Begriffe

- **Client**: Ein SIP Knoten, der SIP Requests verschickt und SIP Responses empfängt.
- **Server**: Ein SIP Knoten, der SIP Requests empfängt und SIP Responses sendet.
- **User Agent (UA)**: Ein SIP Knoten, der am Ende der SIP Verarbeitungskette steht. Man unterscheidet:
  - **UAC**: User Agent Client, erzeugt SIP Requests und verarbeitet SIP Responses
  - **UAS**: User Agent Server, verarbeitet SIP Requests und erzeugt SIP Responses
- **Proxy**: Ein SIP Knoten, der sowohl als Client als auch als Server arbeitet. Hauptaufgabe ist das Routing von SIP Nachrichten.

Grundlagen

Jeder SIP Request muß mindestens die folgenden Felder enthalten:

- **Request URI**: üblicherweise die URI im **To** Header
- **To**: URI, z.B. SIP URI oder Tel URI (RFC2806) des logischen Empfängers
- **From**: URI des logischen Initiators eines SIP Requests
- **Call-Id**: Global eindeutige Kennung für eine Gruppe von Requests (z.B. alle Requests innerhalb eines Dialoges oder einer Registrierung)
- **CSeq**: Sequenznummer und Methode des Requests
- **Max-Forwards**: Maximale Anzahl Hops (vgl. TTL im IP Header, Beispieltrace). Standardwert ist 70
- **Via**: Transportadresse, zu der die Antwort geschickt werden soll. Der Branch Parameter dient zur Zuordnung von Requests/Responses zu Transaktionen.

Identifizieren von SIP Servern (RFC3263)

Anhand einer URI muß ein Server gefunden werden, der die Nachricht weiterleiten kann.

Enthält die URI eine Transportspezifikation, muß diese benutzt werden, andernfalls ist UDP der Standard für SIP URIs, TCP für SIPS.

Im NAPTR Record der DNS Konfiguration findet sich der Name des Servers für die einzelnen Transporttypen einschliesslich ihrer Priorität.

Im SRV Record der DNS Konfiguration finden sich zum jeweiligen Server Namen und Portnummer.

Im A bzw. AAAA Record findet sich die IP Adresse des Servers.

Wird kein NAPTR Record gefunden, wird direkt der SRV Record genutzt.

Beispiel

Wir betrachten die SIP URI sip:name@cesnet.cz (die Ausgaben sind gekürzt):

```
#host -t NAPTR cesnet.cz
cesnet.cz NAPTR "s" "SIP+D2U" _sip._udp.cesnet.cz
cesnet.cz NAPTR "s" "SIP+D2T" _sip._tcp.cesnet.cz

#host -t SRV _sip._udp.cesnet.cz
_sip._udp.cesnet.cz SRV 5060 cyrus.cesnet.cz

#host -t AAAA cyrus.cesnet.cz
cyrus.cesnet.cz AAAA 2001:718:1:...:FE64:1E71
```

Client Routing

Zu welchem Server ein SIP Request weitergeschickt wird, hängt von einer Reihe von Optionen ab:

1. lokale Konfiguration (z.B. Proxy Konfiguration im Client)
2. gespeicherte Routen im Rahmen eines Dialoges
3. die URI im obersten `Route` Header, falls vorhanden
4. die Request-URI

Die gewählte URI ergibt über den DNS Zugriff eine Liste von Zielsystemen (Adresse, Port, Transport) mit Prioritäten.

Server Routing

- Die Response wird zur Quelladresse des Requests zurückgeschickt, wobei der Port im Via Header verwendet wird.
- Bei verbindungsorientiertem Transport ist die Verbindung zu nutzen, auf der der Request empfangen murde.
- Schlägt das fehl, werden Adresse und Transportprotokoll des obersten Via Headers verwendet.
- Enthält der Via Header einen Namen (anstelle einer IP Adresse), wird zuerst der SRV Record aufgelöst, dann der A oder AAAA Record.

Proxies

SIP Proxies gibt es in zwei Ausprägungen, stateless und stateful.

Stateless Proxies dienen zur einfachen Weiterleitung von SIP Nachrichten basierend auf den Regeln für Client und Server.

Ein Stateful Proxy implementiert mindestens den SIP Transaktionsmechanismus. Er verhält sich gegenüber einem Client wie ein vollwertiger Server, gegenüber Servern wie ein Client.

- Prüfung von Requests
- Modifikation von Requests auf Eingangsseite
- Berechnung des Ziels (der Ziele) des Request
- Weiterleitung des Requests in einer eigenen Transaktion
- Verarbeitung der Response(s)

Prüfung des Requests

Ein Proxy muß sicherstellen, daß Minimalanforderungen erfüllt sind, bevor ein Request verarbeitet wird:

- Syntaxprüfung
- Prüfung, ob das URI-Schema (z.B. sip, tel) unterstützt wird (sonst Fehler 416)
- Max-Forwards muß positiv sein (sonst Fehler 483)
- optionale Prüfung auf Routing Schleifen anhand der Adresse in einem `Via` Header
- Authentifizierung und Authorisierung (z.B. Proxy-Authenticate)

Routenprüfung

- Enthält der oberste `Route` Header eine Adresse des Proxies, wird dieser Eintrag entfernt. Damit wird **Loose Routing** implementiert.
- Entspricht die Request-URI einer Adresse, die der Proxy in einen `Record-Route` Header eingetragen hat, wird die Request-URI durch den letzten `Route` Header überschrieben und der Eintrag gelöscht. Dies tritt nur dann auf, wenn der vorherige Proxy (noch) **Strict Routing** nach RFC2543 implementiert. Dies stellt den ursprünglichen Request wieder her.

Berechnung der nächsten Ziele

Für Request-URIs, für die der Proxy zuständig ist, ermittelt er eine Menge von URIs, die als Ziel benutzt werden sollen (sog. Target Set). Hierzu können beliebige Informationsquellen genutzt werden.

Während der Weiterleitung kann das Target Set erweitert werden, z.B. durch Antworten auf weitergeleitete Requests (3xx Redirect).

Ist der Proxy nicht zuständig für die Domain in der Request-URI, enthält das Target Set ausschließlich das in der Request-URI genannte Ziel.

Weiterleiten der Requests

Es ist dem Proxy (bzw. dessen Konfiguration) überlassen, wie und in welcher Reihenfolge die Einträge im Target Set zur Weiterleitung genutzt werden. Üblicherweise sind folgende Schritte nötig:

1. Kopieren des Requests
2. Anpassen der URI
3. Dekrementieren des `Max-Forward` Headers
4. ggfs. einen `Record-Route` Header einfügen
5. weitere optionale Header einfügen
6. Auswerten der Routing Informationen um den nächsten Hop zu bestimmen
7. `Via` Header einfügen und weiterleiten

1. Kopieren des Requests
   - Der Nachrichtenbody darf nicht verändert werden.
   - Die Reihenfolge der Header soll möglichst nicht verändert werden, bei gleichem Namen darf sie nicht verändert werden.
2. Anpassen der URI
   - Die URI des Elementes des Target Sets wird als Request-URI eingesetzt.
3. Dekrementieren des `Max-Forward` Headers
   - Enthält der Request keinen `Max-Forwards` Header, muß er mit Wert 70 eingefügt werden.

4. ggfs. einen `Record-Route` Header einfügen
   - Daß der Proxy Zwischenziel der Kommunikation ist, kann an einem vorkonfigurierten `Route` Header gelegen haben.
   - Ein Proxy kann sich mit einem `Record-Route` Header in den Route Set eines Dialoges eintragen. Dadurch wird der Proxy die weiteren Requests/Responses in diesem Dialog sehen.
6. Auswerten der Routing Informationen um den nächsten Hop zu bestimmen
   - Ist auf dem Proxy eine Liste von weiteren Proxies konfiguriert, über die der Request weitergeleitet werden soll, so werden deren URIs in `Route` Headern vor den bisherigen `Route` Headern eingefügt.
   - Der nächste Hop wird mit den Methoden bestimmt, die für UACs gelten. Dazu muß der Request gegebenenfalls für den Transport entsprechend aufbereitet werden.

7. `Via` Header einfügen und weiterleiten
   - Abhängig vom gewählten Transport, mit dem der nächste Hop erreicht wird, wird ein entsprechender `Via` Header eingefügt.
   - Da es sich um eine neue Clienttransaktion handelt, muß der `Via` einen Branch Parameter enthalten.
   - Ist der Transport verbindungsorientiert, wird ggfs. ein `Content-Length` Header benötigt.

Verarbeitungen der Response

Empfängt ein SIP Knoten eine Response, versucht er zunächst, die zugehörige Transaktion zuzuordnen. Wird keine gefunden, wird die Nachricht wie bei einem Stateless Proxy weitergeleitet.

Andernfalls sind folgende Schritte vorgeschrieben:

1. Bestimmung des Kontextes
2. Entfernen des obersten `Via` Headers
3. Aktualisierung des Kontextes
4. Test, ob die Response sofort weitergeleitet werden muß.
5. Im Fall, daß noch keine abschließende Antwort weitergeleitet worden ist, bestimme die geeignete Antwort.

Der Transaktionskontext kann über `Branch` Parameter und `CSeq` identifiziert werden.

Der oberste `Via` Header wird entfernt. Falls kein weiterer `Via` in der Nachricht enthalten ist, ist die Response für den Proxy bestimmt. Damit darf er nicht weitergeleitet werden.

Die Response wird zum Kontext hinzugefügt, bis eine abschließende Response der zugehörigen Servertransaktion gefunden ist.

Resonses mit Statuscode 1xx außer 100 (Trying) und mit Statuscode 2xx müssen sofort über die Servertransaktion weitergeleitet werden.

Um eine Servertransaktion zu beenden, falls keine abschließende Antwort einer Clienttransaktion gesendet worden ist, wird bei Bedarf ein 408 (Request Timeout) erzeugt.

Responseverarbeitung zur Weiterleitung

- Wird als Antwort ein 401 (Unauthorized) oder 407 (Proxy Authentication Required) ausgewählt, so sind in einer Response alle `WWW-Authenticate` und `Proxy-Authenticate` Header mitzuschicken.
- `Record-Route` Header, die der Proxy selbst eingefügt hat, kann er für die Rückrichtung modifizieren. Das wird z.B. dann benötigt, wenn die Clienttransaktion auf einem anderen Interface (einer anderen IP Adresse) als die Servertransaktion läuft.
- Die ausgewählte Response wird an die Servertransaktion weitergeleitet.
- Ist eine abschließende Antwort zur Servertransaktion weitergeleitet worden, müssen alle zugehörigen Clienttransaktionen mit einem `CANCEL` beendet werden.

INVITE Transaktionen

Transaktionen, die durch ein `INVITE` eingeleitet werden, benutzen einen Handshake bestehend aus `INVITE`, `200 Ok` und `ACK`

- Wird eine Nachricht nicht bestätigt, wird sie wiederholt. Dabei wird die Zeit zwischen Wiederholungen nach jedem Schritt verdoppelt.
- Bestätigung für ein `INVITE` kann ein 1xx oder 2xx Statuscode sein.

Allen anderen Trasaktionen liegt ein einfaches Request-Response Modell zugrunde.

Beispiel

- Bob ist mit zwei Endgeräten bei seinem Provider `example.com` registriert.
- Alice versucht, Bob anzurufen.

1. Alice UA sendet ein `INVITE` an `bob@example.com` und erzeugt damit auf dem Proxy eine Servertransaktion
2. Der Proxy ermittelt beim Registrar ein Target Set mit zwei URIs und antwortet Alice mit einem `100 Trying`
3. Der Proxy erzeugt zwei Clienttransaktionen, die beide den `INVITE` an die jeweilige URI weiterleiten.

Beispiel (Fortsetzung)

1. Bobs UAs erhalten den `INVITE` und erzeugen ihrerseits Servertransaktionen und antworten mit einem `100 Trying`.
2. Die `100 Trying` werden von den Clienttransaktionen gespeichert und nicht weitergereicht.
3. Beide UAs erzeugen ein `180 Ringing`, das von den UAC im Proxy an die zugehörige UAS weitergeleitet wird.
4. Bob nimmt bei einem Endgerät ab, der UA erzeugt ein `200 OK` mit passender SDP Nachricht.
5. Der Proxy empfängt den `200 Ok` und leitet ihn an die Servertransaktion weiter. Die noch nicht abgeschlossene Clienttransaktion wird mit einem `CANCEL` abgebrochen.

Beispiel (Fortsetzung)

1. Der `200 Ok` erreicht den UAC von Alice, der ein `ACK` generiert. Der `ACK` muß nicht notwendigerweise dieselben Transaktionsmerkmale haben, aber im selben Dialog gesendet werden.
2. Zur Beendigung der Session sendet eine Seite ein `BYE`. Da dies zu einem Dialog gehört, wird kein neues Target Set berechnet.

vgl.:
`http://www.it.iitb.ac.in/xnet/sip/parallel_forking.html`

Transaktionen im Dialog

Dialoge dienen dazu, Zustandsinformationen über mehrere Transaktionen hinweg zu transportieren. Diese Informationen werden zur Adressierung und zum Routing benutzt.

UAC und UAS fügen bei der Dialogerzeugung einen `Contact` Header ein, der die SIP URI des Senders enthält. An diese URI werden die folgenden Requests im Dialog adressiert.

Teil des Zustandes eines Dialoges ist das Route Set, d.h. die geordnete Liste von SIP Knoten beschrieben durch ihre URI, die die Nachrichten bekommen müssen.

Server und Clients im Dialog

Erhält ein UAS einen Request, der einen Dialog aufbaut, muß er alle `Record-Route` Header des Requests in die Response kopieren.

Der UAS übernimmt die `Record-Route` Header in sein Route Set.

Erhält ein UAC eine Response, die einen Dialog aufbaut, übernimmt er die Folge der `Record-Route` Header in sein Route Set.

Neue Requests innerhalb des Dialoges benutzen als Request URI den jeweiligen `Contact` Header der Gegenseite und fügen das Route Set in Form von `Route` Headern ein.

ENUM (RFC3761)

Telefonnummern im internationalen Format E.164 können mittels DNS in SIP URIs umgewandelt werden. Die Rufnummern werden in umgekehrter Reihenfolge der Ziffern unter der TLD e164.arpa abgelegt.

ENUM für Deutschland, d.h. die Domain `9.4.e164.arpa`, wird durch die DENIC verwaltet.

Beispiel: Ermittle SIP URI für +420234680499

```
#host -t NAPTR 9.9.4.0.8.6.4.3.2.0.2.4.e164.arpa
9.9.4.0.8.6.4.3.2.0.2.4.e164.arpa NAPTR 200 50
  "u" "E2U+h323" "!^\\+(.*)$!h323:\\1@gk1ext.cesnet.cz!"
9.9.4.0.8.6.4.3.2.0.2.4.e164.arpa NAPTR 100 50
  "u" "E2U+sip" "!^\\+(.*)$!sip:\\1@cesnet.cz!" .
```

Routing

Für die Interaktion zwischen SIP/VoIP und PSTN werden SIP/PSTN Gateways eingesetzt, die sich gegenüber dem PSTN wie eine Vermittlungsstelle und gegenüber dem SIP Netz wie ein Proxy verhalten.

Zur Übertragung/Konvertierung der Datenströme werden sogenannte Media Gateways eingesetzt.

Die Umsetzung der SIP URIs und Telefonnummern erfolgt mittels ENUM.

Beispiel: PTSN ⇒ SIP (RFC3666)

1. Alice wählt auf ihrem PSTN Telefon die Rufnummer von Bob.
2. Der Anruf wird auf das PSTN/VoIP Gateway weitergeleitet.
3. Das Gateway ermittelt über eine ENUM Abfrage die SIP URI von Bob.
4. Das Gateway startet einen SIP Rufaufbau zu Bob (INVITE). RTP Endpunkt in der Session Description ist ein Port des Media Gateways.

Problembeschreibung

- SIP Clients in internen Netzen befinden sich oft in einem nicht öffentlichen IPv4 Bereich.
- Mobile Clients erhalten zur Zeit üblicherweise eine private IPv4 Adresse, die nur eine begrenzte Zeit gültig ist.
- Damit ein Teilnehmer erreichbar ist, muß seine Adresse in einem Registrar hinterlegt sein. Welche Adresse soll das REGISTER Paket enthalten?
- In den SDP Beschreibungen sind IP und Port der Endpunkte des Medienstromes enthalten. Sind diese Endpunkte für die Gegenstelle überhaupt erreichbar?

Workarounds

Abhängig vom Verhalten des NAT Elementes und den Möglichkeiten, die Infrastruktur zu beeinflussen, gibt es verschiedene Ansätze:

- NAT vermeiden
- IP Tunnel zwischen verschiedenen LANs
- Proxies an der Grenze zum Internet
- STUN
- NAT Firewalls/Router, die SIP beherrschen

vgl.:

`http://www.voip-info.org/wiki/view/NAT+and+VOIP`
`http://tools.ietf.org/html/`
`draft-ietf-sipping-nat-scenarios-06`

Anwendung

STUN ist ein Client-Server Protokoll, das es einem Prozess auf einem Host ermöglicht, herauszufinden, ob er sich hinter einem NAT Device befindet und wie dieses arbeitet.

Folgende NAT Typen werden unterstützt:

1. Kein NAT
2. UDP wird vollständig blockiert
3. UDP wird nur zur Quelladresse eines vorausgegangenen UDP Paketes erlaubt
4. Statische IP und Port Übersetzung, eingehende Pakete sind erlaubt
5. Ausgehende Quelladresse hängt von interner Quelladresse und Zieladresse ab
6. Statische IP und Port Übersetzung, eingehende Pakete sind nur als Antwort auf ausgehende Pakete möglich (abhängig von IP oder IP + Port).

STUN Test 1

- Der Client schickt einen STUN Request, der vom Server zur Senderadresse zurückgeschickt wird.
- In der Antwort enthalten sind die QuellIP und der Quellport, die der STUN Server gesehen hat.
- Erhält der Client keine Antwort, ist keine Kommunikation über UDP möglich.
- Sind Quelladresse und Quellport im Paket identisch mit der lokalen Adresse, wird kein NAT verwendet.

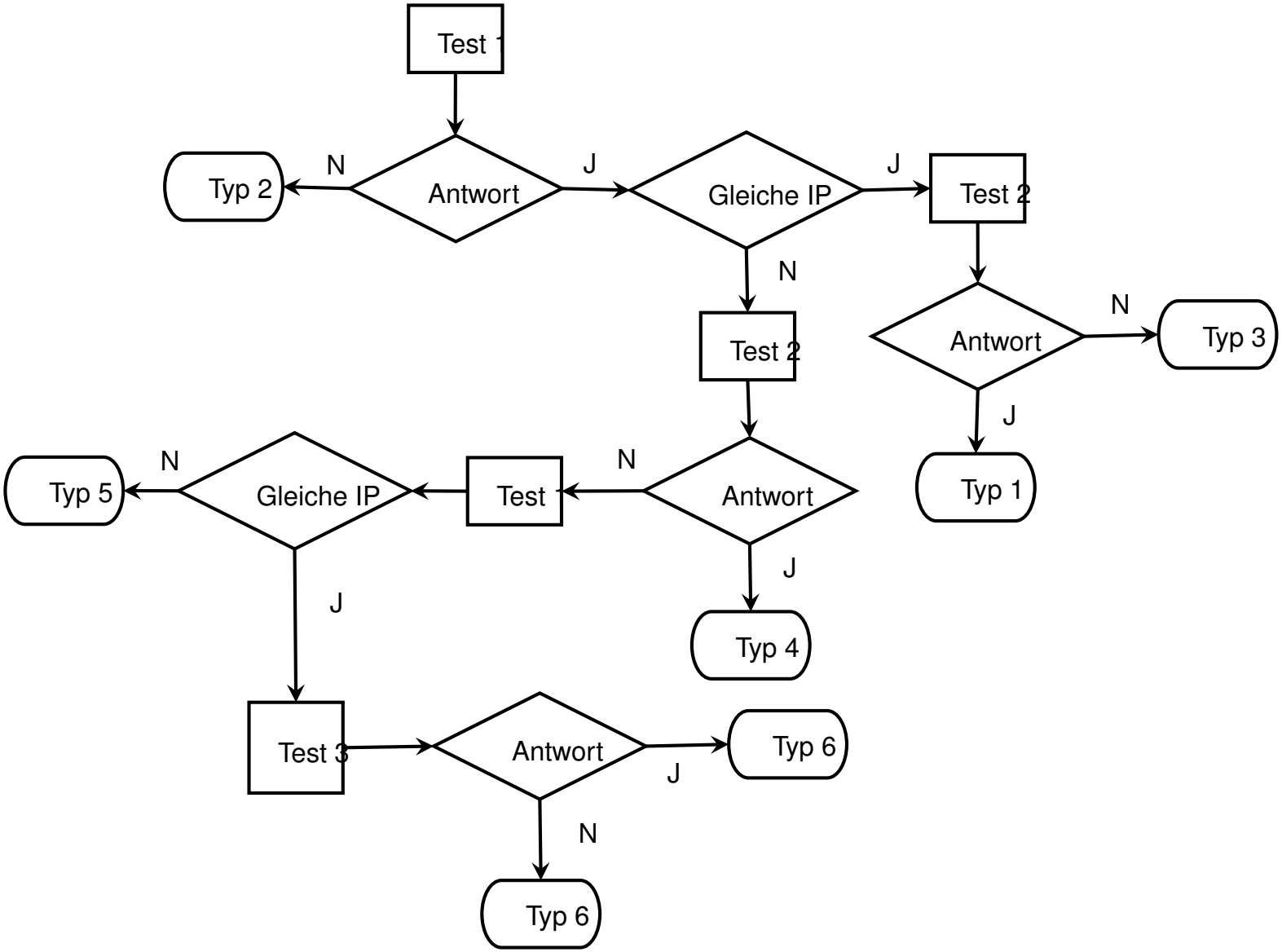
Test 2

- Der Client schickt einen STUN Request mit der Aufforderung, von einer anderen IP und einem anderen Port zu antworten.
- Wird eine Antwort empfangen, gibt es ein statisches Mapping zwischen internen und externen Adressen. Ports werden nicht gesondert gefiltert (sofern nicht vorher ein Request auf diese Adresse geschickt worden ist).
- In Verbindung mit Test 1 können die NAT Typen 1-4 damit erkannt werden.

Test 3

- Der Client sendet einen STUN Request nur mit der Aufforderung, von einem anderen Port aber derselben IP zu antworten.
- Wird eine Antwort empfangen, sind Ports von einer IP Adresse zumindest dann erreichbar, wenn zu dieser IP schon ein UDP Paket gesendet worden ist, unabhängig zu welchem Port.
- Wird keine Antwort empfangen, filtert das NAT Element mindestens nach verwendeten Portpaaren.

STUN Workflow



Beispiel

Test mit dem Vovida STUN Client aus einem öffentlichen WLAN:

```
#stun stunserver.org -v
test I = 1
test II = 1
test III = 1
test I(2) = 1
is nat = 1
mapped IP same = 1
preserver port = 1
Primary: Independent Mapping,
   Independent Filter, preserves ports
Return value is 0x000003
```