Adding Multi-Homing and Dual-Stack Support to the Session Initiation Protocol

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

Abstract— **Although the SIP protocol claims a complete dual-stack support, some aspects, such as interoperability between different address realms and support for multi-homed hosts, are not taken into consideration. This leads to an extensive usage of proxies as gateways, e.g., between different address realms. ALEX ("Address List Extension") is a simple extension to the SIP header that addresses these limitations, providing additional scalability for SIP proxies and allowing the establishment of direct channels between peers, while still guaranteeing backward compatibility with traditional SIP implementations.**

Keywords-component; SIP, IPv6, transition, ALEX, ICE

I. INTRODUCTION

The Session Initiation Protocol [1] (SIP) has been originally developed with a peer-to-peer approach in mind. In the original proposal, a SIP infrastructure is needed only during session establishment, while subsequent SIP messages should be exchanged directly by the two endpoints of the session, called SIP user agents (UAs). However, recent practice is increasingly oriented toward an extended usage of intermediate nodes (SIP proxies) also during the established phase of a SIP session to overcome direct connectivity problems (e.g. UAs with private addresses or dual stack hosts), which results in the protocol operating sub-optimally according to a client/server approach. This is usually accepted due to the low bandwidth required by SIP messages compared to media flows; in fact, most of the work is done for optimizing media sessions rather than SIP session establishment. However, as more applications (e.g. e-presence) are based on SIP signaling, the amount of SIP messages in the network may no longer be negligible.

This paper presents and validates an extension to SIP called Address List Extension (ALEX) that enables direct connectivity for both signaling messages and media flows. This extension is especially targeted to dual-stack and multi-homed hosts since it guarantees better network usage and reduces proxy scalability problems.

This paper is organized as follows. Section II discusses the motivations for such an extension, alternative approaches, and other related work. Section III provides an overview of ALEX and discusses the architectural modifications it brings to the SIP stack. A first implementation is presented in Section IV together with some experimental results. Section V draws some conclusions and outlines future work directions.

II. MOTIVATIONS

The introduction of new generation networks based on IPv6 is creating some problems concerning the interoperability with existing IPv4-only networks: this results in the extensive usage of gateways in order to ensure connectivity between these different address realms. In the SIP context, SIP proxies provide gateway functionality by being forced to handle all SIP messages through the insertion of a `record-route` field in the header of each message and UAs do no longer exchange messages directly. RFC 3261 [1] recommends a moderate usage to avoid performance (RTT increases due to the triangulation) and scalability (proxies are forced to process a huge amount of messages) issues. In fact SIP proxies should be used only to forward the initial messages that establish the session after locating the called party's UA, while all the other messages of the dialog should be exchanged directly.

ALEX has been developed to avoid limitations stemming from deployment of gateways: the idea is to make UAs smart enough to create a direct channel, thus reducing the load on proxies and improving latency for SIP and other (e.g., media) messages. Furthermore ALEX adds an efficient multi-homing support to SIP since it allows a UAs to know the complete list of available addresses, therefore choosing the most suitable pair for the session.

A. The problem with direct connectivity

RFC 3261 states that the *single* `contact` field included in the header of SIP request messages must contain a *single* URI (Universal Resource Identifier) providing information on how to contact the caller. Since most UAs do not have a DNS entry, such URI often contains a caller's IP address (more details have been presented in [3]). The choice of such address is non-trivial for a dual-stack UA, since it does not know whether the called peer and the network between them support IPv6. As a result, connectivity problems might arise even when a large portion of the network is dual-stack, e.g. the entire left-hand side network in Figure 1. For instance, if in this scenario UA_A uses an IPv6 `contact` field, UA_B is not able to send a SIP messages to UA_A directly, but PR_B must acts as a translator, such as in the bottom half of Figure 1. Vice versa, if UA_A used an IPv4 `contact` field, it would loose the possibility to establish direct sessions on IPv6 transport to any UA, because

the called peer cannot know that the calling UA has IPv6 support (neither its IPv6 address).

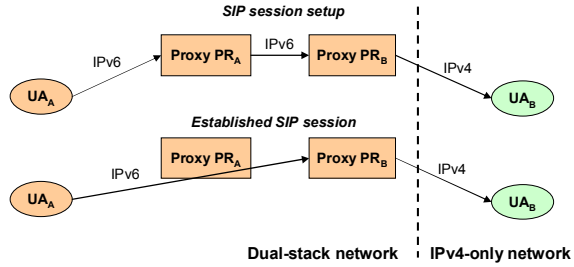


Figure 1. Example of SIP messages in case of dual-stack entities.

While the `record-route` field ensures connectivity between IPv4 and IPv6 networks, this results in additional overhead on PR_B since all SIP messages and possibly media packets are routed through it. In order to quantify the overhead due to the extensive usage of `record-route`, let's consider a sample scenario in which each user has N peers in the buddy list and a decentralized e-presence model is used (the only approach available, for example, in p2pSIP [11]). Each user subscribes the status of its N peers [4] (i.e., 4 messages for each peer: one SUBSCRIBE [2] plus one NOTIFY [2] and the related answers) periodically every SUBSCRIBE Refresh Time. Moreover, some additional messages are sent in case a peer changes its status (e.g. from "online" to "away").

TABLE I. VALUES USED FOR COMPUTING SIP PROXY OVERHEAD

Description	Value
SUBSCRIBE Refresh Time	600 sec (10 min)
Number of status changes per hour	2
Number of contacts in the buddy list	10
Number of users in the Service Provider realm	10 millions
Average SIP Message size	700 bytes

Using the values listed in Table I, the SIP proxy of a medium-sized operator (10 million subscribers) has to sustain a run-time traffic of 780K messages per second, equivalent to an approximate load of 4.4Gbps. It is interesting to note that in case SIP messages are sent directly between peers (i.e. only the messages required to establish the session are routed through proxies), the load on the Service Provider Proxy diminishes to 28K messages per second, equivalent to an approximate load of 156 Mbps (assuming that users establish the first session within a period of 2 hours).

Although decentralized e-presence is not mandatory and not implemented in some domains, this example demonstrates that in real networks the amount of SIP messages traversing these SIP proxies might be significant. Hence, the capability of exchanging SIP messages directly between peers can add a new degree of scalability to the SIP infrastructure. Furthermore, the direct exchange of messages reduces the problem of switching from a faulty SIP proxy to another one, with the obvious burden in terms of maintaining the status of the sessions. In summary, keeping the proxy outside the path of SIP messages is advisable unless there is a specific reason to do otherwise (e.g., monitoring all the signaling traffic).

B. *Multi-homing support*

One of the key features of IPv6 networks is multi-homing support: a host can have multiple IPv6 addresses thus ensuring connectivity across separate networks. Moreover, in future networks the usage of multiple interfaces could be a key to improve performance. However, SIP cannot take advantage of this because a UA can announce only a single address in the `contact` field. It would be desirable, instead, that a UA be able to discover an optimal path to a second UA through a specific interface.

C. *Related work*

SIP deployment with IPv6 is rather inefficient because it does not properly support the transition from IPv4 to IPv6 which relies on dual-stack hosts. Among the solutions proposed in the past, a first one proposed to send a dialog-creating request using the IPv6 protocol first. The problem was that the UA needs to be informed in case of failure, but no dedicated answer code (e.g., "unsupported address family") is available, even though a 3xx answer was considered a possible compromise. Another approach consisted in the UA sending an IPv6 request and an IPv4 one at the same time: at least one of the request was expected to reach the called party.

The current state-of-the-art solution to this problem uses two separate approaches for signaling and media forwarding: for SIP signaling, connectivity is ensured adding a `record-route` field to all the SIP messages of a session [6] so that SIP proxies act as gateways between the IPv4 and IPv6 networks; for media flows, the Interactive Connectivity Establishment (ICE) [7] mechanism is deployed.

ICE adds new fields to the Session Description Protocol (SDP) [5] that are used to negotiate the characteristics of a media session. These additional fields contain the complete list of addresses available to a UA to be used in a media session (both IPv4 and IPv6), a priority associated to each address, and a default address to use. ICE has a major limitation as it is applicable only to media flows, i.e., it does not provide direct connectivity for SIP messages. ALEX, although shares some ideas with ICE, overcomes this issue.

III. ALEX OVERVIEW

ALEX defines a new mechanism within the SIP protocol (while leaving companion protocols, such as SDP, unchanged) that allows direct message exchange and direct session establishment through the addition of a new field (`ALEX-item`) to the SIP header. Each `ALEX-item` contains a network address-port pair related to the new session (i.e. if the host has N network addresses, at least N `ALEX-items` will exist), which results in multiple ALEX items per message to provide information for both SIP signalling and media flows. To deploy the best application-layer connectivity, both UAs start probing all the possible connectivity solutions by exchanging STUN [8][9] messages. Connectivity checking in ALEX is similar to the one used in ICE; the key difference is that ICE aims only at establishing direct media connectivity, while ALEX creates also a direct SIP channel between UAs.

A. *ALEX item format*

`ALEX-items` are used to announce the host addresses and transport-layer ports available as endpoints of SIP and media channels. Currently, two types of `ALEX-items`, both sharing the same structure, are defined: one related to SIP flows and one related to media flows. An `ALEX-item` is constituted by two parts: (*i*) a basic block common to all `ALEX-items` that is used to identify the item and to specify the flow type and the network address, and (*ii*) one or more expansion blocks, depending on the type of `ALEX-item`.

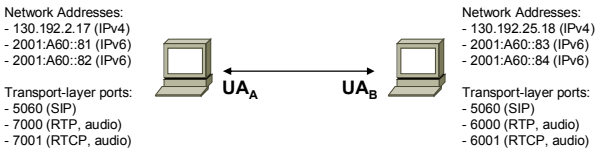


Figure 2: Network addresses and ports used in the ALEX negotiation.

The example in Figure 2 shows two ALEX compliant UAs trying to establish two media flows. UA_A will send the INVITE message reported in Figure 3 (showing only the most important fields) that includes several `ALEX-items` composing the *ALEX Address List*. This message includes three ALEX-items related to the SIP flow and three related to the audio flow.

```
INVITE <sip:UA2@ipv6.polito.it> SIP/2.0
From: UA1 <sip:UA1@ipv6.polito.it >;tag=a73kszlfl
To: UA2 <sip:UA2@ipv6.polito.it >
Contact: <sip:UA1@130.192.2.17:5060>
Supported: ALEX
ALEX-item:sip;0.5;d;130.192.2.17;5060
ALEX-item:audio;0.5;d;rtp;7000;rtcp;7001
ALEX-item:sip;0.8;[ 2001:A60::81];5060
ALEX-item:audio;0.8;rtp;7100;rtcp7101
ALEX-item:sip;0.8;[ 2001:A60::82];5060
ALEX-item:audio;1.0;rtp;7100;rtcp7101

v=0
o=- 2890844526 2890842807 IN IP4 130.192.2.17
m=audio 7000 RTP/AVP 0
c=IN IP4 130.192.2.17
```

Figure 3: INVITE message (SIP + SDP) including ALEX-items.

The components of an ALEX-item are somewhat intuitive. For example, the `ALEX-item` related to the audio flow in the INVITE message can contains the following fields:

- **Component label**: specifies the flow referred by the `ALEX-item`. Currently, values are "sip", "audio", "video".
- **Q-value**: a numeric priority value (in the [0,1] interval; 1 is for the highest priority) used to establish a priority among `ALEX-items` related to the same channel
- **Default flag** ("d"): if present, it indicates the set of default addresses that should ensure immediate connectivity.
- **Expiration time** (not present in Figure 3)**:** validity time in seconds for the `ALEX-item`. It is preceded by the keyword "expires:" and it is optional: if it is not present the expiration time is automatically set to 3600 s.
- **Address**: this field stores the network address related to the `ALEX-item`. IPv6 addresses are surrounded by square brackets. If this field is not present the address is assumed to be equal to the one stored in the previous ALEX-item in the message.
- **Flow component label**: it is a label placed at the beginning of each expansion block. Currently, values are "rtp", "rtcp".
- **Port**: number of the transport-layer port of the specific flow.

Note that the `ALEX-item` considered has two expansion blocks: one related to RTP and one related to RTCP. The presence of such expansion blocks in the same `ALEX-item` shows the correlation between them: if the RTCP flow cannot be established, the related RTP one is useless. Again note that the SIP related `ALEX-items` have no multiple expansion blocks: simply, in the case of SIP, the basic block is followed by the port related to the network address.

B. *ALEX phases*

ALEX modifies the establishment of a SIP session adding four phases, shown in Figure 4, and detailed in the following.

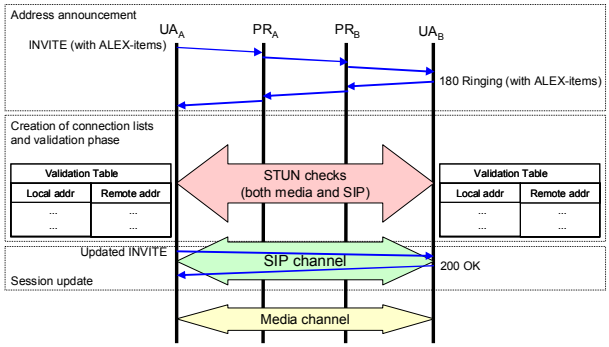


Figure 4: Dialog establishment using ALEX.

1) *Address announcement*

Two UAs exchange the complete lists of their addresses by adding the `ALEX-item` fields to the SIP request that creates the dialog and its related answer, respectively. Each `ALEX-item` field stores exactly one network address and at least one port (more details in the following) related to both SIP and media flows. While UA_A will send `Alex-items` in its INVITE message, UA_B should send its `ALEX-items` back to UA_A as soon as it is possible, e.g. in the 180 RINGING provisional response. This message can include only the SIP related `ALEX-items` (the SDP payload might be unknown at this time, hence `ALEX-items` related to audio and video may be missing); in such a case the `ALEX-items` related to media flows will be sent in the "200 OK" answer. This can be useful because, if the validation phase completes before the sending of the 200 OK, UA_B may decide to modify the list of items included: for example if an IPv6 SIP channel has been discovered, UA_B may include only the `ALEX-items` containing IPv6 network addresses in the final response, thus converging immediately to the optimal choice. Obviously, the called party will add `ALEX-item` fields to its answer only if the request contained the `Supported: ALEX` header, followed by the ALEX list.

2) Creation of the validation tables

When both the UAs exchanged successfully their `ALEX-items`, they start organizing them in validation tables. There will be a validation table for each flow involved. For example, considering the UAs that started the session depicted in Figure 4, there will be a validation table for SIP and one for the audio flow. Both the UAs populate these validation tables exactly in the same way. First of all, the priority of each candidate channel is set to the lowest priority value of the components involved and the expiration time of each candidate channel is set to the lowest priority value of the components involved. All the candidate entries are listed ordering them by decreasing priority values with the exception of the default entries that are placed on top of the table. If two or more entries have the same priority value, they are sorted using the priority of the components sent by the creator of the session. Then, each row is obtained by combining each address-port pair to be used by the UA with each addresses-port pair advertised by the other UA.

Considering the media flows, each entry of the related validation table is made up of two parts, one related to RTP and one to RTCP. These two parts are considered a unique entity and they will be checked at the same time. Entries must be homogeneous, i.e. the RTP item related to an audio session must be paired only with the corresponding RTP audio information on the other peer.

3) Address validation phase

When all the validation tables are ready, the validation phase begins. The connectivity checks are executed sending STUN Binding Requests using the addresses and the ports of each candidate entry. Upon the receipt of a binding request, the STUN server sends back a Binding Response containing in the payload the source network address-port pair seen in the request. By doing this, the sender checks connectivity with the receiver and discovers its network address (and port) seen by the receiver.

Since the validation process aims at establishing a direct SIP channel first, all SIP candidate entries are checked first. The SIP channel helps to discover the optimal channels for the remaining flows: for example if the SIP validation detected IPv4-only connectivity, there is no reason to further test IPv6 connectivity for media flows. It is worth noting that RTP and RTCP network address-port pairs are correlated, hence their validation is interdependent, i.e., both pairs originating from the same network address must be successfully validated in order for such address-port pairs to be usable.

The default channels (i.e. the ones with the "d" flag) are checked first but are used only if the checks on all the other channels fail. ICE performs similar connectivity checks, but ALEX checks also the SIP address pairs and not only the media ones. As soon as the connectivity is verified the UAs can start using the channel: ideally a smart UA can switch automatically from a channel to another, without sending or receiving specific update messages. By the way, if it is not possible, the UA is expected to store this channel in the ALEX-cache in order to use it in subsequent sessions. Note that the validation phase is completely independent from the INVITE transaction: the ACK request may be sent even though the validation phase is not completed (in this case, a sub-optimal couple of addresses can be used). This means that the SIP standard timers need not to be modified to support the ALEX extension.

4) Session Update

As soon as optimal channels for both SIP and media flows are available, UA_A and UA_B can start using them. Optionally UA_A can send a re-INVITE message including a standard SDP payload containing the addresses related to the optimal media channels discovered. This is done to inform intermediate entities that possibly monitor the media traffic between the UAs. UA_B sends a "200 OK" answer with an updated SDP payload for the same purpose.

C. *Backward compatibility*

To ensure backward compatibility with ALEX-unaware UAs, the SIP default component is placed in the `Contact` field. For the same reason the default groups for media streams are placed in the "m=" lines of the SDP payloads. When a standard SIP UA receives the INVITE request depicted in Figure 3, it will start using the default addresses and ports as default remote targets for both SIP and media flows. It is important to note that the ALEX extension does not modify any fields required for the session establishment, such as the `contact` field or the `record-route` field: this approach ensures full backward compatibility (for instance, if an UA cannot fully understand a mandatory field, must discard the message). Obviously, an ALEX-unaware UA must be able to parse a message containing ALEX-items correctly, e.g. discarding unknown headers.

D. *The ALEX cache*

The ALEX cache is a data structure used to store the optimal candidates pairs. This cache is a sort of "neighbor cache" for an ALEX compatible UA, in which address pairs are kept until their expiration time: these pairs can be used to simplify the establishment of subsequent dialogs, e.g., because the best network address pairs can be checked first. This feature is not present in ICE and can be use to speed up connectivity checks, reducing the number of STUN messages exchanged. The ALEX-cache provides information about the involved network address-port pairs and specifies the related flow type together with the priority and the expiration time.

E. *Using ALEX in non-INVITE dialogs*

Formerly the ALEX mechanism has been illustrated in the case of INVITE sessions. By the way ALEX applies to all SIP dialogs. For example, consider the case of a dialog created by a SUBSCRIBE request. Upon the receipt of such a message, an ALEX-aware UA sends back to the subscriber a "101 Dialog Establishment" response containing its ALEX-items: the receipt of this message starts connectivity checks. Only when the checks are over, the UA that received the SUBSCRIBE sends back a "200 OK" answer to complete the transaction. This procedure results in the creation of a direct SIP channel between the UAs: this channel can be optionally used by the subscriber to send an updated SUBSCRIBE message containing the Contact URI related to the channel established.

The same channel is used to send immediately a NOTIFY message to the subscriber. This feature is not available in case of ICE-only UAs, since ICE can be applied only to media flows.

F. *Reliability of ALEX-items*

ALEX-items are inserted in SIP messages, which are always confirmed; hence a loss of an ALEX-item is not a problem, because the SIP implementation will retransmit the entire message anyway. Only provisional messages do not follow this rule; an ALEX-item inserted into such a message must be repeated in the next SIP message (e.g. the 200 OK).

G. *Fitting ALEX into the SIP stack*

One important difference between ICE and ALEX relates to the position of the new mechanism within the SIP protocol stack, which is depicted in Figure 5.

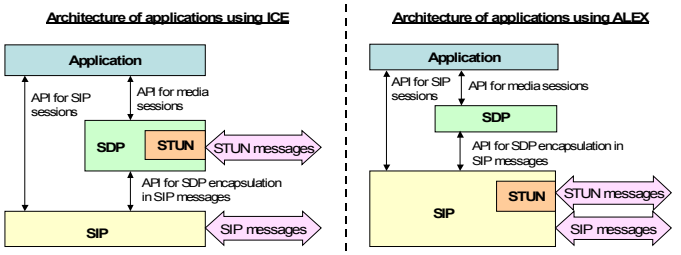


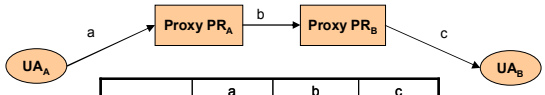
Figure 5 architectural differences between ALEX and ICE.

ALEX relies on a new field within the SIP portion of the message; in addition, ALEX performs connectivity checks to establish direct channels: these checks are handled directly by the SIP module that controls a STUN client/server engine. Since these changes affect only the SIP stack executed on UAs, these are independent from the application that uses SIP for signaling. Vice versa, in the ICE model SDP has to be modified in order to support address exchange, STUN-based validation, etc. Moreover, in the case of ICE SIP cannot benefit from possible direct connectivity. Additionally, ALEX does not require modifications on SIP proxies, and it is compatible with UAs running a non-ALEX SIP stacks.

IV. EXPERIMENTAL RESULTS

In order to test the behavior of an ALEX-capable UA, ALEX has been integrated in the code of an existing UA. OpenWengo [10] was chosen because its source code is publicly available and it offers rich media features. The modifications required to support ALEX were limited to about 5,800 lines of C code, concentrated mostly in the SIP stack whose original size was approximately 31,800 lines. The changes consisted in the modification of the SIP message parser to support the new header fields. The dialog data structure has been updated to store temporary information during connectivity checks. Finally, a new data structure has been introduced to implement the ALEX cache and a control interface has been added to the SIP stack, in order to handle STUN checks and to generate SDP payloads used during the session update procedure.

The ALEX extension has been tested in the scenarios listed in Figure 6, each featuring a different combination of the protocols deployed on the networks between SIP entities. The tests consisted in several media session establishments initiated by UA_A. In all tests ALEX has proven to be effective in enabling UAs to exchange SIP messages and media flows directly. Moreover, channels, i.e., IP addresses (and possibly specific interfaces associated to them), were chosen optimally: IPv6 addresses in the first scenario and IPv4 addresses in the remaining two scenarios.



	a	b	c
Scenario 1	IPv6/IPv4	IPv6/IPv4	IPv6/IPv4
Scenario 2	IPv6/IPv4	IPv4	IPv6/IPv4
Scenario 3	IPv4	IPv6/IPv4	IPv4

Figure 6. Test scenarios.

In the first two scenarios the tests were executed in our labs, while in the last one the two UAs were connected through commercial DSL accesses: under these conditions the average round trip time for the STUN transactions was about 120 ms. Even in this condition, address validation was completed before UA_A received the "200 OK" answer, hence no delay could be perceived by end users. The average time between a "180 RINGING" message and the corresponding "200 OK" message computed during the tests was of about 3 seconds: this value is partly due to the time required by the UA to process the INVITE message and to the reaction time of the person answering the call. On the other hand, the average time to complete the connectivity checks was of about 163.2 ms, confirming the claim that the delay resulting from the validation overhead is not perceivable by end users. Furthermore, as shown in Table II, the tests demonstrated that the overhead due to connectivity checks is limited, even when the establishment of an end-to-end channel is not possible (in which case STUN Binding Requests are retransmitted multiple times).

TABLE II. OVERHEAD INTRODUCED BY CONNECTIVITY CHECKS

UA_B IPv6 addresses	STUN bytes (% of total)		Overall bytes sent (INVITE transaction)	
	Scenario 1	Scenario 2	Scenario 1	Scenario 2
2	5.2%	5.6%	15,586	15,770
3	6.3%	7.1%	16,462	16,604
4	7.5%	8.6%	17,087	17,307

The values in Table II have been computed assigning a single IPv6 address to UA_A and a growing number of IPv6 addresses to UA_B, as specified in the leftmost column; both UAs had a single IPv4 address. The two rightmost columns display the average total number of bytes sent on the network by both the UAs and the proxies during media session establishment, respectively in the case of scenario 1 and scenario 2. The second and third columns show the percentage of these bytes related to connectivity checks, respectively in the case of scenario 1 and scenario 2. Note that in scenario 2 there is no direct IPv6 connectivity between domains. By the way, since the UAs can communicate with their proxies using IPv6

addresses, the IPv6 addresses are inserted in the `ALEX-items`. For this reason, the STUN requests sent to probe the IPv6 addresses get no answer and are retransmitted three times to be sure of the lack of connectivity. It is worthwhile highlighting that even in this case the overhead due to connectivity checks is limited (8.6% versus 7.5% when STUN requests do not need to be retransmitted).

The tests also enabled us to compare the overhead introduced by ALEX with the one introduced by ICE with the aim of demonstrating that direct SIP connectivity comes at a limited cost. For this purpose Table III compares the bytes transmitted by ALEX to announce an IPv4 address together with an increasing number of IPv6 addresses to the ones transmitted by ICE. The number of IPv6 network addresses announced varied from 1 up to 4. Notice that ALEX was used to announce addresses related to SIP connectivity as well as media flows (only one in the tests).

TABLE III. ESTIMATION OF THE OVERHEAD DUE TO THE SIZE OF ALEX-ITEMS

# of IPv6 addresses	ALEX overhead (bytes)	ICE overhead (bytes)	ALEX overhead compared to ICE (%)
1	136	144	-6.25%
2	229	225	+1.78%
3	322	306	+5.23%
4	415	387	+7.23%

As shown in Table III, when four IPv6 addresses are announced the size of SIP messages including `ALEX-items` is just 7.23% greater than the size of SIP messages containing ICE candidates, i.e., ALEX does not make SIP messages significantly larger when compared to ICE.

Interoperability with ALEX-unaware UAs was tested as well: sessions between an ALEX-Openwengo UA and Counterpath Eyebeam UA were completed successfully; the same outcome was achieved with Cisco 7940 VoIP phones. Specifically, in all tests both SIP dialogs and media flows were established successfully, which demonstrated that ALEX-aware UAs can be deployed in standard SIP environments.

V. CONCLUSIONS AND FUTURE WORK

ALEX (Address List Extension) provides a complete solution for both dual-stack and multi-homing support in SIP thus restoring the original peer-to-peer like paradigm for both signaling and media flows.

ALEX extends the concept of direct connectivity introduced with ICE to both signaling channel and media channels. The results obtained from the tests demonstrated the proposed solution to be effective: in most of the considered cases it was possible to establish direct connectivity. Moreover, the tests showed that the network overhead due to the additional information exchanged is comparable to the ICE solution. ALEX can be deployed to ensure direct SIP and media connectivity through NATs, which is the focus of our future work. Furthermore the usage of ALEX to establish direct SIP and media channels in peer-to-peer environments is under investigation.

ACKNOWLEDGMENT

The authors wish to thank Luca De Marco, whose graduation project partly focused on these issues.

REFERENCES

[1] J. Rosemberg et al., *SIP: Session Initiation Protocol*, IETF Network Working Group, RFC 3261, Jun 2002.

[2] J. Rosenberg, H. Shulzrinne, *Session Initiation Protocol (SIP) – Specific Event Notification,* IETF Network Working Group, RFC 3265, Jun 2002.

[3] M. Baldi, F. Marinone, F. Risso, L. Torrero, *ALEX: Improving SIP Support in Systems with Multiple Network Addresses*, Proceedings of the 5th IEEE International Symposium on Signal Processing and Information Technology, Athens, Greece, Dec 2005.

[4] J. Rosenberg, *A Presence Event Package for the Session Initiation Protocol (SIP),* IETF Network Working Group, RFC 3856, Aug 2004.

[5] M. Handley, V. Jacobson, *SDP: Session Description Protocol,* IETF Network Working Group, RFC 2327, Apr 1998.

[6] G. Camarillo et al, *IPv6 Transition in the Session Initiation Protocol (SIP),* IETF Network Working Group, draft-ietf-sipping-v6-transition-05.txt, May 2007.

[7] J. Rosenberg, *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/AnswerProtocols*, IETF Network Working Group, Internet Draft draft-ietf-mmusic-ice-17.txt, Jan 2008.

[8] Rosenberg, J. et al, Simple Traversal of User Datagram Protocol (UDP) through Network Address Translators (NATs), RFC 3489, March 2003

[9] Rosenberg, J. et al, *Session Traversal Utilities for NAT (STUN)*, draft-ietf-behave-rfc3489bis-06, (work in progress), Jan 2008.

[10] The Openwengo Project. Available at http://www.openwengo.org.

[11] P2PSIP, IETF Working Group. Available at http://www.p2psip.org.