OpenSER Admin Course

Voice System SRL

http://www.voice-system.ro

http://www.openser.org

NAT Traversal

NAT is so popular because of

- public IP shortage
- security issues
- network design issues

In Europe, more than 80% of the Internet users are behind NAT

Problem: VoIP does not work over NATs without extra work.

There are many scenarios for which no single solution exists. Solutions include: STUN, ALGs, symmetric communication, media relay, ...

Where NATs affect SIP

INVITE sip:UserB@there.com SIP/2.0
Via: SIP/2.0/UDP 192.168.99.1:5060
From: BigGuy <sip:UserA@here.com>
To: LittleGuy <sip:UserB@there.com>
Call-ID: 12345600@here.com
CSeq: 1 INVITE
Subject: Happy Christmas
Contact: BigGuy <sip:UserA@192.168.99.1>
Content-Type: application/sdp
Content-Length: 147
v=0
o=UserA 2890844526 2890844526 IN IP4 here.com
s=Session SDP
c=IN IP4 100.101.102.103
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000

- Contact
- Route
- Record-Route
- Via header fields (received tag)
- SDP payload

- Application Layer Gateways (ALGs) – built-in application awareness in NATs.
  - Requires ownership of specialized software/hardware and takes app-expertise from router vendors.
  - in real life this type of devices break thinks even more due por implementation

- Geeks' choice: Manual configuration of NAT translations
  - Requires ability of NATs, phones, and humans to configure static NAT translation. (Some have it.) If a phone has no SIP/NAT configuration support, an address-translator can be used.
  - it is not a real life solution

- STUN (RFC 3489): Alignment of phones to NATs
  - Requires NAT-probing ability (STUN support) in end-devices and a simple STUN server. Implementations exist.
  - Does not work over NATs implemented as “symmetric”.
  - Troubles if other party in other routing realm than STUN server.
  - Works even if NAT device not under user’s control.
- Relay: Each party maintains client-server communication
  - Introduces a single point of failure; media relay subject to serious scalability and reliability issues
  - Works over most NATs
- Symmetric clients (RFC3581 for signaling, symmetric media), comedia support

NAT Practices - Conclusions

	ALG	STUN	UPnP	Manual	Relay
Works over ISP's NATs?	N/A	Ltd. (*)	N/A	N/A	Maybe
Symmetric NATs?	N/A	No	N/A	ok	Ltd.
Phone support needed?	No	Yes	Yes	Yes	Yes
NAT support needed?	Yes	Ltd. (*)	Yes	Ltd. (+)	No
Scalability	? (o)	Ok	Ok	Ok	poor ☒
User Effort	Small	Small	Small	Big ☒	Small

*… does not work for symmetric NATs

+ … port translation must be configurable

o … application-awareness affects scalability

- There is no "one size fits it all" solution. All current practices suffer from many limitations.
- **Voice System**'s observations for residential users behind NATs: Affordability wins: SIP-aware users relying on public SIP server use ALGs or STUN.

- Our solution for operation on the public Internet:
  - Let as many phones as possible handle NAT traversal autonomously using STUN (client side)
  - Detect cases which cannot be handled autonomously.
  - If "hard NATs" detected, ignore SIP and help out with RTP (server side)

- the STUN client from the UAC must be able to detect symmetric NATs and not to try to cross them as it is not possible.
- the STUN client is also responsible for keeping the NAT bind open for incoming SIP and RTP traffic
- for the best external IP&port detection, the STUN server must reside on the same IP as the SIP proxy.
- for the same reason, the STUN server requires two public IPs.

- A complete NAT traversal solution includes:
  - manipulation of the signaling traffic (this is done directly from the OpenSER script)
  - media relay (this is done with the help of an external application)

- The NAT traversal may be implemented in two ways:
  - built in the main proxy configuration
    - (+) very compact solution – efficient and scalable as load
    - (-) it increases the complexity of the main proxy cfg.
  - via a specialized SBC in front of the main proxy.
    - (+) detaches the NAT logic and simplifies the proxy cfg
    - (+) is suitable for geographical distribution with one main proxy
    - (-) introduced some overhead as an extra SIP hop
    - (-) not so efficient as data become redundant between SBC and proxy

Solutions – OpenSER module

- OpenSER provides two similar NAT handling module:
  - nathelper
  - mediaproxy
- they offer both functions
  - to mangle the SIP part (headers and SDP)
  - to communicate and use an RTP relay.
- “nathelper” module
  - works with RTPproxy – a C written fast media relay
- “mediaproxy” module
  - works with MediaProxy – a Phyton written media relay

- NAT detection
  - the detection is done for initial SIP requests; sequential requests will rely on the this initial detection.
  - detects if the sender of the request is behind NAT; can be based on :
    - Contact URI, IP part
    - VIA address/port versus received address/port
    - SDP IP class
- reply handling
  - add to the VIA header (of the previous hop) the received IP and port – the source where the request was received from
  - be sure to route the replies back to the same IP and port where the request was received from (done automatically)
- use the same local socket (IP and port) when talking to a natted client (automatically done)

Solution – Algorithm - REGISTER

- detects if the sender of the REGISTER is behind NAT

- save in user location:
  - the received contact – SIP clients expect to see the contact they published;
  - the source IP and port – you need them to be able to cross the NAT;
  - a flag to mark if the contact is NATed or not.

- when sending a request back to the UAC, the received contact URI will be used as RURI and the received source IP and port will be set as outbound proxy.

- After registration, the proxy must be sure that the NAT bind will remain open and it will be able to deliver SIP requests to the registered contact
- solution – perform NAT ping – periodically traffic sent to the SIP UAC with the only purpose of preventing the NAT bind to close.
- types on ping:
  - UDP ping – use a dummy UDP package as probe
    - very simple to generate, but it generates only inbound traffic
  - SIP ping – use the OPTIONS SIP requests as probe
    - more complex to generate, but it triggers a reply from the UAC, so we get bidirectional traffic

REQUEST processing

- detect if the caller is behind NAT
  - if yes, mangle the contact URI and mark it as nated (add a special URI parameter as marker).
- resolve (via the proxy routing logic) the destination
  - via the user location lookup, the destination may be also natted or not
- before forwarding the INVITE, if at least one of the ends was detected as natted, enable to usage of RTPproxy as media relay – this will replace the private IP&port from SDP with the public IP&port of the RTPproxy.

Solution – Algorithm – INVITE (II)

REPLY processing

- if the INVITE is negatively replied, disable RTPproxy

- if the INVITE is 200 OK replied, confirm RTPproxy in order to update the SDP IP and port with RTPproxy coordinates

- if destination was previously detected as natted (at request time), mangle the contact URI and mark it as nated (also add a special URI parameter as marker).

Solution – Algorithm – reINVITE

- NAT traversal algorithm for re-INVITEs is highly similar to the one for INVITEs.

- The only difference is that the NAT detection (for both source and destination) is not performed anymore, but the markers from the contact URIs are used.

- Be sure and re-enable RTPproxy as re-INVITEs usually force new SDP information.

Solution – Algorithm – non-INVITE

- NAT traversal algorithm for non-INVITEs is the same as for INVITEs.
- The difference is that there are no media concerns, so no RTPproxy will be enabled.

- For non-INVITE requests that establish dialogs/sessions, the same guidance lines as for re-INVITEs are to be used.
- Again, no media is to be used.

- nathelper module
- prototype: nat_uac_test(flags)
  - flags parameter specifies the NAT tests to be done
  - return true if one of the tests succeeds
- NAT tests
  - Contact header field is searched for occurrence of RFC1918 addresses.
  - the "received" test: address in Via is compared against source IP address of signaling
  - top most VIA header is searched for occurrence of RFC1918 addresses
  - SDP is searched for occurrence of RFC1918 addresses
  - test if the source port is different from the port in Via

- If SDP is negotiated via 200 OK and ACK (instead of INVITE 200 OK), the same algorithm is used, but the RTPproxy must be enabled and re-enabled for 200 OK and ACK.
- To allow chaining of several RTP relays (on the media path), check if the SDP IP is public or not.
- If public, instruct RTPproxy to start using that IP for sending the media traffic – doing so, you avoid dead-locks between two RTPproxies that waits one after the other in order to discover the source of media.

- when source is not behind the nat and it is a parallel forking
  - you can enable RTP relaying only for NATted destinations using onbranch_route
- for serial forking, the failure_route can be used to disable RTP relaying if the new destination is on public IP. You should enable RTP relaying after creating the transaction, to avoid the changes in SDP for new branches
- if calls go to media servers in the public network, you can rely on comedia support present in most of such entities (asterisk, cisco)
- same for PSTN gateways
- you can use configuration file tricks to detect when the source and destination are behind same NAT
  - compare source IP with destination URI domain after the lookup("location")
  - potential risk when dealing with multiple levels of NAT

walk through openser nat traversal configuration file

BREAK