ICE: the ultimate way of beating NAT in SIP

Saúl Ibarra Corretgé | AG Projects

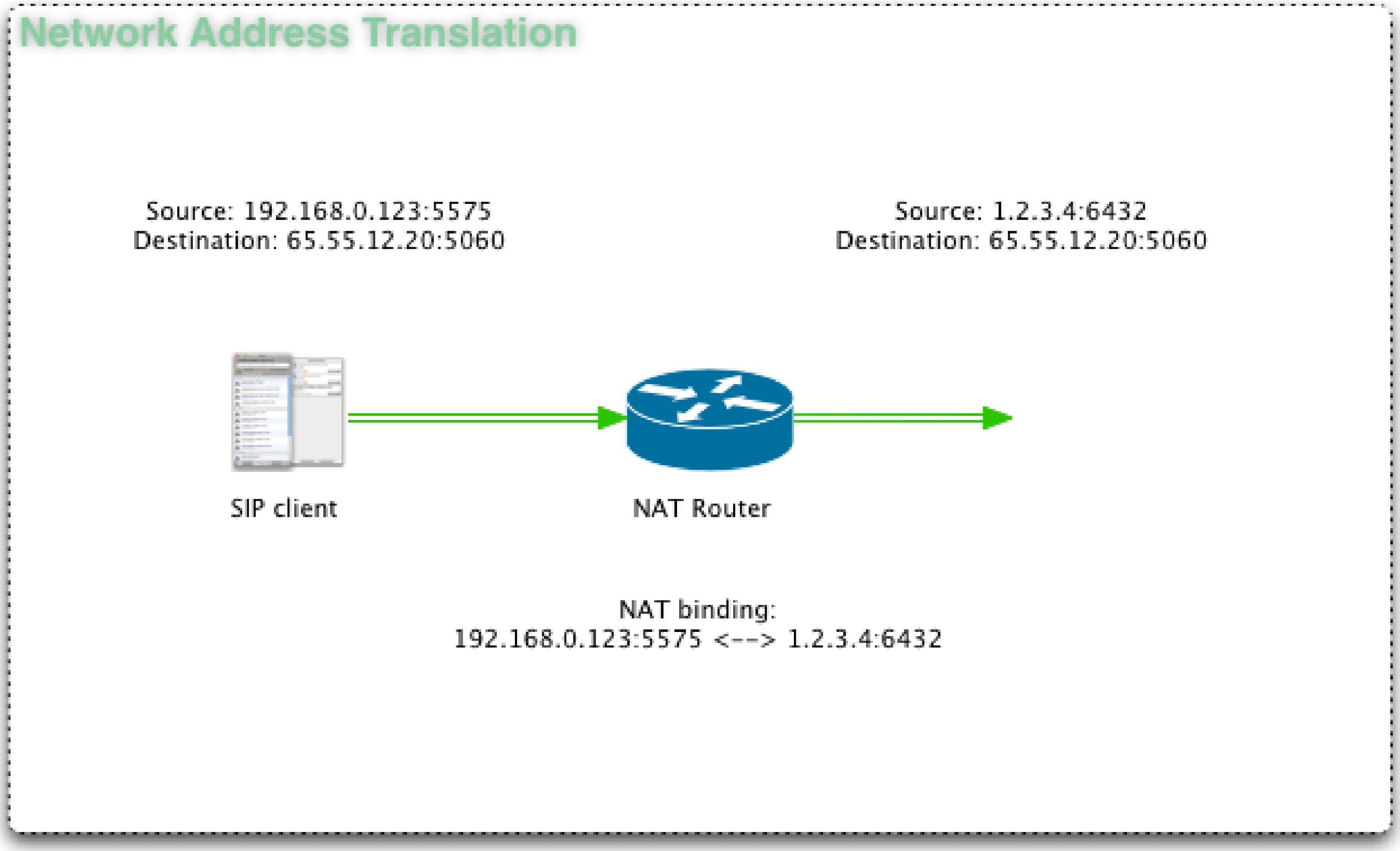
Index

How NAT affects SIP

- Internet providers use NATs
  - Multiplex private addresses into a single public one
  - 'Hide' inner network from the outside

- NATs create a binding between the internal/private address and the external/public
- IP and port in the packets is modified with the binding information

How NAT affects SIP (II)


Network Address Translation
Source: 192.168.0.123:5575
Destination: 65.55.12.20:5060
Source: 1.2.3.4:6432
Destination: 65.55.12.20:5060
SIP client
NAT Router
NAT binding:
192.168.0.123:5575 <--> 1.2.3.4:6432

How NAT affects SIP (III)

- This changes in the source IP/port affect SIP because it will contain private IP addresses
    - Contact header: in REGISTER requests it will be used for targeting incoming INVITEs
    - SDP: target address and port for media
- This results in one way audio / no media at all!
- Can this be solved?
    - Contact header for REGISTER: a proxy can use the received IP/port.
    - SDP: hard to solve, as ports are dynamically allocated

How NAT affects SIP (IV)

```
INVITE sip:3333@sip2sip.info SIP/2.0
Via: SIP/2.0/UDP 192.168.99.23:49919;rport;branch=z9hG4bKPj.OB8RPYvcZIaBcu.uom4xvbsyw9RBwlW
Max-Forwards: 70
From: "saul" <sip:saghul@sip2sip.info>;tag=N0mSaBvIOXOLC0sNpJ9oJvrpJMuSeC8p
To: <sip:3333@sip2sip.info>
Contact: <sip:dezruwmf@192.168.99.23:49919>
Call-ID: PQ4m4UxA9VHDJ.uLGXzKOQm-9ljIZGvH
CSeq: 24149 INVITE
Route: <sip:81.23.228.150;lr>
Allow: SUBSCRIBE, NOTIFY, PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, MESSAGE
Supported: 100rel
User-Agent: blink-0.18.2
Content-Type: application/sdp
Content-Length:   387

V=0
o=- 3484383368 3484383368 IN IP4 192.168.99.23
s=blink-0.18.2
c=IN IP4 192.168.99.23
t=0 0
m=audio 50076 RTP/AVP 0 8 9 101
a=rtcp:50077 IN IP4 192.168.99.23
a=rtpmap:9 G722/8000
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=sendrecv
```

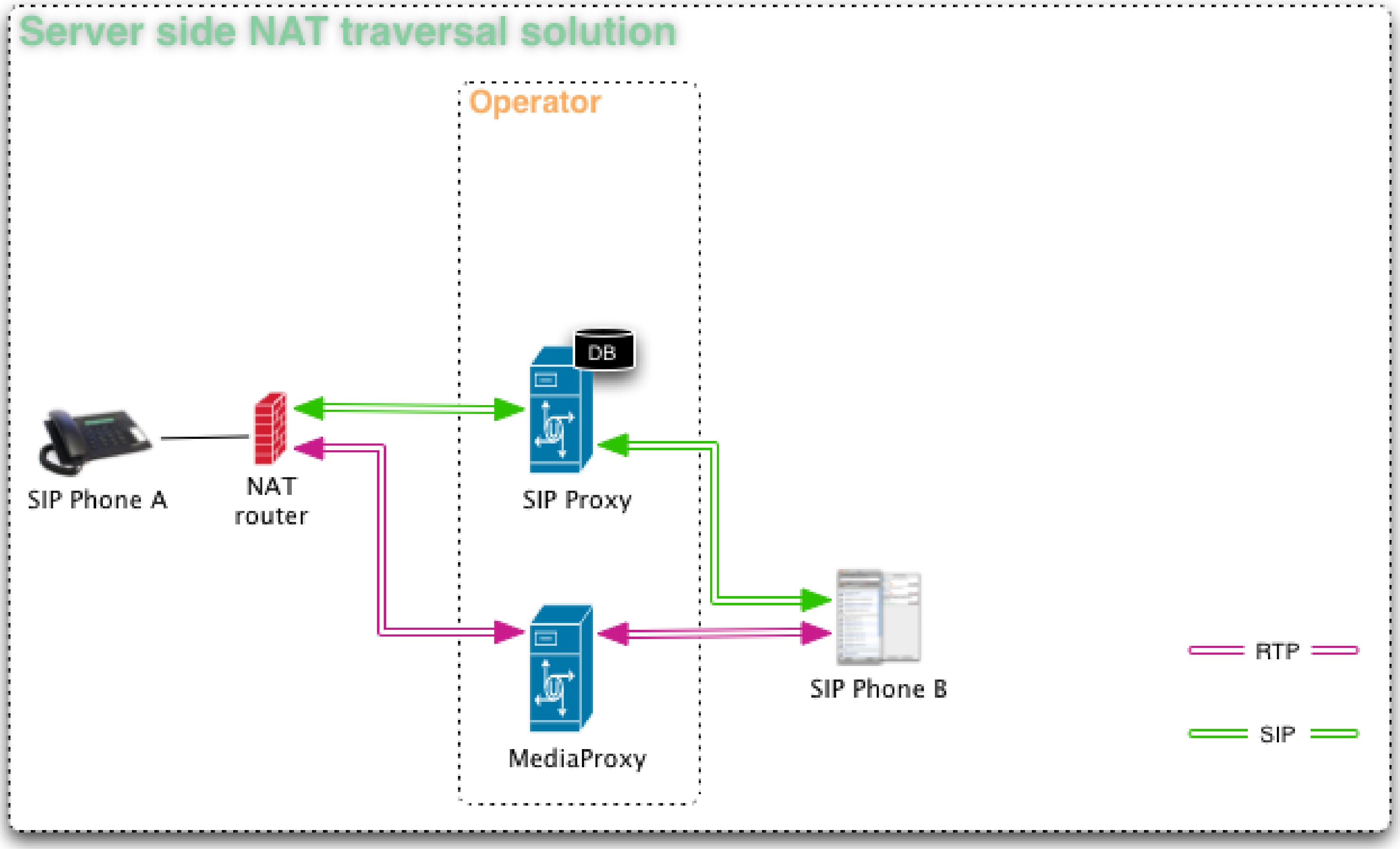
Solving the problem in the client

- Clients may try to solve their NAT issue by using client-side NAT traversal techniques
  - Session Traversal Utilities for NAT (STUN) – RFC 5389
  - Traversal Using Relays around NAT (TURN) – RFC 5766
- However...
  - TURN hasn't been widely deployed
  - STUN can't be used in case of symmetric NAT
    - Most common type of NAT?
- Cooperation from the server side
  - Deployment of STUN/TURN servers
- Servers don't trust clients

Solving the problem in the server

- Insert a media relay in the path so that 2 way media works in the worst case
- SDP mangling
- Ugly hacks to avoid using a media relay every time
  - If both users come from the same network
  - Other local policies

Solving the problem in the server (II)



ICE: the ultimate solution!

- Interactive Connection Establishment
  - **RFC 5245**. Yes, it's an RFC!
- Combines client-side techniques with server support to find the most appropriate way of communicating with the other endpoint
  - STUN + TURN
- Media should only be relayed in the worst case
  - Both endpoints behind symmetric NATs
- Start sending media when it's guarantied that there will be a successful communication
- Clients don't need to know their NAT type
- **A complex protocol**
  - It took ICE 6 years to become an RFC!
  - Not many fully capable ICE clients... but you can Blink! :)

ICE Step 1: Allocation

- Before sending the INVITE
- Gather all possible **candidates**
- Candidate types
    - Host candidates: machines local network interfaces
    - Server reflexive candidates: learnt by using STUN
    - Relayed candidates: allocated with STUN Relay Usage requests (RFC 5766)

ICE Step 2: Prioritization

priority = (2^24)*(type preference) +
(2^8)*(local preference) +
(2^0)*(256 -componentID)

- **Type preference**: Depends on candidate type (0 for relayed candidate, 126 for host candidate)
- **Local preference**: Local policy for selecting different priority if candidates are same type. Also IPv4 / IPv6.
- **Component ID**: 1 for RTP, 2 for RTCP

ICE Step 3: Offer encoding

ICE attributes

ICE candidates

```
V=0
o=- 3484389594 3484389594 IN IP4 192.168.99.23
s=blink-0.18.2
c=IN IP4 192.168.99.23
t=0 0
m=audio 64249 RTP/AVP 104 103 102 9 0 8 101
a=rtcp:64250 IN IP4 62.131.6.55
a=rtpmap:104 speex/32000
a=rtpmap:103 speex/16000
a=rtpmap:102 speex/8000
a=rtpmap:9 G722/8000
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=ice-ufrag:241ffa10
a=ice-pwd:2f5a42f7
a=candidate:Sc0a86317 1 UDP 1694498815 62.131.6.55 64249 typ srflx raddr 192.168.99.23 rport 49306
a=candidate:Hc0a86317 1 UDP 2130706431 192.168.99.23 49306 typ host
a=candidate:Sc0a86317 2 UDP 1694498814 62.131.6.55 64250 typ srflx raddr 192.168.99.23 rport 49519
a=candidate:Hc0a86317 2 UDP 2130706430 192.168.99.23 49519 typ host
a=sendrecv
```

ICE Step 3: Offer encoding (II)

a=candidate:Sc0a86317 1 UDP 1694498815 62.131.6.55 64249 typ srflx raddr 192.168.99.23 rport 49306

- **Foundation** (Sc0a86317): Unique identifier for each candidate of the same type, same interface and STUN server (if applicable)
- **Component ID** (1): Identifier, 1 for RTP, 2 for RTCP
- **Transport** (UDP): Candidate transport type
- **Priority** (1694498815): Priority for the given component
- **IP address and port** (62.131.6.55 64249): Component's IP and port
- **Type** (srflx): Component type
- **Related address** (raddr 192.168.99.23 rport 49306): Optional information: for relayed candidates it contains the server reflexive address and for server reflexive candidates it contains the host address.

- After encoding the offer it's sent out as a regular INVITE

ICE Step 4: Allocation

- The callee receives the offer and starts his own process.
  - Gather candidates
  - Prioritize
  - Encode SDP answer
  - Send 200 OK with SDP

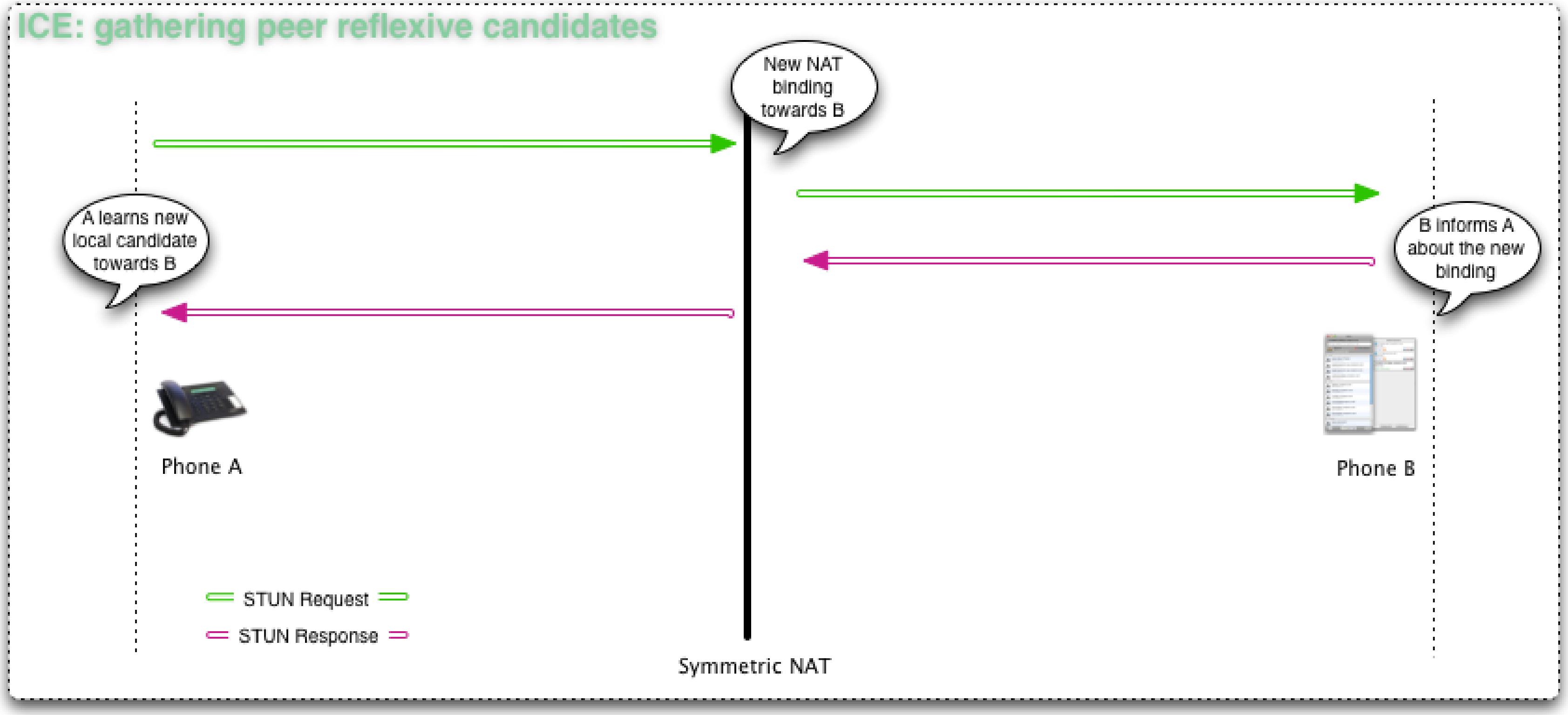
ICE Step 5: Verification

- Both parties have each other's candidates
- Each party pairs it's own local candidates with the candidates from the remote party
- List is pruned for duplicated candidates
  - Both endpoints will have the same list

- Each endpoints sends a connectivity check every 20ms
  - STUN Binding Request from the local candidate to the remote
  - The receiver generates an answer with the received IP and port included
  - If the response is received the check is successful

ICE Step 5: Verification (II)

- During the connectivity checks new candidates can be found
  - Peer reflexive candidates
  - P2P media is possible if **only one** of the parties if behind a symmetric NAT


ICE: gathering peer reflexive candidates
New NAT binding towards B
A learns new local candidate towards B
B informs A about the new binding
Phone A
Phone B
STUN Request
STUN Response
Symmetric NAT

ICE Step 6: Coordination + Communication

- After all checks both endpoints will have the same set of valid candidates
- **All negotiation has taken place at the media level, through STUN**
- Controlling agent will decide which of the valid candidates to use
  - In ICE full implementations the offerer is the controlling agent
  - It will do a connectivity check again, but with a “use candidate” flag included in the STUN request
  - If check succeeds both endpoints know where to send media to each other :)

ICE Step 7: Confirmation

- Negotiation took place at the media level
  - At SIP level we don't know where media is!
- If the transport address where the media is received changed due to ICE negotiation, a re-INVITE must be sent to update the status of any possible middle box.

Why ICE ~~doesn't~~ didn't work

- Currently SDP mangling + media relaying is the most common NAT traversal mechanism
- If a SIP proxy mangles the SDP without taking ICE into account the negotiation will be broken

Why ICE ~~doesn't~~ didn't work (II)

```
V=0
o=- 3484393780 3484393780 IN IP4 192.168.99.53
s=sipsimple 0.14.2
c=IN IP4 85.17.186.6
t=0 0
m=audio 51354 RTP/AVP 9 8 101
a=rtcp:51355 IN IP4 85.17.186.6
a=rtpmap:9 G722/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=ice-ufrag:76e08623
a=ice-pwd:4e2db26f
a=candidate:Sc0a86335 1 UDP 1694498815 62.131.6.55 49732 typ srflx raddr 192.168.99.53 rport 51641
a=candidate:Hc0a86335 1 UDP 2130706431 192.168.99.53 51641 typ host
a=candidate:Sc0a86335 2 UDP 1694498814 62.131.6.55 49733 typ srflx raddr 192.168.99.53 rport 40568
a=candidate:Hc0a86335 2 UDP 2130706430 192.168.99.53 40568 typ host
a=sendrecv
```

- IP in the c line doesn't match any IP in the candidate list!
  - ICE mismatch!

- **OpenSIPS + MediaProxy will come to the rescue!**

Fixing ICE in the server

- The server needs to be aware of ICE
  - Mangle necessary information in the SDP
  - Don't block STUN checks
  - Think about accounting!

- Tools that needed to be modified
  - OpenSIPS (http://opensips.org)
  - MediaProxy (http://mediaproxy.ag-projects.com)
  - CDRTool (http://cdrtool.ag-projects.com)

Fixing ICE in the server: OpenSIPS

- Detect that a request is offering ICE
- Allow the user to select if a ICE candidate should be inserted and the priority
- Allow the user to dynamically change the behavior though an AVP
- Complete design: http://mediaproxy.ag-projects.com/wiki/ICE

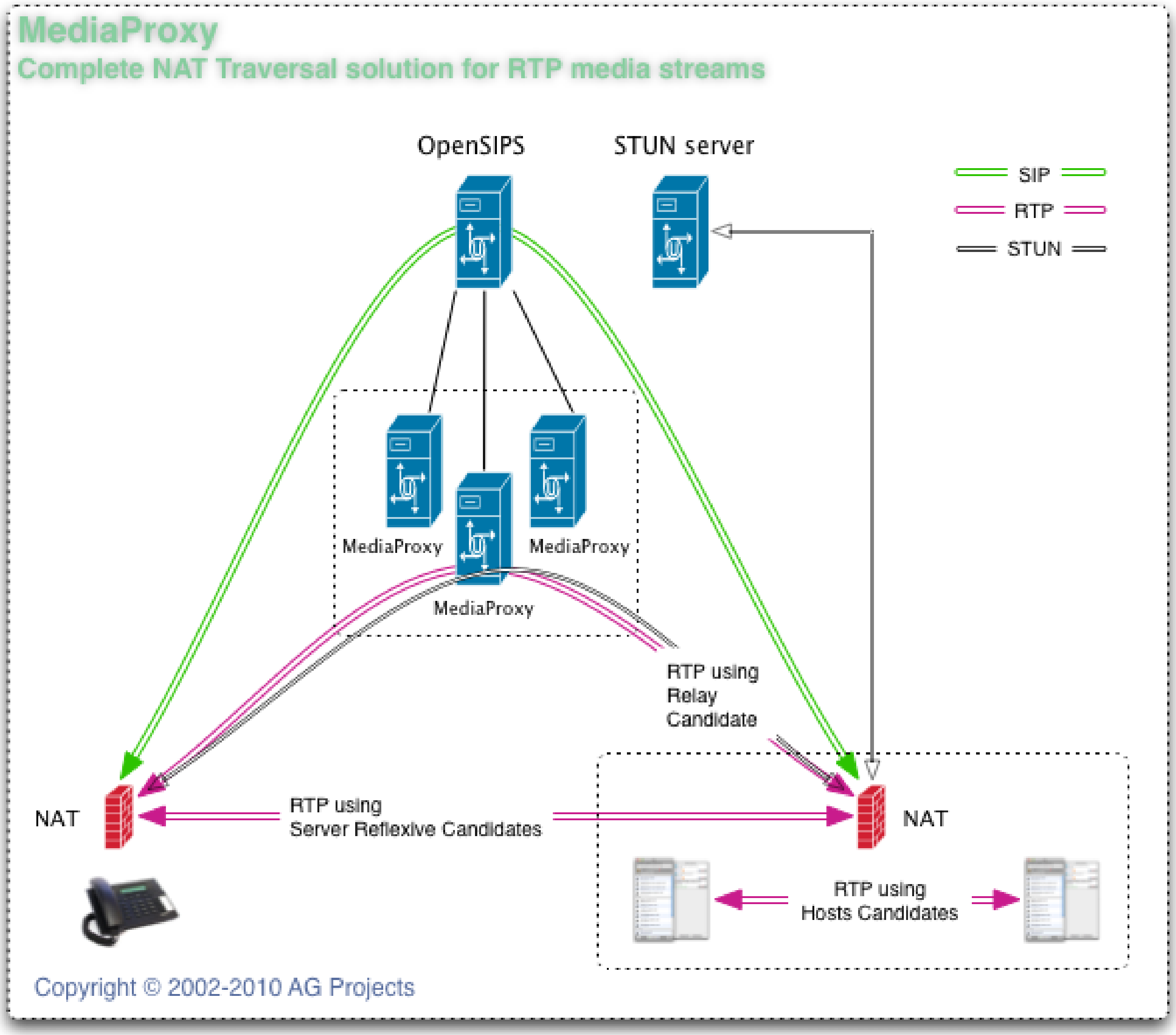
Fixing ICE in the server: OpenSIPS (II)

```
V=0
o=- 3484393780 3484393780 IN IP4 192.168.99.53
s=sipsimple 0.14.2
c=IN IP4 85.17.186.6
t=0 0
m=audio 51354 RTP/AVP 9 8 101
a=rtcp:51355 IN IP4 85.17.186.6
a=rtpmap:9 G722/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=ice-ufrag:76e08623
a=ice-pwd:4e2db26f
a=candidate:R6ba1155 1 UDP 16777215 85.17.186.6 51354 typ relay
a=candidate:R6ba1155 2 UDP 16777214 85.17.186.6 51355 typ relay
a=candidate:Sc0a86335 1 UDP 1694498815 62.131.6.55 49732 typ srflx raddr 192.168.99.53 rport 51641
a=candidate:Hc0a86335 1 UDP 2130706431 192.168.99.53 51641 typ host
a=candidate:Ha45450a 1 UDP 2130706431 10.69.69.10 51641 typ host
a=candidate:Sc0a86335 2 UDP 1694498814 62.131.6.55 49733 typ srflx raddr 192.168.99.53 rport 40568
a=candidate:Hc0a86335 2 UDP 2130706430 192.168.99.53 40568 typ host
a=candidate:Ha45450a 2 UDP 2130706430 10.69.69.10 40568 typ host
a=sendrecv
```

Fixing ICE in the server: MediaProxy

- MediaProxy needs to be aware about ICE negotiation taking place
- **Ability to relay STUN requests**
- Bail out silently if it was not the chosen candidate
  - Both endpoints had ICE information in the SDP
  - STUN checks were received from both of them

Fixing ICE in the server: MediaProxy (II)

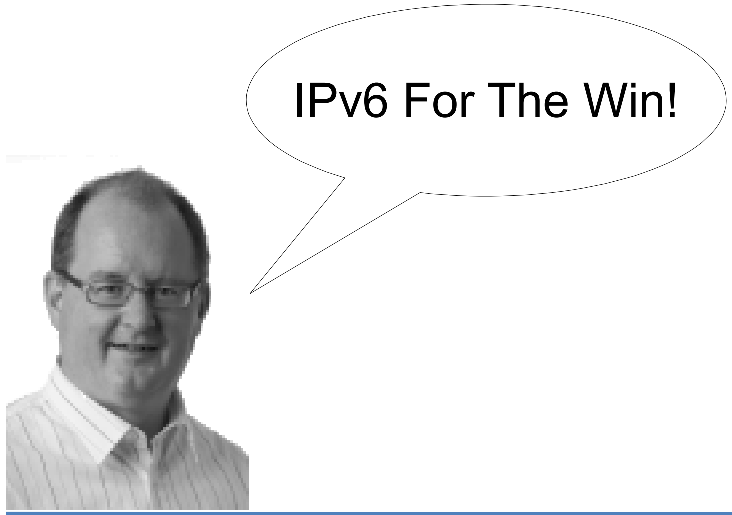

MediaProxy
Complete NAT Traversal solution for RTP media streams
OpenSIPS
STUN server
SIP
RTP
STUN
MediaProxy
MediaProxy
MediaProxy
RTP using Relay Candidate
NAT
RTP using Server Reflexive Candidates
NAT
RTP using Hosts Candidates
Copyright © 2002-2010 AG Projects

Fixing ICE in the server: recap

- This solution was **successfully tested at past SIPit26**
- **OpenSIPS + MediaProxy** is the first software combination to fix ICE this way ever
- Software versions
  - OpenSIPS >= 1.6.2)
  - MediaProxy >= 2.4.2
  - CDRTool >= 7.1
- Free public platform available: **http://sip2sip.info**

What about IPv6?

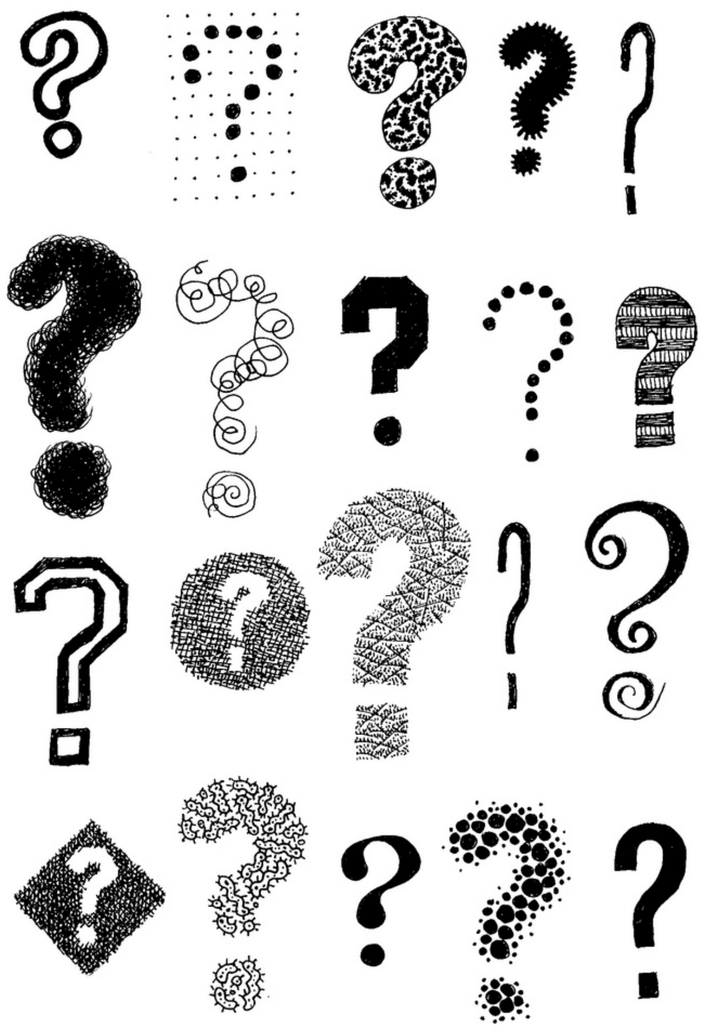
- Adoption will not begin tomorrow!
  - Meantime: IPv6 in the backbones and IPv4 elsewhere
- Still, NATs won't disappear!
- ICE can be used to select between IPv6 and IPv4 candidates



Recap

- ICE will allow endpoints to try to communicate by all means
- Server cooperation is needed
  - STUN servers
  - Mangle all necessary information not to break ICE
- Published as an RFC!
  - Go and implement it!
- Operators will want ICE
  - Who will relay HD video calls?

Questions?



BYE

```
BYE sip:audience@amoocon.de SIP/2.0
Via: SIP/2.0/UDP 192.168.99.23:49919;rport;branch=z9hG4bKPjDb30Dx0sH-ozn9QB.cCCboyU.atR97aM
Max-Forwards: 70
From: "saul" <sip:saul@ag-projects.com>;tag=UCpGKVZbQQx7BUKYtiuPEX668oa9jaU7
To: <sip:audience@amoocon.de>;tag=as59aef35c
Call-ID: DEWDfu63OACwYeQk7MrhmRhRq.1cqqis
CSeq: 10633 BYE
Route: <sip:81.23.228.129;lr;ftag=UCpGKVZbQQx7BUKYtiuPEX668oa9jaU7;did=641.a8a9c553>
User-Agent: blink-0.18.2
Content-Length:  0
```

You can Blink tomorrow at 14:00

twitter

@saghul
@agprojects

saul@ag-projects.com

sip:saul@ag-projects.com