Abstract

Most operational experience with SIP to date has been over the IPv4 network; however, SIP implementations that support IPv6 are starting to emerge. IPv6 support in Session Initiation Protocol (SIP) goes beyond merely running a SIP stack on a host supporting a dual IP-stack (i.e., IPv4/IPv6). In addition to host-level support for IPv6,
a SIP stack itself must exhibit certain behavior if it is to support IPv6. This document describes such behavior in the form of recommendations that SIP implementors can use while constructing IPv6-aware SIP clients and servers.

This work is being discussed on the sipping@ietf.org mailing list.

Table of Contents

1. Introduction ........................................... 3
2. Background: IPv6 Transition Strategies .................. 3
   2.1 IPv6 Transition Strategies for Nodes ................ 4
       2.1.1 IPv4 Client, IPv6 Server ....................... 5
       2.1.2 IPv6 Client, IPv4 Server ....................... 7
   2.2 IPv6 Transition Strategies for Applications .......... 9
3. SIP and IPv6 Network Configuration ..................... 10
4. Parser Torture Tests .................................... 10
   4.1 Valid SIP request with raw IPv6 addresses .......... 11
   4.2 Which port should I knock on? ....................... 11
   4.3 Knock on this port, please .......................... 12
   4.4 SIP request with IPv6 header parameter .............. 13
   4.5 SIP request with IPv6 identifiers in SDP body ....... 13
   4.6 Via headers from different networks in a request .... 14
   4.7 SIP request with multiple network identifiers in SDP . 14
   4.8 More test cases .................................... 15
5. Insulating Your Implementation with IPv6 APIs: Source Code Examples .................................. 15
6. Security Considerations .................................. 15
7. IANA Considerations ..................................... 15
8. Acknowledgments ......................................... 15
9. References .............................................. 15
   9.1 Normative References ................................ 15
   9.2 Informative References ............................... 16
   Authors’ Addresses ..................................... 16
A. Bit-exact archive of each test message ................. 16
   A.1 Encoded Reference Messages ......................... 17
   Intellectual Property and Copyright Statements .......... 18
1. Introduction

The Session Initiation Protocol (SIP [1]) is a protocol to establish, maintain, and tear down multimedia sessions. Most operational experience with SIP to date has been over the IPv4 network; however, SIP implementations that support IPv6 are starting to emerge. In SIP, IPv6 support needs to be provided not only by the host on which a SIP element is executing on, but support is also expected from the application (i.e., the SIP element) itself. For instance, the presentation format of an IPv6 address is much different from its IPv4 counterpart, and a SIP element must be intelligent enough to converse with its upstream or downstream peer using the network that the peer supports. Furthermore, within the SIP application itself, support for IPv6 must be provided in message signaling header as well as the message payload (the Session Description Protocol (SDP) contains network identifiers that will be in IPv6 presentation format).

In this draft, a set of recommendations is detailed that will be of use to implementors for ensuring IPv6 support. These include parser torture tests and related strategies for encouraging the use of network application programming interfaces that render the application code agnostic of the underlying network.

This document address implementation-specific issues for introducing IPv6 in SIP agents. Systems-level issues like the interplay between DNS A/AAAA queries and IPv6 and the use of DHCP is discussed in the SIP and IPv6 transition document [3], which is also being discussed in the IETF SIPPING Working Group.

2. Background: IPv6 Transition Strategies

The transition to IPv6 will proceed in stages. As IPv6 is introduced in the predominantly IPv4-based Internet, routers, hosts and applications must be made aware of this transition. In certain cases (like newly developed applications), this transition will be easy owing to the fact that IPv6 has been implicitly made to be compatible with IPv4. In other cases (like legacy applications that cannot be recompiled), this transition will be somewhat challenging as other techniques such as tunneling are employed to ease the transition. To this end, migration strategies have been defined for hosts, routers and applications. As a brief background, we summarize the relevant transition strategies here. Interested readers are urged to consult Gilligan et al. [4] for the transition mechanisms for IPv6 hosts and routers, and Shin et al. [5] for application-specific transition strategies.

Before looking at these strategies, the following terminology
definitions are required.

IPv4-only node
  A host, router or an application that implements and understands IPv4 only.

IPv6-only node
  A host, router or an application that implements and understands IPv6 only.

IPv4/IPv6
  A host, router or an application that is implements and understands both IPv4 and IPv6.

Dual-Stack node

IPv6-native address
  A network identifier that identifies an IPv6 endpoint. The presentation of such an address follows the conventions established in section 2.2 of RFC3513 [2] that uses colon-separated hexadecimal values.

IPv4-native address
  A network identifier that identifies an IPv4 endpoint. The presentation of such an address consists of four 8-bit hexadecimal values separated by a period (the dotted-decimal format).

IPv4-mapped IPv6 addresses
  These addresses allow IPv6 applications on dual-stack nodes to communicate with IPv4-only nodes. These addresses are not stored in any DNS data files, instead, they are created dynamically by the resolver. The low order 32-bits contain an IPv4 address, preceded by the fixed value of "ffff" in the previous 16 bits. The high order 80 bits are set to zeroes. Example ::ffff:192.0.2.1.

2.1 IPv6 Transition Strategies for Nodes

The most widely used transition strategy is the dual-stack node (or IPv4/IPv6) strategy described in Section 2 of [4]. Many, if not all, vendors that provide IPv6 support do so while maintaining IPv4 support in the operating system. Under this strategy, an IPv4/IPv6 node has the ability to send and receive both IPv4 and IPv6 packets. Such hosts can directly interoperate (at least at the network layer) with IPv4 nodes using IPv4 packets and also directly with IPv6 nodes using IPv6 packets [4]. While most operating systems come equipped with IPv6 support, the default behavior is that IPv6 is dormant and only IPv4 is active. Special configuration is required to make IPv6 active as well (since this configuration will vary among operating systems, we do not cover it in this document.

A general property of a dual-stack node is that an IPv6 server can entertain requests from both IPv4 and IPv6 clients, and an IPv6
client can converse with both an IPv6 and IPv4 server [6] (the cases where the network protocol matches between the client and the server are straightforward and not discussed further in this document). From this general property, it is apparent that IPv6-related APIs are a superset of IPv4 ones. Implementations that judiciously use the IPv6 APIs will be able to execute their code in a dual-stack node where IPv6 support has been disabled (i.e., the code will run under IPv4 semantics). We will revisit this topic in Section 5.

Finally, the dual-stack node strategy assumes that the DNS is populated with both IPv4 and IPv6 addresses corresponding to the same fully qualified domain name of the host. Over time, as the infrastructure moves to a pure IPv6 deployment, IPv4 addresses can be removed from DNS.

2.1.1 IPv4 Client, IPv6 Server

Figure 1 (abridged from Figure 12.2 in [6]) depicts how an IPv6 server supports both IPv4 and IPv6 clients. The ensuing discussion in this section and the next is based on Chapter 12 of [6].
In Figure 1, an IPv6 server is hosted on a dual-stack node. The IP layer of the node has two addresses, an IPv4 address (192.0.2.1) and an IPv6 address (2001:db8::1/32). An IPv6 server has been started on this node, which has created a listening socket and has bound itself to the IPv6 wildcard address (0::0) and TCP port 9999. The steps
that will allow an IPv4 TCP client to communicate with an IPv6 server are summarized as follows:

1. The IPv4 client calls gethostbyname() and finds a DNS A record for the server. Recall that in the dual-stack transition strategy, DNS is populated with A and AAAA records.

2. The client calls connect() and the client's node sends a TCP SYN to the server.

3. The server's node receives the IPv4 SYN packet. The destination port in the packet indicates an IPv6 listening socket, thus a flag is set to let the server know that this connection is using an IPv4-mapped IPv6 address (::ffff:192.0.2.1). The server responds with an IPv4 SYN/ACK. When the connection is established, the address returned to the server by accept() is the IPv4-mapped IPv6 address (a server, using the IPv6 sockets API [7], can explicitly check whether a given address is an IPv4-mapped IPv6 address; but otherwise, it never knows that it is communicating with an IPv4 client).

4. When the server sends messages to the IPv4-mapped IPv6 address, its IP stack generates IPv4 datagrams to the IPv4 address. All communications between the client and server occur over IPv4.

2.1.2 IPv6 Client, IPv4 Server

The protocols used by the client and server from the example used in the previous section are now swapped. Figure 2 shows an IPv4 server listening on an IPv4-only node while a IPv6 client sends a connection request to it from a dual-stack node.
IPv4-only node
+-----------+
| IPv4 Server |
+-----------+
IPv4 listening socket
bound to INADDR_ANY,
port 9999
+----+
| TCP |
+----+
+-----+
| IPv4 |
+-----+
192.0.2.1
+-----+
| Datalink |
+--------+

+------------------------------+      :
| Enet | IPv4 | TCP | TCP data |      :
| hdr  | hdr  | hdr |          | .....:
+------------------------------+

type           dport
0800           9999
IPv4 link-layer packet
transmitted from an IPv6 client using the
IPv4-mapped IPv6 address of the server

Figure 2: An IPv4-only server communicating with an IPv6 client.

In Figure 2, an IPv4-only node hosts an IPv4 server that has created
a listening socket and bound itself to the wildcard address and TCP
port 9999. The steps that will allow an IPv6 client to communicate
with the IPv4 server are summarized as follows:

1. The IPv6 client invokes getaddrinfo() (the IPv4/IPv6 replacement
   for gethostbyname(); see [4]) with the AI_ALL and AI_V4MAPPED
   bits set. This causes the resolver to return an IPv4-mapped
   IPv6 address (since the IPv4 server will not have an IPv6 AAAA
   record).
2. The IPv6 client calls connect() with the IPv4-mapped IPv6 address. The kernel detects the mapped address and automatically sends an IPv4 SYN to the server.
3. The server responds with an IPv4 SYN/ACK and the connection is established. Datagrams are exchanged over IPv4.

2.2 IPv6 Transition Strategies for Applications

Besides support for IPv6 in the node, many applications have to be updated to support IPv6 as well. This is true for SIP. The protocol contains network identifiers in many headers and the SDP body. Implementations must ensure that such identifiers are parsed and understood correctly. Section 4 will discuss a suite of torture tests in this context.

This section presents a discussion on application transition scenarios as outlined in [5]. That document classifies the transition in four different classifications; of these, only the last two -- case 3 and case 4 -- are pertinent to the discussion contained in this document.

The first case -- labeled as case 3 in [5] -- is depicted in Figure 3:

```
+-------------------+
|     appv4/v6      | (appv4/v6 - applications supporting
+-------------------+ both IPv4 and IPv6)
| TCP / UDP / others| (transport protocols - TCP, UDP,
+-------------------+ SCTP, DCCP, etc.)
|    IPv4 | IPv6    | (IP protocols supported/enabled in
+-------------------+ the OS)
```

Figure 3: Applications supporting both IPv4 and IPv6 in a dual-stack node

Here, an application has been ported (or written from the beginning) to run over IPv4 and IPv6. This transition case is the most advisable [5]. Some recommendations on writing SIP clients and servers that use this strategy will be provided in Section 6.

The second case -- labeled as case 4 in [5] -- is depicted in Figure 4:
Figure 4: Applications supporting both IPv4 and IPv6 in an IPv4-only node

Here, an application has been ported (or written from scratch) to run over IPv4 and IPv6, however, the base operating system only supports IPv4. IPv6 protocol may be supported on the operating system, but it may not be enabled. This will be the most commonly deployed scenario for SIP systems in the near future.

It is instructive to consider a specialized case, namely, an application is deployed on a legacy system that does not support IPv6 at all. The implications of this are that unlike the case where IPv6 is supported but not enabled, applications will not compile at all on a legacy system with no IPv6 support. Such systems will not have IPv6-specific libraries and system include files. The application designer will have to make a conscious decision on how to restructure the code so that it can be compiled for a legacy IPv4-only node (for instance, using compile-time macros, or by creating a "shim layer" that insulates the application from the underlying network representation). Such a legacy system is highlighted for the purpose of completeness only. The discussion in this document and the source code examples assume that the underlying operating system is a dual-stack node, where the worst that can happen is that IPv6 is supported but disabled.

3. SIP and IPv6 Network Configuration

System-level issues like deploying a dual-stack proxy server, populating DNS with A and AAAA RRs, zero-configuration discovery of outbound proxies for IPv4 and IPv6 networks, when should a dual-stack proxy Record-Route itself, and media issues also play a major part in the transition to IPv6. While this document addresses implementation-specific issues for the transition, system-level issues are addressed in a companion transition document [3]. Readers are thus urged to be familiar with both these documents in order to get an accurate picture of the transition.

4. Parser Torture Tests

This section is informational, and is NOT NORMATIVE on any aspect of
This section contains test messages based on the current version (2.0) of SIP as defined in [1]. Some messages exercise SIP’s use of SDP.

The test messages are organized into several sections. Some stress only a SIP parser and others stress both the parser and the application above it. Some messages are valid, and some are not. Each example clearly calls out what makes any invalid messages incorrect.

This section does not attempt to catalog every way to make an invalid message, nor does it attempt to be comprehensive in exploring unusual, but valid, messages. Instead, it catalogues some of the most common errors that implementations may exhibit during the parsing of IPv6 addresses.

Please refer to the ABNF in [1] on representing IPv6 addresses in SIP. IPv6 addresses are delimited by a ’[‘ and ‘]’.

The appendix contains an encoded binary form of all the messages and the algorithm needed to decode them into files.

### 4.1 Valid SIP request with raw IPv6 addresses

This REGISTER request is well-formatted per the grammar in [1]. An IPv6 address in presentation form appears in the Request-URI (R-URI), Via header, and Contact header.

Message Details: reg-good

```
REGISTER sip:[2001:db8::10] SIP/2.0
To: sip:user@example.com
From: sip:user@example.com;tag=81x2
Via: SIP/2.0/UDP [2001:db8::9:1];branch=z9hG4bKas3-111
Call-ID: SSG9559905523997077@hlau_4100
Contact: "Caller" <sip:caller@[2001:db8::1]>
CSeq: 98176 REGISTER
Content-Length: 0
```

### 4.2 Which port should I knock on?

IPv6 uses the colon to delimit octets. This may lead to ambiguity if the port number on which to contact a SIP server is inadvertently
conflated with the IPv6 address. Consider the REGISTER request below. The sender of the request intended to specify a port number (5070). Unfortunately, however, since the IPv6 address in the R-URI is compressed, it makes it hard to tell whether the 5070 is a port number or the last octet in the address.

From a pure parsing point of view, the REGISTER request is well-formed. However, from a semantic point of view, it will not yield the desired result. Implementations must take care to ensure that when a raw IPv6 address appears in a SIP URI, then any port number must appear outside the closing ‘[‘ of the URI.

Message Details: reg-ambiguous

REGISTER sip:[2001:db8::10:5070] SIP/2.0
To: sip:user@example.com
From: sip:user@example.com;tag=81x2
Via: SIP/2.0/UDP [2001:db8::9:1];branch=z9hG4bKas3-111
Call-ID: SSG9559905523997077@hlau_4100
Contact: "Caller" <sip:caller@[2001:db8::1]>
CSeq: 98176 REGISTER
Content-Length: 0

4.3 Knock on this port, please

In contrast to the example in Section 4.2, the following REGISTER request leaves no ambiguity whatsoever on where the IPv6 address begins and where it ends. This REGISTER request is well formatted per the grammar in [1].

Message Details: reg-good-port

REGISTER sip:[2001:db8::10]:5070 SIP/2.0
To: sip:user@example.com
From: sip:user@example.com;tag=81x2
Via: SIP/2.0/UDP [2001:db8::9:1];branch=z9hG4bKas3-111
Call-ID: SSG9559905523997077@hlau_4100
Contact: "Caller" <sip:caller@[2001:db8::1]>
CSeq: 98176 REGISTER
Content-Length: 0
4.4 SIP request with IPv6 header parameter

This REGISTER request contains an IPv6 address in a header parameter. The request itself is well formatted per the grammar in [1].

Message Details: reg-param

```
REGISTER sip:[2001:db8::10] SIP/2.0
To: sip:user@example.com
From: sip:user@example.com;tag=81x2
Via: SIP/2.0/UDP [2001:db8::9:1];received=[2001:db8::9:255];
    branch=z9hG4bKas3-111
Call-ID: SSG9559905523997077@hlau_4100
Contact: "Caller" <sip:caller@[2001:db8::1]>
CSeq: 98176 REGISTER
Content-Length: 0
```

4.5 SIP request with IPv6 identifiers in SDP body

This INVITE request is valid and well-formed. Notice the IPv6 addresses in the SDP body.

Message Details: inv-good

```
INVITE sip:user@[2001:db8::10] SIP/2.0
To: sip:user@[2001:db8::10]
From: sip:user@example.com;tag=81x2
Via: SIP/2.0/UDP [2001:db8::9:1];branch=z9hG4bKas3-111
Call-ID: SSG9559905523997077@hlau_4100
Contact: "Caller" <sip:caller@[2001:db8::1]>
CSeq: 8612 INVITE
Content-Type: application/sdp
Content-Length: 268

v=0
o=assistant 971731711378798081 0 IN IP6 2001:db8::20
s=Live video feed for today’s meeting
c=IN IP6 2001:db8::1
i=3338481189 3370017201
m=audio 6000 RTP/AVP 2
a=rtpmap:2 G726-32/8000
m=video 6024 RTP/AVP 107
a=rtpmap:107 H263-1998/90000
```
4.6 Via headers from different networks in a request

This BYE request is valid and well-formed. The Via list contains a mix of IPv4 and IPv6 addresses.

Message Details: bye-good

BYE sip:user@host.example.com SIP/2.0
Via: SIP/2.0/UDP [2001:db8::9:1]:6050;branch=z9hG4bKas3-111
Via: SIP/2.0/UDP 192.0.2.1;branch=z9hG4bKjhja8781hjuaij65144
Via: SIP/2.0/TCP [2001:db8::9:255];branch=z9hG4bK451jj;
    received=192.0.2.200
Call-ID: 997077@lau_4100
CSeq: 89187 BYE
To: sip:user@example.net;tag=9817--94
From: sip:user@example.com;tag=81x2

4.7 SIP request with multiple network identifiers in SDP

This INVITE request is valid and well-formed. It contains multiple network identifiers in the SDP body.

Message Details: inv-mult-sdp

INVITE sip:user@[2001:db8::10] SIP/2.0
To: sip:user@[2001:db8::10]
From: sip:user@example.com;tag=81x2
Via: SIP/2.0/UDP [2001:db8::9:1];branch=z9hG4bKas3-111
Call-ID: SSG9559905523997077@hlau_4100
Contact: "Caller" <sip:caller@[2001:db8::1]>
CSeq: 8912 INVITE
Content-Type: application/sdp
Content-Length: 181

v=0
o=bob 280744730 28977631 IN IP4 host.example.com
s=
t=0 0
m=audio 22334 RTP/AVP 0
c=IN IP4 192.0.2.1
m=video 6024 RTP/AVP 107
c=IN IP6 2001:db8::1
a=rtpmap:107 H263-1998/90000
4.8 More test cases

TBD. Looking for more test cases...suggestions welcome.

5. Insulating Your Implementation with IPv6 APIs: Source Code Examples

TBD.

Things to write here include the use of struct sockaddr_storage to insulate the application, creating sockets using AF_INET6, impact of these on IPv4 nodes and so on.

6. Security Considerations

This document does not introduce any new security considerations beyond those that are already well known and documented in [1].

This document is not a comprehensive compilation of attacks possible on SIP systems. It contains some common pitfalls that the authors have discovered while parsing IPv6 identifiers in SIP implementations.

7. IANA Considerations

This document has no actions for IANA.

8. Acknowledgments

The authors acknowledge the gracious help provided by Dennis Bijwaard, Gonzalo Camarillo, Bob Gilligan, Larry Kollasch, Erik Nordmark, Kumiko Ono and Robert Sparks.

The appendix contains a bit-exact archive of each message following the convention established by Robert Sparks.

9. References

9.1 Normative References


9.2 Informative References


Authors’ Addresses

Vijay Gurbani (Ed.)
Lucent Technologies/Bell Laboratories
2000 Lucent Lane
Rm 6G-440
Naperville, IL  60566
USA
Phone: +1 630 224 0216
Email: vkg@lucent.com

Chris Boulton
Ubiquity Software Corporation
Building 3
West Fawr Lane
St Mellons
Cardiff, South Wales  CF3 5EA

Email: cboulton@ubiquitysoftware.com

Appendix A.  Bit-exact archive of each test message

The following text block is an encoded, gzip compressed TAR archive of files that represent each of the example messages discussed in Section 4.
To recover the compressed archive file intact, the text of this
document may be passed as input to the following Perl script (the
output should be redirected to a file or piped to "tar -xzvf -").

#!/usr/bin/perl
use strict;
my $bdata = "";
use MIME::Base64;
while(<>) {
  if (/-- BEGIN MESSAGE ARCHIVE --/ .. /-- END MESSAGE ARCHIVE --/) {
    if ( m/^\s*[^\s]*\s*$/) {
      $bdata = $bdata . $; ;
    }
  }
}
print decode_base64($bdata);

Alternatively, the base-64 encoded block can be edited by hand to
remove document structure lines and fed as input to any base-64
decoding utility.

A.1 Encoded Reference Messages

-- BEGIN MESSAGE ARCHIVE --
H4sICPXFRkMAA3h4LnRhcgDtWVtvzYUzrN+BdGXpck+h6REUqGbmWGSsG/YCFENQ0
D3yz7Muk2Sj2a8f2Ve+50alqBJ00QfbMsVzdCjyfB9jaXRj3GmaTk4aBCCAz/mj/QXG/b
3jBh7zTxAYRcp9hsyeR2oJ9Bko2osi1LnNuTqr+mjdsfqxxth1FZhgt7lklsEj+\/Pyd
F1AXLwuRvZm1RdswnHwncL0xmnrNMr0+13aAecq0kFD67xtk/+PaHaQgsliBoEK8DrrwYPT
Ua6T8szLR80u+Og3XTAExe/6InTsh3bLYYf5bK6lkDibL3U09z2k/\NB5eHYrMPW861sX4
R7056e5Gzo2ZH4Qs0GQ6sB7bw0iIAh3iz0890gqemcDczfA2EKpSC2z5xhGuz6pO
60xJSn26G6ShjxwXwDkX/\IOvsf0dtvaTuInj3B5H1Kyem/\9c2fDf+DB33smpT8\fwL
0f/qDFcY1gC\v\\\nHEz+i\\\\\nH\n\n
-- END MESSAGE ARCHIVE --
Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2005). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.