what a network administrator would expect.

But anyone who has used the increasingly popular internet telephony software Skype knows that it works as smoothly behind a NAT firewall as it does if the PC is connected directly to the internet. The reason for this is that the inventors of Skype and similar software have come up with a solution. Naturally every firewall must also let packets through into the local network - after all the user wants to view websites, read e-mails, etc. The firewall must therefore forward the relevant data packets from outside, to the workstation computer on the LAN. However it only does so, when it is convinced that a packet represents the response to an outgoing data packet. A NAT router therefore keeps tables of which internal computer has communicated with which external computer and which ports the two have used.

The trick used by VoIP software consists of persuading the firewall that a connection has been established, to which it should allocate subsequent incoming data packets. The fact that audio data for VoIP is sent using the

connectionless UDP protocol acts to Skype's advantage. In contrast to TCP, which includes additional connection information in each packet, with UDP, a firewall sees only the addresses and ports of the source and destination systems. If, for an incoming UDP packet, these match an NAT table entry, it will pass the packet on to an internal computer with a clear conscience. This technique is precisely called UDP Hole Punching. It enables two clients to set up a direct peer-to-peer UDP session with the help of a well-known rendezvous server, even if the clients are both behind NATs.

4.1 The Rendezvous Server

Hole punching assumes that the two clients, A and B, already have active UDP sessions with a rendezvous server S. When a client registers with S, the server records two endpoints for that client: the (IP address, UDP port) pair that the client believes itself to be using to talk with S, and the (IP address, UDP port) pair that the server observes the client to be using to talk with it. We refer to the first pair as the client's private endpoint and the second as the client's public endpoint. The server might obtain the client's private endpoint from the client itself in a field in the body of the client's registration message, and obtain the client's public endpoint from the source IP address and source UDP port fields in the IP and UDP headers of that registration message. If the client is not behind a NAT, then its private and public endpoints should be identical.

A few poorly behaved NATs are known to scan the body of UDP datagrams for 4-byte fields that look like IP addresses, and translate them as they would the IP address fields in the IP header. To be robust against such behavior, applications may wish to obfuscate IP addresses in messages bodies slightly, for example by transmitting the one's complement of the IP address instead of the IP address itself. Of course, if the application is encrypting its messages, then this behavior is not likely to be a problem.

4.2 Establishing p2p sessions

Suppose client A wants to establish a UDP session directly with client B. Hole punching proceeds as follows:

1. A initially does not know how to reach B, so A asks S for help establishing a UDP session with B.

2. S replies to A with a message containing B's public and private endpoints. At the same time, S uses its UDP session with B to send B a connection request message containing A's public and private endpoints. Once these messages are received, A and B know each other's public and private endpoints.

3. When A receives B's public and private endpoints from S, A starts sending UDP packets to both of these endpoints, and subsequently *'locks in'* whichever endpoint first elicits a valid response from B. Similarly, when B receives A's public and private endpoints in the forwarded connection request, B starts sending UDP packets to A at each of A's known endpoints, locking in the first endpoint that works. The order and timing of these messages are not critical as long as they are asynchronous.

We now consider how UDP hole punching handles each of the three specific network scenarios.

1. The two clients reside behind the same NAT on one private network (the *easy* case).

2. The most common case is when the two clients reside behind different NATs.

3. The clients each reside behind two levels of NAT: a common *first-level* NAT deployed by an ISP for example, and distinct *second-level* NATs such as consumer NAT routers for home networks.

It is in general difficult or impossible for the application itself to determine the exact physical layout of the network, and thus which of these scenarios (or the many other possible ones) actually applies at a given time. Protocols such as STUN can provide some information about the NATs present on a communication path, but this information may not always be complete or reliable, especially when multiple levels of NAT are involved. Nevertheless, hole punching works automatically in all of these scenarios without the application having to know the specific network organization, as long as the NATs involved behave in a reasonable fashion.

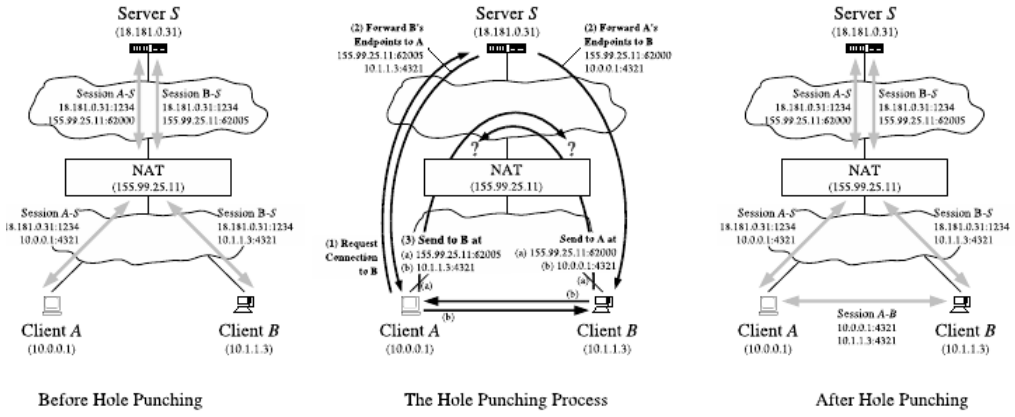


Figure 3: UDP Hole Punching: Peer behind a common NAT

4.3 Peers Behind a Common NAT

First consider the simple scenario in which the two clients (probably unknowingly) happen to reside behind the same NAT, and are therefore located in the same private IP address realm, as shown in Fig.3. Client A has established a UDP session with server S, to which the common NAT has assigned its own public port number 62000. Client B has similarly established a session with S, to which the NAT has assigned public port number 62005.

Suppose that client A uses the hole punching technique outlined above to establish a UDP session with B, using server S as an introducer. Client A sends S a message requesting a connection to B. S responds to A with B's public and private endpoints, and also forwards A's public and private endpoints to B. Both clients then attempt to send UDP datagrams to each other directly at each of these endpoints. The messages directed to the public end-points may or may not reach their destination, depending on whether or not the NAT supports hairpin translation. The messages directed at the private endpoints do reach their destinations, however, and since this direct route through the private network is likely to be faster than an indirect route through the NAT anyway, the clients are most likely to select the private endpoints for subsequent regular communication.

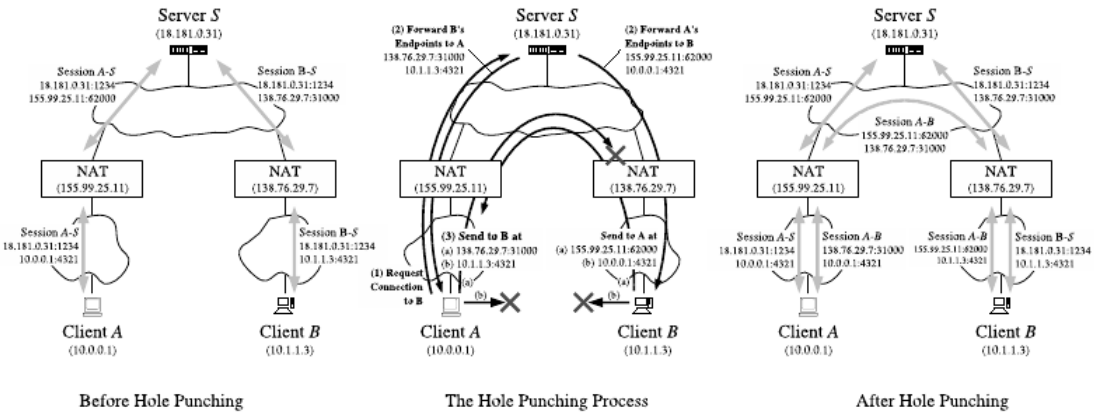


Figure 4: UDP Hole Punching: Peers behind different NATs

4.4 Peers Behind Different NATs

Suppose clients A and B have private IP addresses behind different NATs, as shown in Fig. 4. A and B have each initiated UDP communication sessions from their local port 4321 to port 1234 on server S. In handling these outbound sessions, NAT A has assigned port 62000 at its own public IP address, 155.99.25.11, for the use of A's session with S, and NAT B has assigned port 31000 at its IP address, 138.76.29.7, to B's session with S.

In A's registration message to S, A reports its private endpoint to S as 10.0.0.1:4321, where 10.0.0.1 is A's IP address on its own private network. S records A's reported private endpoint, along with A's public endpoint as observed by S itself. A's public endpoint in this case is 155.99.25.11:62000, the temporary endpoint assigned to the session by the NAT. Similarly, when client B registers, S records B's private endpoint as 10.1.1.3:4321 and B's public endpoint as 138.76.29.7:31000.

Now client A follows the hole punching procedure described above to establish a UDP communication session directly with B. First, A sends a request message to S asking for help connecting with B. In response, S sends B's public and private endpoints to A, and sends A's public and private endpoints to B. A and B each start trying to send UDP datagrams directly to each of

these endpoints.

Since A and B are on different private networks and their respective private IP addresses are not globally routable, the messages sent to these endpoints will reach either the wrong host or no host at all. Because many NATs also act as DHCP servers, handing out IP addresses in a fairly deterministic way from a private address pool usually determined by the NAT vendor by default, it is quite likely in practice that A's messages directed at B's private endpoint will reach some (incorrect) host on A's private network that happens to have the same private IP address as B does. Applications must therefore authenticate all messages in some way to filter out such stray traffic robustly. The messages might include application specific names or cryptographic tokens, for example, or at least a random nonce pre-arranged through S.

Now consider A's first message sent to B's public end-point, as shown in Fig. 4. As this outbound message passes through A's NAT, this NAT notices that this is the first UDP packet in a new outgoing session. The new session's source endpoint (10.0.0.1:4321) is the same as that of the existing session between A and S, but its destination endpoint is different. If NAT A is well-behaved, it preserves the identity of A's private endpoint, consistently translating all outbound sessions from private source end-point 10.0.0.1:4321 to the corresponding public source endpoint 155.99.25.11:62000. A's first outgoing message to B's public endpoint thus, in effect, *punches a hole* in A's NAT for a new UDP session identified by the endpoints (10.0.0.1:4321, 138.76.29.7:31000) on A's private network, and by the endpoints (155.99.25.11: 62000, 138.76.29.7:31000) on the main Internet.

If A's message to B's public endpoint reaches B's NAT before B's first message to A has crossed B's own NAT, then B's NAT may interpret A's inbound message as unsolicited incoming traffic and drop it. B's first message to A's public address, however, similarly opens a hole in B's NAT, for a new UDP session identified by the end-points (10.1.1.3:4321, 155.99.25.11:62000) on B's private network, and by the endpoints (138.76.29.7:31000, 155.99.25.11:62000) on the Internet. Once the first messages from A and B have crossed their respective NATs, holes are open in each direction and UDP communication can proceed normally. Once the clients have verified that the public endpoints

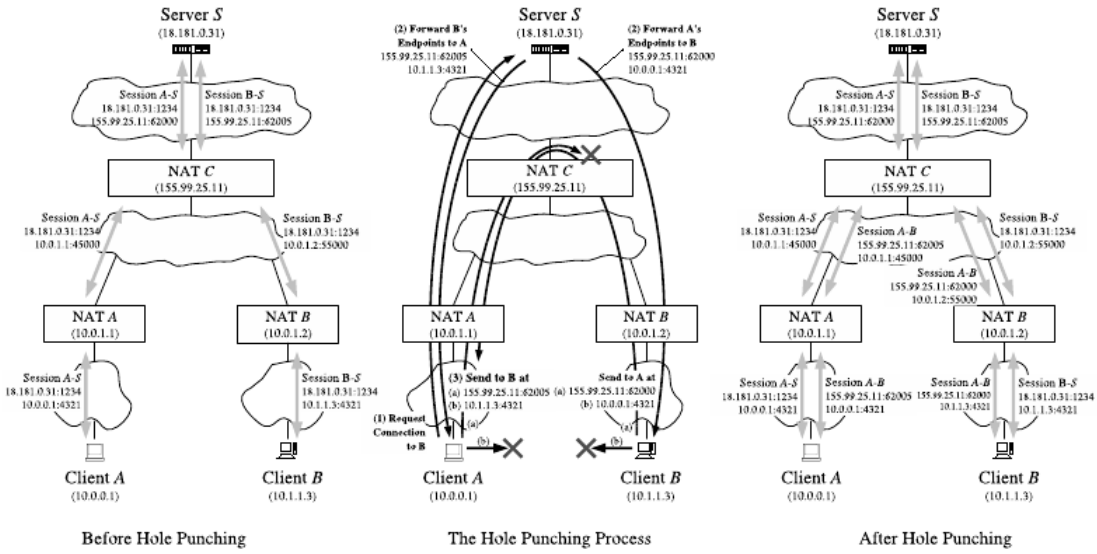


Figure 5: UDP Hole Punching: Peers behind multiple levels of NAT

work, they can stop sending messages to the alternative private endpoints.

4.5 Peers Behind Multiple Levels of NAT

In some topologies involving multiple NAT devices, two clients cannot establish an optimal P2P route between them without specific knowledge of the topology. Consider a final scenario, depicted in Fig. 5. Suppose NAT C is a large industrial NAT deployed by an internet service provider (ISP) to multiplex many customers onto a few public IP addresses, and NATs A and B are small consumer NAT routers deployed independently by two of the ISP's customers to multiplex their private home networks onto their respective ISP-provided IP addresses. Only server S and NAT C have globally routable IP addresses; the 'public' IP addresses used by NAT A and NAT B are actually private to the ISP's address realm, while client A's and B's addresses in turn are private to the addressing realms of NAT A and NAT B, respectively. Each client initiates an outgoing connection to server S as before, causing NATs A and B each to create a single public/private translation, and causing NAT C to establish a public/private translation for each

session.

Now suppose A and B attempt to establish a direct peer-to-peer UDP connection via hole punching. The optimal routing strategy would be for client A to send messages to client B's 'semi-public' endpoint at NAT B, 10.0.1.2:55000 in the ISP's addressing realm, and for client B to send messages to A's 'semi-public' end-point at NAT B, namely 10.0.1.1:45000. Unfortunately, A and B have no way to learn these addresses, because server S only sees the truly global public endpoints of the clients, 155.99.25.11:62000 and 155.99.25.11:62005 respectively. Even if A and B had some way to learn these addresses, there is still no guarantee that they would be usable, because the address assignments in the ISP's private address realm might conflict with unrelated address assignments in the clients' private realms. (NAT A's IP address in NAT C's realm might just as easily have been 10.1.1.3, for example, the same as client B's private address in NAT B's realm.)

The clients therefore have no choice but to use their global public addresses as seen by S for their P2P communication, and rely on NAT C providing hairpin or loopback translation. When A sends a UDP datagram to B's global endpoint, 155.99.25.11:62005, NAT A first translates the datagram's source endpoint from 10.0.0.1:4321 to 10.0.1.1:45000. The datagram now reaches NAT C, which recognizes that the datagram's destination address is one of NAT C's own translated public endpoints. If NAT C is well-behaved, it then translates both the source and destination addresses in the datagram and 'loops' the datagram back onto the private network, now with a source endpoint of 155.99.25.11:62000 and a destination endpoint of 10.0.1.2:55000. NAT B finally translates the datagram's destination address as the datagram enters B's private network, and the datagram reaches B. The path back to A works similarly. Many NATs do not yet support hairpin translation, but it is becoming more common as NAT vendors become aware of this issue.

5 STUN: A Popular NAT Traversal Protocol for VoD

STUN, Session Traversal Utilities for NAT, is a NAT traversal protocol used in applications involving real-time voice, video, messaging, and other

interactive IP communications. It is a light-weight client-server protocol requiring only simple query and response via UDP. The client side is implemented in the user's communications application, such as a Voice over Internet Protocol (VoIP) phone or instant messaging client. The client, operating inside the NAT-masqueraded network, initiates a short sequence of requests to a STUN protocol server listening at two IP addresses in the network on the public side of the NAT, traversing the NAT. The server responds with the results, which are the mapped IP address and port on the 'outside' of the NAT for each request to its two listeners. From the results of several different types of requests, the client application can learn the operating method of the network address translator, including the lifetime of the NAT's port bindings.

STUN usually operates on a User Datagram Protocol (UDP) messaging transport. Since UDP does not provide reliable transport guarantees, reliability is achieved by application-controlled retransmissions of the STUN requests. STUN servers do not implement any reliability mechanism for their responses. When reliability is mandatory, Transmission Control Protocol (TCP) may be used, but induces extra networking overhead. In security-sensitive applications, STUN may be transported and encrypted via TCP/TLS.

An application may automatically determine a suitable STUN server for communications with a particular peer by querying for the stun (for UDP) or stuns (for TCP/TLS) SRV Domain Name System (DNS) resource record. The standard listening port number for a STUN server is 3478, for UDP and TCP, and 5349 for TLS. Alternately, TLS may also be run on the TCP port if the server implementation can de-multiplex TLS and STUN packets. In case no STUN server is found using SRV lookups, the standard recommends that the destination domain name should be queried for address records (A or AAAA) which would be used with the default port numbers. In addition to using protocol encryption via TLS, STUN also has built-in authentication and message-integrity mechanisms via specialized STUN packet types.

When a client has discovered its external address, it can communicate with communication peers by advertising its external NAT address to its peers, rather than the masqueraded (internal) address that is not reachable for its peers on the public network. In some cases of NATs (e.g., full-cone type)

then either side can initiate communication. With other types (restricted cone or restricted port cone types) both sides must commence transmitting at about the same time to establish the port bindings in the network address translator.

Protocols like the Real-time Transport Protocol (RTP) and the Session Initiation Protocol (SIP) use UDP packets for the transfer of media and signaling traffic over the Internet. In many application scenarios it is common that both endpoints are located behind a NAT. This double-NAT problem is often not easily overcome even with STUN and sometimes an intermediate application proxy server is required.

References

[1] Y. Huang, T. Z. Fu, D.M. Chiu, J. C. Lui, and C. Huang. Challenges, design and analysis of a large-scale p2p-vod system. *in ACM SIGCOMM 2008 conference on Data communication. ACM, 2008*, pp. 375388.

[2] B. Ford, P. Srisuresh, D. Kegel. Peer-to-peer Communication Across Network Address Translators. *in Proceedings of USENIX*, 2005.

[3] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN: Simple Traversal of User Datagram Protocol (UDP)Through Network Address Translators (NATs). *RFC 3489, IETF, Mar. 2003.*

[4] S. A. Baset and H. G. Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. *in In proceedings INFOCOM 2006. 25th IEEE International Conference on Computer Communications. IEEE*, 2006, pg. 1-11.

[5] Wikipedia. http://en.wikipedia.org/wiki/NAT_traversal

[6] H-Online. http://www.h-online.com/security/features/How-Skype-Co-get-round-firewalls-747197.html