ALEX: Improving SIP Support in Systems with Multiple Network Addresses

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

***Abstract* — The successful and increasingly adopted Session initiation Protocol (SIP) does not adequately support hosts with multiple network addresses, such as dual-stack (IPv4-IPv6) or IPv6 multi-homed devices. This paper presents the Address List EXtension (ALEX) to SIP that adds effective support to systems with multiple addresses, such as dual-stack hosts or multi-homed IPv6 hosts. ALEX enables IPv6 transport to be used for SIP messages, as well as for communication sessions between SIP User Agents (UAs), whenever possible and without compromising compatibility with ALEX-unaware UAs and SIP servers.**

Keywords — SIP; IPv4-IPv6 transition; multi-homed devices; VoIP and ToIP in dual-stack environments

I. INTRODUCTION

SIP (Session Initiation Protocol) is a general purpose signaling protocol that aims at establishing voice, video, gaming, and other types of application-level sessions between two or more peers. However, this protocol and its related infrastructure do not support end-systems that have multiple network addresses, e.g. multi-homed machines or dual-stack ones, which are hereafter called *multi-address devices*. This is a major issue because new generation of mobile phones will probably use mainly IPv6 and a larger number of dual-stack machines will probably be deployed to make the transition to the new network protocol easier. In the case of multi-homed devices the choice of an address pair for communication might impact performance and communication feasibility since different network addresses might be reachable through different routing paths. Hence, the capability to support multiple addresses on the same device and choose the best match between two peers is a must.

The current best practice to identify the target for a SIP interaction is by means of a user agent (UA) identifier (UAID) that is inserted in the `contact` field of the header of SIP messages. Since each UA must have only one UAID, — usually derived from one of its network addresses — this approach is critical when a target UA can be reached through multiple network addresses. Currently proposed solutions, surveyed in Section IV, lack generality, being effective only in some specific cases.

This work has been carried out in the context of a research contract granted by CSI-Piemonte (http://www.csi.it).

This paper proposes an extension to the SIP protocol called ALEX (Address List EXtension) to add effective support of multi-address devices. ALEX is simple, requires only minimal modifications to SIP user agents and SIP servers, guarantees compatibility with ALEX-unaware implementations, and works with any application (e.g. voice/video calls, instant messaging, etc.). Section II provides a brief overview of SIP in order to lay the basis for the discussion in Section III on the inadequacy of SIP when it comes to multi-address devices. Section IV presents existing approaches to multi-address device support with SIP and highlights their shortcomings. Section V describes the proposed SIP extension in detail highlighting its main features and strengths. Conclusions are drawn in Section VI that also outlines further work directions.

II. SIP

The Session Initiation Protocol (SIP) is an application-level signalling protocol that aims at establishing, modifying and terminating a communication between peers. This protocol provides a mechanism to locate the other peer, negotiate capabilities, and start an application-dependent (voice, video, shared whiteboard, messaging, and more) session. The protocol encompasses a number of entities interacting with each other: a *User Agent Client* (UAC) sending request messages and processing the corresponding responses, a *User Agent Server* (UAS) responsible for processing incoming request messages and generating related responses, and a set of intermediate servers. Although SIP UAs may communicate directly in order to establish a connection without the presence of any intermediate nodes, SIP servers make the signalling process more flexible and allow simpler UA implementation.

SIP resources, such as users, mailboxes on a message system, etc., are identified by *Uniform Resource Identifiers* (SIP URIs). A special resource is the *address-of-record* (AOR) that has a global scope and represents the “public address” (i.e. the unique identifier) of a *user* (e.g. bob@foo.com). The AOR cannot be used ‘as is’ and it has to be mapped to a physical resource identifying the actual device (e.g. a SIP user agent such as a soft-phone, or a mailbox if the user is offline) associated to that user. In general, the AOR can be mapped to several resources at the same time because a user can be deploying several physical devices (mobile phone, office phone, etc.).

Communications between entities rely on SIP messages, text-based messages that are carried within a TCP/UDP session. Among the most important information contained in SIP messages (and relevant to this work), the `contact` field contains a SIP URI that uniquely identifies a physical resource associated to the user that generated the message. Often, the `contact` field contains a SIP UA identifier consisting of a user name and a fully qualified domain name (FQDN) or a network address, e.g. *bob@bobpc.foo.com* or *bob@1.2.3.4*. The deployment of a network address within the `contact` field is very common as it offers a very simple method to generate a unique identifier. However, it should be kept in mind that the sole purpose of such address is the creation of a unique identifier and (in principle) it should not have any relationship with the IP address from which the SIP message is arriving or the IP address to which a response should be sent.

Another relevant part of the SIP header is the `via` field that specifies the address of each SIP node that forwards the message. This field provides a way to track the path followed by a SIP request so that the related SIP response can be forwarded on the reverse path. A UA has to fill the `via` field with its address before sending any message. When SIP servers forward a message, they update its `via` by adding their address on the top of the list.

Among SIP servers, the most important are the SIP Registrar and the SIP Proxy. The *SIP Registrar* server keeps track of all the UAs in its *domain*. A UA registers itself by sending a REGISTER request, which contains the AOR of its SIP user and one (or more) `contact` fields containing the URIs of the devices used by the user; this information is stored in the *Location Service* database that is often integrated with the Registrar server, although it could be in principle separate. The *SIP Proxy* server forwards incoming SIP messages to a destination AOR; this process often requires the interaction with the Location Service in order to get the list of contact addresses associated to the AOR— i.e., the URI of the devices the user is deploying. Routing of SIP messages can be fully delegated to Proxy servers if Record Route field is included (usually by a proxy), which forces SIP UAs to send SIP messages through a SIP proxy even if a shortest path (e.g. UA to UA interaction) is available. The Record Route field may be essential in some cases, such as when UAs behind NATs are able to communicate to specific servers, such as their SIP proxy, but are not allowed to generally exchange messages with any host.

Figure 1 shows a sample SIP session: a SIP UA (left) wanting to start a media session sends an INVITE message to a target SIP UA through two proxies. Only when the called party accepts the invitation (the *"200 OK"* message) the media session begins and data is exchanged directly between the peers. Figure 1 shows also two key concepts of SIP: dialogs and transactions. A *transaction* consists of one request and corresponding responses. SIP responses are identified by a code: the 2xx responses are called final responses and they close a transaction. For instance, Figure 1 contains two transactions, T1 and T2. *Dialogs* are relationships between UAs initiated when the initiating peer receives a non-failure response. Figure 1 shows a dialog created as a result of an INVITE request, which is outside of the dialog itself. If a Record Route field is not included in the messages, direct communication between the two UAs is possible since each UA can learn the IP address of its peer from the `contact` field of received SIP messages. For example, in Figure 1 the callee UA finds out the address of the caller UA from the URI in `contact` field of the INVITE request, while the caller UA devises the peer UA address from the URI in the `contact` field of the "200 OK" response.

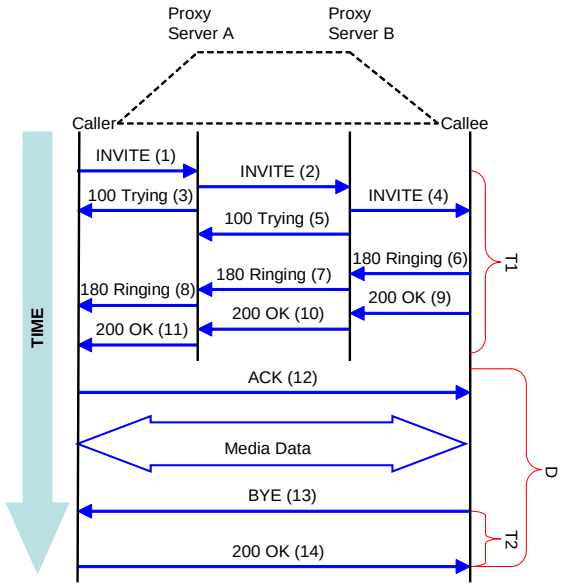

Figure 1 Media session establishment through SIP INVITE message.

III. MOTIVATIONS

When direct interaction should take place between multi-address SIP entities, each of them is faced with two problems: (*i*) obtaining the complete list of network addresses of the remote device and (*ii*) selecting the "best" source/destination address pair to be used for the dialog. The reminder of this section presents existing solutions to these problems in general and discusses why these are not suitable to the case of SIP devices.

A. *Multi-address devices in a client-server environment*

With common client-server protocols (such as HTTP, POP, SMTP, etc.), a list of network addresses can be easily associated to a host by assigning the host a unique name, such as a Fully Qualified Domain Name (FQDN) and storing several records (e.g. A and AAAA records) in the DNS associated to the FQDN. Referring to such unique name when communicating with the host indicates that any address contained in the associated DNS records is equivalent for the purposes of the communication.

A client that wants to contact the multi-address host obtains from the DNS all the records associated to that name, getting to know the capabilities of the server in terms of address families (IPv4, IPv6) and network addresses. At this point the client has

to solve the second problem previously listed: which source/destination address pair should be the best to use. At first sight the solution seems straightforward: if both devices are IPv6-capable the "best" address family is IPv6; in case of failure, or if one of the hosts is IPv4 only, then IPv4 addresses should be used. Moreover, the choice among addresses of the same family is usually done statically in IPv4 (often the first IPv4 address returned is used), and dynamically in IPv6 through the Source Address Selection [6] mechanism. However, in general the scenario might be complicated since some of the addresses of one host might not be reachable by the other or might be associated to "long" routes.

B. Multi-address devices in a SIP environment

The SIP protocol is not a simple client-server protocol. Although some of its functions are based on the classical client-server paradigm (e.g. the SIP REGISTER message which is used to register a SIP UA within its SIP server), many others are not. For instance, a SIP INVITE message usually travels from a UA to its SIP server, gets forwarded to the SIP server of the target domain, and finally to the target UA; in addition, the response message can be sent directly from the contacted UA to the session originator.

Intermediate devices possibly participating into a SIP transaction (often an INVITE message involves two UAs and two proxy servers) result in a much more complicated "capability negotiation" with regard to network addresses. For instance, the message between the originating UA and its SIP server can be sent through IPv6 if both are supporting this protocol, but the message coming back from the called UA may be using IPv4 and yet the originating client must be able to understand that this message is part of the dialog that was started in IPv6. Moreover, the destination of some SIP messages (e.g. INVITE, but not REGISTER) is specified in terms of the called user's SIP AOR, which can be associated to several devices. Therefore an additional resolution step[1] is needed to locate the set of UAs associated to a SIP user. Then, for each UA, the list of its network addresses must be obtained and one chosen among them for the session.

In order to better define the problem, the reminder of this section analyzes the possible interactions between SIP devices, pointing out the most problematic issues.

a) Messages from SIP UAs to SIP servers

Messages from SIP UAs to SIP servers (e.g. the first step for an INVITE message) do not present any new problem as the interaction between these devices follows the typical client-server paradigm. The SIP UA obtains the address list of the SIP server through a set of DNS queries that involve NAPTR and SRV records, in addition to the well-known A and AAAA ones. Hence, existing methods to select the best source/destination address pair can be applied. It is important to point out that an address possibly specified in the `contact` field of SIP messages (e.g. `contact: bob@1.2.3.4`) might be different from the source address in the IP packet carrying the message to the server (e.g. `2001:760:400::1`), according to the OSI layering model.

b) Messages from a SIP server to another SIP server

Messages from a SIP server to another SIP server (e.g. the second step for an INVITE message, when the proxy server in the caller's domain forwards it to the proxy server in the callee's domain) are also based on a typical client-server paradigm; hence addresses are handled as in the previous case.

c) Messages from a SIP server to a SIP UA

Messages from a SIP server to a SIP UA (e.g. the third step for an INVITE message, when the proxy server in the callee's domain delivers the message to the called UA) are more critical since the association of a list of addresses to a UA cannot be effectively done through a FQDN and the DNS for a number of reasons. First, it is not uncommon that a device running a UA is not registered in a DNS server. Second, even if it is registered in the DNS, keeping DNS entries updated is difficult when addresses change rather often, which is common in IPv6.[2] In principle the SIP Registrar server could be used to store the list of network addresses associated to each UA, but SIP does not specify a way to do so. In fact, multiple contacts can be associated within the SIP Registrar to an AOR, but they are viewed as distinct UAs working for the same user, i.e., messages may be duplicated and sent concurrently to all listed UAs. Instead when two addresses are associated to the same UA only one must be used to transfer messages to the UA.

In any case, knowledge of the UA address list is not essential in order to just deliver a SIP message since the contact URI previously registered by the UA with the SIP Registrar can be used. However, this does not enable the server to make the best address choice or to switch to a different address in case the one corresponding to the registered URI is no longer working (e.g. due to a network failure).

d) Messages from a SIP UA to another SIP UA

Responses from a SIP UA to another SIP UA (e.g., the "200 OK" message that closes an INVITE transaction) undergo the same issues described for the previous case, since reply are sent back on the same path of the original message (hence using the `via` header). Hence, a called UA wanting to start another transaction to the calling UA (e.g. a BYE message) can rely only on information in the `contact` field of the received SIP message. However, differently from previous case, the approach is not necessarily effective even when just aiming at delivering the message (without necessarily aiming at optimising delivery). If, for example, the `contact` field contains a URI formed from an IPv6 address and the called device does not support IPv6, a direct communication cannot take place, even if the calling UA is a dual-stack device — which is unknown by the callee.

Message delivery can be ensured by including a Record Route field which forces all messages to follow the same path

[1] See the two general problems mentioned at the beginning of Section III.

[2] An IPv6 address changes, for example, after a variation of the MAC address of a network interface, when the lifetime of the "privacy address" [7] expires, or when a host changes its location.

— through proxies — and it does not allow direct peer-to-peer interaction. However, this solution imposes an additional overhead on the proxy servers, which may not be negligible especially with applications (e.g. instant messaging) that exchange SIP messages periodically. Moreover, while providing a solution for delivering SIP messages, this approach does not solve the problem for applications that require direct UA-to-UA interaction (e.g. voice sessions), thus requiring each UA to know the network addresses of the other one.

It can be concluded that an extension to SIP is required to exchange an address list between SIP entities running on multi-address devices, such as dual-stack and multi-homed hosts, in order to efficiently enable dialog setup.

IV. Related work

The problem of supporting multi-address devices in SIP has not been explored in depth in the literature. The reason is that mostly IPv4-only hosts are currently being used, which rarely have multiple addresses. Additionally, inadequate support of multi-address IPv4 devices most likely results simply in under-optimized SIP message transfer. However, when dual-stack IPv4-IPv6 machines are deployed — which is going to become more widespread — SIP UAs may not be able to successfully start a dialog, as discussed in the previous section.

To the best of our knowledge, the only existing approach for supporting multi-address devices in a dual-stack scenario stems from the combination of ICE and ANAT. ICE (Interactive Connectivity Establishment) [2] allows listing the several network addresses associated to a UA, although they must belong to the same address family. ANAT (Alternative Network Address Types) [3][4] overcomes this limitation by extending ICE to support dual-stack clients.

The ICE-ANAT combination is interesting as it supports also "virtual" network addresses, such as the network address dynamically substituted by a NAT (Network Address Translator) into packets generated by or addressed to a host. However, the ICE-ANAT solution inserts additional addressing information in the SDP (Session Description Protocol) portion of a SIP message, making this solution suitable only for media flow establishment. In other words, this approach is not applicable to a range of SIP-based applications that do not deploy media flows, among which, for example, instant messaging. Other ANAT open issues, described in [4], are related to situations in which one of the UAs does not support ANAT. In addition, the ICE-ANAT solution has a high degree of complexity.

The solution proposed in the next section aims at overcoming the issues of the ICE-ANAT, i.e., (*i*) providing general (not only for media flow-based applications) support for direct communication between UAs running on multi-address devices (*ii*) with full backward compatibility.

V. The Address List EXtension (ALEX)

The solution to the issues discussed in Section III is presented in the following and is based on a SIP extension that encompasses an information exchange to devise network layer identification of a UA, rather than deriving it from the URI communicated within the `contact` field. The SIP protocol has been designed to be flexible and it supports ad-hoc extensions that must be negotiated during the first message exchange in a session between UAs. If both UAs support the proposed extension then the added information can be used to devise the best source/destination address pair for direct communications between the UAs. The proposed extension is beneficial when supported by UAs, and assures interoperation of dual-stack UAs with IPv4-only UAs and proxies.

ALEX aims at optimising direct peering between two UAs and is applicable only to SIP dialogs, provided that a SIP proxy does not insert the Record Route field; ALEX related information is included in SIP messages setting up the dialog. ALEX is not suitable outside dialogs because in this case messages are expected to be exchanged through proxies, either because the data exchange is expected to be very small (hence the dialog creation inserts an unnecessary overhead), or because messages might be delivered to multiple destinations (hence proxies are the only entities that can perform a message forking); hence the address list of the other peer is useless.

A. *Address field format*

ALEX (Address List EXtension) requires the definition of an `address` field whose specification in Backus-Naur Form (BNF) [RFC2234] is shown in Figure 2. The `address` field is present (one ore more times) in every request message that creates a dialog with another peer to provide it with network level information. A UA supporting ALEX builds an `address` field for each (logical) interface it is listening to. For instance, a dual-stack UA includes at least two `address` fields in every message, one for its IPv4 address and one for its IPv6 address. A `require` field with value `ALEX` precedes the sequence of `address` fields to indicate that ALEX is required.

```
ADDRESS EQUAL network-address \
      [; transport-address ] ; c-p-q [; expire-t]
network-address = 1*64(alphanum)
transport-address = "port" EQUAL ( alex-port )
c-p-q = "q" EQUAL qvalue
qvalue = ( "0" [ "." 0*3DIGIT ] )  \
            / ( "1" [ "." 0*3("0") ] )
alex-port = 1*8[DIGIT]
expire-t = "expires" EQUAL time-exp
time-exp = 1*8[DIGIT]
```

Figure 2 `address:` field specification.

The `address` field contains a network address, a `q` parameter that defines the "quality" of the address within as a number between 0 and 1 (higher numbers means higher preferences; e.g. IPv6 addresses should have higher preference, leaving IPv4 addresses as a "fallback" solution), an optional `expires` parameter that defines a time validity for the corresponding network address, and an optional `port` parameter to communicate the SIP port in case it differs from the default value.

Figure 3 shows a SIP `INVITE` message with ALEX extension. A called UA supporting ALEX responds with a SIP `200 OK` message that includes the list of its addresses — in a sequence of `address` fields. Otherwise, the callee refuses the

connection and forces the caller to retry through an `INVITE` message without ALEX. The choice of bounding the deployment of ALEX to the `require` field, and consequently imposing the above behaviour, stems the experimental nature of ALEX. In the future, ALEX support might be requested through the `support` field instead, which does not force to abort the dialog in case the called party does not support this extension.

```
INVITE sip:bob@biloxi.com SIP/2.0
   To: Bob <sip:bob@biloxi.com>
   From: Alice <sip:alice@atl.com>
   Contact: <sip:alice@192.168.225.1>
   Require: ALEX
   Address: 192.168.225.1 ; q=0.1 ; expires=600
   Address: 2001:760:250:::1 ; q=0.9
```

Figure 3 SIP request from a UA with ALEX extension

B. Deployment principles

An ALEX capable UA must bind its SIP stack to every network address available on the host. Both local and public addresses suitable for communicating with other hosts are included into its address list.

A UAC initiates a dialog with a UAS based on the URI of the UAS or of its user (i.e., an AOR). The UAC sends the initiating message (e.g., an `INVITE` message or a `SUBSCRIBE`) to its SIP proxy server and the SIP infrastructure (i.e., location service, SIP proxies, SIP redirectors) delivers such message to the UAS. A UAC supporting ALEX includes the extension in such first message, as exemplified by the `SUBSCRIBE` message in Figure 4. A UAS supporting ALEX extracts from received messages the information in the `address` fields (see the right column of the SIP Dialog Address Table of the called UA in Figure 4). When preparing a reply message, the UAS includes one or more `address` fields containing the list of its addresses, as exemplified by the first `200 OK` message in Figure 4. Upon reception of the `200 OK` message the caller UA extracts and stores the addresses carried by ALEX, as shown by its SIP Dialog Address Table in Figure 4.

Each of the two UAs involved in a dialog must select the address pair to be used for direct communication with the peer among all the pairs resulting from the combinations of compatible local addresses and remote peer addresses learned through ALEX. The first step of this selection consists in identifying which of the possible pairs do actually enable successful packet exchanges. To this purpose, UAs carry out an *Address Validation Process*, as shown in Figure 4. The `q` parameter associated to each address within ALEX extensions is used to select a validated address pair when more than one is available. The selected address pair is used in the direct message exchanges within the dialog, as shown in the bottom part of Figure 4, possibly including media packets.

The Address Validation Process is based on using the address pair to be validated for exchanging IP packets that carry a new SIP messages called `VALIDATE`. For the sake of brevity, the mechanisms and protocols involved in the address validation process are not described here and will be included in future publications.

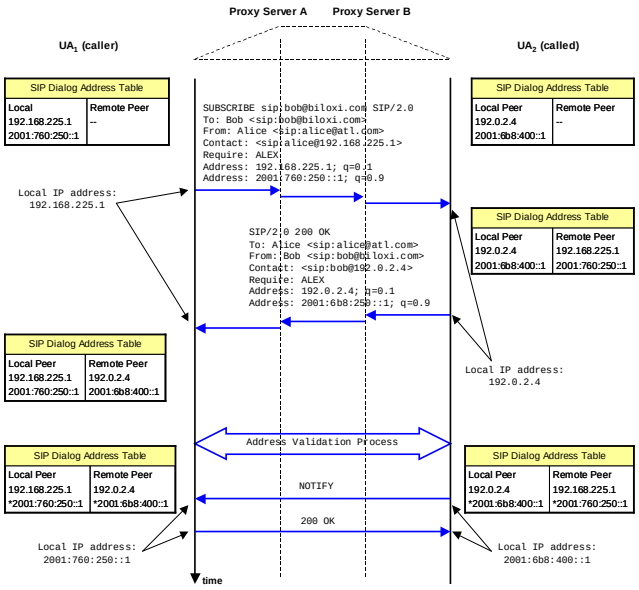

Figure 4 Example of message exchange between UAs with ALEX support. Only the relevant messages are shown.

Remote peer addresses are cached for later communications with the same UA. A UA must not consider and store remote peer addresses that could not be validated, i.e., cannot be used for communication with the remote peer. For instance, an IPv6-only UA will keep only entries of the SIP dialog address table marked as IPv6 addresses.

When a new dialog, like the SUBSCRIBE in Figure 4, is negotiated between the same UAs, they compare the received address list with the previously stored one. Pairs not changed and not expired are used without need for validation, while new pairs are to undergo the address validation process.

C. Backward compatibility

While an ALEX-aware UA uses the URI in the `contact` field only for identifying its remote peer, a traditional UA uses the information contained in this field for establishing a direct communication with its remote peer. For this reason, an ALEX-compliant UA shall anyway fill out the `contact` field as it is done by ALEX unaware devices, which is stored in the registar Proxy and it is used by all SIP devices (ALEX-aware or not). When using an address in the `contact` field it shall be IPv4 for three reasons: (*i*) in a dual-stack environment, IPv4 is understood by every UA, (*ii*) an IPv6 addresses size could create problems to some IPv4-only UAs (due to implementation-related bugs), and (*iii*) proxies could be IPv4 only.

VI. CONCLUSIONS AND FUTURE WORKS

This paper presents the Address List EXtension (ALEX) to the Session Initiation Protocol (SIP) that adds effective support to direct communications between User Agents (UAs) running on hosts with multiple addresses, such as dual-stack hosts or multi-homed IPv6 hosts. ALEX enables IPv6 transport to be used for SIP messages whenever possible, without compromising compatibility with ALEX-unaware UAs. Future work is required to quantitatively assess the benefits of ALEX in a generic multi-homed host scenario, where there could be hosts with many interfaces and dual-stack capabilities. Moreover, ALEX-based, lightweight, and low latency support for NAT (network address translator) and firewall traversal should be developed. Finally, the usage of ALEX for optimizing the message exchange between UAs and SIP proxies will be investigated.

REFERENCES

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

[2] J. Rosenberg, *Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for Multimedial Session Establishment Protocols*, IETF Network Working Group, http://www.ietf.org/internet-drafts/draft-ietf-mmusic-ice-04.txt, February 2005.

[3] G. Camarillo, J. Rosenberg, *The Alternative Network Address Types (ANAT) Semantics for the Session Description Protocol (SDP) Grouping Framework*, IETF Network Working Group, RFC 4091, June 2005.

[4] G. Camarillo, J. Rosenberg, *Usage of the Session Description Protocol (SDP) Alternative Network Address Types (ANAT) Semantics in the Session Initiation Protocol (SIP)*, RFC 4092, June 2005.

[5] J. Mulahusic, H. Persson, *SIP Issues in Dual-stack Environments*, IETF Network Working Group, draft-persson-sipping-sip-issues-dual-stack-00.txt, February 2003.

[6] R. Draves, *Default Address Selection for Internet Protocol version 6 (IPv6)*, IETF Network Working Group, RFC 3484, February 2003.

[7] T. Narten, R. Draves, *Privacy Extensions for Stateless Address Autoconfiguration in IPv6*, IETF Network Working Group, RFC 3041, Jan 2001.