E7311 Software Defined Networking – Lecture Notes

Lecture 2: Network Address Translators and NAT Traversal

Introduction

When the Internet was becoming commercial in mid 1990s, it faced two scalability problems: address exhaustion and routing table growth in the core networks. Two “short term solutions” were created. The first was Classless Inter Domain Routing (CIDR) and the second was Network Address Translation[1] or NAT at the borders of stub networks. A stub network does not carry transit traffic but only provides access to users connected to it. CIDR was discussed on lecture 1.

An extension to NAT is Network Address and Port Translation (NAPT). We lump both under the simple term NAT.

NATs let stub networks reuse IPv4 addresses from the range of private addresses allocated by IANA:

10.0.0.0 - 10.255.255.255	2^{24} = 16M addresses
172.16.0.0 - 172.31.255.255	2^{20} = 1M addresses

[1] RFC 1631 - The IP Network Address Translator (NAT) from 1994.

192.168.0.0 - 192.168.255.255 2^{16} = 65K addresses

The task of a NAT device is to provide a globally unique IP address to a host that wants to access a server in the global address space. More specifically, this is called the source-NAT. Allocation is on-demand and dynamic. We call the dynamically allocated address, a NAT outbound-IP. Usually, at the same time, the NAT maps the source port the client is using to a dynamic port number. Since not all hosts are always active and since one host does not need all 2^{16} ports at the same time, many clients can share a public IP address that is owned by the NAT. As a rule, clients can use any port numbers from a large subrange of "ephemeral ports[2]", while there are a number of "well-known" server ports such as 80 and 8080 for http. Because two clients, might have co-incidentally allocated the same "ephemeral" source port, it would not be a good idea to let the two clients share a source address and let them keep their arbitrarily chosen source ports in the wide area packet.

The NAT tries to stay ***invisible*** to the applications. Because it manipulates the source address and source port fields in the TCP/IP or UDP/IP message, the NAT has to recalculate checksums on IP and UDP/TCP layers.

Upon a new message from a client in the internal network, the NAT device creates *connection state or NAT binding* that contains the mapping of the client's (IP:port) -pair to the *outbound (IP:port) -pair*. The latter are owned by the NAT. The state also contains a timeout. The value of the timeout may be protocol state (TCP or UDP) dependent. It follows that a NAT may know quite a bit about the Internet protocols such as TCP, UDP, SCTP etc. The binding state is necessary for routing the responses back to the client.

The limitation of a NAT is that usually, upon a new message arriving from the external network while there is no suitable state where the sender's IP address is found, the message must be dropped. This is the case in the destination network and destination-NAT. We call this issue the *NAT reachability problem*. Another way to look at this property is that this is the intended behavior of hiding the private network from the global Internet. Depending on the policy, the NAT may or may not respond to the sender by ICMP indicating that the message has been dropped. A NAT may also allow *port forwarding*. This means that upon a message to port XY and to address xy owned by the NAT, the NAT will send the incoming message to private address zx. The rule is static configuration information and opens a hole in the "firewall" formed by the NAT. It would be possible to let the client know about the destination port XY in response to its DNS query, if the client happens to support SRV records. However, not many applications support SRV records. SRV -record appends the normal A-record containing just the destination IP as response to a domain name query with a destination port number associated with the domain name.

[2] See about the differences on different platforms: https://en.wikipedia.org/wiki/Ephemeral_port#:~:text=4%20Notes,Range,the%20port%20range%2032768%E2%80%9360999.

The ideology called the “end-to-end” principle has been very strong in IETF for a long time. After the introduction of NATs in 1994, IETF and many protocol engineers made believe that NATs do not exist but the network practitioners applied them widely. The result was that IEFT has been spending a lot of engineering effort in fixing problems caused by NATs.

Under the end-to-end principle, the assumption is that an address is both an identifier of a host and the routing locator. (Most people seem to make this assumption although this is not mentioned in how D. Clark et. al. initially formulated the end-to-end principle). When an application protocol uses an IP address as an identifier and the NAT changes the address on the way, the application protocol is broken. A protocol, (e.g. FTP or SIP/SDP) may for the purpose of host identification send the local IP address in a control message to a remote party. If there is a NAT between the two parties, usually the application protocol does not work. At the same time, due to dynamic IP addresses, it has been obvious for a long time that IP addresses are really poor IDs. As an ID, IP address can be valid just for the current session.

With the introduction of peer-to-peer applications and in particular interactive multimedia including Voice over IP, the need for better ways of NAT traversal became urgent. To address the issue, the BEHAVE group of IETF created the descriptions of NAT behavior, best practices of configuring NATs, protocols for snooping on the types of mapping the NAT is creating, relay services for inbound traffic through NATs etc. Other groups have created complete solutions for NAT traversal for particular needs.

In corporate networks, NAT are often integrated with a Firewall. Intelligent *Firewalls create connection state* and have protocol specific state machines for many protocols besides FTP. The connection state is needed for the Firewall to be able to process the protocol properly, i.e. provide maximum protection for the internal network while letting the application protocol flow through in a controlled manner.

What are NATs and why are they needed

For NAT, the term “translation” was adopted. Actually, using a reasonable definition of the term “switching”, a *NAT is a TCP/IP -layer switching device*. (Switching is a mode of data forwarding, where the switch uses a flow or connections state to mangle the forwarded data and/or to choose the path or outlet for the data.) Because of political reasons, this terminology has not been used. Instead, we talk about *translation*. Fair enough, a NAT, besides a simple swap operation of addresses and ports, has to recalculate checksums and it may have some protocol specific treatment of the time-out. We however see that checksum calculation and modification falls also under the process of “mangling the packet” based on the network held state.

Routing is a process of determining path or next hop to the location of the destination of communication. The mechanism of delivering the traffic to the target is either *switching* or *forwarding*.

Delivery is always based on network state. It may be an *aggregate state* that applies to many hosts and users or it can be a *connection state (or flow state)* that applies to a single sender or receiver or a pair of a sender and a receiver. A state that applies to many users can be seen as *network state*. Rather than being dependent on the actual communication that is taking place, it is a function of the network topology. On lecture 1, we saw how these two types of state can be generalized for SDN.

In the Internet, network state takes the form of routing entries in the node's routing table. An entry is identified by an IP address prefix and therefore applies to many hosts. In Ad hoc networks, routing entries may be created *on-demand* but they can still be used by many hosts (e.g. AODV protocol). Normally, in the Internet, the routing entries are created by a *proactive background process* of running intra and inter domain routing protocols such as OSPF, IS-IS and BGP.

When delivery is based on globally unique addresses in packets, there is no need to modify the packets (on IP layer) in the network nodes. In this case we talk about *forwarding*. Even in this case, packets are modified on L2 that uses locally significant addresses (e.g. 802.1). This L2 modification is based on topology rather than awareness of a user's connection. Also, this modification is hidden from transport and application protocols.

When a packet contains a locally significant identifier and a node modifies all packets using its state, we may talk about mapping, translation or switching. For example, in a similar case of MPLS, we talk openly about (label) switching.

Coming back to our discussion of lecture 1 of maintaining network or connection state, we can see that NAT state is created and maintained mainly by *implicit signaling*. The state is created by the usual message pattern of a client and while the client is active the traffic itself will keep the state alive. Only if a host behind a NAT needs to be reachable at all times or an application has long periods of inactivity but needs to be reachable, some explicit signaling for keeping the mapping alive is needed. This signaling is called *keep-alive signaling*.

NATs allow address reuse. A single host is not always active. Even when it is active, it does not use the whole ports space of 2^{16} arbitrarily. Rather, it usually makes use of just a few port numbers. Therefore, with a dynamic allocation of IP addresses and ports by the NAT, several hundreds of hosts (or even several thousands) can be served by a single public IP address. The factor of reuse depends on how active the hosts are.

Address saving was the original motivation for NATs. They do, however, have the side effect of *hiding the private network from the global Internet*. This is seen useful for protecting the local network from *unwanted traffic*. An unsolicited DOS attack to a host that only has a private address is not possible. The attack is blocked by the NAT.

This property of network hiding is particularly important for corporate networks and for mobile hosts. Neither operators nor users want to see a lot of unwanted traffic on the air interface in a wireless network. Even if we could tolerate the loss of air interface capacity, the unwanted traffic would lead to a highly active firewall in the mobile device and this would deplete the battery in no time at all.

We also note that in networks where denial of service is possible by flooding the target with traffic, the quality of legitimate service is unpredictable. No amount of traffic modeling will be able to give valid predictions of the quality.

Problems created by NATs to Voice Applications

Telephony is a service that requires that the targets of communication are reachable at all times. In this respect, telephony is inherently not a client-server but a peer-to-peer service. If the B-party or the callee is behind a NAT, the first signaling message cannot reach the party and calls to that party cannot be set up. Even if somehow, we solve the problem of signaling traversal through the NAT, media usually uses another UDP port and consequently media traversal is a separate problem.

Types of NATs

An example of the usual NAT behavior is presented in Figure 2.1. Internal (or stub) network is on the left and the public net on the right.

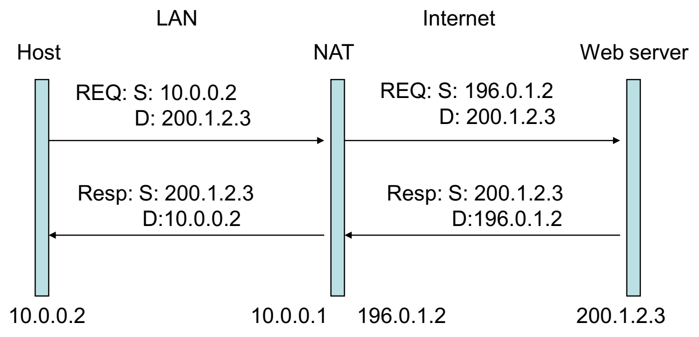
LAN
Internet
Host
NAT
Web server
REQ: S: 10.0.0.2 D: 200.1.2.3
REQ: S: 196.0.1.2 D: 200.1.2.3
Resp: S: 200.1.2.3 D:10.0.0.2
Resp: S: 200.1.2.3 D:196.0.1.2
10.0.0.2
10.0.0.1
196.0.1.2
200.1.2.3

Figure 2.1: Usual NAT behavior.

Figure 2.1 shows how for the first message of a flow the NAT maps the private source address (and port) to a NAT outbound address (and port) and creates a binding (storing it in the connection state) between the two. Based on the stored binding, the NAT is able to map the response message into the right destination address (and port).

For client-server applications such as Email or WWW this model works fine. We call application protocols that normally have no conflicts with NATs *NAT-friendly*. Similarly, application protocols that may have a conflict with a NAT are *NAT-unfriendly*. Another pair of a bit more generic terms is IP-agnostic and non-agnostic.

NATs may be *nested*. I.e. a client may actually have to traverse two or more NATs before getting its message to the public Internet. It means that actually, a NAT may be mapping between two private address spaces. This model is, for example, used by some broadband access operators. For simplicity, in this lesson, we ignore nested NATs and talk as if the mapping is between a private and a public address space. This simplification is valid because the internal NAT can be seen as invisible to the global communication.

NATs vary in terms of what kind of mapping they decide to create and in particular what users on the public side can use the created mapping. IETF made the first attempt to describe NAT behavior more than 10 years from the emergence of NATs using terms like symmetric NAT or full cone NAT. The description turned out to be not exact enough. This terminology was abandoned and a new terminology[3] was adopted. We will make use of it.

NAT behavior is classified in several dimensions. These are: *mapping behavior*, *address pooling behavior*, *port assignment* and *filtering behavior*.

Mapping behavior

NAT Mapping behavior defines what kind of mapping is created by the NAT upon the initial message from the internal network for which no state can be found. Figure 2.2 shows an *Endpoint independent mapping*.

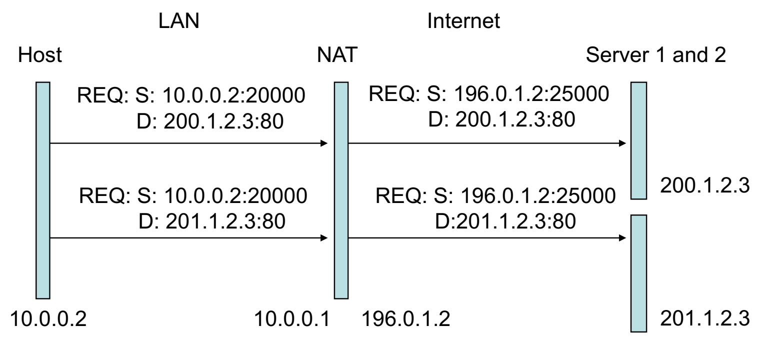
LAN
Internet
Host
NAT
Server 1 and 2
REQ: S: 10.0.0.2:20000 D: 200.1.2.3:80
REQ: S: 196.0.1.2:25000 D: 200.1.2.3:80
200.1.2.3
REQ: S: 10.0.0.2:20000 D: 201.1.2.3:80
REQ: S: 196.0.1.2:25000 D:201.1.2.3:80
10.0.0.2
10.0.0.1
196.0.1.2
201.1.2.3

Figure 2.2: Endpoint Independent Mapping in a NAT

The Figure shows how the client can use the same mapping to communicate with any number of independent endpoints in the public network. The weakness of this model is that any arbitrary host (e.g. an attacker) can try to reuse an active mapping of this kind.

The direct opposite is Address dependent mapping shown in Figure 2.3

[3] E.g. Network Address Translation (NAT) Behavioral Requirements for Unicast UDP (RFC 4787), 2007.

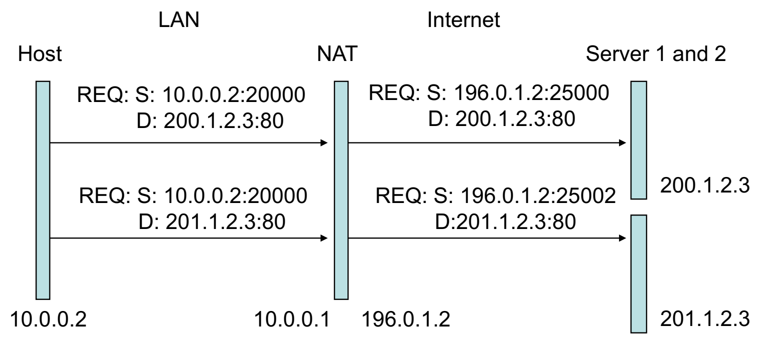
LAN
Internet
Host
NAT
Server 1 and 2
REQ: S: 10.0.0.2:20000 D: 200.1.2.3:80
REQ: S: 196.0.1.2:25000 D: 200.1.2.3:80
REQ: S: 10.0.0.2:20000 D: 201.1.2.3:80
REQ: S: 196.0.1.2:25002 D:201.1.2.3:80
200.1.2.3
10.0.0.2
10.0.0.1
196.0.1.2
201.1.2.3

Figure 2.3: Address dependent mapping

The Figure shows how the NAT allocates a new outbound source port for the communication each time the client initiates a communication to a new partner on the public network. This type of NAT will however let the client use the same NAT mapping to communicate with a single partner on the public net using any set of destination ports.

The strictest behavior in terms of the mapping is address and port dependent. This is shown in Figure 2.4.

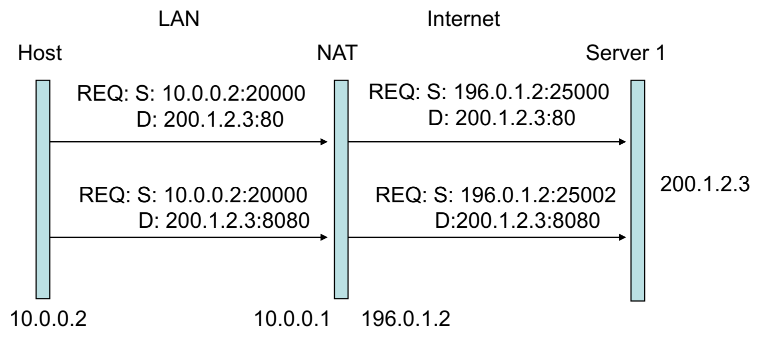
LAN
Internet
Host
NAT
Server 1
REQ: S: 10.0.0.2:20000 D: 200.1.2.3:80
REQ: S: 196.0.1.2:25000 D: 200.1.2.3:80
REQ: S: 10.0.0.2:20000 D: 200.1.2.3:8080
REQ: S: 196.0.1.2:25002 D:200.1.2.3:8080
200.1.2.3
10.0.0.2
10.0.0.1
196.0.1.2

Figure 2.4: Address and port dependent mapping.

The figure shows how a new mapping is created each time the client initiates communication to a public (address:port) pair. This type of NAT is safest for the client. It protects the client of any communication even from a different application residing on the destination host except the one with which the client has started communication. This is important additional protection: an attacker that has managed to lure an unsuspecting user to a site cannot easily probe and possibly break in through the other ports on the client's machine. Instead, a successful attacker must know a vulnerability in the particular application protocol (or its implementation) the client happens to be using.

In all the previous examples, the NAT makes use of a single outbound IP address. For a small stub network this may be sufficient. However, a larger stub network may use a NAT that owns several public IP addresses.

Address pooling behavior

The case when the NAT owns several public IP addresses opens several new options in NAT behavior. First, it may be that the NAT allocates any of the public outbound IP addresses for a new communication to a new destination (IP:port) -pair that it sees from a client. This is shown in Figure 2.5 and is called arbitrary pooling behavior.

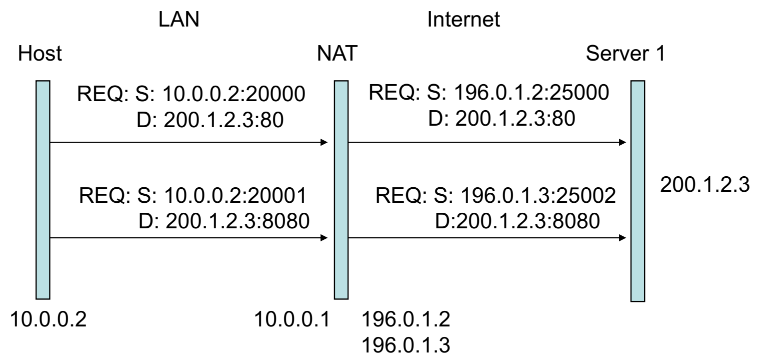
LAN
Internet
Host
NAT
Server 1
REQ: S: 10.0.0.2:20000 D: 200.1.2.3:80
REQ: S: 196.0.1.2:25000 D: 200.1.2.3:80
REQ: S: 10.0.0.2:20001 D: 200.1.2.3:8080
REQ: S: 196.0.1.3:25002 D:200.1.2.3:8080
200.1.2.3
10.0.0.2
10.0.0.1
196.0.1.2
196.0.1.3

Figure 2.5: Arbitrary pooling behavior.

Arbitrary pooling behavior is troublesome from the communicating partner's point of view. It is difficult for the partner to figure out whether it is communicating with a single host or several different hosts using different applications. Therefore, IETF recommends that the NAT should instead stick to a *paired behavior*. In this recommended behavior, the NAT maps all simultaneous communication flows using any set of source ports from a single client to the same outbound public IP address. For each new source port, the NAT must allocate a new outbound source port in order to be able to route the responses back correctly.

Port Assignment

A NAT that owns many IP addresses has the option of trying to *preserve the source port* number and allocate a new public IP address to each new active client.

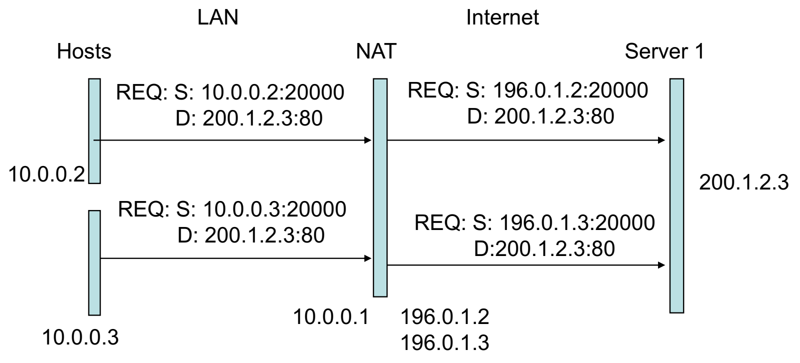
LAN
Internet
Hosts
NAT
Server 1
REQ: S: 10.0.0.2:20000 D: 200.1.2.3:80
REQ: S: 196.0.1.2:20000 D: 200.1.2.3:80
10.0.0.2
200.1.2.3
REQ: S: 10.0.0.3:20000 D: 200.1.2.3:80
REQ: S: 196.0.1.3:20000 D:200.1.2.3:80
10.0.0.1
196.0.1.2
196.0.1.3
10.0.0.3

Figure 2.6: Port preserving in NAT

This does not scale very well because of the shortage to public IP addresses. It helps in terms of the address shortage only to the extent there are non-active hosts in the internal network. Therefore, such a NAT may abandon the port preserving behavior once all public outbound addresses are in use.

Instead of abandoning the port preservation policy, the NAT could choose to use *port overloading*. This is shown in Figure 2.7.

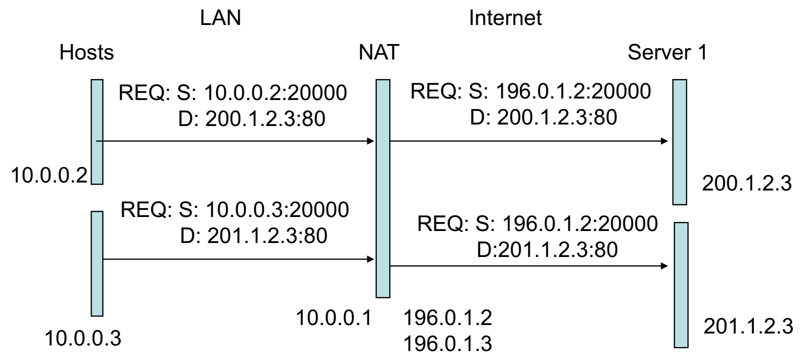

Figure 2.7: Port overloading: not recommended!

In port overloading, several communications from different clients seem to be coming from the same address and port. The responses can still be correctly routed back to the right client using the source address in the response message. In the Internet all routing is normally based on the destination address although for exceptional reasons, the source address may also be used (e.g. for load sharing). For NATs this works but is not recommended.

Filtering behavior

The question of which external partner can make use of the mapping is classified separately by IETF. In practice, the

- endpoint independent mapping gives raise to endpoint independent filtering
- address dependent mapping gives raise to address dependent filtering
- address and port dependent mapping gives raise to address and port dependent filtering.

In addition, sometimes it is important that two hosts in the same internal network can communicate using public IP addresses assigned to them for the communication by the NAT. This is called *hairpinning*.

The most common type of NAT used in practice is one with *Address and port dependent mapping and filtering* (APDMF). It provides the best protection of the internal network and is the most difficult to traverse.

Need for optimization of NAT traversal

A NAT mapping, that is endpoint independent, can be reused by several remote hosts. For example, such hosts might be running a signaling application and a media application and reside in completely different computer systems. Even an address dependent (but not port dependent) mapping could be reused for signaling and for media provided that both originate in the same remote IP address. However, if the NAT mapping happens to be address and port dependent, under all circumstances signaling and media will have to use two different mappings.

It follows that either an application has to assume the strictest NAT behavior always or be aware of the NAT type. The downside of assuming the strictest NAT behavior (although the NAT might be less strict) is cost. For reaching a callee, a VOIP client would have always to route all its packets through a *special server*[4] with a public IP address that would then tunnel the packets to the callee residing in a private address space. Also, the callee would have to continuously keep its mapping alive in the NAT and the server. This was seen as wasteful both in terms of address space and computing power when the BEHAVE group was active. Also, when the callee has a battery-powered device, this puts an additional drain on the battery.

Based on this reasoning, IETF has developed the UNSAF or the Unilaterial Self Address Fixing architecture. UNSAF lets an application snoop on the NAT behavior and choose to fix its identity expressed as an IP address that it shows to a remote party (on the application layer) so that the NAT behind which the application runs can be traversed.

Since it is assumed that NATs may change a mapping dynamically (after all it is assumed that a NAT is invisible), there are no guarantees that the snooping result will stay fixed in the NAT for the time the application is being used. The application is forced to keep snooping and fix its identity once more if it sees that the NAT has changed the mapping dynamically.

This makes NAT traversal a *responsibility for an application*. The target of UNSAF is to find out what kind of mapping and filtering behavior is used in the NAT and choose the least expensive way of traversing the NAT.

At the hearth of the idea of self-address fixing is that NAT and the host are at cross-purposes. NAT tries to be invisible, while host uses lots of efforts to find out what the NAT is doing, how it is behaving and thus avoid the limitations set by the NAT.

With hindsight, when we now look at the most common Internet meeting applications, like Teams, Zoom etc, they all run in the cloud and always in a session one or more cloud servers are involved. Also, a modern data center grade computer is able to run packet processing at 200Gbps, which would be sufficient for something like 100 000 to 200 000 users. So, while the heavy-duty data transfers can be seen as client server, the meeting

[4] Nowadays, we are moving data plane processing apps also to the cloud. Such apps run on generic data center hardware. One can e.g. take data center grade Linux computer with multiple CPU-cores and multiple NIC cards and process a packet flow of even at 200 Gbps on a single computer. If one session for voice and video uses e.g. 1 Mbps, we could run more than 100 000 simultaneous sessions thru a single computer

applications do not actually need huge capacity. Therefore, it has been a commercially valid choice to go for a simple approach to NAT traversal in many application scenarios.

STUN a toolbox for NAT traversal

STUN stands for *Session Traversal Utilities for NAT* and has been specified in RFC 5389. It specifies an extensible packet format, defines operation over several transport protocols and provides for two forms of authentication. The latter is needed because it should be ensured that outbound IP addresses allocated to one user are not revealed to other users. The transport protocols include UDP and TCP.

STUN is intended to be used in particular contexts. These are called STUN usages. Each usage describes how STUN is used to achieve the desired NAT traversal for the particular application.

It may be necessary to multiplex STUN with another application protocol using a single source port on a host.

The first attempt of STUN specification was Simple Traversal of UDP through NAT (RFC 3489). It did not work reliably and is obsoleted by the current RFC 5389.

Figure 2.8 shows the STUN in operation.

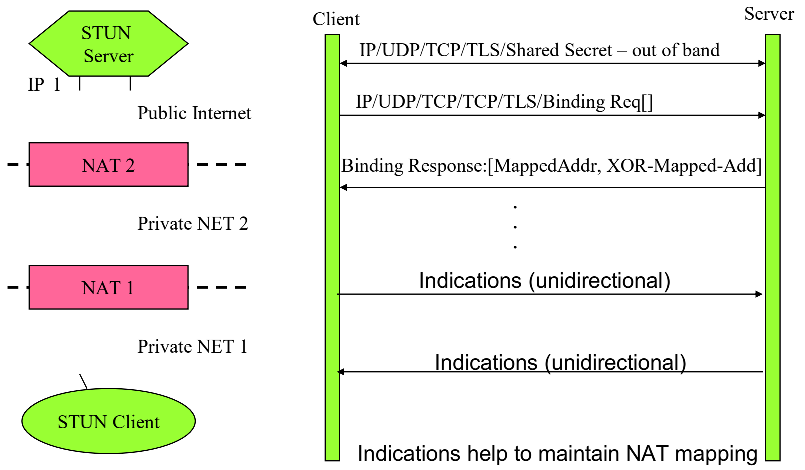
STUN Server
IP 1
Public Internet
NAT 2
Private NET 2
NAT 1
Private NET 1
STUN Client
Client
Server
IP/UDP/TCP/TLS/Shared Secret – out of band
IP/UDP/TCP/TCP/TLS/Binding Req[]
Binding Response:[MappedAddr, XOR-Mapped-Add]
Indications (unidirectional)
Indications (unidirectional)
Indications help to maintain NAT mapping

Figure 2.8: STUN protocol model and operation.

STUN assumes that a client residing behind several NAT devices wishes to fix its identity that it shows to remote hosts in the public network. In addition to the client, the model assumes that in the public net there is a special STUN server that will respond to STUN requests. So, STUN is a client-server protocol. Naturally, a STUN server can be integrated with other network servers/entities in real products.

It is assumed that the STUN server may have several IP addresses and that it may respond on several ports.

The 2 STUN agents (client and server) form a security association. The base protocol defines one request, called the *Binding request* by which the client can ask the server to let it know, how the server sees the client's public address that was allocated for the binding message by the outermost NAT. The allocated address is returned in the *Mapped Address information element* in the STUN response. Naturally, the response itself is routed through the NATs like any other response in any other client server communication. It is assumed that a complete usage of STUN may define other requests and responses.

In addition to the requests, a STUN agent can send an Indication that does not require the other agent to respond. An indication can for example be used to maintain the bindings in the traversed NATs or the possible state in the receiving agent.

An extension to STUN: Relay operation

For address and port dependent mapping NATs, STUN can snoop the type of the NAT but each application is forced to maintain its own mapping in the NAT. For example, a VOIP client must maintain one mapping for signaling and another for each media port it uses. This does not scale nicely for example to mobile hosts because of the extra battery drain.

Also, for the case of address and port dependent mapping, since only a given destination can send messages on a single port to the host protected by the NAT, that host can be reachable for telephony only through a single point in the global network. That single point needs to be addressable.

An extension to STUN has been specified for Traversal Using Relays around NAT (TURN). The idea is to allocate a public IP address independent of the NAT itself to the client on-demand.

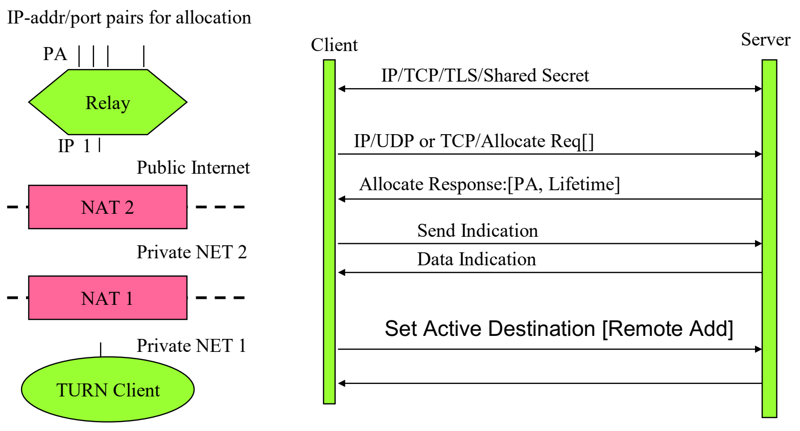

Figure 2.9: TURN operation

TURN defines a request and response for address allocation. Send and Data Indications are used to tunnel the application level packets through the NAT. By the Set Active Destination, the client can ask the TURN server to drop the tunnel mode data transfer and move to switching mode data transfer with a single destination.

NAT traversal solutions: SIP Outbound

A proposed standard, RFC 5626: Managing Client-Initiated Connections in the Session Initiation Protocol (SIP) defines a solution for NAT traversal for the SIP signaling protocol.

SIP requires that a User Agent (UA) must register in a Registrar that resides in the public network. SIP-Outbound co-locates a SIP proxy and a STUN server with the Registrar. SIP-Outbound explains how the resulting connection through a NAT or NAT binding (e.g. an address and port dependent NAT) can be later kept alive by regular binding requests using STUN and how the binding can be later reused for the delivery of inbound SIP messages for example for session setup.

The SIP-Outbound explains that for the sake of reliable connectivity, the User Agent may decide to keep alive several simultaneous bindings in the NAT. The RFC explains how to recover a failed binding. The timeout proposed for keep-alive messages either using STUN binding requests or TCP CRLF messages is 24 to 29 seconds. Unfortunately, this scales poorly for battery powered user agents. It does not seem like much to require a mobile to send a simple enough message once in about 25 seconds but unfortunately it is likely to be enough ***to deplete the battery in a few hours (at least on some brands of mobile devices this was the case until the device vendors invented their own tricks to reduce the power drain).*** The situation becomes worse if there are several applications on a battery powered device that wish to stay reachable for an extended period or by-default always.

The advantage of SIP-outbound compared to some other methods of NAT traversal is simplicity. This simplicity is based on the fact that SIP requires a registration of User Agents (UA) in a Registrar and the Registrar is centralized for each UA. In Peer-to-Peer SIP the registrar is distributed and thus SIP-outbound is not sufficient.

NAT traversal solutions: ICE

The recommended solution for *media traversal* through NATs is described by the MMUSIC WG on some 120 pages: Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols (http://tools.ietf.org/html/rfc5245).

In this case STUN needs to be multiplexed (i.e. they use the same IP-port) with the Real time protocol (RTP).

ICE can also be used for signaling in the case of Peer-to-peer SIP.

ICE collects several candidate IP addresses for the destination User Agent. These include the UA's own IP address, the outermost outbound NAT address (called the reflexive address) and a TURN address. ICE runs through a number of connectivity checks using STUN and determines the least expensive IP address that it can use for media. This is then communicated to the remote end using SIP/SDP.

Due to the erratic behavior of NATs, in ICE implementation repeats each query to a STUN/TURN server up-to 7 times. ICE must wait for the query timeout to expire to figure out that some address does not work). The queries are paced like the corresponding application paces its messages (e.g. for RTP one message can be sent about each 20ms). As a result, one full ICE round in case of address and port dependent mapping and filtering takes up-to 100 messages and creates significant delay in session establishment. Now, there has been some work in IETF trying to optimize ICE from this original set of ideas.

Jouni Mäenpää showed in his doctoral thesis[5] that for the case of mobile devices and APDMF NATs and P2PSIP, the session setup delay can be up-to about 30s with the required 2 rounds of ICE (one for signaling and the second for media). He showed that by applying several optimizations it is possible to squeeze the delay to about 10s. Since the ITU-T requirements for session setup are 2s for national calls and 8s for international calls, the result is not quite satisfactory in terms of delay.

Other NAT traversal solutions

Other NAT traversal solutions have been developed. For example, we already mentioned that it is possible to configure the NAT to always send messages addressed to a certain port to a particular IP address in the internal network. This is called *port forwarding*. The configuration in the NAT is static and a question remains, (a) how do we manage the allocation of ports to hosts in the stub network. Another question is (b) how does the remote party know that a particular host in a private address space with which it wants to communicate with e.g. ssh is not in port 443 that is the default port for ssh but in e.g port 36745?

It would be natural to answer the above questions: (a) use the dynamic host configuration protocol (DHCP) to obtain a port and (b) use SRV records in DNS to communicate the port to the client. Both require changes in host behavior. They also require that NAT administrator would *trust* the hosts to do the right thing. This approach is not popular.

Some of the solutions set additional requirements to DNS that may have to notify the NAT about an inbound new packet flow.

Also, researchers have proposed more generic solutions for NAT traversal such as TRIAD. TRIAD proposes an additional protocol layer between IP and Transport layer

[5] http://lib.tkk.fi/Diss/2013/isbn9789526051215/isbn9789526051215.pdf

and uses source routing to traverse any number of address realm boundaries on the end-to-end path.

None of the well-known solutions, recommended or hacked is perfect.

Customer Edge Switching and Realm Gateway

A generic solution is also proposed by Comnet (among them the author of these notes) in www.re2ee.org. You will find a demonstrator of the technology in GitHub/Aalto5G. The solution is called customer edge switching that proposes to replace NATs by devices called Customer Edge Switches (CES). The CES approach introduces host/service/ identities in the edge node. In practice, in our CES implementation, the host IDs are domain names. For legacy interworking we propose the Realm Gateway. For CES-to-CES signaling we use the Customer Edge Traversal Protocol (CETP) that sets up tunnels for the user packets over the core network and to do so carries control signaling edge to edge. The control signaling allows the inbound CES to make an informed decision on flow admission. Due to this property, we can say that a CES is a co-operative Firewall.

The Realm Gateway (RGW) among other things contains a DNS leaf node. The first traversal method supported by RGW works like this: upon a DNS address query, it reserves an outbound address for a short period of time (we have used 2s timeout to guard the reservation) for the client and returns this address in the DNS response. Upon the first message of the flow itself the reservation is released and the address can be used for a fresh flow arrival while the current flow is using it as well. To make this possible, the current flow has a binding state, just like NAT has a binding state. As a result, a host with a fully qualified domain name (FQDN) becomes reachable on the Internet even if it only has a private address and sits behind a RGW without the cumbersome NAT traversal techniques recommended by IETF. This method of NAT traversal is suitable for applications that follow the pattern: DNS-query followed by one data flow and no IP addresses carried blindly on Application layer.

It immediately appears that RGW is vulnerable to many kinds of attacks. We have developed many heuristic algorithms to tackle such attacks. In case that the RGW is a carried grade device in an operator network with many outbound addresses, the policies and heuristics that can be applied to tackle attacks are more diverse than in case of a single outbound address device. Single address devices suffer from low efficiency of address reuse, while for devices with more than 3 outbound addresses, efficiency is high. RGW can also translate between IPv4 and IPv6 and thus it can be used as a transition mechanism to IPv6.

RGW supports a second traversal method that is suitable e.g. for HTTP and HTTPS versions that have a plain text header element called SSI, essentially a domain name of the destination in their first message of the flow. This method is called Application Layer Gateway and it gleans the domain name from the first suitable HTTPS or TLS packet and creates a destination NAT binding based on that. The solution uses a protocol grammar definition to describe the lexical analysis and parsing needed to find the domain name in TLS or HTTPS etc., so the solution can be declaratively expanded to new protocols without the need to write by hand a new lexer-parser for the protocol.

In case that both communicating hosts are behind CES devices, the CES devices communicate using the CETP protocol that allows the inbound CES to make an informed decision on flow admission. The CETP sets up tunnels for the the data plane packets edge to edge using an SDN controller and OpenFlow. With the two components (CETP and RGW) widely adopted, the wide area communication can be based on globally unique names and globally or locally significant identities and addresses. Globally unique addresses can continue to be used for heavy-duty servers. At the same time, it will be sufficient for any normal end-host owned/used by a user to have just a private address.

Global communication is not just a matter of packet delivery across the address boundaries. It also raises the question of trust. Trust has many definitions in the literature. For the purpose of communications, we can *define trust as the willingness of the receiver to accept a risk when communicating with a remote party*. CES proposes to manage all communication admission by policy. All aspects of the CETP protocol are controlled by policy.

We see CES devices as perfect tools to collect evidence of behavior. To secure global communication we have proposed that an Internet wide or alliance wide trust management system should be created. Its task would be collecting all that evidence, aggregating it and assigning trust/reputation values for each host, customer network and application. In addition, it should assign a second trust value for the evidence sending behavior of the hosts and customer networks (credibility of evidence given).

CES gives the benefits of being able to detect and eliminate source address spoofing. It gives the benefit of being able to block DDoS and isolate the hosts that are used to carry out the attack. CES favors application protocols that are IP-agnostic; these can traverse a CES node without difficulty. Non-IP-agnostic protocols can keep using UNSAF, ICEs etc[6].

Conclusion

NAT traversal has turned out to be a hard issue for the IETF. In my opinion the core problem is ideology or politics. IETF has not been quick to adopt NATs as part of the Internet architecture. Instead, it has chosen to avoid attacking the problem directly and created poorly scalable problem avoidance solutions. The recommended solution, UNSAF makes NAT traversal a matter for the applications. This in itself is a hack. UNSAF does not scale well to battery-powered devices with private only addresses.

[6] E.g. Session Initiation Protocol (SIP) was designed by IETF from late 1990's over a period of more than 10 years. When the work started, NATs were widely in use. However, the design unfortunately made believe that NATs do not exists or that they should be taken out from the IP networks, when it would have been rather simple to adopt a different attitude and avoid NATs becoming a problem that then needed to be solved with great efforts.

Better solutions are needed and undoubtedly will be developed in the future. For example, we have proposed a solution titled "Customer Edge Switching" that supports smooth global communication between devices that reside in different private address spaces. In this solution, a host has a private, locally significant address, one or more separate locally significant identity and one or more globally unique names. The result would seem to be that for example IPv6 is not needed at all although its introduction will become technically easier due to CES and the RGW. This is because CES tunnels traffic edge to edge and the RGW translates between IPv4/IPv6 thus providing technology isolation between networks. CES is still a research prototype rather than something that people could just pick off the shelf and use.

Another example is the SCION architecture that uses addresses that are triples: ISD:AS:A where the address (A) is only significant within one Autonomous System.

The challenge is how to get the global community to agree on any solution and how to deploy it widely. In our work on CES and trust management, we have argued that for protocol and solution deployability, costs and benefits should be perfectly aligned: the stakeholder making an investment decision into a technology should benefit irrespective of what the other players are doing. We have argued that an ideal solution can be deployed one network at a time and each adopter should benefit. CES/RGW can be adopted one network at a time; no changes in hosts are mandatory.