Introduction to Network Address Translation (NAT) and NAT Traversal

This page describes NAT and the problems caused by it and the solutions. More...

Introduction to NAT

NAT (Network Address Translation) is a mechanism where a device performs modifications to the TCP/IP address/port number of a packet and maps the IP address from one realm to another (usually from private IP address to public IP address and vice versa). This works by the NAT device allocating a temporary port number on the public side of the NAT upon forwarding outbound packet from the internal host towards the Internet, maintaining this mapping for some predefined time, and forwarding the inbound packets received from the Internet on this public port back to the internal host.

NAT devices are installed primarily to alleviate the exhaustion of IPv4 address space by allowing multiple hosts to share a public/Internet address. Also due to its mapping nature (i.e. a mapping can only be created by a transmission from an internal host), NAT device is preferred to be installed even when IPv4 address exhaustion is not a problem (for example when there is only one host at home), to provide some sort of security/shield for the internal hosts against threats from the Internet.

Despite the fact that NAT provides some shields for the internal network, one must distinguish NAT solution from firewall solution. NAT is not a firewall solution. A firewall is a security solution designed to enforce the security policy of an organization, while NAT is a connectivity solution to allow multiple hosts to use a single public IP address. Understandably both functionalities are difficult to separate at times, since many (typically consumer) products claims to do both with the same device and simply label the device a “NAT box”. But we do want to make this distinction rather clear, as PJNATH is a NAT traversal helper and not a firewall bypass solution (yet).

The NAT traversal problems

While NAT would work well for typical client server communications (such as web and email), since it's always the client that initiates the conversation and normally client doesn't need to maintain the connection for a long time, installation of NAT would cause major problem for peer-to-peer communication, such as (and especially) VoIP. These problems will be explained in more detail below.

Peer address problem

In VoIP, normally we want the media (audio, and video) to flow directly between the clients, since relaying is costly (both in terms of bandwidth cost for service provider, and additional latency introduced by relaying). To do this, each client informs its media transport address to the other client , by sending it via the VoIP signaling path, and the other side would send its media to this transport address.

And there lies the problem. If the client software is not NAT aware, then it would send its private IP address to the other client, and the other client would not be able to send media to this address.

Traditionally this was solved by using STUN. With this mechanism, the client first finds out its public IP address/port by querying a STUN server, then send sthis public address instead of its private address to the other client. When both sides are using this mechanism, they can then send media packets to these addresses, thereby creating a mapping in the NAT (also called opening a "hole", hence this

mechanism is also popularly called "hole punching") and both can then communicate with each other.

But this mechanism does not work in all cases, as will be explained below.

Hairpinning behavior

Hairpin is a behavior where a NAT device forwards packets from a host in internal network (lets call it host A) back to some other host (host B) in the same internal network, when it detects that the (public IP address) destination of the packet is actually a mapped IP address that was created for the internal host (host B). This is a desirable behavior of a NAT, but unfortunately not all NAT devices support this.

Lacking this behavior, two (internal) hosts behind the same NAT will not be able to communicate with each other if they exchange their public addresses (resolved by STUN above) to each other.

Symmetric behavior

NAT devices don't behave uniformly and people have been trying to classify their behavior into different classes. Traditionally NAT devices are classified into Full Cone, Restricted Cone, Port Restricted Cone, and Symmetric types, according to RFC 3489 section 5. A more recent method of classification, as explained by RFC 4787, divides the NAT behavioral types into two attributes: the mapping behavior attribute and the filtering behavior attribute. Each attribute can be one of three types: *Endpoint-Independent*, *Address-Dependent*, or *Address and Port-Dependent*. With this new classification method, a Symmetric NAT actually is an Address and Port-Dependent mapping NAT.

Among these types, the Symmetric type is the hardest one to work with. The problem is because the NAT allocates different mapping (of the same internal host) for the communication to the STUN server and the communication to the other (external) hosts, so the IP address/port that is informed by one host to the other is meaningless for the recipient since this is not the actual IP address/port mapping that the NAT device creates. The result is when the recipient host tries to send a packet to this address, the NAT device would drop the packet since it does not recognize the sender of the packet as the "authorized" hosts to send to this address.

There are two solutions for this. The first, we could make the client smarter by switching transmission of the media to the source address of the media packets. This would work since normally clients uses a well known trick called symmetric RTP, where they use one socket for both transmitting and receiving RTP/media packets. We also use this mechanism in PJMEDIA media transport. But this solution only works if a client behind a symmetric NAT is not communicating with other client behind either symmetric NAT or port-restricted NAT.

The second solution is to use media relay, but as have been mentioned above, relaying is costly, both in terms of bandwidth cost for service provider and additional latency introduced by relaying.

Binding timeout

When a NAT device creates a binding (a public-private IP address mapping), it will associate a timer with it. The timer is used to destroy the binding once there is no activity/traffic associated with the binding. Because of this, a NAT aware application that wishes to keep the binding open must periodically send outbound packets, a mechanism known as keep-alive, or otherwise it will ultimately loose the binding and unable to receive incoming packets from Internet.

The NAT traversal solutions

Old STUN (RFC 3489)

The original STUN (Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)) as defined by RFC 3489 (published in 2003, but the work was started as early as 2001) was meant to be a standalone, standard-based solution for the NAT connectivity problems above. It is equipped with NAT type detection algoritm and methods to hole-punch the NAT in order to let traffic to get through and has been proven to be quite successful in traversing many types of NATs, hence it has gained a lot of popularity as a simple and effective NAT traversal solution.

But since then the smart people at IETF has realized that STUN alone is not going to be enough. Besides its nature that STUN solution cannot solve the symmetric-to-symmetric or port-restricted

connection, people have also discovered that NAT behavior can change for different traffic (or for the same traffic overtime) hence it was concluded that NAT type detection could produce unreliable results hence one should not rely too much on it.

Because of this, STUN has since moved its efforts to different strategy. Instead of attempting to provide a standalone solution, it's now providing a part solution and framework to build other (STUN based) protocols on top of it, such as TURN and ICE.

STUN/STUNbis (RFC 5389)

The Session Traversal Utilities for NAT (STUN) is the further development of the old STUN. While it still provides a mechanism for a client to query its public/mapped address to a STUN server, it has deprecated the use of NAT type detection, and now it serves as a framework to build other protocols on top of it (such as TURN and ICE).

Old TURN (draft-rosenberg-midcom-turn)

Traversal Using Relay NAT (TURN), a standard-based effort started as early as in November 2001, was meant to be the complementary method for the (old) STUN to complete the solution. The original idea was the host to use STUN to detect the NAT type, and when it has found that the NAT type is symmetric it would use TURN to relay the traffic. But as stated above, this approach was deemed to be unreliable, and now the prefered way to use TURN (and it's a new TURN specification as well) is to combine it with ICE.

TURN (draft-ietf-behave-turn)

Traversal Using Relays around NAT (TURN) is the latest development of TURN. While the protocol details have changed a lot, the objective is still the same, that is to provide relaying control for the application. As mentioned above, preferably TURN should be used with ICE since relaying is costly in terms of both bandwidth and latency, hence it should be used as the last resort.

B2BUA approach

A SIP Back to Back User Agents (B2BUA) is a SIP entity that sits in the middle of SIP traffic and acts as SIP user agents on both call legs. The primary motivations to have a B2BUA are to be able to provision the call (e.g. billing, enforcing policy) and to help with NAT traversal for the clients. Normally a B2BUA would be equipped with media relaying or otherwise it wouldn't be very useful.

Products that fall into this category include SIP Session Border Controllers (SBC), and PBXs such as Asterisk are technically a B2BUA as well.

The benefit of B2BUA with regard to helping NAT traversal is it does not require any modifications to the client to make it go through NATs. And since basically it is a relay, it should be able to traverse symmetric NAT successfully.

However, since it is a relay, the usual relaying drawbacks apply, namely the bandwidth and latency issue. More over, since a B2BUA acts as user agent in either call-legs (i.e. it terminates the SIP signaling/call on one leg, albeit it creates another call on the other leg), it may also introduce serious issues with end-to-end SIP signaling.

ALG approach

Nowdays many NAT devices (such as consumer ADSL routers) are equipped with intelligence to inspect and fix VoIP traffic in its effort to help it with the NAT traversal. This feature is called Application Layer Gateway (ALG) intelligence. The idea is since the NAT device knows about the mapping, it might as well try to fix the application traffic so that the traffic could better traverse the NAT. Some tricks that are performed include for example replacing the private IP addresses/ports in the SIP/SDP packet with the mapped public address/port of the host that sends the packet.

Despite many claims about its usefullness, in reality this has given us more problems than the fix. Too many devices such as these break the SIP signaling, and in more advanced case, ICE negotiation. Some examples of bad situations that we have encountered in the past:

- NAT device alters the Via address/port fields in the SIP response message, making the response fail to pass SIP response verification as defined by SIP RFC.
- In other case, the modifications in the Via headers of the SIP response hides the important information from the SIP server, nameny the actual IP address/port of the client as seen by the SIP server.
- Modifications in the Contact URI of REGISTER request/response makes the client unable to detect it's registered binding.
- Modifications in the IP addresses/ports in SDP causes ICE negotiation to fail with ice-mismatch status.
- The complexity of the ALG processing in itself seems to have caused the device to behave erraticly with managing the address bindings (e.g. it creates a new binding for the second packet sent by the client, even when the previous packet was sent just second ago, or it just sends inbound packet to the wrong host).

Many man-months efforts have been spent just to troubleshoot issues caused by these ALG (mal)functioning, and as it adds complexity to the problem rather than solving it, in general we do not like this approach at all and would prefer it to go away.

UPnP

The Universal Plug and Play (UPnP) is a set of protocol specifications to control network appliances and one of its specification is to control NAT device. With this protocol, a client can instruct the NAT device to open a port in the NAT's public side and use this port for its communication. UPnP has gained popularity due to its simplicity, and one can expect it to be available on majority of NAT devices.

The drawback of UPnP is since it uses multicast in its communication, it will only allow client to control one NAT device that is in the same multicast domain. While this normally is not a problem in household installations (where people normally only have one NAT router), it will not work if the client is behind cascaded routers installation. More over uPnP has serious issues with security due to its lack of authentication, it's probably not the prefered solution for organizations.

Other solutions

Other solutions to NAT traversal includes:

- SOCKS, which supports UDP protocol since SOCKS5.

ICE Solution - The Protocol that Works Harder

A new protocol is being standardized (it's in Work Group Last Call/WGLC stage at the time this article was written) by the IETF, called Interactive Connectivity Establishment (ICE). ICE is the ultimate weapon a client can have in its NAT traversal solution arsenals, as it promises that if there is indeed one path for two clients to communicate, then ICE will find this path. And if there are more than one paths which the clients can communicate, ICE will use the best/most efficient one.

ICE works by combining several protocols (such as STUN and TURN) altogether and offering several candidate paths for the communication, thereby maximising the chance of success, but at the same time also has the capability to prioritize the candidates, so that the more expensive alternative (namely relay) will only be used as the last resort when else fails. ICE negotiation process involves several stages:

- candidate gathering, where the client finds out all the possible addresses that it can use for the communication. It may find three types of candidates: host candidate to represent its physical NICs, server reflexive candidate for the address that has been resolved from STUN, and relay candidate for the address that the client has allocated from a TURN relay.
- prioritizing these candidates. Typically the relay candidate will have the lowest priority to use since it's the most expensive.
- encoding these candidates, sending it to remote peer, and negotiating it with offer-answer.
- pairing the candidates, where it pairs every local candidates with every remote candidates that it receives from the remote peer.
- checking the connectivity for each candidate pairs.
- concluding the result. Since every possible path combinations are checked, if there is a path to communicate ICE will find it.

There are many benetifs of ICE:

- it's standard based.
- it works where STUN works (and more)
- unlike standalone STUN solution, it solves the hairpinning issue, since it also offers host candidates.
- just as relaying solutions, it works with symmetric NATs. But unlike plain relaying, relay is only used as the last resort, thereby minimizing the bandwidth and latency issue of relaying.
- it offers a generic framework for offering and checking address candidates. While the ICE core standard only talks about using STUN and TURN, implementors can add more types of candidates in the ICE offer, for example UDP over TCP or HTTP relays, or even uPnP candidates, and this could be done transparently for the remote peer hence it's compatible and usable even when the remote peer does not support these.
- it also adds some kind of security particularly against DoS attacks, since media address must be acknowledged before it can be used.

Having said that, ICE is a complex protocol to implement, making interoperability an issue, and at this time of writing we don't see many implementations of it yet. Fortunately, PJNATH has been one of the first hence more mature ICE implementation, being first released on mid-2007, and we have been testing our implementation at SIP Interoperability Test (SIPit) events regularly, so hopefully we are one of the most stable as well.

PJNATH - The building blocks for effective NAT traversal solution

PJSIP NAT Helper (PJNATH) is a library which contains the implementation of standard based NAT traversal solutions. PJNATH can be used as a stand-alone library for your software, or you may use PJSUA-LIB library, a very high level library integrating PJSIP, PJMEDIA, and PJNATH into simple to use APIs.

PJNATH has the following features:

- STUNbis implementation, providing both ready to use STUN-aware socket and framework to implement higher level STUN based protocols such as TURN and ICE.
- NAT type detection, useful for troubleshooting purposes.
- TURN implementation.
- ICE implementation.

More protocols will be implemented in the future.

Go back to **PJNATH - Open Source ICE, STUN, and TURN Library**.
