[Proceeding] Providing End-to-End Connectivity to SIP User Agents Behind NATs

Providing End-to-End Connectivity to SIP User Agents Behind NATs

Mario Baldi, Luca De Marco, Fulvio Risso, Livio Torrero
Dipartimento di Automatica e Informatica,
Politecnico di Torino, Torino, Italy
{mario.baldi,fulvio.risso,livio.torrero}@polito.it, luca.demarco@studenti.polito.it

***Abstract*— The widespread diffusion of private networks in SOHO scenarios is fostering an increased deployment of Network Address Translators (NATs). The presence of NATs seriously limits end-to-end connectivity and prevents protocols like the Session Initiation Protocol (SIP) from working properly. This document shows how the Address List Extension (ALEX), which was originally developed to provide dual-stack and multi-homing support to SIP, can be used, with minor modifications, to ensure end-to-end connectivity for both media and signaling flows, without relying on intermediate relay nodes whenever it is possible.**

Index Terms- NAT, SIP, ALEX, STUN, ICE, hole punching.

I. INTRODUCTION

The number of hosts connected to the Internet grows day by day making public Internet addresses a scarce and precious resource. The shortage of the IPv4 public addresses imposed the rationalization of address assignment policies and the use of IPv4 private addresses through the creation of private networks. This led to a wide adoption of Network Address Translators (NATs) [1][2]), which dynamically map the hosts of a private network over a restricted pool of public addresses and ports. The improved utilization of the IPv4 address space introduced by the deployment of NATs limited the need for new IPv6 addresses. However, NATs were developed with a client-server paradigm in mind: a private client can contact a public server and obtain the required services. For this reason the deployment of NATs highly limits the end-to-end connectivity of all the applications that use different paradigms (e.g., peer-to-peer and multimedia software). This article focuses on establishing direct sessions (particularly SIP [3] and RTP [4] media ones) between two UAs, even in the presence of NATs. The goal of this article is to prove that a slightly modified version of the SIP protocol including ALEX (Address List Extension [5]) can achieve this. ALEX has been formerly introduced as a solution to support dual-stack UAs and multi-homed UAs as well, making it possible to the choose at runtime the best network addresses and ports to use for communication: the extension proposed here adds the support to UAs in NAT controlled networks. The idea is that each UA behind a NAT can be considered a multi-homed host since it has at least two network addresses when it tries to communicate with an external node: its private network address and a public network address dynamically assigned by the NAT. Note that it is not possible to know a priori which of these network addresses should be best used to communicate: for example, if two UAs are behind the same NAT, the best choice would be to use internal network addresses and ports. These considerations suggest that ALEX does not need deep changes to be an optimal solution also for NAT traversal, in order to provide an integrated solution for end-to-end connectivity in a large variety of network scenarios. This paper presents the modification to ALEX required to support NAT traversal and the results that come from its experimental evaluation. This paper is structured as follows. Section II explains why NATs may affect communications between hosts and how to overcome the problem. In the same section ALEX is introduced. Section III covers the related works. Then section IV explains how ALEX has been modified to support hosts in NAT controlled networks. Section V discusses the experimental results obtained comparing a modified softphone supporting ALEX to other reference softphones. Section VI summarizes the results obtained and introduces future directions.

II. BACKGROUND

A. NAT operations and traversal

The Network Address Translator (NAT) is a function, most often implemented in edge routers, that changes the source network address/port of outgoing packets (*base address/port* in NAT terminology), with new ones (*reflexive address/port* in NAT terminology). The inverse operation is performed on the destination network address/port of incoming packets. One of the main applications of NAT is allowing a host with a private address to communicate to hosts in to the public Internet by substituting the private address (i.e., the base address) with a public one (reflexive address) before packets transit from the private part of the network to the public one. When two hosts placed in networks controlled by separate NATs want to start a session, first of all they have to determine their respective reflexive addresses/ports in order to be mutually reachable: the STUN [6][7] mechanism achieves this goal. The private host discovers its network public address/port by sending a STUN Binding request to a special public entity called STUN server. When the STUN server receives the request that has possibly traversed a NAT, it simply copies the source network address/port of the request in the payload of the related STUN Binding Response. In this way, the private host can extract its public network address and port pair from the payload of the Binding response. The public network address and port obtained are called *server reflexive address and port*

(according to [8]). The mutual knowledge of the respective server reflexive addresses and ports does not ensure that the hosts are able to communicate: indeed NATs typically drop sessions started from external hosts to prevent network attacks. To establish a communication, both the hosts have to start the session making each NAT believe that the initiator is the host in the internal network: by doing this each NAT will create a temporary binding with the remote host, thus allowing incoming packets delivery. The procedure that results in the creation of such bindings will be referred in the following as connectivity establishment between hosts. Depending on the NAT policies, there are two mechanisms to achieve this result: *hole punching* [9] and *relaying* [9]. The *hole punching* technique tries to establish direct bidirectional flows by exchanging the server reflexive addresses and ports associated to the two hosts through an external support node and start send each other probe packets, using the reflexive addresses and ports as targets: since both the hosts are senders, the NATs will create the bindings (or "holes", according to the hole punching terminology). Typically *hole punching* implementations consist in the integration of STUN servers directly in the hosts and the probe packets are STUN Binding Requests. In this case, the reflexive addresses and ports contained in the related STUN Binding Responses are referred as *peer reflexive addresses and ports* [8].

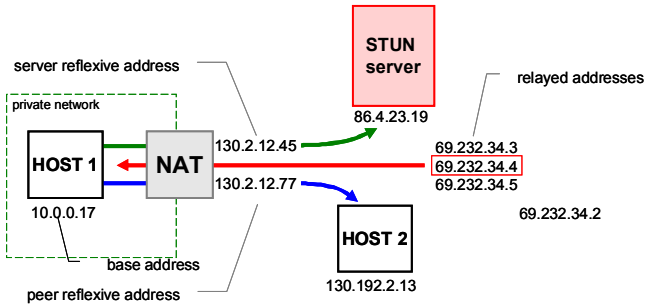

Figure 1. Example of base address and derived addresses.

It is important to notice that, depending on the NAT type, the server reflexive address/port may not coincide with the peer reflexive address/port as the NAT might assign different public network addresses/ports to a host when it sends packets to different destinations. Furthermore if both the hosts are placed in networks controlled by NATs implementing the policy described, the hole punching will fail, since there is no chance to deliver the probe packets (both the server reflexive addresses/ports are different from the peer reflexive ones). If this is the case, the only solution available is *relaying*: an external public node with relay capability is required to exchange packets. This functionality has been recently integrated in the STUN entity [10] (i.e. *STUN relay*). An internal host can request one of the network address and port pairs of the relay (called *relayed address and port*) by sending a STUN Allocate Request message to the STUN relay, which returns the chosen network address and port in the related STUN Allocate response. From that point on, each packet received by the STUN relay on the relayed address and port will be forwarded to the host. When the session has been established, if for some reason there is no traffic between the hosts for a given period, the NATs may erase the temporary bindings: to refresh them the hosts have to exchange periodically *keep-alive packets* (again this is typically done exchanging STUN messages).

Server reflexive, *peer reflexive* and *relayed addresses* will be referred in the following as *derived addresses*, meaning that they do not belong to any interface of the host, but they are mapped to it. Figure 1 shows an overview of base and derived network addresses.

B. *ALEX*

In [5] we have defined an extension to the SIP protocol called ALEX, providing full dual-stack and multi-homing support for both signaling and media flows. This extension allows a UA to send all its IPv4 and IPv6 addresses to a remote UA in order to discover which network address and port pair (or *network endpoint*, according to ALEX terminology) is the best choice. Indeed standard SIP UAs can announce only a network endpoint for the signaling flow and one for each media flow: so they have to make the best choice a priori. On the contrary ALEX defines a new SIP header field called *ALEX-item*, which consists in a network address with the related ports to be associated to a flow (either a SIP session, or audio/video media), together with a priority value. The ALEX-items allows the UAs to exchange all their network endpoints in order to choose the most feasible ones at runtime. ALEX defines four steps executed by each UA during session establishment, as exemplified in Figure 2.

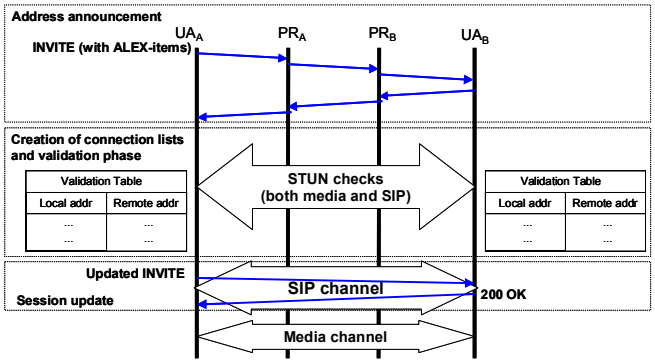

Figure 2. Dialog establishment using ALEX.

The first one consists in *gathering network endpoints*: both the UAs collect all the network endpoints they can use to send and receive IP packets. Then the UAs *exchange network endpoints* placing them in the ALEX-items inside the SIP messages. The third step is the *creation of the validation tables*: each UA pairs each one of its endpoint with the ones obtained by the ALEX-items received by the other UA: these endpoints pairs are referred as *candidate channels* and are the entries of the *validation tables*. Validation tables on both UAs contain the same data. Concluding the last step is the *validation step*: this step consists in checking the candidate channels using STUN messages to verify connectivity: the channels with the highest priority successfully checked are referred as *optimal channels*. ALEX stores the optimal channels in the ALEX cache: these will be checked first during subsequent session establishments. Tests performed in [5] comparing a standard UA with an ALEX-enabled UA demonstrated the effectiveness and the limited overhead of ALEX. However, one of the assumptions of ALEX is direct connectivity between UAs, such as in case

UAs have public addresses. Therefore, ALEX does not work in other cases, e.g., when one UA is behind a NAT.

III. RELATED WORK

Several proposals have been done to enable NAT traversal for signaling and media flows.

NAT traversal for signaling flows is related to the establishment and maintenance of SIP dialogs[1] between UAs that share at least one NAT in their path. Two separate problems are to be faced: the delivery of out-of-dialog SIP messages and the delivery of mid-dialog ones. Out-of-dialog messages are SIP messages used to establish a dialog (i.e. the INVITE message that creates a media session). Mid-dialog messages are used to update an existing dialog or to terminate it (e.g., the BYE message closing a media session between two UAs). The delivery of out-of-dialog requests implies that a UA be reachable by any other UA that can start a dialog. The solution proposed by the IETF [11] is based on the fact that in a typical SIP infrastructure, SIP UAs send out-of-dialog requests through support nodes called outbound (or edge) proxies. The first message sent by a UA is a REGISTER message: the request reaches a node called *registrar* that stores the registering information. According to this solution, since multiple registrations through different outbound proxies are possible, multiple *flows* (i.e., bidirectional streams of datagrams over UDP or TCP) towards the proxies can be created. These flows will be reused to deliver all the incoming out-of-dialog messages to the UA, thus placing proxies in the path of these messages. For mid-dialog messages, the solution proposed in [12] exploits the fact that usually SIP UAs exchange directly these messages using the addresses placed in the *contact* header of the SIP messages that created the dialog: the problem is that such addresses may not be globally routable when NATs are on the path of the messages. The proposed solution consists in adding a *record-route* field to the SIP header of each message forwarded: this field forces the UA to send all the mid-dialog messages through an edge proxy. Since the edge proxies may fail due to excessive load, the Registrar should stay in the path as well: if an edge proxy goes down, the registrar redirects the messages to an edge proxy that has an active flow with the UA.

For the problem concerning NAT traversal of media flows, typically SIP uses SDP (Session Description Protocol) [13] for the negotiation of media-flow parameters and RTP (Real-Time Transport Protocol) for the delivery of the media flows themselves. During the SDP negotiation private UAs announce their network private endpoints. In this fashion the establishment of incoming RTP media flows is not possible. Interactive Connectivity Establishment (ICE) [8] is the state-of-the art solution to ensure the establishment of media flows. ICE is not intended for signaling flows since ICE extends only SDP. Each UA must discover its server reflexive addresses/ports and must obtain at least a relayed address/port: these addresses and ports are inserted in new SDP fields called candidate fields, used by ICE to announce them. Both the UAs start the hole punching procedure using the candidates exchanged to open a path through the NATs.

The biggest problem of these solutions is that often signaling messages are still exchanged through relays even when direct connectivity is available (with an increased and unnecessary load on proxy servers). Our solution will overcome this problem.

IV. OPERATING PRINCIPLES

The aim of this paragraph is to discuss in detail how ALEX has been modified to support NAT traversal for both SIP and media flows: although this is a modified version of the original ALEX protocol presented in [5], will be referred with the same name for the sake of clarity. The new ALEX is still based on the four steps discussed in Section II.B, but some of them need to be modified in order to support new functionalities, which become active only when NAT traversal is required, therefore ensuring compatibility with traditional ALEX UAs.

A. Gathering network address and port information

Before trying to establish a SIP and/or multimedia session, a UA must collect all the addresses/ports that can be useful to establish the communication with other endpoints. If a UA is behind a NAT, its private address can be used only to communicate with other UAs in the internal network. To ensure connectivity, the UA must obtain at least a server reflexive address and port pair for each flow, plus a relayed address and port pair as a "fallback" solution in case direct connectivity is not possible: typically NATs assign the same network address to each flow and simply change the port. To reduce the overhead on STUN relays, the UA must gather no more than one relayed address and port pair for each flow. The gathering phase should be repeated every time a new SIP dialog is going to be established. These addresses (and ports) can be seen as "virtual addresses" (and ports) associated to the host, which therefore becomes a sort of multi-homed host (i.e. it is associated to more than one address and port). Moreover, as the discovering and management of all the available addresses of a UA is one of the key features of the NAT traversal, it is a logical consequence to think that ALEX can be beneficial in establishing optimal signaling and media flows among NATted user agents.

B. Format of the ALEX-item field in SIP headers

This section describes the changes in the format of the ALEX-item field needed to support NAT traversal.

Figure 3 shows two sample ALEX-item fields. The new parameters are shown in italic bold. The first new parameter is *seq*: it is a sequence number increased by one each time the gathering procedure is repeated in order to point out the most updated network addresses and optimize the validation step. The second new parameter is *addr-type*, which specifies the type of the network address announced in the ALEX-item. The possible values of this parameter are *base*, *relayed*, *srflx* and *prflx*. *Base* is referred to a network address associated with an interface of the UA. *Relayed*, is meant for a relayed address, while *srflx* is intended for a server reflexive address. The last admitted value is *prflx*, standing for *peer reflexive address*. If

[1] Dialogs are end-to-end relationships between UAs, established to exchange SIP messages.

two UAs have established a previous SIP dialog, thus discovering peer reflexive addresses/ports different from their server reflexive ones, the peer reflexive addresses/ports can be announced when the UAs attempt to create a new dialog. This may help to rapidly discover optimal channels. In the case of derived addresses the ALEX-item must include the base address as well: for such purpose the *addr* parameter is followed by the *base* one, containing the base address.

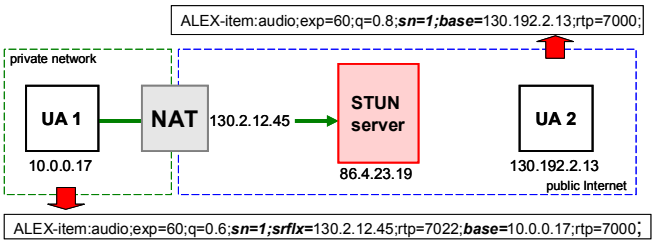

Figure 3. ALEX-item format.

C. *Address priority*

ALEX defines a priority field used to rank the gathered network endpoints. The highest priority will be assigned to network endpoints containing base addresses, permitting a direct connection in case either no NAT is present, or two internal UAs want to communicate. A lower priority value will be assigned to the network endpoints containing server reflexive addresses, followed by the ones containing relayed addresses. However, these ones will be the default network endpoints, meaning that they will be used to deliver packets temporarily until the validation step is completed.

D. *Creation of the validation tables*

UAs create validation tables following the standard procedure defined by ALEX. The only change is that each derived address/port is stored with the related base address/port.

E. *Address validation step*

The address validation step starts as soon as the validation tables are ready. It consists in the reiterated execution of a channel probing procedure based on the mutual exchange of STUN Binding Request and STUN Binding Response messages between the UAs. The goal is to ensure that both UAs have a chance to send a STUN Binding Request message and to receive related response directly from the other UA, thus punching a hole through NATs (if they are present) and informing the UAs that the channel is available. Considering for example the situation depicted in Figure 4, UA1 starts sending a STUN Binding request message and, when UA2 sends back the related response, UA1 concludes that the channel is operative. Notice that UA2 is not aware of UA1's conclusion and consequently UA2 anyway sends a STUN Binding Request to UA1. When UA2 receives the related answer, UA2 knows that the channel can be used. In order to cope with NATs (if present), both UAs should ideally send STUN Binding Requests simultaneously. If these hosts are behind two different NATs, this allows the STUN requests opening "holes" through each NAT, thus making the delivery of the STUN Binding Responses possible. In order to approximate a simultaneous transmission, each of UA1 and UA2 sends STUN Binding Requests as soon as it realizes it needs to open a media session with the other one.

Figure 4. User Agents validating a channel.

The finite state machine depicted in Figure 5 implements the connectivity check procedure. If a UA receives the initial STUN Binding Request, the connectivity check moves to the *receive and transmit* state: the other endpoint can send packets without fear of being dropped by a NAT. Then the UA sends back the related STUN Binding Response together with another STUN Binding Request: if the response arrives, the UA knows that the channel is operative in both directions, thus moving to the *confirmed* state. On the other hand, if a UA sends a STUN Binding Request before receiving one, the connectivity check transitions to the *receive* state. If the UA is behind a NAT, the STUN message has opened a channel toward the other endpoint and the UA is expecting to receive packets through such channel. When an answer is received it shows that the other endpoint can receive packets from the UA, which in turn can receive the other endpoint's messages: a transition to the *confirmed* state occurs.

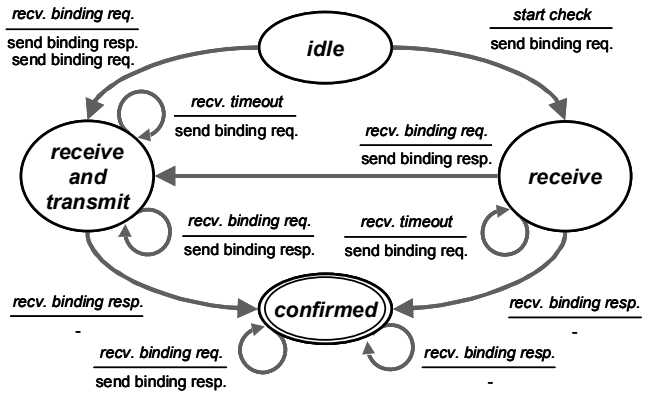

Figure 5. FSM handling the connectivity check procedure.

If a STUN Binding Request is received while in the *receive* state, a STUN Binding Response is sent and a transition to the *receive and transmit* state occurs. In the original ALEX specification, i.e., when NATs are not involved, the only reason for packet retransmission was network loss. When dealing with NAT traversal, packets might be dropped by NATs: potentially more STUN message retransmissions are required to probe each channel. To reduce the delay due to multiple retransmissions, the validation step has been split in two sub-steps: the *real-time validation step* and the *background validation step*. The *real-time validation step* aims at discovering optimal channels as soon as possible: STUN messages are resent quickly and up to three times in case of missing answers. The *real-time validation step* begins probing the default channels (i.e. channels made up by default network endpoints) and stops as soon as an optimal channel has been discovered or when a dedicated timer fires. Upon timer

expiration, the *background validation step* begins and concurrently both SIP and media sessions start using the best channels successfully checked. The *background validation step* is needed to probe again high priority channels that may have not been successfully checked in the *real-time validation step* or to probe the remaining channels that have not been tested yet because of the timer expiring. The goal of the background step is to discover optimal channels without delaying the beginning of the SIP and media communication and in a less aggressive way, in order to limit additional network load due to connectivity checks. The retransmission intervals of STUN messages in the background step are longer than in the real-time step.

F. The ALEX cache

To fully support NAT traversal, each entry of the ALEX cache contains an additional tag to keep its current state. At first the state of an optimal channel is set to "live". Periodically the UA sends a keep-alive message on the channel to ensure that it is still open: this is done at least until there is a session involving it. If no keep-alive messages are sent for a given period, the state of the channel is set to "probed": the channel is no longer active, but it is kept in the cache as a hint for the establishment of subsequent sessions.

G. Using the SIP channel to probe media flows

As it is likely to happen, the real-time validation step concludes with the discovery of an optimal SIP channel before the `200 OK` SIP message is sent. The information gathered can be used to reduce the number of STUN messages needed to detect optimal media channels. This is done setting the priority fields of the ALEX-items related to media flows placed in the `200 OK` answer. For example if the optimal SIP channel included a relayed address, the highest priority is assigned to the ALEX-item containing a relayed address for each media flow.

V. Experimental Results

To evaluate the effectiveness of ALEX for NAT traversal, ALEX has been implemented in the OpenWengo NG [14], the same UA that has been used in [5] to demonstrate and assess the first ALEX version for multi-homing support. During the validation phase each STUN binding request is retransmitted up to three times if an answer is not received. The modified OpenWengo NG has been tested against two reference UAs: CounterPath X-lite [15] and PJSUA [16]. CounterPath X-lite is one of the most complete and easy-to-use existing UAs. X-lite supports both STUN and ICE functionalities, but ICE support was disabled during the tests since its implementation refers to an obsolete specification. PJSUA has been used as the reference UA to test ICE functionalities against ALEX. This UA is a sample textual application that is part of the PJSIP project. The tests performed aim at (*i*) demonstrating that ALEX provides connectivity between UAs independently of the kind of NAT present in the path (*ii*) comparing ALEX performance in terms of network overhead required to identify the optimal channel with the one of the alternative solutions. In order to define our test scenarios we analyzed the different typologies of NATs [17]. For instance, an application-friendly NAT always assigns the same reflexive address/port to an internal host regardless of the remote target. Furthermore, this NAT has a deterministic behavior, meaning that its behavior does not change without explicit reconfiguration. Entry-level NATs typically implement this policy since it is the least resource consuming. For the sake of completeness more complex NAT implementations with a non-deterministic behavior are also used in the experiments. Most of the times, these NATs may assign different reflexive addresses/ports when forwarding packets to different remote targets. All the NATs considered during the tests implement address and port-dependent filtering policies. Using these NATs the following test scenarios, depicted in Figure 6, are considered in this work (*i*) UA1 and UA2 are in the same internal network, (*ii*) UA2 is behind an non-deterministic NAT, (*iii*) UA2 is behind an non-deterministic NAT, (*iv*) UA1 and UA2 are behind two different application-friendly NATs, (*v*) UA1 is behind an application-friendly NAT while UA2 is behind a non-deterministic NAT and (*vi*) both UAs are behind non-deterministic NATs. OpenWengo NG, X-lite and PJSUA are tested in each scenario trying to establish 20 subsequent media sessions, in order to be able to significantly average the obtained measurements. The UAs are executed on hosts connected to the Internet through ADSL connections at 4 Mbit/s (384 Kbps in download) and NATs, when deployed, are at the boundary of a private network (internal) and the public Internet (external). UAs established only media sessions consisting of a single audio flow, without any RTCP flows. The average end-to-end round trip time between the UAs is about 163 ms. SIP proxies, STUN servers and STUN relays are placed on the public Internet. The RTT from the UAs to the STUN server and the STUN relay is about 70 ms.

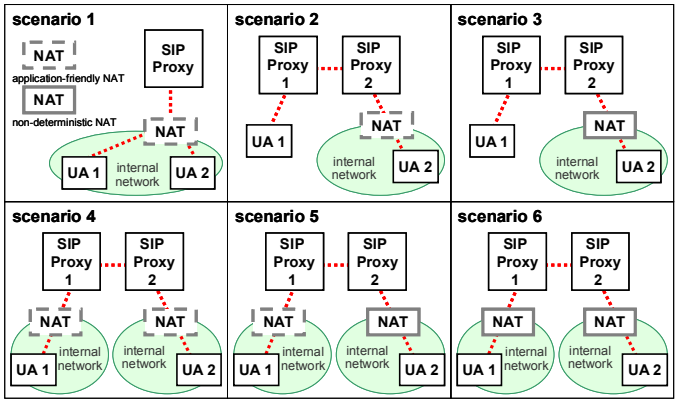

Figure 6. Scenarios used for the tests.

A. X-lite

In a first set of tests the Record-Routing functionality is disabled on the proxies. X-lite performs standard STUN operations to discover a network endpoint that can be registered to receive out-of-dialog messages. While this is enough in scenario 1, in scenario 2 and 3 session establishment fails when UA1 is the caller. In fact, the ACK message cannot be delivered directly to UA2 since the NAT drops it. Furthermore, session establishment fails also in the remaining scenarios as the ACK cannot be delivered in any case since there are multiple NATs on the path between the UAs. The only way to complete the session establishment is to have the

record-routing functionality enabled on the proxies: this solution guarantees the delivery of all SIP messages, but it does not solve the problem of media sessions whose packets are usually dropped (depending on the typology of NAT). Since both UAs send each other RTP packets simultaneously, holes through the NATs might be successfully opened, thus allowing the delivery of subsequent media packets (although the first media packets are lost): this is possible only if the NATs are application-friendly. This may work in scenario 2 and 3, but does not work in scenario 4, 5 and 6: so this approach cannot be considered a proper solution.

B. PJSUA

The SIP signaling considerations done above for X-lite are valid for PJSUA as well. Since PJSUA supports the ICE extension, direct media connectivity is possible in the first four scenarios: the STUN messages exchanged by the UAs open direct channels through the NATs. This does not apply to the remaining two scenarios because PJSUA does not support relayed sessions: since there is at least a non deterministic NAT on the path, direct media connectivity is not possible.

C. ALEX-enabled OpenWengo NG

Differently from the other UAs, the ALEX-enabled OpenWengo NG tries to open a direct SIP channel: in the first four scenarios this results in direct connectivity for both SIP and media flows. Out-of-dialog SIP messages (like the initial INVITE) still need to be delivered through a proxy that can reach the UA, while mid-dialog SIP messages are exchanged without intermediaries. In scenarios 5 and 6 the situation is more complex because it is not possible to establish a direct communication due to the non deterministic behavior of NATs. In scenario 5 UA2 hides the presence of a NAT announcing a relayed network address and port. In this fashion UA2 can announce itself as a sort of "virtual" UA on the public Internet and, because of this, UA1 has only to punch a hole through its NAT contacting the relay. In scenario 6 both the UAs behave like UA2 did in the previous scenario: to ensure connectivity both the UAs have to announce relayed network addresses and ports in order to masquerade the presence of non deterministic NATs, thus media and signaling flows are delivered to the relay present on the public network.

D. Performance evaluation

Table 1 shows the results obtained measuring the time overhead introduced by ALEX, when compared to the behavior of a standard SIP UA. The first column shows the validation time for the SIP channel, the second one shows the same value for the media flow and the last one displays the latency reduction (in terms of end-to-end delay) measured when SIP messages are exchanged directly instead of being delivered through a SIP proxy.

The time spent in the validation phase of the SIP channel is about 900 ms in scenario 1, 5 and 6 since UAs retransmit every unanswered STUN binding request up to three times every 300 ms. In scenario 1 multiple retransmissions occur because server reflexive addresses and ports cannot be checked. The cause is the lack of hairpin translation [9] support on the NAT (i.e. the NAT drops sessions between hosts in the same internal network if they use server reflexive addresses and ports instead of the internal ones). In scenario 5 and 6 multiple retransmission are due to NAT policies. The values in the second column are lower because the SIP channel properly works as a probe channel, i.e., the time needed to discover an optimal channel for media flows is reduced by using information already gathered during the identification of the optimal SIP channel. In scenario 5 and 6 the time spent to check the media flow is about 70 ms: this time is needed to send a STUN allocate request to the STUN relay and to receive the related response, since media packets have to be exchanged using a relay.

TABLE I. ALEX PERFORMANCE EVALUATION

Scenario	validation time [ms]		latency reduction
	SIP	Media	
1	867	34	50 %
2	207	156	23.38 %
3	192	167	25.76 %
4	198	135	22.41 %
5	874	70	23.11 %
6	891	75	23.09 %

Table 2 shows the results obtained comparing OpenWengo NG to PJSUA, i.e., comparing ALEX and ICE. Specifically, the lefthand half of the table shows the results obtained when UA1 sends the INVITE message that commences the session, while the righthand half shows the results obtained when the roles are inverted. The first two columns in each half display the number of STUN messages exchanged respectively during the ALEX validation step and the ICE validation phase.

TABLE II. COMPARING ICE AND ALEX

Scenario	#of STUN messages		Overhead (% in bytes)
	ALEX	ICE	
1	14	6	-1.34 %
2	9	6	-12.34 %
3	11	4	+1.9 %
4	8	4	-6.83 %

Note that the number of messages exchanged during the ALEX validation step is higher since the algorithm implementation uses multiple retransmissions during the real-time validation step, while the ICE implementation does not. Furthermore the algorithm used by ALEX is slightly more complex since both UAs send STUN binding requests, which is not always true for the validation algorithm used by in ICE. The last column shows the overhead of ALEX computed as the number of additional bytes exchanged by ALEX when compared to ICE, over the total number of bytes exchanged by ICE (i.e. the overhead includes the size of ALEX-items as well as the one of STUN messages). The overhead observed for ALEX is limited when compared to ICE mainly because the format of ALEX-items is more compact than the format of the candidate fields included in ICE. Indeed each STUN binding

request has a transaction identifier used to correlate it with the corresponding STUN binding response. ICE has to store a parameter specific for such purpose in each candidate field. On the contrary ALEX computes this identifier from the identifier of the SIP dialog: since ALEX is an extension of SIP protocol, it is easy to exploit dialog-related data structures. The overall byte overhead is not affected by the STUN messages because of their limited size compared to SIP messages. Scenario 5 and 6 have not been considered since PJSUA does not support relaying.

VI. CONCLUSIONS AND FUTURE DIRECTIONS

NAT traversal techniques for media session establishment typically ensure direct communication only for media flows, relegating the delivery of SIP messages (both out-of-dialog and mid-dialog ones) to SIP proxies according to the solution currently proposed within IETF [11][12]. ALEX moves the state information needed to route messages from the proxies to the UAs. Moreover, in some cases ALEX ensures the delivery of out-of-dialog messages using relayed addresses obtained from a STUN relay. Furthermore, since the delivery of mid-dialog messages is completely handled by the UAs, the registrar does not need to elaborate and forward all the mid-dialog messages while edge-proxies are no longer necessary to make the SIP session fault-tolerant. In addition to that, end-to-end sessions enable a significant reduction of network latencies and ALEX UAs succeed in establishing communication in all the network scenarios considered, while the ICE based UAs used during tests considered do not. The overhead introduced by ALEX validation step is negligible because of the small dimension of STUN messages. Thanks to ALEX effectiveness the need to have intermediate nodes in the path of the messages to ensure connectivity is considerably reduced, bringing the SIP protocol back to its original *centralized peer-to-peer* paradigm.

Future work includes the deployment of ALEX in a *peer-to-peer infrastructure* able to provide distributed relaying functionalities, which can be included directly in UAs. This will help to guarantee connectivity in every situation, while providing scalability since as the number of the UAs grows so does the number of potential relay nodes.

A prototype of our solution is freely downloadable from our research web site, http://netgroup.polito.it.

REFERENCES

[1] K. Egevang, P. Francis, "*The IP Network Address Translator*," http://www.ietf.org/rfc/rfc1631.txt, RFC 1631, May 1994.

[2] P. Srisuresh, M. Holdrege, "*IP NAT Terminology and Considerations*," http://www.ietf.org/rfc/rfc2663.txt, RFC 2663, Aug. 1999.

[3] J. Rosemberg et al., "*SIP: Session Initiation Protocol*," IETF Network Working Group, http://www.ietf.org/rfc/rfc3261.txt, RFC 3261, June 2002.

[4] H. Schulzrinne et al., "*RTP: a transport protocol for real time Applications,"* IETF Network Working Group, http://www.ietf.org/rfc/rfc3550.txt, RFC 3550, July 2003.

[5] Mario Baldi, Fulvio Risso, Livio Torrero, "*Adding Multi-Homing and Dual-Stack Support to the Session Initiation Protocol," Internet Protocol Symposium, Washinghton D.C., USA, november 2007*

[6] J. Rosenberg, J. Weinberger, C. Huitema, R. Mahy, "*STUN - Simple Traversal of User Datagram Protocol (UDP) Through NATs*," http://www.ietf.org/rfc/rfc3489.txt, RFC 3489, Mar. 2003.

[7] Rosenberg, J., "*Session Traversal Utilities for NAT (STUN)*," http://tools.ietf.org/html/draft-ietf-behave-rfc3489bis-06, Oct. 2006.

[8] J. Rosenberg, "*Interactive Connectivity Establishment (ICE): A Protocol for NAT Traversal for Offer/Answer Protocols,"* http://tools.ietf.org/html/draft-ietf-mmusic-ice-18, September. 2007.

[9] Bryan Ford, Pyda Srisuresh, Dan Kegel, "*Peer-to-Peer Communication Across NATs,*" USENIX 2005

[10] J. Rosenberg, R. Mahy, C. Huitema, "*Traversal Using Relays around NATs (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)*," http://tools.ietf.org/html/draft-ietf-behave-turn-04, July 2007.

[11] C. Jennings, Ed., R. Mahy, Ed., "*Managing Client Initiated Connections in SIP*," http://tools.ietf.org/html/draft-ietf-sip-outbound-10, July 2007.

[12] K. Johns, "*Routing of mid dialog requests using sip-outbound*," http://tools.ietf.org/html/draft-johns-sip-outbound-middialog-draft-02, Jan. 2007.

[13] M. Handley, V. Jacobson, "*SDP: Session Description Protocol*," http://www.ietf.org/rfc/rfc2327.txt, RFC 2327, Apr. 1998.

[14] The OpenWengo Project. Available at http://www.openwengo.org.

[15] CounterPath X-lite. Available at http://www.counterpath.com.

[16] Pjsip.org. Available at http://www.pjsip.org.

[17] F. Audet, Ed., C. Jennings, "*NAT Behavioral Requirements for Unicast UDP,*" http://www.ietf.org/rfc/rfc4787.txt, RFC 4787, Jan. 2007 .