Demystifying SIP NAT Traversal

Alex Balashov
Evariste Systems LLC
abalashov@evaristesys.com
(678) 954-0670

28 July 2009
What's NAT?

- **Network Address Translation**
  - Allows hosts on non-routable, private IP networks to interact with other IP networks, or the rest of the “public” Internet.
  - Requires special gateway aware of “connections” (not necessarily in TCP sense) to rewrite packets traversing between private and outside network.
  - Being aware of such connections requires packet header inspection and insertion of data about the endpoints and their present orientation into memory – keeping “state.” NAT gateways are “stateful.”
  - RFC1918 defines private subnets:
    - 192.168.0.0/16
    - 172.16.0.0/12
    - 10.0.0.0/8
What's NAT? (cont'd)

- AKA:
  - “IP masquerading”
  - “Network masquerading”
  - “Internet connection sharing”

  - Most common application of NAT familiar to pedestrians.
  - Ubiquitous.
  - Example: Allows sharing of home broadband connection among several computers – share one public IP address from many private addresses.
  - Done automatically by ISP customer premise equipment or home “broadband routers” (D-Link, Netgear, Linksys, etc.) - generally taken for granted.
What's NAT? (cont'd)

- Two main categories of NAT:
  - Source NAT
    - Private endpoint initiates connection to outside, NAT gateway rewrites request and statefully rewrites replies from far end.
    - IP masquerading
    - Internet Connection Sharing
    - Clients behind NAT – what we care about in this discussion
  - Destination NAT
    - Outside (public) endpoint connects to private host through NAT gateway – AKA server is behind NAT.
    - For SIP this is generally a VERY BAD IDEA, except one-to-one NAT. It's still a bad idea, just less so.
    - Don't do that.
What's NAT? (cont'd)

- Many different NAT schemes and nuances:
  - Full cone NAT
  - Restricted cone NAT
  - Symmetric NAT.
  - Etc.
- For SIP, for the most part – who cares? NAT is NAT.
  - All NATs have uniting characteristic.
  - Endpoint that is being addressed is not the “real” endpoint.
  - Scheme is largely irrelevant.
  - Exception: For SIP server to be behind destination NAT, NAT must be one-to-one.
So?

- We are talking about SIP server on public IP ↔ clients behind NAT.
  - Example: Phones on a private home LAN registered to a private SIP provider.
Why is SIP+NAT so hard?

- SIP architecture → separate signaling / bearer plane
  - AKA: SIP does not actually carry the media (voice, video, etc.)
  - SIP sets up RTP sessions on other ports to carry media
  - So, NAT gateway has to be aware of more than just SIP message endpoints → to be aware of full state of SIP “call”
  - By default, NAT gateways are dumb – not aware
  - Not like NAT'ing those “simple” protocols: HTTP, SMTP, ICMP, POP, IMAP
  - More like NAT'ing Quake, FTP, MSN Messenger video, H.323, etc.
    - Linux folks know “smart” conntrack modules exist for these
    - For a reason – straight NAT logic alone won't work
Why is SIP+NAT so hard?

- SIP uses UDP as a transport
  - TCP is supported, but not often used – too much delay/overhead/PITA
  - Cookie didn't crumble that way
  - NAT gateways do TCP statekeeping fine:
    - It's how they're used by most applications
    - TCP → built-in keepalive type messages
    - TCP → is inherently stateful – that's the point
  - UDP is connectionless & stateless
    - Fire and forget packets
    - “Datagram-oriented,” not “connection-oriented” sockets
    - But UDP connections “interactions” still have to be NAT'd
      - NAT devices vary in ability to do this well
      - Not as straightforward as TCP
Why is SIP+NAT so hard?

- SIP is a leaky abstraction!
  - From point of view of OSI model / protocol stack
SIP = leaky abstraction?

- Abstractions supposed to isolate other layers (more and less abstract) from any knowledge of their internals
  - Cleanly
  - Like well-formed object-oriented design in programming
- SIP's design is 100% opposite that – not at all isolated!
How non-leaky protocols work

- Normal OSI layer protocols minding their own business:

```
<table>
<thead>
<tr>
<th>OSI Model</th>
<th>data unit</th>
<th>layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>application</td>
<td>Network Process to Application</td>
</tr>
<tr>
<td>data</td>
<td>presentation</td>
<td>Data Representation &amp; Encryption</td>
</tr>
<tr>
<td>data</td>
<td>session</td>
<td>Interhost Communication</td>
</tr>
<tr>
<td>segments</td>
<td>transport</td>
<td>End-to-End Connections and Reliability</td>
</tr>
<tr>
<td>packets</td>
<td>network</td>
<td>Path Determination &amp; Logical Addressing (IP)</td>
</tr>
<tr>
<td>frames</td>
<td>data link</td>
<td>Physical Addressing (MAC &amp; LLC)</td>
</tr>
<tr>
<td>bits</td>
<td>physical</td>
<td>Media, Signal and Binary Transmission</td>
</tr>
</tbody>
</table>
```

← self-contained
← self-contained
← self-contained
← Some other, SEPARATE protocols here and there to create links between these layers
← self-contained
← self-contained
← self-contained
← self-contained
How non-leaky protocols work

• Example:
  – Ethernet

• MAC layer/Link Layer/Layer 2 only know how to get Ethernet frames between Ethernet stations
• Only by MAC address!
• IP (Layer 3) only knows how to get IP packets to IP endpoints
• IP to MAC resolution required to keep abstraction clean
  – That's what ARP (Address Resolution Protocol) is for
But SIP is “special” ...

- SIP protocol semantics (directly) operate on:

<table>
<thead>
<tr>
<th>OSI Model</th>
<th>data unit</th>
<th>layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>application</td>
<td>Network Process to Application</td>
</tr>
<tr>
<td>data</td>
<td>presentation</td>
<td>Data Representation &amp; Encryption</td>
</tr>
<tr>
<td>data</td>
<td>session</td>
<td>Interhost Communication</td>
</tr>
<tr>
<td>segments</td>
<td>transport</td>
<td>End-to-End Connections and Reliability</td>
</tr>
<tr>
<td>packets</td>
<td>network</td>
<td>Path Determination &amp; Logical Addressing (IP)</td>
</tr>
<tr>
<td>frames</td>
<td>data link</td>
<td>Physical Addressing (MAC &amp; LLC)</td>
</tr>
<tr>
<td>bits</td>
<td>physical</td>
<td>Media, Signal and Binary Transmission</td>
</tr>
</tbody>
</table>

← Session
← Transport (TCP & UDP – for “ports”)
← Network (IP)

Involved in almost as much as Federal government
But SIP is “special” ...

• AKA:
  – Endpoint reachability information in SIP is represented in Layer 3 & Layer 4 (network & transport) terms
  – No intermediate protocol (except DNS) for resolution – the information is literal, not encapsulated
  – SIP URIs don't refer to SIP reachability “entities,” but to literal network and transport layer information!
  – Not just URIs: Via headers, SDP, etc. all guilty
But SIP is “special” ...

• Maybe SIP is not only protocol that works that way.
  – HTTP URLs go to IP addresses & domains too?
  – That's not exactly the same thing, though, because:
    • HTTP server decides how to provide the “host” requested – Host: header or dedicated IP address
    • That's how name-based and IP-based virtual hosts work
  
• But in SIP:
  – INVITE sip:6789540670@192.168.5.3:5060 SIP/2.0
  – … means EXACTLY that (on L3 & L4)
But SIP is “special” ...

Why was SIP made that way?
But SIP is “special” ...

Why was SIP made that way?

- Hard for me to speculate
- Simplicity?
  - But this approach adds a lot of complexity!
  - SIP is simple!?!?!?

  - More than 100 RFCs other than RFC 3261 commonly relevant in comprehensive implementation of SIP stacks
  - SIP is many things – but not simple!
But SIP is “special” ...

Why was SIP made that way?

- Complete lack of accounting for NAT?
  - On which much of the Internet depends
  - That wasn't necessarily the case when SIP was conceived
  - Lots of backward compatibility requirements subsequently
But SIP is “special” ...

IETF SIP Working Group???

Opinions vary :-) -- but I don't think that's a plausible explanation
But SIP is “special” ...  

Why was SIP made that way?

- More likely, the leak is deliberate.
- SIP is an “IP-oriented” protocol – comes from IP heritage, not telephony, etc.
- It's the “IP protocol.” It's for setting up all kinds of sessions over IP.
- So why shouldn't its semantics be … “IP-affirming?”
- What better way to do that than to actually use IP (and Transport layer) within it?
But SIP is “special” ...

- SIP and IP are ontologically married
  - Let's grant that.
  - But this presents us with some inconveniences and problems
  - When IP routing techniques are in use of which SIP is not intrinsically aware
  - Like NAT!
I thought you were a “solutions” expert or whatnot...

- Nice job rambling about the problem
- PLZ FIX K THX
Solutions we aren't talking about

- **STUN**
  - Old
  - Too hard to deal with
  - Not all equipment supports
  - Not feasible for service providers to force users to use
  - Annoying
- Anything else like STUN
  - No shortage of near-end NAT traversal schemes
  - They all have the same flaw –
    - too hard
    - not general enough
Solution(1) we are talking about

- Far-end NAT traversal
  - All done on side of service provider/PBX/softswitch/SBC/you name it
    - That's the “far end”
    - Nothing required on client side
  - Step 1: Detect client-side NAT
  - Step 2: Work around client-side NAT
  - Step 3: Success/profit/etc.
Far-end NAT traversal

- How to “detect” NAT, exactly?
  - On the signaling (SIP) level.
  - SIP endpoint will send IP and transport-layer reachability info in SIP messages and replies based on what it knows its local IP to be
    - Phone on local LAN (192.168.1.24) will send SIP messages riddled with references to 192.168.1.24
  - Look for inconsistencies between that information and where (IP and port) SIP packet actually comes from
  - “Where SIP packet actually comes from” = terminology: “received” IP and port
    - “Actually received from” vs. “allegedly received from”
  - If inconsistencies found, ignore information in SIP message and use received IP:port
Far-end NAT traversal (cont'd)

• How to “detect” NAT, exactly?
  – Not RFC 3261 compliant at all
    • If NAT breaks RFC SIP functionality, NAT workarounds will break the RFC SIP specification
  – Compliant with other RFCs that deal specifically with far-end NAT traversal schemes
  – That's reality of how protocols and standards are actually defined vs. implemented
REGISTER requests

- Registrar and/or location server at sip.evaristesys.com receives REGISTER message from client at 192.168.1.24:5060 (source port 5060).
- NAT gateway maps this request on egress to another IP:port tuple:
  - 64.4.41.187:17688
- Contact URI is not rewritten by NAT gateway. It still contains:
  - Contact: <sip:s@192.168.1.24:5060>
  - SIP registrar supposed to send request to that Request URI when it looks up AOR (Address of Record) – AKA “username” or line identifier
- NAT-aware registrar notices that IP:port in Contact URI != received IP:port
- Contact URI is ignored, sip:s@64.4.41.187:17688 is stored as Contact binding for that AOR instead
REGISTER requests (cont'd)

- Registrar gets call at SIP URI corresponding to AOR:
  - Ex: <sip:abalashov@sip.evaristesys.com>
  - Translates to stored contact binding
    - It was overridden with received IP:port
    - Registrar forwards INVITE to:
      
      sip:s@64.4.41.187:17688
INVITE requests (inbound from client)

- INVITE request from client arrives at SIP server from 192.168.1.24:5060 → 64.4.41.187:17688

- Via header contains 192.168.1.24:5060, but received IP:port is 64.4.41.187:17688.
  - Red flag – ignore Via: header and send response to the received IP:port instead.
  - Via header indicates where provisional replies to INVITE request go.

- Contact URI is <sip:something@192.168.1.24:5060>
  - That's where sequential in-dialog requests (like BYE, or a re-INVITE) go
  - Ignore it and mangle the domain/port portion to received IP:port before processing on UAS transaction layer
INVITE requests (outbound to client)

- Same as scenario in previous slide, except outbound INVITE to client is not mangled.
- Final replies in particular are mangled – that includes Contact URI in 200 OK received from client.
  - That's where sequential/in-dialog requests go.
- Sequential in-dialog requests (such as a re-INVITE) are similarly mangled – overridden with received IP:port if inconsistency detected.
What about media?!

- Media is described by SDP (Session Description Protocol) payload contained in INVITEs and in final replies to INVITEs (200 OKs)
  - Pedantic note: SDP offers/answers can also be contained in ACKs, per RFC 3261
  - Less pedantic note: Also in early dialogs for early media (183 Session in Progress)
  - Actually, in any non-100 1xx-class provisional message.
  - Who cares – just check for Content-Type: application/sdp :-)

- In incoming INVITEs, mangle the media IP endpoint in the SDP payload to the received IP if NAT detected.

- In outgoing INVITEs, mangle all SDP-containing replies to contain received IP if NAT detected.
What about media?!  
Wait! Mangle the SDP payload to the received IP –  

... what about media **ports**?
What about media?!

- Media ports:
  - Clearly, the SIP received port cannot be used.
  - Media does not go to the SIP port.
  - The SDP payload does contain ports, but they are internal (NAT'd) host ports – who knows what the outside ports will be?
  - There is no way to know the externally mapped (by the NAT gateway) media ports in advance until the media stream actually starts.

- Additional intelligence needed for far-end NAT traversal-capable media endpoint: **symmetric RTP**
Symmetric RTP detection

- Symmetric RTP is logical correlate to the same kind of logic being used with SIP in this scenario.

- Actual process is not called symmetric RTP – symmetric RTP refers to endpoints that receive media on same port they send it from. The NAT traversal process described here is most accurately described as “draft-comedia style RTP NAT traversal” - the essence of the technique was described in that IETF draft.
  - It'd be nice if someone came up with a more succinct name for this technique.
Symmetric RTP detection

- Ignore the RTP ports in the SDP payload – they are no more valid than `<sip:s@192.168.1.24:5060>` is valid from external perspective.
- Don't send RTP packets to NAT'd client – instead, **wait to receive** media packets first!
- Look at the **actual** (received) source port of the incoming RTP stream from NAT'd client.
- Start sending media there, instead.
  - Requires that endpoint expect to receive media at same port it sends media from – most endpoints nowadays do this, but not all
- No appreciable media start delay from point of view of user – happens in milliseconds.
  - RTP is very high PPS – with typical packetization durations
Symmetric RTP det. (cont'd)

- What if my SIP endpoint is not also my media endpoint? (Distributed / softswitch assembly-like architecture)
  - The media endpoint needs to be capable of symmetric RTP
  - Setting needs to be enabled (if SBC and/or softswitch and is supported)
  - If symmetric RTP not supported, you're out of luck, unless...

- **Option #2**: Inline media relay.
  - Use some kind of media proxy/gateway that does symmetric RTP
  - To NAT'd endpoint, send INVITE or replies w/SDP that point to media proxy as endpoint
  - To other endpoint, send INVITE and/or replies w/SDP that also point to the media proxy as the endpoint
  - SIP signaling agent activates media proxy via API and/or control socket and/or media gateway control protocol on correct ports, with provision for symmetric RTP
Problems

- **Asymmetrical signaling/media endpoints**: Some user agents (NAT'd endpoints) are not symmetrical in signaling and/or media
  - Receive on different port than they send from
  - This is not how NAT works – NAT statefully forwards replies to a “tracked” connection (or “connection” in case of UDP) to the source port of the original request that initiated that “connection”
  - But most user agents are symmetrical in their behaviour these days
  - One reason not to buy really, really cheap SIP equipment
Problems (cont'd)

- Application Layer Gateways (ALGs):
  - Some routers/firewalls/CPE are SIP-aware
  - They actually read SIP packets and mangle the semantics themselves
  - They're trying to be “helpful” - good idea, in principle
  - However, they conflict with far-end NAT traversal detection
  - Often they are incomplete – for instance, will mangle SIP but not appropriately mangle SDP ports
  - If you are using far-end NAT traversal, you're going to do well to just turn your SIP ALG off – if you can.
    - Cisco routers / firewalls often have it enabled by default
    - Needs to be explicitly turned off.
  - Surprisingly diverse range of equipment (even very cheap stuff) often has a SIP ALG – or tries to, anyway
Problems (cont'd)

- Poor UDP statekeeping:
  - Compared to TCP, firewalls “forget” UDP state mappings (inside IP:port ↔ outside IP:port) rather quickly
  - Example:
    - **REGISTER** request creates UDP state “pinhole” in firewall.
    - 2 minutes later it is expired due to lack of subsequent activity.
    - 5 minutes after **REGISTER** request, **INVITE** comes in to last known received IP:port.
    - NAT gateway already forgot about that tuple – sends ICMP port unreachable message.
    - No response from client – SIP server retransmits **INVITE** some more, eventually fails.
Problems (cont'd)

- **Poor UDP statekeeping (cont'd):**
  - Difference in how NAT equipment handles this very large
  - Some devices keep UDP state for fairly long time
  - Others are astoundingly bad – expiration in as little as 30 seconds to 1 minute!
  - **Solutions:**
    - Send periodic “pings” of some sort to reset the expiration timer on the UDP state mapping.
      - **SIP OPTIONS** pings commonly used for this
    - Low re-registration intervals to create activity on that state mapping.
Problems (cont'd)

- **Diversity in the wild:**
  - One-size-fits-all solution needed.
  - Lots of different NAT implementations
  - Lots of SIP endpoints
  - Lots of equipment
  - Occasional interop issues
  - Implementations of conflicting IETF drafts
    - You thought the hundreds of RFCs were it? Lots of stuff gets implemented (often conflicting, multiple versions of same functionality) long before stuff becomes RFC.
    - Even one-size-fits-all common denominator will only work for 90% of customers – some customers just won't work
      - Asymmetric signaling and/or media endpoints
      - ALGs that can't be disabled
      - Multiple layers of NAT
      - Interop bugs
**Asterisk**

- Doing far-end NAT detection & traversal with Asterisk
  - Really easy.
  - Set `nat=yes` and `qualify=yes` on the `sip.conf` SIP peer for the (possibly NAT'd) client.
  - What `nat=yes` does:
    - SIP-level far-end NAT detection procedure as described earlier
    - Draft-comedia style symmetric RTP NAT traversal, as described earlier
  - What `qualify=yes` does:
    - Periodic SIP OPTIONS pings.
    - Used for Asterisk's “dead peer detection.”
    - Have the very useful side effect of keeping UDP NAT state pinholes open, though!
If you're not using Asterisk...

- **OpenSER family of projects (Kamailio/OpenSIPS/SER/SIP-Router):**
  - Nathelper module
  - Rtpproxy or MediaProxy 2.0 (or, alternatively, whatever IPTel's kernel-bound one is called these days – I forget)
  - Shameless plug: Evariste Systems deploys OpenSER-based far-end NAT traversal solutions!

- **Common session border controllers (SBCs) and switches:**
  - There is probably an option to enable these behaviours
    - Unless your equipment and/or firmware and/or software is unbearably old and the SIP stack hasn't gotten the memo on NAT
    - Or unless your equipment and/or firmware and/or software is really, really, really low-end
If you're not using Asterisk...

- Some other VoIP network element:

:-)
Public Service Announcement

- If you are trying to run a SIP server behind destination NAT...
  - Server on private IP
  - Clients connect to it through NAT gateway from outside networks, addressing it by the NAT gateway's outside interface address:

    … DON'T!!!!

- Pointless waste of time
  - Exception: one-to-one NAT
    - It's still a bad idea, but it'll probably work, depending on your NAT equipment
    - Hidden gotchas abound.
THE END