Internet Engineering Task Force
INTERNET-DRAFT
draft-ietf-mmusic-sip-07.ps

MMUSIC WG
Handley/Schulzrinne/Schooler/Rosenberg
ISI/Columbia U./Caltech/Bell Labs.

July 16, 1998
Expires: December 1998

SIP: Session Initiation Protocol

Status of this Memo

Copyright Notice

Abstract

The Session Initiation Protocol (SIP) is an application-layer control (signaling) protocol for creating, modifying and terminating sessions with one or more participants. These sessions include Internet multimedia conferences, Internet telephone calls and multimedia distribution. Members in a session can communicate via multicast or via a mesh of unicast relations, or a combination of these.

SIP invitations used to create sessions carry session descriptions which allow participants to agree on a set of compatible media types. It supports user mobility by proxying and redirecting requests to the user's current location. Users can register their current location. SIP is not tied to any particular conference control protocol. SIP is designed to be independent of the lower-layer transport protocol and can be extended with additional capabilities.

This document is a product of the Multi-party Multimedia Session Control (MMUSIC) working group of the Internet Engineering Task Force. Comments are solicited and should be addressed to the working group's mailing list at confctrl@isi.edu and/or the authors.

Contents

1 Introduction

1.1 Overview of SIP Functionality

The Session Initiation Protocol (SIP) is an application-layer control protocol that can establish, modify and terminate multimedia sessions or calls. These multimedia sessions include multimedia conferences, distance learning, Internet telephony and similar applications. SIP can invite both persons and "robots", such as a media storage service. SIP can invite parties to both unicast and multicast sessions; the initiator does not necessarily have to be a member of the session to which it is inviting. Media and participants can be added to an existing session.

SIP can be used to initiate sessions as well as invite members to sessions that have been advertised and established by other means. Sessions may be advertised using multicast protocols such as SAP, electronic mail, news groups, web pages or directories (LDAP), among others.

SIP transparently supports name mapping and redirection services, allowing the implementation of ISDN and Intelligent Network telephony subscriber services. These facilities also enable *personal mobility*. In the parlance of telecommunications intelligent network services, this is defined as: "Personal mobility is the ability of end users to originate and receive calls and access subscribed telecommunication services on any terminal in any location, and the ability of the network to identify end users as they move. Personal mobility is based on the use of a unique personal identity (i.e., 'personal number')." [1, p. 44]. Personal mobility complements terminal mobility, i.e., the ability to maintain communications when moving a single end system from one subnet to another.

SIP supports five facets of establishing and terminating multimedia communications:

User location: determination of the end system to be used for communication;

User capabilities: determination of the media and media parameters to be used;

User availability: determination of the willingness of the called party to engage in communications;

Call setup: "ringing", establishment of call parameters at both called and calling party;

Call handling: including transfer and termination of calls.

SIP can also initiate multi-party calls using a multipoint control unit (MCU) or fully-meshed interconnection instead of multicast. Internet telephony gateways that connect PSTN parties may also use SIP to set up calls between them.

SIP is designed as part of the overall IETF multimedia data and control architecture currently incorporating protocols such as RSVP (RFC 2205 [2]) for reserving network resources, the real-time transport protocol (RTP) (RFC 1889 [3]) for transporting real-time data and providing QOS feedback, the real-time streaming protocol (RTSP) (RFC 2326 [4]) for controlling delivery of streaming media, the session announcement protocol (SAP) for advertising multimedia sessions via multicast and the session description protocol (SDP) (RFC 2327 [5]) for describing multimedia sessions. However, the functionality and operation of SIP does not depend on any of these protocols.

SIP may also be used in conjunction with other call setup and signaling protocols. In that mode, an end system uses SIP exchanges to determine the appropriate end system address and protocol from a given address that is protocol-independent. For example, SIP could be used to determine that the party may be reached via H.323, obtain the H.245 gateway and user address and then use H.225.0 to establish the call. In another example, it may be used to determine that the callee is reachable via the public switched telephone network (PSTN) and indicate the phone number to be called, possibly suggesting an Internet-to-PSTN gateway to be used.

SIP does not offer conference control services such as floor control or voting and does not prescribe how a conference is to be managed, but SIP can be used to introduce conference control protocols. SIP does not allocate multicast addresses.

SIP can invite users to sessions with and without resource reservation. SIP does not reserve resources, but may convey to the invited system the information necessary to do this. Quality-of-service guarantees, if required, may depend on knowing the full membership of the session; this information may or may not be known to the agent performing session invitation.

1.2 Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [6] and indicate requirement levels for compliant SIP implementations.

1.3 Definitions

This specification uses a number of terms to refer to the roles played by participants in SIP communications. The definitions of client, server and proxy are similar to those used by the Hypertext Transport Protocol (HTTP) (RFC 2068 [7]). The following terms have special significance for SIP.

Call: A call consists of all participants in a conference invited by a common source. A SIP call is identified by a globally unique call-id (Section 6.12). Thus, if a user is, for example, invited to the same multicast session by several people, each of these invitations will be a unique call. A point-to-point Internet telephony conversation maps into a single SIP call. In a MCU-based call-in conference, each participant uses a separate call to invite himself to the MCU.

Call leg: A call leg is identified by the combination of Call-ID, To and From.

Client: An application program that establishes connections for the purpose of sending requests. Clients may or may not interact directly with a human user. *User agents* and *proxies* contain clients (and servers).

Conference: A multimedia session (see below), identified by a common session description. A conference may have zero or more members and includes the cases of a multicast conference, a full-mesh conference and a two-party "telephone call", as well as combinations of these. Any number of calls may be used to create a conference.

Downstream: Requests sent in the direction from the caller to the callee.

Final response: A response that terminates a SIP transaction, as opposed to a *provisional response* that does not. All 2xx, 3xx, 4xx, 5xx and 6xx responses are final.

Initiator, calling party, caller: The party initiating a conference invitation. Note that the calling party does not have to be the same as the one creating the conference.

Invitation: A request sent to a user (or service) requesting participation in a session. A successful SIP invitation consists of two transactions: an INVITE request followed by an ACK request.

Invitee, invited user, called party, callee: The person or service that the calling party is trying to invite to a conference.

Isomorphic request or response: Two requests or responses are defined to be *isomorphic* for the purposes of this document if they have the same values for the Call-ID, To, From and CSeq header fields. In addition, requests have to have the same Request-URI.

Location server: See *location service*.

Location service: A location service is used by a SIP redirect or proxy server to obtain information about a callee's possible location(s). Location services are offered by location servers. Location servers may be co-located with a SIP server, but the manner in which a SIP server requests location services is beyond the scope of this document.

Parallel search: In a parallel search, a proxy issues several requests to possible user locations upon receiving an incoming request. Rather than issuing one request and then waiting for the final response before issuing the next request as in a *sequential search*, a parallel search issues requests without waiting for the result of previous requests.

Provisional response: A response used by the server to indicate progress, but that does not terminate a SIP transaction. 1xx responses are provisional, other responses are considered *final*.

Proxy, proxy server: An intermediary program that acts as both a server and a client for the purpose of making requests on behalf of other clients. Requests are serviced internally or by passing them on, possibly after translation, to other servers. A proxy must interpret, and, if necessary, rewrite a request message before forwarding it.

Redirect server: A redirect server is a server that accepts a SIP request, maps the address into zero or more new addresses and returns these addresses to the client. Unlike a *proxy server*, it does not initiate its own SIP request. Unlike a *user agent server*, it does not accept calls.

Registrar: A registrar is server that accepts REGISTER requests. A registrar is typically co-located with a proxy or redirect server and may offer location services.

Ringback: Ringback is the signaling tone produced by the calling client's application indicating that a called party is being alerted (ringing).

Server: A server is an application program that accepts requests in order to service requests and sends back responses to those requests. Servers are either proxy, redirect or user agent servers or registrars.

Session: "A multimedia session is a set of multimedia senders and receivers and the data streams flowing from senders to receivers. A multimedia conference is an example of a multimedia session." (RFC 2327 [5]) (A session as defined for SDP may comprise one or more RTP sessions.) As defined, a callee may be invited several times, by different calls, to the same session. If SDP is used, a session is defined by the concatenation of the *user name*, *session id*, *network type*, *address type* and *address* elements in the origin field.

(SIP) transaction: A SIP transaction occurs between a client and a server and comprises all messages from the first request sent from the client to the server up to a final (non-1xx) response sent from the server to the client. A transaction is identified by the CSeq sequence number (Section 6.16) within a single *call leg*. The ACK request has the same CSeq number as the corresponding INVITE request, but comprises a transaction of its own.

Upstream: Responses sent in the direction from the called client to the caller.

URL-encoded: A character string encoded according to RFC 1738, Section 2.2 [8].

User agent client (UAC), calling user agent: A user agent client is a client application that initiates the SIP request.

User agent server (UAS), called user agent: A user agent server is a server application that contacts the user when a SIP request is received and that returns a response on behalf of the user. The response may accept, reject or redirect the request.

An application program may be capable of acting both as a client and a server. For example, a typical multimedia conference control application would act as a user agent client to initiate calls or to invite others to conferences and as a user agent server to accept invitations. The properties of the different SIP server types are summarized in Table 1.

1.4 Summary of SIP Operation

This section explains the basic protocol functionality and operation. Callers and callees are identified by SIP addresses, described in Section 1.4.1. When making a SIP call, a caller first locates the appropriate server (Section 1.4.2) and then sends a SIP request (Section 1.4.3). The most common SIP operation is the invitation (Section 1.4.4). Instead of directly reaching the intended callee, a SIP request may be redirected or may trigger a chain of new SIP requests by proxies (Section 1.4.5). Users can register their location(s) with SIP servers (Section 4.2.6).

property	redirect server	proxy server	user agent server	registrar
also acts as a SIP client	no	yes	no	no
returns 1xx status	yes	yes	yes	rare
returns 2xx status	no	yes	yes	yes
returns 3xx status	yes	yes	yes	yes
returns 4xx status	yes	yes	yes	yes
returns 5xx status	yes	yes	yes	yes
returns 6xx status	no	yes	yes	no
inserts Via header	no	yes	no	no
accepts ACK	yes	yes	yes	no

Table 1: Properties of the different SIP server types

1.4.1 SIP Addressing

The "objects" addressed by SIP are users at hosts, identified by a SIP URL. The SIP URL takes the form similar to a mailto or telnet URL, i.e., *user@host*. The *user* part is a user name, a civil name or a telephone number. The *host* part is either a domain name having a DNS SRV (RFC 2052 [9]), MX (RFC 974 [10], CNAME or A record (RFC 1035 [11]), or a numeric network address.

A user's SIP address can be obtained out-of-band, can be learned via existing media agents, can be included in some mailers' message headers, or can be recorded during previous invitation interactions. In many cases, a user's SIP URL can be guessed from his email address.

Examples of SIP URLs include:

```
sip:mjh@metro.isi.edu
sip:watson@bell-telephone.com
sip:root@193.175.132.42
sip:info@ietf.org
```

A SIP URL address can designate an individual (possibly located at one of several end systems), the first available person from a group of individuals or a whole group. The form of the address, e.g., `sip:sales@example.com`, is not sufficient, in general, to determine the intent of the caller.

If a user or service chooses to be reachable at an address that is guessable from the person's name and organizational affiliation, the traditional method of ensuring privacy by having an unlisted "phone" number is compromised. However, unlike traditional telephony, SIP offers authentication and access control mechanisms and can avail itself of lower-layer security mechanisms, so that client software can reject unauthorized or undesired call attempts.

1.4.2 Locating a SIP Server

A SIP client MUST follow the following steps to resolve the *host* part of a callee address. If a client supports only TCP or UDP, but not both, the client omits the respective address type. If the SIP address contains a port number, that number is to be used, otherwise, the default port number 5060 is to be used. The default port number is the same for UDP and TCP. In all cases, the client first attempts to contact the server using

UDP, then TCP.

A client SHOULD rely on ICMP "Port Unreachable" messages rather than time-outs to determine that a server is not reachable at a particular address. (For socket-based programs: For TCP, `connect()` returns `ECONNREFUSED` if there is no server at the designated address; for UDP, the socket should be bound to the destination address using `connect()` rather than `sendto()` or similar so that a second `write()` fails with `ECONNREFUSED`.)

If the SIP address contains a numeric IP address, the client contacts the SIP server at that address. Otherwise, the client follows the steps below.

1. If there is a SRV DNS resource record (RFC 2052 [9]) of type sip.udp, contact the listed SIP servers in the order of the preference values contained in those resource records, using UDP as a transport protocol at the port number given in the URL or, if none provided, the one listed in the DNS resource record.

2. If there is a SRV DNS resource record (RFC 2052 [9]) of type sip.tcp, contact the listed SIP servers in the order of the preference value contained in those resource records, using TCP as a transport protocol at the port number given in the URL or, if none provided, the one listed in the DNS resource record.

3. If there is a DNS MX record (RFC 974 [10]), contact the hosts listed in their order of preference at the port number listed in the URL or the default SIP port number if none. For each host listed, first try to contact the SIP server using UDP, then TCP.

4. Finally, check if there is a DNS CNAME or A record for the given *host* and try to contact a SIP server at the one or more addresses listed, again trying first UDP, then TCP.

If all of the above methods fail to locate a server, the caller MAY contact an SMTP server at the user's *host* and use the SMTP EXPN command to obtain an alternate address and repeat the steps above. As a last resort, a client MAY choose to deliver the session description to the callee using electronic mail.

A client MAY cache the result of the reachability steps for a particular address and retry that host address for the next call. If the client does not find a SIP server at the cached address, it MUST start the search at the beginning of the sequence.

> This sequence is modeled after that described for SMTP, where MX records are to be checked before A records (RFC 1123 [12]).

1.4.3 SIP Transaction

Once the *host* part has been resolved to a SIP server, the client sends one or more SIP requests to that server and receives one or more responses from the server. A request (and its retransmissions) together with the responses triggered by that request make up a SIP transaction. The ACK request following an INVITE is *not* part of the transaction since it may traverse a different set of hosts.

If TCP is used, request and responses within a single SIP transaction are carried over the same TCP connection (see Section 10). Several SIP requests from the same client to the same server may use the same TCP connection or may open a new connection for each request.

If the client sent the request via unicast UDP, the response is sent to the address contained in the next Via header field (Section 6.40) of the response. If the request is sent via multicast UDP, the response is directed to the same multicast address and destination port. For UDP, reliability is achieved using retransmission (Section 10).

The SIP message format and operation is independent of the transport protocol.

1.4.4 SIP Invitation

A successful SIP invitation consists of two requests, INVITE followed by ACK. The INVITE (Section 4.2.1) request asks the callee to join a particular conference or establish a two-party conversation. After the callee has agreed to participate in the call, the caller confirms that it has received that response by sending an ACK (Section 4.2.2) request. If the caller no longer wants to participate in the call, it sends a BYE request instead of an ACK.

The INVITE request typically contains a session description, for example written in SDP (RFC 2327 [5]) format, that provides the called party with enough information to join the session. For multicast sessions, the session description enumerates the media types and formats that may be distributed to that session. For a unicast session, the session description enumerates the media types and formats that the caller is willing to receive and where it wishes the media data to be sent. In either case, if the callee wishes to accept the call, it responds to the invitation by returning a similar description listing the media it wishes to receive. For a multicast session, the callee should only return a session description if it is unable to receive the media indicated in the caller's description or wants to receive data via unicast.

The protocol exchanges for the INVITE method are shown in Fig. 1 for a proxy server and in Fig. 2 for a redirect server. (Note that the messages shown in the figures have been abbreviated slightly.) In Fig. 1, the proxy server accepts the INVITE request (step 1), contacts the location service with all or parts of the address (step 2) and obtains a more precise location (step 3). The proxy server then issues a SIP INVITE request to the address(es) returned by the location service (step 4). The user agent server alerts the user (step 5) and returns a success indication to the proxy server (step 6). The proxy server then returns the success result to the original caller (step 7). The receipt of this message is confirmed by the caller using an ACK request, which is forwarded to the callee (steps 8 and 9). All requests and responses have the same Call-ID.

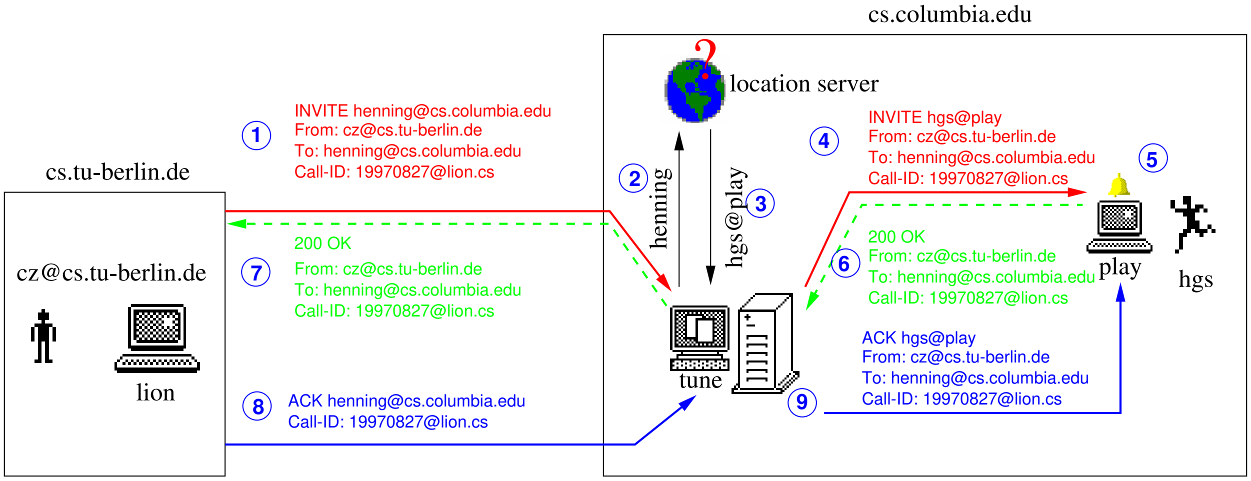

Figure 1: Example of SIP proxy server

The redirect server shown in Fig. 2 accepts the INVITE request (step 1), contacts the location service as before (steps 2 and 3) and, instead of contacting the newly found address itself, returns the address to the caller (step 4), which is then acknowledged via an ACK request (step 5). The caller issues a new request, with the same call-ID but a higher CSeq, to the address returned by the first server (step 6). In the example, the call succeeds (step 7). The caller and callee complete the handshake with an ACK (step 8).

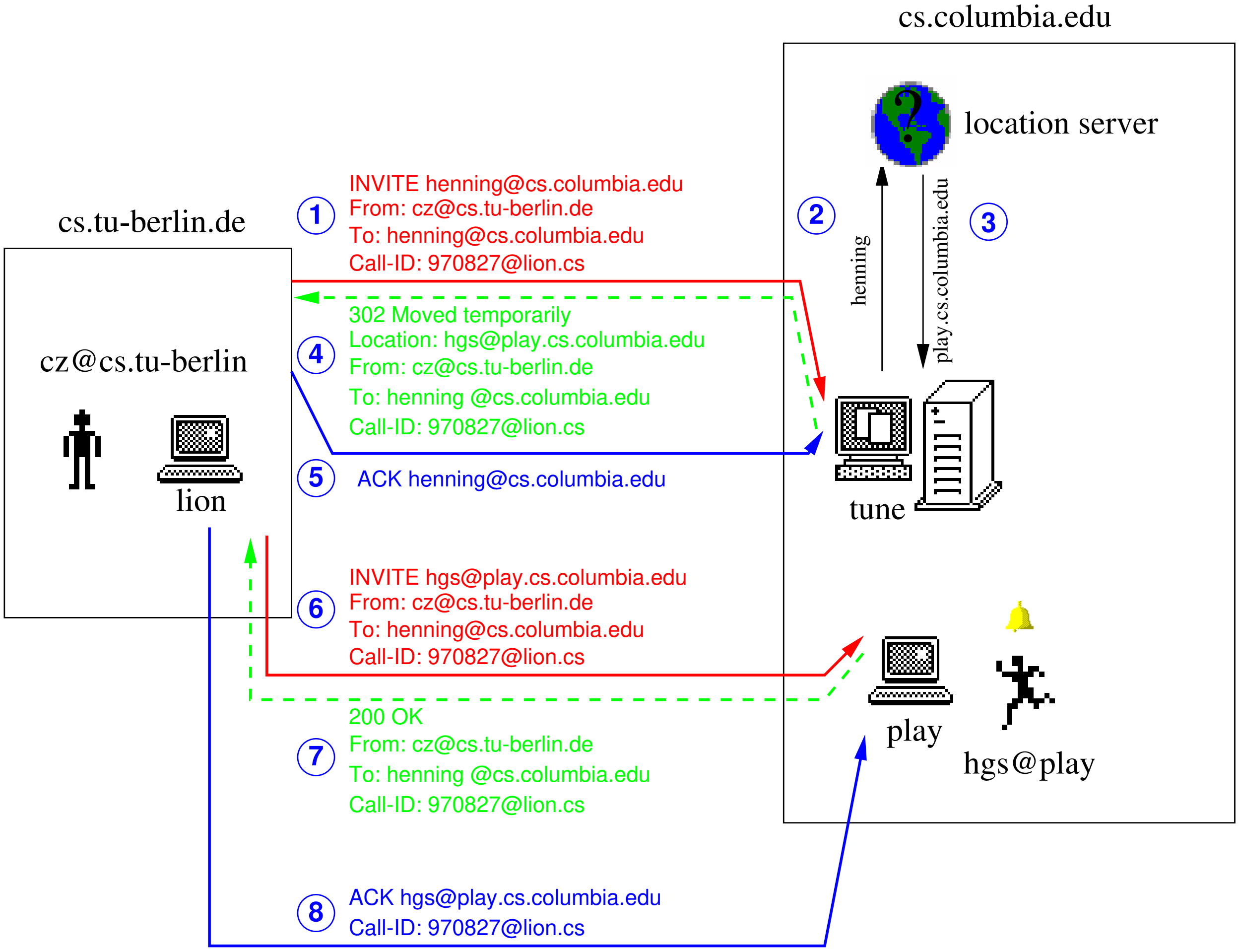

Figure 2: Example of SIP redirect server

The next section discusses what happens if the location service returns more than one possible alternative.

1.4.5 Locating a User

A callee may move between a number of different end systems over time. These locations can be dynamically registered with the SIP server (Sections 1.4.7, 4.2.6). A location server may also use one or more other protocols, such as finger (RFC 1288 [13]), rwhois (RFC 2167 [14]), LDAP (RFC 1777 [15]), multicast-based protocols [16] or operating-system dependent mechanisms to actively determine the end system where a user might be reachable. A location server may return several locations because the user is logged in at several hosts simultaneously or because the location server has (temporarily) inaccurate information. The SIP server combines the results to yield a list of a zero or more locations. It is recommended that each location server sorts results according to the likelihood of success.

The action taken on receiving a list of locations varies with the type of SIP server. A SIP redirect server returns the list to the client as Location headers (Section 6.22). A SIP proxy server can sequentially or in parallel try the addresses until the call is successful (2xx response) or the callee has declined the call (6xx response). With sequential attempts, a proxy server can implement an “anycast” service.

If a proxy server forwards a SIP request, it MUST add itself to the end of the list of forwarders noted in the Via (Section 6.40) headers. The Via trace ensures that replies can take the same path back, ensuring correct operation through compliant firewalls and avoiding request loops. On the response path, each host MUST remove its Via, so that routing internal information is hidden from the callee and outside networks. When a multicast request is made, first the host making the request, then the multicast address itself are added to the path. A proxy server MUST check that it does not generate a request to a host listed in the Via list. (Note: If a host has several names or network addresses, this may not always work. Thus, each host also checks if it is part of the Via list.)

A SIP invitation may traverse more than one SIP proxy server. If one of these “forks” the request, i.e., issues more than one request in response to receiving the invitation request, it is possible that a client is reached, independently, by more than one copy of the invitation request. Each of these copies bears the same Call-ID. The user agent MUST return the appropriate status response. Duplicate requests are not an error.

1.4.6 Changing an Existing Session

In some circumstances, it may be necessary to change the parameters of an existing session. For example, two parties may have been conversing and then want to add a third party, switching to multicast for efficiency. One of the participants invites the third party with the new multicast address and simultaneously sends an INVITE to the second party, with the new multicast session description, but with the old call identifier.

1.4.7 Registration Services

The REGISTER request allows a client to let a proxy or redirect server know at which address(es) it may be reached. A client may also use it to install call handling features at the server.

1.5 Protocol Properties

1.5.1 Minimal State

A single conference session or call may involve one or more SIP request-response transactions. Proxy servers do not have to keep state for a particular call, however, they MAY maintain state for a single SIP transaction, as discussed in Section 11.

For efficiency, a server may cache the results of location service requests.

1.5.2 Lower-Layer-Protocol Neutral

SIP makes minimal assumptions about the underlying transport and network-layer protocols. The lower-layer may provide either a packet or a byte stream service, with reliable or unreliable service.

In an Internet context, SIP is able to utilize both UDP and TCP as transport protocols, among others. UDP allows the application to more carefully control the timing of messages and their retransmission, to perform parallel searches without requiring TCP connection state for each outstanding request, and to use multicast. Routers can more readily snoop SIP UDP packets. TCP allows easier passage through existing firewalls, and given the similar protocol design, allows common servers for SIP, HTTP and the Real Time Streaming Protocol (RTSP) (RFC 2326 [4]).

When TCP is used, SIP can use one or more connections to attempt to contact a user or to modify parameters of an existing conference. Different SIP requests for the same SIP call may use different TCP connections or a single persistent connection, as appropriate.

For concreteness, this document will only refer to Internet protocols. However, SIP may also be used directly with protocols such as ATM AAL5, IPX, frame relay or X.25. The necessary naming conventions are beyond the scope of this document. User agents SHOULD implement both UDP and TCP transport, proxy and redirect servers MUST.

1.5.3 Text-Based

SIP is text-based, using ISO 10646 in UTF-8 encoding throughout. This allows easy implementation in languages such as Java, Tcl and Perl, allows easy debugging, and most importantly, makes SIP flexible and extensible. As SIP is used for initiating multimedia conferences rather than delivering media data, it is believed that the additional overhead of using a text-based protocol is not significant.

2 SIP Uniform Resource Locators

SIP URLs are used within SIP messages to indicate the originator (From), current destination (Request-URI) and final recipient (To) of a SIP request, and to specify redirection addresses (Location). A SIP URL can also be embedded in web pages or other hyperlinks to indicate that a user or service may be called.

Because interaction with some resources may require message headers or message bodies to be specified as well as the SIP address, the SIP URL scheme is defined to allow setting SIP request-header fields and the SIP message-body.

A SIP URL follows the guidelines of RFC 1630 [17], as revised [18], and has the syntax shown in Fig. 3. Note that reserved characters have to be escaped.

The URI character classes referenced above are described in Section C. The URI specification is currently being revised. It is anticipated that future versions of this specification will reference the revised edition. Note that all URL reserved characters MUST be encoded.

host: The mailto: URL and RFC 822 email addresses require that numeric host addresses ("host numbers") are enclosed in square brackets (presumably, since host names might be numeric), while host numbers without brackets are used for all other URLs. The SIP URL requires the latter form, without brackets.

userinfo: The SIP scheme MAY use the format "user:password" in the userinfo field. The use of passwords in the userinfo is NOT RECOMMENDED, because the passing of authentication information in clear text (such as URIs) has proven to be a security risk in almost every case where it has been used.

If the host is an Internet telephony gateway, the userinfo field can also encode a telephone number using the notation of telephone-subscriber (Fig. 4). The telephone number is a special case of a user name and cannot be distinguished by a BNF. Thus, a URL parameter, user, is added to distinguish telephone numbers from user names. The phone identifier is to be used when connecting to a telephony gateway. Even without this parameter, recipients of SIP URLs MAY interpret the pre-@ part as a phone number if local restrictions on the name space for user name allow it.

```
SIP-URL          =  "sip:" [ userinfo ] "@" hostport
                    url-parameters [ headers ]
userinfo         =  user [ ":" password ]
user             =  *( unreserved | escaped |
                    ";" | "&" | "=" | "+" | "$" | "," )
password         =  *( unreserved | escaped |
                    ";" | "&" | "=" | "+" | "$" | "," )
hostport         =  host [ ":" port ]
host             =  hostname | IPv4address
hostname         =  *( domainlabel "." ) toplabel [ "." ]
domainlabel      =  alphanum
                 |  alphanum *( alphanum | "-" ) alphanum
toplabel         =  alpha | alpha *( alphanum | "-" ) alphanum
IPv4address      =  1*digit "." 1*digit "." 1*digit "." 1*digit
port             =  *digit
url-parameters   =  *( ";" url-parameter)
url-parameter    =  transport-param | user-param
                 |  ttl-param | maddr-param | tag-param
                 |  other-param
transport-param  =  "transport=" ( "udp" | "tcp" )
ttl-param        =  "ttl=" ttl
ttl              =  1*3DIGIT ; 0 to 255
maddr-param      =  "maddr=" maddr
maddr            =  IPv4address ; multicast address
user-param       =  "user=" ( "phone" )
tag-param        =  "tag=" UUID
other-param      =  *uric
headers          =  "?" header *( "&" header )
header           =  hname "=" hvalue
hname            =  *uric
hvalue           =  *uric
uric             =  reserved | unreserved | escaped
reserved         =  ";" | "/" | "?" | ":" | "@" | "&"
                 |  "=" | "+" | "$" | ","
digits           =  1*DIGIT
UUID             =  1*( hex | "-" )
```

Figure 3: SIP URL syntax

If a server handles SIP addresses for another domain, it MUST URL-encode the "@" character (%40).

URL parameters: SIP URLs can define specific parameters of the request, including the transport mechanism (UDP or TCP) and the use of multicast to make a request. These parameters are added after the host and are separated by semi-colons. For example, to specify to call `j.doe@big.com` using

```
telephone-subscriber  =  global-phone-number | local-phone-number
global-phone-number   =  "+" 1*phonedigit [ isdn-subaddress ]
                         [ post-dial ]
local-phone-number    =  1*( phonedigit | dtmf-digit
                      |  pause-character) [ isdn-subaddress ]
                         [ post-dial ]
isdn-subaddress       =  ";isub=" 1*phonedigit
post-dial             =  ";postd=" 1*( phonedigit | dtmf-digit
                      |  pause-character )
phonedigit            =  DIGIT | visual-separator
visual-separator      =  "-" | "."
pause-character       =  one-second-pause | wait-for-dial-tone
one-second-pause      =  "p"
wait-for-dial-tone    =  "w"
dtmf-digit            =  "*" | "#" | "A" | "B" | "C" | "D"
```

Figure 4: SIP URL syntax; telephone subscriber

multicast to 239.255.255.1 with a ttl of 15, the following URL would be used:

```
sip:j.doe@big.com;maddr=239.255.255.1;ttl=15
```

The transport protocol UDP is to be assumed when a multicast address is given.

Transport parameters MUST NOT be used in the From and To header fields and the Request-URI; they are ignored if present.

Headers: Headers of the SIP request can be defined with the "?" mechanism within a SIP URL. The special hname "body" indicates that the associated hvalue is the message-body of the SIP INVITE request. Headers MUST NOT be used in the From and To header fields and the Request-URI; they are ignored if present.

Tag: The tag parameter allows several instances of a user that share the same host and port values to be distinguished from each other, for example, where the host designates a firewall or proxy. The tag value is a random string consisting of hex digits. The use of version-1 (time-based) or version-4 (random) UUID [19] is OPTIONAL. The tag value is designed to be globally unique and cryptographically random with at least 32 bits of randomness. It SHOULD NOT be included in long-lived SIP URLs, e.g., those found on web pages or user databases. A single user maintains the same tag throughout the call identified by the Call-ID. The tag parameter in To headers is ignored when matching responses to requests that did not contain a tag in their To header. (See Section 6.37.)

Table 2 summarizes where the components of the SIP URL can be used.
Examples of SIP URLs are:

```
sip:j.doe@big.com
```

	Request-URI	To	From	Location	external
user	x	x	x	x	x
password		x		x	x
host	x	x	x	x	x
tag	x	x	x	x	
headers				x	x
transport para.				x	x

Table 2: Use of URL elements for SIP headers, Request-URI and references

```
sip:j.doe:secret@big.com;transport=tcp
sip:j.doe@big.com?subject=project%20x&priority=urgent
sip:+1-212-555-1212:1234@gateway.com;user=phone
sip:1212@gateway.com
sip:alice@10.1.2.3
sip:alice@example.com;tag=f81d4fae-7dec-11d0-a765-00a0c91e6bf6
sip:alice%40example.com@gateway.com
```

Within a SIP message, URLs are used to indicate the source and intended destination of a request, redirection addresses and the current destination of a request. Normally all these fields will contain SIP URLs.

SIP URLs are case-insensitive, so that for example the two URLs `sip:j.doe@example.com` and `SIP:J.Doe@Example.com` are equivalent. All URL parameters are included when comparing SIP URLs for equality.

In some circumstances a non-SIP URL may be used in a SIP message. An example might be making a call from a telephone which is relayed by a gateway onto the internet as a SIP request. In such a case, the source of the call is really the telephone number of the caller, and so a SIP URL is inappropriate and a phone URL might be used instead. To allow for this flexibility, SIP headers that specify user addresses allow these addresses to be SIP and non-SIP URLs.

Clearly not all URLs are appropriate to be used in a SIP message as a user address. The correct behavior when an unknown scheme is encountered by a SIP server is defined in the context of each of the header fields that use a SIP URL.

3 SIP Message Overview

SIP is a text-based protocol and uses the ISO 10646 character set in UTF-8 encoding (RFC 2279 [20]). Lines are terminated by CRLF, but receivers should be prepared to also interpret CR and LF by themselves as line terminators.

Except for the above difference in character sets, much of the message syntax is identical to HTTP/1.1; rather than repeating it here we use [HX.Y] to refer to Section X.Y of the current HTTP/1.1 specification (RFC 2068 [7]). In addition, we describe SIP in both prose and an augmented Backus-Naur form (BNF) [H2.1] described in detail in RFC 2234 [21].

Unlike HTTP, SIP MAY use UDP. When sent over TCP or UDP, multiple SIP transactions can be carried in a single TCP connection or UDP datagram. UDP datagrams, including all headers, should not normally be larger than the path maximum transmission unit (MTU) if the MTU is known, or 1400 bytes if the MTU is unknown.

The 1400 bytes accommodates lower-layer packet headers within the "typical" MTU of around 1500 bytes. Recent studies [22, p. 154] indicate that an MTU of 1500 bytes is a reasonable assumption. The next lower common MTU values are 1006 bytes for SLIP and 296 for low-delay PPP (RFC 1191 [23]). Thus, another reasonable value would be a message size of 950 bytes, to accommodate packet headers within the SLIP MTU without fragmentation.

A SIP message is either a request from a client to a server, or a response from a server to a client.

```
SIP-message  =  Request | Response
```

Both Request (section 4) and Response (section 5) messages use the generic-message format of RFC 822 [24] for transferring entities (the body of the message). Both types of messages consist of a start-line, one or more header fields (also known as "headers"), an empty line (i.e., a line with nothing preceding the carriage-return line-feed (CRLF)) indicating the end of the header fields, and an optional message-body. To avoid confusion with similar-named headers in HTTP, we refer to the header describing the message body as entity headers. These components are described in detail in the upcoming sections.

```
generic-message  =  start-line
                    *message-header
                    CRLF
                    [ message-body ]

start-line       =  Request-Line |     Section 4.1
                    Status-Line        Section 5.1

message-header   =  *( general-header
                    | request-header
                    | response-header
                    | entity-header )
```

In the interest of robustness, any leading empty line(s) MUST be ignored. In other words, if the Request or Response message begins with a CRLF, CR, or LF, these characters should be ignored.

```
general-header   =  Call-ID               ; Section 6.12
                 |  CSeq                  ; Section 6.16
                 |  Date                  ; Section 6.17
                 |  Encryption            ; Section 6.18
                 |  Expires               ; Section 6.19
                 |  From                  ; Section 6.20
                 |  Record-Route          ; Section 6.29
                 |  Timestamp             ; Section 6.36
                 |  To                    ; Section 6.37
                 |  Via                   ; Section 6.40
entity-header    =  Content-Encoding      ; Section 6.13
                 |  Content-Length        ; Section 6.14
                 |  Content-Type          ; Section 6.15
request-header   =  Accept                ; Section 6.7
                 |  Accept-Encoding       ; Section 6.8
                 |  Accept-Language       ; Section 6.9
                 |  Authorization         ; Section 6.11
                 |  Hide                  ; Section 6.21
                 |  Location              ; Section 6.22
                 |  Max-Forwards          ; Section 6.23
                 |  Organization          ; Section 6.24
                 |  Priority              ; Section 6.25
                 |  Proxy-Authorization   ; Section 6.27
                 |  Proxy-Require         ; Section 6.28
                 |  Route                 ; Section 6.33
                 |  Require               ; Section 6.30
                 |  Response-Key          ; Section 6.31
                 |  Subject               ; Section 6.35
                 |  User-Agent            ; Section 6.39
response-header  =  Allow                 ; Section 6.10
                 |  Location              ; Section 6.22
                 |  Proxy-Authenticate    ; Section 6.26
                 |  Retry-After           ; Section 6.32
                 |  Server                ; Section 6.34
                 |  Unsupported           ; Section 6.38
                 |  Warning               ; Section 6.41
                 |  WWW-Authenticate      ; Section 6.42
```

Table 3: SIP headers

4 Request

The Request message format is shown below:

```
Request  =  Request-Line          ; Section 4.1
            *( general-header
            | request-header
            | entity-header )
            CRLF
            [ message-body ]      ; Section 8
```

4.1 Request-Line

The Request-Line begins with a method token, followed by the Request-URI and the protocol version, and ending with CRLF. The elements are separated by SP characters. No CR or LF are allowed except in the final CRLF sequence.

```
Request-Line  =  Method SP Request-URI SP SIP-Version CRLF
```

4.2 Methods

The methods are defined below. Methods that are not supported by a proxy or redirect server are treated by that server as if they were an INVITE method and forwarded accordingly. Methods that are not supported by a user agent server cause a 501 (Not Implemented) response to be returned (Section 7).

```
Method  =  "ACK" | "BYE" | "CANCEL" | "INVITE"
        |  "OPTIONS" | "REGISTER"
```

4.2.1 INVITE

The INVITE method indicates that the user or service is being invited to participate in a session. The message body contains a description of the session to which the callee is being invited. For two-party calls, the caller indicates the type of media it is able to receive as well as their parameters such as network destination. If the session description format allows this, it may also indicate "send-only" media. A success response indicates in its message body which media the callee wishes to receive.

A server MAY automatically respond to an invitation for a conference the user is already participating in, identified either by the SIP Call-ID or a globally unique identifier within the session description, with a 200 (OK) response.

If a user agent receives an INVITE request for an existing Call-ID with a higher CSeq sequence number than any previous INVITE for the same Call-ID, it MUST check any version identifiers in the session description or, if there are no version identifiers, the content of the session description to see if it has changed. It MUST also inspect any other header fields for changes and act accordingly. If the session description has changed, the user agent server MUST adjust the session parameters accordingly, possibly after asking the user for confirmation. (Versioning of the session description may be used to accommodate the capabilities of new arrivals to a conference, add or delete media or change from a unicast to a multicast conference.)

This method MUST be supported by SIP proxy, redirect and user agent servers as well as clients.

4.2.2 ACK

The ACK request confirms that the client has received a final response to an INVITE request. (ACK is used *only* with INVITE requests.) 2xx responses are acknowledged by client user agents, all other final responses by the first proxy or client user agent to receive the response. The Via is always initialized to the host that originates the ACK request, i.e., the client user agent after a 2xx response or the first proxy to receive a non-2xx final response. The ACK request is forwarded as the corresponding INVITE request, based on its Request-URI. See Section 10 for details.

The ACK request MAY contain a message body with the final session description to be used by the callee. If the ACK message body is empty, the callee uses the session description in the INVITE request.

This method MUST be supported by SIP proxy, redirect and user agent servers as well as clients.

4.2.3 OPTIONS

The client is being queried as to its capabilities. A server that believes it can contact the user, such as a user agent where the user is logged in and has been recently active, MAY respond to this request with a capability set. A called user agent MAY return a status reflecting how it would have responded to an invitation, e.g., 600 (Busy).

This method MUST be supported by SIP proxy, redirect and user agent servers, registrars and clients.

4.2.4 BYE

The user agent client uses BYE to indicate to the server that it wishes to abort the call. A BYE request is forwarded like an INVITE request. A caller SHOULD issue a BYE request before aborting a call ("hanging up"). Note that a BYE request may also be issued by the callee.

If the INVITE request contained a Location header, the callee sends the BYE request to that address rather than the From address.

This method MUST be supported by proxy servers and SHOULD be supported by redirect and user agent SIP servers.

4.2.5 CANCEL

The CANCEL request cancels a pending request with the same Call-ID, To, From and CSeq (sequence number only) header values, but does not affect a completed request. (A request is considered completed if the server has returned a final status response.)

A user agent client or proxy client MAY issue a CANCEL request at any time. A proxy, in particular, MAY choose to send a CANCEL to destinations that have not yet returned a final response after it has received a 2xx or 6xx response for one or more of the parallel-search requests. A proxy that receives a CANCEL request forwards the request to all destinations with pending requests. The Call-ID, To and From in the CANCEL request are identical to those contained in the request being canceled, but the Via header field is initialized to the proxy issuing the CANCEL request. (Thus, responses to this CANCEL request only reach the issuing proxy.)

Once a user agent server has received a CANCEL, it MUST NOT issue a 2xx response for the cancelled original request.

A redirect server or user agent server returns 200 (OK) if the Call-ID exists and 481 (Invalid Call-ID) if not, but takes no further action. In particular, any existing call is unaffected.

> The BYE request cannot be used to cancel branches of a parallel search, since several branches may, through intermediate proxies, find the same user agent server and then terminate the call. To terminate a call instead of just pending searches, the UAC must use BYE instead of or in addition to CANCEL. While CANCEL can terminate any pending request other than ACK or CANCEL, it is typically useful only for INVITE. 200 responses to INVITE and 200 responses to CANCEL are distinguished by the method in the Cseq header field, so there is no ambiguity.

This method MUST be supported by proxy servers and SHOULD be supported by all other SIP server types.

4.2.6 REGISTER

A client uses the REGISTER method to register the address listed in the To header with a SIP server.

A user agent SHOULD register with a local server on startup by sending a REGISTER request to the well-known "all SIP servers" multicast address, 224.0.1.75, with a time-to-live value of 1. SIP user agents on the same subnet MAY listen to that address and use it to become aware of the location of other local users [16]; however, they do not respond to the request.

The REGISTER request-header fields are defined as follows. We define "address-of-record" as the SIP address that the registry knows the registrand, typically of the form "user@domain" rather than "user@host". In third-party registration, the entity issuing the request is different from the entity being registered.

To: The To header field contains the address-of-record whose registration is to be created or updated.

From: The From header field contains the address-of-record of the person responsible for the registration. For first-party registration, it is identical to the To header field value.

Request-URI: The Request-URI names the destination of the registration request, i.e., the domain of the registrar. The user name MUST be empty. Generally, the domains in the Request-URI and the To header have the same value; however, it is possible to register as a "visitor", while maintaining one's name. For example, a traveller `sip:alice@acme.com` (To) may register under the Request-URI `sip:@atlanta.ayh.org`, with the former as the To field and the latter as the Request-URI. The request is no longer forwarded once it reached the server whose authoritative domain is the one listed in the Request-URI.

Location: The request MUST contain a Location header field; requests for the Request-URI will be directed to the address(es) given. It is RECOMMENDED that user agents include SIP URLs with both UDP and TCP transport parameters in their registration. If the registration contains a Location field whose URL includes a transport parameter, future requests will use that protocol. Otherwise, requests use the same transport protocol as used by the registration. However, a multicast REGISTER request still causes future requests to be unicast unless the maddr URL parameter explicitly requests otherwise. If the Location header does not contain a port number, the default SIP port number is used for future requests.

> We cannot require that registration and subsequent INVITE requests use the same transport protocol, as multicast registrations may be quite useful.

Registrations are additive, but all current locations must share the same action value. A proxy server ignores the q parameter, while a redirect server simply returns the parameter in its Location header.

> Having the proxy server interpret the q parameter is not sufficient to guide proxy behavior, as it is not clear, for example, how long it should wait between trying addresses.

If the registration is changed while a user agent or proxy server processes an invitation, the new information should be used.

This allows a service known as "directed pick-up".

A server SHOULD silently drop the registration after one hour, unless refreshed by the client. A client may request a lower or higher refresh interval through the Expires header (Section 6.19). Based on this request and its configuration, the server chooses the expiration interval and indicates it through the Expires header in the response. A single address (if host-independent) may be registered from several different clients.

A client cancels an existing registration by sending a REGISTER request with an expiration time (Expires) of zero seconds for a particular Location or the wildcard Location designated by a "*" for all registrations.

The server SHOULD return the current list of registrations in the 200 response as Location header fields.

It is particularly important that REGISTER requests are authenticated since they allow to redirect future requests.

Beyond its use as a simple location service, this method is needed if there are several SIP servers on a single host. In that case, only one of the servers can use the default port number. Each server that cannot would register with a server for the administrative domain. Since a client may not have easy access to the host address or port number, using the source address and port from the request itself seems simpler.

Support of this method is RECOMMENDED.

4.3 Request-URI

The Request-URI field is a SIP URL as described in Section 2 or a general URI. It indicates the user or service to which this request is being addressed. Unlike the To field, the Request-URI field may be re-written by proxies.

The SIP-URL MUST NOT contain the transport-param, maddr-param, ttl-param, or headers elements. A server that receives a SIP-URL with these elements removes them before further processing.

Typically, the UAC sets the Request-URI and To to the same SIP URL, presumed to remain unchanged over long time periods. However, if the UAC has cached a more direct path to the callee, e.g., from the Location header of a response to a previous request, the To would still contain the long-term, "public" address, while the Request-URI would be set to the cached address.

Proxy and redirect servers may use the information in the Request-URI and request header fields to handle the request and possibly rewrite the Request-URI. For example, a request addressed to the generic address `sip:sales@acme.com` might be proxied to the particular person, e.g., `sip:bob@ny.acme.com`, with the To remaining as `sales@acme.com`. At `ny.acme.com`, Bob may have designated Alice as the temporary substitute.

The host part of the Request-URI typically agrees with one of the host names of the server. If it does not, the server SHOULD proxy the request to the address indicated or return a 404 (Not Found) response if it is unwilling or unable to do so. For example, the Request-URI and server host name may disagree in the case of a firewall proxy that handles outgoing calls. This mode of operation similar to that of HTTP proxies.

If a SIP server receives a request with a URI indicating a scheme other than SIP which that server does not understand, the server MUST return a 400 (Bad Request) response. It MUST do this even if the To field contains a scheme it does understand.

4.3.1 SIP Version

Both request and response messages include the version of SIP in use, and basically follow [H3.1], with HTTP replaced by SIP. To be conditionally compliant with this specification, applications sending SIP messages MUST include a SIP-Version of "SIP/2.0".

4.4 Option Tags

Option tags are unique identifiers used to designate new options in SIP. These tags are used in Require (Section 6.30) and Unsupported (Section 6.38) fields.
Syntax:

option-tag = 1*uric

The creator of a new SIP option should either prefix the option with a reverse domain name or register the new option with the Internet Assigned Numbers Authority (IANA). For example, "com.foo.mynewfeature" is an apt name for a feature whose inventor can be reached at "foo.com". Options registered with IANA have the prefix "org.ietf.sip.", options described in RFCs have the prefix "org.ietf.rfc.N", where N is the RFC number. Option tags are case-insensitive.

4.4.1 Registering New Option Tags with IANA

When registering a new SIP option, the following information should be provided:

- Name and description of option. The name may be of any length, but SHOULD be no more than twenty characters long. The name MUST NOT contain any spaces, control characters or periods.
- Indication of who has change control over the option (for example, IETF, ISO, ITU-T, other international standardization bodies, a consortium or a particular company or group of companies);
- A reference to a further description, if available, for example (in order of preference) an RFC, a published paper, a patent filing, a technical report, documented source code or a computer manual;
- For proprietary options, contact information (postal and email address);

Borrowed from RTSP and the RTP AVP.

5 Response

After receiving and interpreting a request message, the recipient responds with a SIP response message. The response message format is shown below:

```
Response = Status-Line         ; Section 5.1
           *( general-header
           | response-header
           | entity-header )
           CRLF
           [ message-body ]    ; Section 8
```

[H6] applies except that HTTP-Version is replaced by SIP-Version. Also, SIP defines additional response codes and does not use some HTTP codes.

5.1 Status-Line

The first line of a Response message is the Status-Line, consisting of the protocol version (Section 4.3.1) followed by a numeric Status-Code and its associated textual phrase, with each element separated by SP characters. No CR or LF is allowed except in the final CRLF sequence.

Status-Line = SIP-version SP Status-Code SP Reason-Phrase CRLF

5.1.1 Status Codes and Reason Phrases

The Status-Code is a 3-digit integer result code that indicates the outcome of the attempt to understand and satisfy the request. The Reason-Phrase is intended to give a short textual description of the Status-Code. The Status-Code is intended for use by automata, whereas the Reason-Phrase is intended for the human user. The client is not required to examine or display the Reason-Phrase.

Status-Code	=	Informational	Fig. 5
	|	Success	Fig. 5
	|	Redirection	Fig. 6
	|	Client-Error	Fig. 7
	|	Server-Error	Fig. 8
	|	Global-Failure	Fig. 9
	|	extension-code	
extension-code	=	3DIGIT	
Reason-Phrase	=	*<TEXT, excluding CR, LF>	

We provide an overview of the Status-Code below, and provide full definitions in Section 7. The first digit of the Status-Code defines the class of response. The last two digits do not have any categorization role. SIP/2.0 allows 6 values for the first digit:

1xx: Informational – request received, continuing to process the request;

2xx: Success – the action was successfully received, understood, and accepted;

3xx: Redirection – further action must be taken in order to complete the request;

4xx: Client Error – the request contains bad syntax or cannot be fulfilled at this server;

5xx: Server Error – the server failed to fulfill an apparently valid request;

6xx: Global Failure – the request is invalid at any server.

Figures 5 through 9 present the individual values of the numeric response codes, and an example set of corresponding reason phrases for SIP/2.0. These reason phrases are only recommended; they may be replaced by local equivalents without affecting the protocol. Note that SIP adopts many HTTP/1.1 response codes. SIP/2.0 adds response codes in the range starting at x80 to avoid conflicts with newly defined HTTP response codes, and adds a new class, 6xx, of response codes.

SIP response codes are extensible. SIP applications are not required to understand the meaning of all registered response codes, though such understanding is obviously desirable. However, applications MUST

```
Informational  =  "100"  ; Trying
               |  "180"  ; Ringing
               |  "181"  ; Call Is Being Forwarded
               |  "182"  ; Queued
Success        =  "200"  ; OK
```

Figure 5: Informational and success status codes

```
Redirection  =  "300"  ; Multiple Choices
             |  "301"  ; Moved Permanently
             |  "302"  ; Moved Temporarily
             |  "303"  ; See Other
             |  "305"  ; Use Proxy
             |  "380"  ; Alternative Service
```

Figure 6: Redirection status codes

understand the class of any response code, as indicated by the first digit, and treat any unrecognized response as being equivalent to the x00 response code of that class, with the exception that an unrecognized response MUST NOT be cached. For example, if a client receives an unrecognized response code of 431, it can safely assume that there was something wrong with its request and treat the response as if it had received a 400 (Bad Request) response code. In such cases, user agents SHOULD present to the user the message body returned with the response, since that message body is likely to include human-readable information which will explain the unusual status.

6 Header Field Definitions

SIP header fields are similar to HTTP header fields in both syntax and semantics [H4.2, H14]. In general the ordering of the header fields is not of importance (with the exception of Via fields, see below), but proxies MUST NOT reorder or otherwise modify header fields other than by adding a new Via or other hop-by-hop field. Proxies MUST NOT, for example, change how header fields are broken across lines. This allows an authentication field to be added after the Via fields that will not be invalidated by proxies.

The header fields required, optional and not applicable for each method are listed in Table 4. The table uses "o" to indicate optional, "m" mandatory and "-" for not applicable. A "*" indicates that the header fields are needed only if message body is not empty: The Content-Type and Content-Length headers are required when there is a valid message body (of non-zero length) associated with the message (Section 8).

The "type" column describes the request and response types for which the header field may be used. A numeric value indicates the status code for a response, while "R" refers to any request header, "r" to any response header. "g" and "e" designate general (Section 6.1) and entity header (Section 6.2) fields, respectively.

The "enc." column describes whether this message header may be encrypted end-to-end. A "n" designates fields that MUST NOT be encrypted, while "c" designates fields that SHOULD be encrypted if encryption is used.

```
Client-Error  =  "400"  ; Bad Request
              |  "401"  ; Unauthorized
              |  "402"  ; Payment Required
              |  "403"  ; Forbidden
              |  "404"  ; Not Found
              |  "405"  ; Method Not Allowed
              |  "406"  ; Not Acceptable
              |  "407"  ; Proxy Authentication Required
              |  "408"  ; Request Timeout
              |  "409"  ; Conflict
              |  "410"  ; Gone
              |  "411"  ; Length Required
              |  "413"  ; Request Message Body Too Large
              |  "414"  ; Request-URI Too Large
              |  "415"  ; Unsupported Media Type
              |  "420"  ; Bad Extension
              |  "480"  ; Temporarily not available
              |  "481"  ; Invalid Call-ID
              |  "482"  ; Loop Detected
              |  "483"  ; Too Many Hops
              |  "484"  ; Address Incomplete
              |  "485"  ; Ambiguous
```

Figure 7: Client error status codes

```
Server-Error  =  "500"  ; Internal Server Error
              |  "501"  ; Not Implemented
              |  "502"  ; Bad Gateway
              |  "503"  ; Service Unavailable
              |  "504"  ; Gateway Timeout
              |  "505"  ; SIP Version not supported
```

Figure 8: Server error status codes

The “e-e” column has a value of “e” for end-to-end and a value of “h” for hop-by-hop headers.
Other headers may be added as required; a server MAY ignore optional headers that it does not under-

```
Global-Failure  |  "600"  ; Busy
                |  "603"  ; Decline
                |  "604"  ; Does not exist anywhere
                |  "606"  ; Not Acceptable
```

Figure 9: Global failure status codes

	type	enc.	e-e	ACK	BYE	CAN	INV	OPT	REG
Accept	R		e	-	-	-	o	o	o
Accept-Encoding	R		e	-	-	-	o	o	o
Accept-Language	R	n	e	-	o	o	o	o	o
Allow	405		e	o	o	o	o	o	o
Authorization	R		e	o	o	o	o	o	o
Call-ID	g	n	e	m	m	m	m	m	m
Content-Encoding	e		e	*	-	-	*	*	*
Content-Length	e		e	m	-	-	m	m	m
Content-Type	e		e	*	-	-	*	*	*
CSeq	g	n	e	m	m	m	m	m	o
Date	g		e	o	o	o	o	o	o
Encryption	g	n	e	o	o	o	o	o	o
Expires	g		e	-	-	-	o	o	o
From	g		e	m	m	m	m	m	m
Hide	R	n	h	o	o	o	o	o	o
Location	R		e	o	-	-	o	-	m
Location	2xx		e	-	-	-	o	o	-
Location	3xx		e	-	o	-	o	o	o
Location	485		e	-	o	-	o	o	o
Max-Forwards	R	n	e	o	o	o	o	o	o
Organization	R	c	e	-	-	-	o	o	o
Proxy-Authenticate	407	n	h	o	o	o	o	o	o
Proxy-Authorization	R	n	h	o	o	o	o	o	o
Proxy-Require	R	n	h	o	o	o	o	o	o
Priority	R	c	e	-	-	-	o	-	-
Require	R	n	e	o	o	o	o	o	o
Retry-After	R	c	e	-	-	-	-	-	o
Retry-After	600,603	c	e	-	-	-	o	-	-
Response-Key	R	c	e	-	o	o	o	o	o
Record-Route	R		h	o	o	o	o	o	o
Record-Route	2xx		h	o	o	o	o	o	o
Route	R		h	-	o	o	o	o	o
Server	r	c	e	o	o	o	o	o	o
Subject	R	c	e	-	-	-	o	-	-
Timestamp	g		e	o	o	o	o	o	o
To	g	n	e	m	m	m	m	m	m
Unsupported	420		e	o	o	o	o	o	o
User-Agent	R	c	e	o	o	o	o	o	o
Via	g	n	e	m	m	m	m	m	m
Warning	r		e	o	o	o	o	o	o
WWW-Authenticate	401	c	e	o	o	o	o	o	o

Table 4: Summary of header fields

stand. A compact form of these header fields is also defined in Section 9 for use over UDP when the request has to fit into a single packet and size is an issue.

Table 5 in Appendix A indicates which system components should be capable of parsing which header fields.

6.1 General Header Fields

General header fields apply to both request and response messages. The general-header field names can be extended reliably only in combination with a change in the protocol version. However, new or experimental header fields may be given the semantics of general header fields if all parties in the communication recognize them to be general-header fields. Unrecognized header fields are treated as entity-header fields.

6.2 Entity Header Fields

The entity-header fields define meta-information about the message-body or, if no body is present, about the resource identified by the request. The term "entity header" is an HTTP 1.1 term where the response body may contain a transformed version of the message body. The original message body is referred to as the "entity". We retain the same terminology for header fields but usually refer to the "message body" rather then the entity as the two are the same in SIP.

6.3 Request Header Fields

The request-header fields allow the client to pass additional information about the request, and about the client itself, to the server. These fields act as request modifiers, with semantics equivalent to the parameters of a programming language method invocation.

The request-header field names can be extended reliably only in combination with a change in the protocol version. However, new or experimental header fields MAY be given the semantics of request-header fields if all parties in the communication recognize them to be request-header fields. Unrecognized header fields are treated as entity-header fields.

6.4 Response Header Fields

The response-header fields allow the server to pass additional information about the response which cannot be placed in the Status-Line. These header fields give information about the server and about further access to the resource identified by the Request-URI.

Response-header field names can be extended reliably only in combination with a change in the protocol version. However, new or experimental header fields MAY be given the semantics of response-header fields if all parties in the communication recognize them to be response-header fields. Unrecognized header fields are treated as entity-header fields.

6.5 End-to-end and Hop-by-hop Headers

End-to-end headers must be transmitted unmodified across all proxies, while hop-by-hop headers may be modified or added by proxies.

6.6 Header Field Format

Header fields (general-header, request-header, response-header, and entity-header) follow the same generic header format as that given in Section 3.1 of RFC 822 [24]. Each header field consists of a name followed by a colon (":") and the field value. Field names are case-insensitive. The field value may be preceded by any amount of leading white space (LWS), though a single space (SP) is preferred. Header fields can be extended over multiple lines by preceding each extra line with at least one SP or horizontal tab (HT). Applications SHOULD follow HTTP "common form" when generating these constructs, since there might exist some implementations that fail to accept anything beyond the common forms.

message-header	=	field-name ":" [field-value] CRLF
field-name	=	token
field-value	=	*(field-content | LWS)
field-content	=	<the OCTETs making up the field-value and consisting of either *TEXT or combinations of token, tspecials, and quoted-string>

The order in which header fields are received is not significant if the header fields have different field names. Multiple header fields with the same field-name may be present in a message if and only if the entire field-value for that header field is defined as a comma-separated list (i.e., #(values)). It MUST be possible to combine the multiple header fields into one "field-name: field-value" pair, without changing the semantics of the message, by appending each subsequent field-value to the first, each separated by a comma. The order in which header fields with the same field-name are received is therefore significant to the interpretation of the combined field value, and thus a proxy MUST NOT change the order of these field values when a message is forwarded.

Field names are not case-sensitive, although their values may be.

6.7 Accept

See [H14.1] for syntax. This request-header field is used only with the INVITE, OPTIONS and REGISTER request methods to indicate what media types are acceptable in the response.

Example:

```
Accept: application/sdp;level=1, application/x-private, text/html
```

6.8 Accept-Encoding

The Accept-Encoding request-header field is similar to Accept, but restricts the content-codings [H3.4.1] that are acceptable in the response. See [H14.3].

6.9 Accept-Language

See [H14.4] for syntax. The Accept-Language request header can be used to allow the client to indicate to the server in which language it would prefer to receive reason phrases, session descriptions or status responses carried as message bodies. A proxy may use this field to help select the destination for the call, for example, a human operator conversant in a language spoken by the caller.

Example:

```
Accept-Language: da, en-gb;q=0.8, en;q=0.7
```

6.10 Allow

See [H14.7]. The Allow entity-header field lists the set of methods supported by the resource identified by the Request-URI. The purpose of this field is strictly to inform the recipient of valid methods associated with the resource. An Allow header field MUST be present in a 405 (Method Not Allowed) response.

6.11 Authorization

See [H14.8]. A user agent that wishes to authenticate itself with a server – usually, but not necessarily, after receiving a 401 response – MAY do so by including an Authorization request-header field with the request. The Authorization field value consists of credentials containing the authentication information of the user agent for the realm of the resource being requested.

6.12 Call-ID

The Call-ID general header uniquely identifies a particular invitation or all registrations of a particular client. Note that a single multimedia conference may give rise to several calls with different Call-IDs, e.g., if a user invites a single individual several times to the same (long-running) conference.

For an INVITE request, a callee user agent server SHOULD NOT alert the user if the user has responded previously to the Call-ID in the INVITE request. If the user is already a member of the conference and the conference parameters contained in the session description have not changed, a callee user agent server MAY silently accept the call, regardless of the Call-ID. An invitation for an existing Call-ID or session may change the parameters of the conference. A client application MAY decide to simply indicate to the user that the conference parameters have been changed and accept the invitation automatically or it MAY require user confirmation.

A user may be invited to the same conference or call using several different Call-IDs. If desired, the client may use identifiers within the session description to detect this duplication. For example, SDP contains a session id and version number in the origin (o) field.

The REGISTER and OPTIONS methods use the Call-ID value to unambiguously match requests and responses. All REGISTER requests issued by a single client MUST use the same Call-ID.

The Call-ID may be any string consisting of the unreserved URI characters that can be guaranteed to be globally unique for the duration of the request. Call-IDs are case-sensitive and are *not* URL-encoded.

> Since the Call-ID is generated by and for SIP, there is no reason to deal with the complexity of URL-encoding and case-ignoring string comparison.

```
Call-ID   =  ( "Call-ID" | "i" ) ":" local-id "@" host
local-id  =  *uric
```

host MUST be either a fully qualified domain name or a globally routable IP address, while the local-id is a random identifier unique within host. The use of a UUID as local-id is OPTIONAL. The UUID is a version-4 (random) UUID [19].

Using cryptographically random identifiers provides some protection against session hijacking. Call-ID, To and From are needed to identify a *call leg*. The distinction between call and call leg matters in calls with third-party control.

Example:

```
Call-ID: f81d4fae-7dec-11d0-a765-00a0c91e6bf6@foo.bar.com
```

6.13 Content-Encoding

The Content-Encoding entity-header field is used as a modifier to the media-type. When present, its value indicates what additional content codings have been applied to the entity-body, and thus what decoding mechanisms MUST be applied in order to obtain the media-type referenced by the Content-Type header field. Content-Encoding is primarily used to allow a document to be compressed without losing the identity of its underlying media type. See [H14.12].

6.14 Content-Length

The Content-Length entity-header field indicates the size of the message-body, in decimal number of octets, sent to the recipient.

Content-Length = "Content-Length" ":" 1*DIGIT

An example is

```
Content-Length: 3495
```

Applications MUST use this field to indicate the size of the message-body to be transferred, regardless of the media type of the entity. Any Content-Length greater than or equal to zero is a valid value. If no body is present in a message, then the Content-Length header MUST be set to zero. If a server receives a message without Content-Length, it MUST assume it to be zero. Section 8 describes how to determine the length of the message body.

6.15 Content-Type

The Content-Type entity-header field indicates the media type of the message-body sent to the recipient. The media-type element is defined in [H3.7].

Content-Type = "Content-Type" ":" media-type

Examples of this header field are

```
Content-Type: application/sdp
Content-Type: text/html; charset=ISO-8859-4
```

6.16 CSeq

Clients MUST add the CSeq (command sequence) general-header field to every request. A CSeq request header field contains a single decimal sequence number chosen by the requesting client, unique within a single value of Call-ID. The sequence number MUST be expressible as a 32-bit unsigned integer. The initial value of the sequence number is arbitrary, but MUST be less than 2**31. Consecutive requests that differ in request method, headers or body, but have the same Call-ID MUST contain strictly monotonically increasing and contiguous sequence numbers; sequence numbers do not wrap around. Retransmissions of the same request carry the same sequence number, but an INVITE with a different message body or different header fields (a "re-invitation") acquires a new, higher sequence number. A server MUST echo the CSeq value from the request in its response. If the Method value is missing, the server fills it in appropriately.

The ACK and CANCEL requests MUST contain the same CSeq value as the INVITE request that it refers to, while a BYE request cancelling an invitation MUST have a higher sequence number.

A user agent server MUST remember the highest sequence number for any INVITE request with the same Call-ID value. The server MUST respond to, but ignore any INVITE request with a lower sequence number.

All requests spawned in a parallel search have the same CSeq value as the request triggering the parallel search.

CSeq = "CSeq" ":" 1*DIGIT Method

Strictly speaking, CSeq header fields are needed for any SIP request that can be cancelled by a BYE or CANCEL request or where a client can issue several requests for the same Call-ID in close succession. Without a sequence number, the response to an INVITE could be mistaken for the response to the cancellation (BYE or CANCEL). Also, if the network duplicates packets or if an ACK is delayed until the server has sent an additional response, the client could interpret an old response as the response to a re-invitation issued shortly thereafter. Using CSeq also makes it easy for the server to distinguish different versions of an invitation, without comparing the message body.

The Method value allows the client to distinguish the response to an INVITE request from that of a CANCEL response. CANCEL requests can be generated by proxies; if they were to increase the sequence number, it might conflict with a later request issued by the user agent for the same call.

With a length of 32 bits, a server could generate, within a single call, one request a second for about 136 years before needing to wrap around. The initial value of the sequence number is chosen so that subsequent requests within the same call will not wrap around. A non-zero initial value allows to use a time-based initial sequence number, which protects against ambiguities when clients are re-invited to the same call after rebooting. A client could, for example, choose the 31 most significant bits of a 32-bit second clock as an initial sequence number.

Forked requests must have the same CSeq as there would be ambiguity otherwise between these forked requests and later BYE issued by the client user agent.

Example:

```
CSeq: 4711 INVITE
```

6.17 Date

General header field. See [H14.19].

The Date header field can be used by simple end systems without a battery-backed clock to acquire a notion of current time.

6.18 Encryption

The Encryption general-header field specifies that the content has been encrypted. Section 12 describes the overall SIP security architecture and algorithms. This header field is intended for end-to-end encryption of requests and responses. Requests are encrypted with a public key belonging to the entity named in the To header field. Responses are encrypted with the public key conveyed in the Response-Key header field.

SIP chose not to adopt HTTP's Content-Transfer-Encoding header because the encrypted body may contain additional SIP header fields as well as the body of the message.

For any encrypted message, at least the message body and possibly other message header fields are encrypted. An application receiving a request or response containing an Encryption header field decrypts the body and then concatenates the plaintext to the request line and headers of the original message. Message headers in the decrypted part completely replace those with the same field name in the plaintext part. (Note: If only the body of the message is to be encrypted, the body has to be prefixed with CRLF to allow proper concatenation.) Note that the request method and Request-URI cannot be encrypted.

Encryption only provides privacy; the recipient has no guarantee that the request or response came from the party listed in the From message header, only that the sender used the recipients public key. However, proxies will not be able to modify the request or response.

```
Encryption          =  "Encryption" ":" encryption-scheme 1*SP
                       #encryption-params
encryption-scheme   =  token
encryption-params   =  token "=" ( token | quoted-string )
```

The token indicates the form of encryption used; it is described in section 12.

The following example for a message encrypted with ASCII-armored PGP was generated by applying "pgp -ea" to the payload to be encrypted.

```
INVITE sip:watson@boston.bell-telephone.com SIP/2.0
Via: SIP/2.0/UDP 169.130.12.5
From: <sip:a.g.bell@bell-telephone.com>
To: T. A. Watson <sip:watson@bell-telephone.com>
Call-ID: 187602141351@worcester.bell-telephone.com
Content-Length: 885
Encryption: PGP version=2.6.2,encoding=ascii

hQEMAxkp5GPd+j5xAQf/ZDIfGD/PDOM1wayvwdQAKgGgjmZWe+MTy9NEX8O25Red
h0/pyrd/+DV5C2BYs7yzSOSXaj1C/tTK/4do6rtjhP8QA3vbDdVdaFciwEVAcuXs
ODxlNAVqyDi1RqFC28BJIvQ5KfEkPuACKTK7WlRSBc7vNPEA3nyqZGBTwhxRSbIR
RuFEsHSVojdCam4htcqxGnFwD9sksqs6LIyCFaiTAhWtwcCaN437G7mUYzy2KLcA
zPVGq1VQg83b99zPzIxRdlZ+K7+bAnu8Rtu+ohOCMLV3TPXbyp+err1YiThCZHIu
X9dOVj3CMjCP66RSHa/ea0wYTRRNYA/G+kdP8DSUcqYAAAE/hZPX6nFIqk7AVnf6
IpWHUPTelNUJpzUp5Ou+q/5P7ZAsn+cSAuF2YWtVjCf+SQmBR13p2EYYWHoxlA2/
GgKADYe4M3JSwOtqwU8zUJF3FIfk7vsxmSqtUQrRQaiIhqNyG7KxJt4YjWnEjF5E
WUIPhvyGFMJaeQXIyGRYZAYvKKklyAJcm29zLACxU5alX4M25lHQd9FR9Zmq6Jed
wbWvia6cAIfsvlZ9JGocmQYF7pcuz5pnczqP+/yvRqFJtDGD/v3s++G2R+ViVYJO
z/lxGUZaM4IWBCf+4DUjNanZM0oxAE28NjaIZ0rrldDQmO8V9FtPKdHxkqA5iJP+
```

```
6vGOFti1Ak4kmEz0vM/Nsv7kkubTFhRl05OiJIGr9S1UhenlZv9l6RuXsOY/EwH2
z8X9N4MhMyXEVuC9rt8/AUhmVQ==
=bOW+
```


Since proxies may base their forwarding decision on any combination of SIP header fields, there is no guarantee that an encrypted request "hiding" header fields will reach the same destination as an otherwise identical un-encrypted request.

6.19 Expires

The Expires entity-header field gives the date and time after which the message content expires.

This header field is currently defined only for the REGISTER and INVITE methods. For REGISTER, it is a request and response-header field and allows the client to indicate how long the registration is to be valid; the server uses it to indicate when the client has to re-register the addresses contained in the request. The server's choice overrides that of the client. The server MAY choose a shorter time interval than that requested by the client, but SHOULD NOT choose a longer one.

For INVITE, it is a request and response-header field. In a request, the callee can limit the validity of an invitation. For example, if a client wants to limit how long a search should take at most or when a conference invitation is time-limited. A user interface may take this as a hint to leave the invitation window on the screen even if the user is not currently at the workstation. This also limits the duration of a search. If the request expires before the search completes, the proxy returns a 408 (Request Timeout) status. In a 302 (Moved Temporarily) response, a server can advise the client of the maximal duration of the redirection.

The value of this field can be either an HTTP-date or an integer number of seconds (in decimal), measured from the receipt of the request. The latter approach is preferable for short durations, as it does not depend on clients and servers sharing a synchronized clock.

```
Expires = "Expires" ":" ( HTTP-date | delta-seconds )
```

Two examples of its use are

```
Expires: Thu, 01 Dec 1994 16:00:00 GMT
Expires: 5
```

6.20 From

Requests and responses MUST contain a From general-header field, indicating the initiator of the request. The server copies the To and From header fields from the request to the response. The optional display-name is meant to be rendered by a human-user interface.

The SIP-URL MUST NOT contain the transport-param, maddr-param, ttl-param, or headers elements. A server that receives a SIP-URL with these elements removes them before further processing.

```
From         = ( "From" | "f" ) ":" ( name-addr | addr-spec )
name-addr    = [ display-name ] "<" addr-spec ">"
addr-spec    = SIP-URL | URI
display-name = *token | quoted-string
```

Examples:

```
From: A. G. Bell <sip:agb@bell-telephone.com>
From: sip:+12125551212@server.phone2net.com
From: Anonymous <sip:c8oqz84zk7z@privacy.org>
```

Call-ID, To and From are needed to identify a *call leg*. The distinction between call and call leg matters in calls with third-party control. The format is similar to the equivalent RFC 822 [24] header, but with a URI instead of just an email address.

6.21 Hide

The Hide request header field indicates that the path comprised of the Via header fields (Section 6.40) should be hidden from subsequent proxies and user agents. It can take two forms: Hide: route and Hide: hop. Hide header fields are typically added by the client user agent, but MAY be added by any proxy along the path.

If a request contains the "Hide: route" header field, all following proxies SHOULD hide their previous hop. If a request contains the "Hide: hop" header field, only the next proxy SHOULD hide the previous hop and then remove the Hide option unless it also wants to remain anonymous.

A server hides the previous hop by encrypting the host and port parts of the top-most Via header with an algorithm of its choice. Servers SHOULD add additional "salt" to the host and port information prior to encryption to prevent malicious downstream proxies from guessing earlier parts of the path based on seeing identical encrypted Via headers. Hidden Via fields are marked with the hidden Via option, as described in Section 6.40.

A server that is capable of hiding Via headers MUST attempt to decrypt all Via headers marked as "hidden" to perform loop detection. Servers that are not capable of hiding can ignore hidden Via fields in their loop detection algorithm.

If hidden headers were not marked, a proxy would have to decrypt all headers to detect loops, just in case one was encrypted, as the Hide: Hop option may have been removed along the way.

A host MUST NOT add such a "Hide: hop" header field unless it can guarantee it will only send a request for this destination to the same next hop. The reason for this is that it is possible that the request will loop back through this same hop from a downstream proxy. The loop will be detected by the next hop if the choice of next hop is fixed, but could loop an arbitrary number of times otherwise.

A client requesting "Hide: route" can only rely on keeping the request path private if it sends the request to a trusted proxy. Hiding the route of a SIP request may be of limited value if the request results in data packets being exchanged directly between the calling and called user agent.

The use of Hide header fields is discouraged unless path privacy is truly needed; Hide fields impose extra processing costs and restrictions for proxies and can cause requests to generate 482 (Loop Detected) responses that could otherwise be avoided.

The encryption of Via header fields is described in more detail in Section 12.

The Hide header field has the following syntax:

Hide = "Hide" ":" ("route" | "hop")

6.22 Location

The Location general-header field can appear in requests, 2xx responses and 3xx responses.

REGISTER requests: REGISTER requests MUST contain a Location header field indicating at which locations the user may be reachable. The REGISTER request defines a wildcard Location field, "*", which is only used with Expires: 0 to remove all registrations for a particular user.

INVITE and ACK requests: INVITE and ACK requests SHOULD contain Location headers indicating from which location the request is originating. If the SIP address does not refer to the user agent server, the SIP URL MUST contain a tag parameter uniquely identifying the user agent. (The same person may be logged on at several locations within the same domain served by the proxy.)

This allows the callee to send a BYE directly to the caller instead of through a series of proxies. The Via header is not sufficient since the desired address may be that of a proxy.

INVITE 2xx responses: A user agent server sending a definitive, positive response (2xx) MAY insert a Location response header indicating the SIP address under which it is reachable most directly for future SIP requests, such as ACK. This may be the address of the server itself or that of a proxy, e.g., if the host is behind a firewall. If the SIP address does not refer to the user agent server, the SIP URL MUST contain a tag parameter uniquely identifying the user agent. (The same person may be logged on at several locations within the same domain served by the proxy.) The value of this Location header is copied into the Request-URI of subsequent requests for this call.

REGISTER 2xx responses: Similarly, a REGISTER response SHOULD return all locations at which the user is currently reachable.

3xx responses: The Location response-header field can be used with a 3xx response codes to indicate one or more addresses to try. It can appear in responses to BYE, INVITE and OPTIONS methods. The Location header field contains URIs giving the new locations or user names to try, or may simply specify additional transport parameters. A 301 (Moved Permanently) or 302 (Moved Temporarily) response SHOULD contain a Location field containing URIs of new addressed to be tried. A 301 or 302 response may also give the same location and username that was being tried but specify additional transport parameters such as a multicast address to try or a change of SIP transport from UDP to TCP or vice versa.

Note that the Location header may also refer to a different entity than the one originally called. For example, a SIP call connected to GSTN gateway may need to deliver a special information announcement such as "The number you have dialed has been changed."

A Location response header may contain any suitable URI indicating where the called party may be reached, not limited to SIP URLs. For example, it may contain a phone or fax a mailto: (RFC 2368, [25]) or irc: URL.

The following parameters are defined. Additional parameters may be defined in other specifications.

q: The qvalue indicates the relative preference among the locations given. qvalue values are decimal numbers from 0.0 to 1.0, with higher values indicating higher preference.

action: The action is only used when registering with the REGISTER request. It indicates whether the client wishes that the server proxies or redirects future requests intended for the client. The action taken if this parameter is not specified depends on server configuration. In its response, the registrar SHOULD indicate the mode used. This parameter is ignored for other requests.

```
Location = ( "Location" | "m" ) ":"
           ("*" | (1# (( SIP-URL | URI )
           [ LWS *( ";" location-params ) ] ))

location-params      =   "q"                "="    qvalue
                     |   "action"           "="    "proxy" | "redirect"
                     |   extension-attribute
extension-attribute  =   extension-name   [ "="    extension-value ]
```

Example:

```
Location: sip:watson@worcester.bell-telephone.com;tag=123
   ;q=0.7,
   mailto:watson@bell-telephone.com ;q=0.1
```

6.23 Max-Forwards

The Max-Forwards request-header field may be used with any SIP method to limit the number of proxies or gateways that can forward the request to the next inbound server. This can also be useful when the client is attempting to trace a request chain which appears to be failing or looping in mid-chain. [H14.31]

```
Max-Forwards = "Max-Forwards" ":" 1*DIGIT
```

The Max-Forwards value is a decimal integer indicating the remaining number of times this request message may be forwarded.

Each proxy or gateway recipient of a request containing a Max-Forwards header field MUST check and update its value prior to forwarding the request. If the received value is zero (0), the recipient MUST NOT forward the request. Instead, for the OPTIONS and REGISTER methods, it MUST respond as the final recipient. For all other methods, the server returns 483 (Too many hops).

If the received Max-Forwards value is greater than zero, then the forwarded message MUST contain an updated Max-Forwards field with a value decremented by one (1).

Example:

```
Max-Forwards: 6
```

6.24 Organization

The Organization request-header field conveys the name of the organization to which the callee belongs. It may also be inserted by proxies at the boundary of an organization and may be used by client software to filter calls.

```
Organization = "Organization" ":" *text
```

6.25 Priority

The Priority request header signals the urgency of the call to the callee.

```
Priority        =  "Priority" ":" priority-value
priority-value  =  "emergency" | "urgent" | "normal"
                |  "non-urgent"
```

The value of "emergency" should only be used when life, limb or property are in imminent danger. Examples:

```
Subject: A tornado is heading our way!
Priority: emergency

Subject: Weekend plans
Priority: non-urgent
```

These are the values of RFC 2076 [26], with the addition of "emergency".

6.26 Proxy-Authenticate

The Proxy-Authenticate response-header field MUST be included as part of a 407 (Proxy Authentication Required) response. The field value consists of a challenge that indicates the authentication scheme and parameters applicable to the proxy for this Request-URI. See [H14.33] for further details.

A client SHOULD cache the credentials used for a particular proxy server and realm for the next request to that server. Credentials are, in general, valid for a specific value of the Request-URI at a particular proxy server. If a client contacts a proxy server that has required authentication in the past, but the client does not have credentials for the particular Request-URI, it MAY attempt to use the most-recently used credential. The server responds with 401 (Unauthorized) if the client guessed wrong.

This suggested caching behavior is motivated by proxies restricting phone calls to authenticated users. It seems likely that in most cases, all destinations require the same password. Note that end-to-end authentication is likely to be destination-specific.

6.27 Proxy-Authorization

The Proxy-Authorization request-header field allows the client to identify itself (or its user) to a proxy which requires authentication. The Proxy-Authorization field value consists of credentials containing the authentication information of the user agent for the proxy and/or realm of the resource being requested. See [H14.34] for further details.

6.28 Proxy-Require

The Proxy-Require header is used to indicate proxy-sensitive features that MUST be supported by the proxy. Any Proxy-Require header features that are not supported by the proxy MUST be negatively acknowledged by the proxy to the client if not supported. Servers treat this field identically to the Require field.

See Section 6.30 for more details on the mechanics of this message and a usage example.

6.29 Record-Route

The Record-Route request and response header field is added to an INVITE request by any proxy that insists on being in the path of subsequent ACK and BYE requests for the same call. It contains a globally reachable Request-URI that identifies the proxy server. Each proxy server adds its Request-URI to the beginning of the list.

The server copies the Record-Route header unchanged into the response. (Record-Route is only relevant for 2xx responses.)

The calling user agent client copies the Record-Route header into a Route header of subsequent requests, *reversing* the order of requests, so that the first entry is closest to the caller. If the response contained a Location header field, the calling user agent adds its content as the last Route header. Unless this would cause a loop, any client MUST send any subsequent requests for this Call-ID to the first Request-URI in the Route request header and remove that entry.

Some proxies, such as those controlling firewalls or in an automatic call distribution (ACD) system, need to maintain call state and thus need to receive any BYE and ACK packets for the call.

The Record-Route header field has the following syntax:

Record-Route = "Record-Route" ":" 1# request-uri

Example for a request that has traversed the hosts `ieee.org` and `bell-telephone.com`, in that order:

```
Record-Route: sip:a.g.bell@bell-telephone.com, sip:a.bell@ieee.org
```

6.30 Require

The Require request header is used by clients to tell user agent servers about options that the client expects the server to support in order to properly process the request. If a server does not understand the option, it MUST respond by returning status code 420 (Bad Extension) and list those options it does not understand in the Unsupported header.

Require = "Require" ":" 1#option-tag

Example:

```
C->S:   INVITE sip:watson@bell-telephone.com SIP/2.0
        Require: com.example.billing
        Payment: sheep_skins, conch_shells

S->C:   SIP/2.0 420 Bad Extension
        Unsupported: com.example.billing
```

This is to make sure that the client-server interaction will proceed without delay when all options are understood by both sides, and only slow down if options are not understood (as in the example above). For a well-matched client-server pair, the interaction proceeds quickly, saving a round-trip often required by negotiation mechanisms. In addition, it also removes ambiguity when the client requires features that the server does not understand. Some features, such as call handling fields, are only of interest to end systems.

Proxy and redirect servers MUST ignore features that are not understood. If a particular extension requires that intermediate devices support it, the extension should be tagged in the Proxy-Require field instead (see Section 6.28).

6.31 Response-Key

The Response-Key request header field can be used by a client to request the key that the called user agent SHOULD use to encrypt the response with. The syntax is:

```
Response-Key  =  "Response-Key" ":" key-scheme 1*SP #key-param
key-scheme    =  token
key-param     =  token "=" ( token | quoted-string )
```

The key-scheme gives the type of encryption to be used for the response. Section 12 describes security schemes.

If the client insists that the server return an encrypted response, it includes a

Require: org.ietf.sip.encrypt-response

header field in its request. If the client cannot encrypt for whatever reason, it MUST follow normal Require header field procedures and return a 420 (Bad Extension) response. If this Require header is not present, a client SHOULD still encrypt, but MAY return an unencrypted response if unable to.

6.32 Retry-After

The Retry-After response header field can be used with a 503 (Service Unavailable) response to indicate how long the service is expected to be unavailable to the requesting client and with a 404 (Not Found), 600 (Busy), or 603 (Decline) response to indicate when the called party may be available again. The value of this field can be either an HTTP-date or an integer number of seconds (in decimal) after the time of the response.

A REGISTER request may include this header field when deleting registrations with Location: *; Expires: 0. The Retry-After value then indicates when the user might again be reachable. The registrar MAY then include this information in responses to future calls.

An optional comment can be used to indicate additional information about the time of callback. An optional duration parameter indicates how long the called party will be reachable starting at the initial time of availability. If no duration parameter is given, the service is assumed to be available indefinitely.

```
Retry-After  =  "Retry-After" ":" ( HTTP-date | delta-seconds )
                [ comment ] [ ";duration" "=" delta-seconds ]
```

Examples of its use are

```
Retry-After: Mon, 21 Jul 1997 18:48:34 GMT (I'm in a meeting)
Retry-After: Mon,  1 Jan 9999 00:00:00 GMT
  (Dear John: Don't call me back, ever)
Retry-After: Fri, 26 Sep 1997 21:00:00 GMT;duration=3600
Retry-After: 120
```

In the third example, the callee is reachable for one hour starting at 21:00 GMT. In the last example, the delay is 2 minutes.

6.33 Route

The Route request header determines the route taken by a request. Each host removes the first entry and then proxies the request to the host listed in that entry, also using it as the Request-URI. The operation is further described in Section 6.29.

The Route header field has the following syntax:

Route = "Route" ":" 1# request-uri

6.34 Server

The Server response-header field contains information about the software used by the user agent server to handle the request. See [H14.39].

6.35 Subject

This is intended to provide a summary, or to indicate the nature, of the call, allowing call filtering without having to parse the session description. (Also, the session description may not necessarily use the same subject indication as the invitation.)

Subject = ("Subject" | "s") ":" *text

Example:

```
Subject: Tune in - they are talking about your work!
```

6.36 Timestamp

The timestamp general header describes when the client sent the request to the server. The value of the timestamp is of significance only to the client and may use any timescale. The server MUST echo the exact same value and MAY, if it has accurate information about this, add a floating point number indicating the number of seconds that have elapsed since it has received the request. The timestamp is used by the client to compute the round-trip time to the server so that it can adjust the timeout value for retransmissions.

Timestamp = "Timestamp" ":" *(DIGIT) ["." *(DIGIT)] [delay]
delay = *(DIGIT) ["." *(DIGIT)]

6.37 To

The To general-header field specifies recipient of the request, with the same SIP URL syntax as the From field.

To = ("To" | "t") ":" (name-addr | addr-spec)

The UAS copies the To header into its response, but SHOULD add a tag parameter if not already present. It MAY forego adding the tag parameter if there is no chance that another UAS responds to the same request.

A SIP server returns a 400 (Bad Request) response if it receives a request with a To header field containing a URI with a scheme it does not recognize.

Example:

```
To: The Operator <sip:operator@cs.columbia.edu>
To: sip:+12125551212@server.phone2net.com
```

Call-ID, To and From are needed to identify a *call leg*. The distinction between call and call leg matters in calls with third-party control. The tag is added to the To header in the response to allow forking of future requests for the same call by proxies, while addressing only one of the possibly several responding user agent servers. It also allows several instances of the callee to send requests that can be distinguished.

6.38 Unsupported

The Unsupported response header lists the features not supported by the server. See Section 6.30 for a usage example and motivation.

6.39 User-Agent

The User-Agent request-header field contains information about the client user agent originating the request. See [H14.42].

6.40 Via

The Via field indicates the path taken by the request so far. This prevents request looping and ensures replies take the same path as the requests, which assists in firewall traversal and other unusual routing situations.

6.40.1 Requests

The client originating the request MUST insert into the request a Via field containing its host name or network address and, if not the default port number, the port number at which it wishes to receive responses. (Note that this port number may differ from the UDP source port number of the request.) A fully-qualified domain name is RECOMMENDED. Each subsequent proxy server that sends the request onwards MUST add its own additional Via field before any existing Via fields. A proxy that receives a redirection (3xx) response and then searches recursively, MUST use the same Via headers as on the original request.

A proxy SHOULD check the top-most Via header to ensure that it contains the sender's correct network address, as seen from that proxy. If the sender's address is incorrect, the proxy should add an additional received attribute, as described 6.40.2.

A host behind a network address translator (NAT) or firewall may not be able to insert a network address into the Via header that can be reached by the next hop beyond the NAT. Hosts behind NATs or NAPTs should insert the local port number of the outgoing socket, rather than the port number for incoming requests, as NAPTs assume that responses return with reversed source and destination ports.

Additionally, if the message goes to a multicast address, an extra Via field is added by the sender before all the other Via fields giving the multicast address and TTL.

If a proxy server receives a request which contains its own address, it MUST respond with a 482 (Loop Detected) status code.

This prevents a malfunctioning proxy server from causing loops. Also, it cannot be guaranteed that a proxy server can always detect that the address returned by a location service refers to a host listed in the Via list, as a single host may have aliases or several network interfaces.

6.40.2 Receiver-tagged Via Fields

Normally, every host that sends or forwards a SIP message adds a Via field indicating the path traversed. However, it is possible that Network Address Translators (NAT) may change the source address of the request (e.g., from net-10 to a globally routable address), in which case the Via field cannot be relied on to route replies. To prevent this, a proxy SHOULD check the top-most Via header to ensure that it contains the sender's correct network address, as seen from that proxy. If the sender's address is incorrect, the proxy should add a received tag to the Via field inserted by the previous hop. Such a modified Via field is known as a receiver-tagged Via field. An example is:

```
Via: SIP/2.0/UDP erlang.bell-telephone.com:5060
Via: SIP/2.0/UDP 10.0.0.1:5060 ;received=199.172.136.3
```

In this example, the message originated from 10.0.0.1 and traversed a NAT with the external address `border.ieee.org` (199.172.136.3) to reach `erlang.bell-telephone.com`. The latter noticed the mismatch, and tagged the previous hop's Via field with the address that it actually came from.

6.40.3 Responses

In the return path, Via fields are processed by a proxy or client according to the following rules:

1. The first Via field should indicate the proxy or client processing this message. If it does not, discard the message. Otherwise, remove this Via field.
2. If the second Via field is a receiver-tagged field (Section 6.40.2), send the message to the address in the received tag. Otherwise, if the Via header contains a maddr multicast address, send the response to that multicast address, using the value of the ttl parameter if given. Otherwise, send the message to the address indicated in the sent-by parameter.
3. If there is no second Via field, this response is destined for this client.

These rules ensure that a client only has to check the first Via field in a response to see if it needs processing.

A user agent server or redirect server returns the response to the network address where the request came from. (Since these servers do not forward the request, they do not add a Via header field or received tag.)

6.40.4 Syntax

The format for a Via header is shown in Fig. 10.

The defaults for "protocol-name" and "transport" are "SIP" and "UDP", respectively. The "maddr" parameter, designating the multicast address, and the "ttl" parameter, designating the time-to-live (TTL) value, are included only if the request was sent via multicast. The "received" parameter is added only for receiver-added Via fields (Section 6.40.2). For reasons of privacy, a client or proxy may wish to hide its Via information by encrypting it (see Section 6.21). The "hidden" parameter is included if this header was hidden by the upstream proxy (see 6.21).

The "branch" parameter is included by every forking proxy. The token uniquely identifies a branch of a particular search. The identifier has to be unique only within a set of isomorphic requests.

```
Via              =  ( "Via" | "v" ) ":" 1#( sent-protocol sent-by
                    *( ";" via-params ) [ comment ] )
via-params       =  via-hidden | via-ttl | via-maddr
                 |  via-received | via-branch
via-hidden       =  "hidden"
via-ttl          =  "ttl" "=" ttl
via-maddr        =  "maddr" "=" maddr
via-received     =  "received" "=" host
via-branch       =  "branch" "=" token
sent-protocol    =  [ protocol-name "/" ] protocol-version
                    [ "/" transport ]
protocol-name    =  "SIP" | token
protocol-version =  token
transport        =  "UDP" | "TCP" | token
sent-by          =  ( host [ ":" port ] ) | ( concealed-host )
concealed-host   =  token
ttl              =  1*3DIGIT                              ; 0 to 255
```

Figure 10: Syntax of Via header field

Note that privacy of the proxy relies on the cooperation of the next hop, as the next-hop proxy will, by necessity, know the IP address and port number of the source host.

```
Via: SIP/2.0/UDP first.example.com:4000;ttl=16
  ;maddr=224.2.0.1 (Example)
Via: SIP/2.0/UDP adk8%20.8x%fe%03 ;hidden
```

6.41 Warning

The Warning response-header field is used to carry additional information about the status of a response. Warning headers are sent with responses and have the following format:

```
Warning       =  "Warning" ":" 1#warning-value
warning-value =  warn-code SP warn-agent SP warn-text
warn-code     =  3DIGIT
warn-agent    =  ( host [ ":" port ] ) | pseudonym
                 ; the name or pseudonym of the server adding
                 ; the Warning header, for use in debugging
warn-text     =  quoted-string
```

A response may carry more than one Warning header.

The warn-text should be in a natural language that is most likely to be intelligible to the human user receiving the response. This decision may be based on any available knowledge, such as the location of the cache or user, the Accept-Language field in a request, the Content-Language field in a response, etc. The default language is English.

Any server may add Warning headers to a response. Proxy servers MUST place additional Warning headers before any Authorization headers. Within that constraint, Warning headers MUST be added after any existing Warning headers not covered by a signature. A proxy server MUST NOT delete any Warning header that it received with a response.

When multiple Warning headers are attached to a response, the user agent SHOULD display as many of them as possible, in the order that they appear in the response. If it is not possible to display all of the warnings, the user agent first displays warnings that appear early in the response. Systems that generate multiple Warning headers should order them with this user agent behavior in mind.

The warn-code consists of three digits. The first digit indicates the significance of the warning, with 3xx indicating a warning that did not cause the request to fail and 4xx indicating a fatal error condition that contributed to the failure of the request.

This is a list of the currently-defined warn-codes, each with a recommended warn-text in English, and a description of its meaning. Additional warn-codes may be defined through IANA. Note that these warnings describe failures induced by the session description.

x01 Insufficient bandwidth: The bandwidth specified in the session description or defined by the media exceeds that known to be available.

x02 Incompatible transport protocol: One or more transport protocols described in the request are not available.

x03 Incompatible network protocol: One or more network protocols described in the request are not available.

x04 Incompatible network address formats: One or more network address formats described in the request are not available.

x05 Incompatible media format: One or more media formats described in the request are not available.

x06 Incompatible bandwidth description: One or more bandwidth descriptions contained in the request were not understood.

x07 Multicast not available: The site where the user is located does not support multicast.

x08 Unicast not available: The site where the user is located does not support unicast communication (usually due to the presence of a firewall).

x09 Media type not available: One or more media types contained in the request are not available.

x10 Attribute not understood: One or more of the media attributes in the request are not supported.

x09 Session description parameter not understood: A parameter other than those listed above was not understood.

x99 Miscellaneous warning: The warning text may include arbitrary information to be presented to a human user, or logged. A system receiving this warning MUST NOT take any automated action.

1xx and 2xx have been taken by HTTP/1.1.

Examples:

```
Warning: 309 isi.edu "Session parameter 'foo' not understood"
Warning: 404 isi.edu "Incompatible network address type 'E.164'"
```

6.42 WWW-Authenticate

The WWW-Authenticate response-header field MUST be included in 401 (Unauthorized) response messages. The field value consists of at least one challenge that indicates the authentication scheme(s) and parameters applicable to the Request-URI. See [H14.46] and [27].

The content of the realm parameter SHOULD be displayed to the user. A user agent SHOULD cache the authorization credentials for a given value of the destination (To header) and realm and attempt to re-use these values on the next request for that destination.

In addition to the “basic” and “digest” authentication schemes defined in the specifications cited above, SIP defines a new scheme, PGP (RFC 2015, [28]), Section 13. Other schemes, such as S-MIME, are for further study.

7 Status Code Definitions

The response codes are consistent with, and extend, HTTP/1.1 response codes. Not all HTTP/1.1 response codes are appropriate, and only those that are appropriate are given here. Other HTTP/1.1 response codes should not be used. Response codes not defined by HTTP/1.1 have codes x80 upwards to avoid clashes with future HTTP response codes. Also, SIP defines a new class, 6xx. The default behavior for unknown response codes is given for each category of codes.

7.1 Informational 1xx

Informational responses indicate that the server or proxy contacted is performing some further action and does not yet have a definitive response. The client SHOULD wait for a further response from the server, and the server SHOULD send such a response without further prompting. Typically a server should send a 1xx response if it expects to take more than 200 ms to obtain a final response. A server can issue zero or more 1xx responses, with no restriction on their ordering or uniqueness. Note that 1xx responses are not transmitted reliably, that is, they do not cause the client to send an ACK. Servers are free to retransmit informational responses and clients can inquire about the current state of call processing by re-sending the request.

7.1.1 100 Trying

Some unspecified action is being taken on behalf of this call (e.g., a database is being consulted), but the user has not yet been located.

7.1.2 180 Ringing

The called user agent has located a possible location where the user has registered recently and is trying to alert the user.

7.1.3 181 Call Is Being Forwarded

A proxy server MAY use this status code to indicate that the call is being forwarded to a different set of destinations. The new destinations are listed in Location headers. Proxies SHOULD be configurable not to reveal this information.

7.1.4 182 Queued

The called party is temporarily unavailable, but the callee has decided to queue the call rather than reject it. When the callee becomes available, it will return the appropriate final status response. The reason phrase MAY give further details about the status of the call, e.g., "5 calls queued; expected waiting time is 15 minutes". The server MAY issue several 182 responses to update the caller about the status of the queued call.

7.2 Successful 2xx

The request was successful and MUST terminate a search.

7.2.1 200 OK

The request has succeeded. The information returned with the response depends on the method used in the request, for example:

BYE: The call has been terminated. The message body is empty.

CANCEL: The search has been cancelled. The message body is empty.

INVITE: The callee has agreed to participate; the message body indicates the callee's capabilities.

OPTIONS: The callee has agreed to share its capabilities, included in the message body.

REGISTER: The registration has succeeded. The client treats the message body according to its Content-Type.

7.3 Redirection 3xx

3xx responses give information about the user's new location, or about alternative services that may be able to satisfy the call. They SHOULD terminate an existing search, and MAY cause the initiator to begin a new search if appropriate.

Any redirection (3xx) response MUST NOT suggest any of the addresses in the Via (Section 6.40) path of the request in the Location header field. (Addresses match if their host and port number match.)

To avoid forwarding loops, a user agent client or proxy MUST check whether the address returned by a redirect server equals an address tried earlier.

7.3.1 300 Multiple Choices

The address in the request resolved to several choices, each with its own specific location, and the user (or user agent) can select a preferred communication end point and redirect its request to that location.

The response SHOULD include an entity containing a list of resource characteristics and location(s) from which the user or user agent can choose the one most appropriate, if allowed by the Accept request header. The entity format is specified by the media type given in the Content-Type header field. The choices SHOULD also be listed as Location fields (Section 6.22). Unlike HTTP, the SIP response may contain several Location fields or a list of addresses in a Location field. User agents MAY use the Location field value for automatic redirection or MAY ask the user to confirm a choice. However, this specification does not define any standard for such automatic selection.

This header is appropriate if the callee can be reached at several different locations and the server cannot or prefers not to proxy the request.

7.3.2 301 Moved Permanently

The user can no longer be found at the address in the Request-URI and the requesting client should retry at the new address given by the Location header field (Section 6.22). The caller SHOULD update any local directories, address books and user location caches with this new value and redirect future requests to the address(es) listed.

7.3.3 302 Moved Temporarily

The requesting client should retry the request at the new address(es) given by the Location header field (Section 6.22). The duration of the redirection can be indicated through an Expires (Section 6.19) header.

7.3.4 380 Alternative Service

The call was not successful, but alternative services are possible. The alternative services are described in the message body of the response.

7.4 Request Failure 4xx

4xx responses are definite failure responses from a particular server. The client SHOULD NOT retry the same request without modification (e.g., adding appropriate authorization). However, the same request to a different server may be successful.

7.4.1 400 Bad Request

The request could not be understood due to malformed syntax.

7.4.2 401 Unauthorized

The request requires user authentication.

7.4.3 402 Payment Required

Reserved for future use.

7.4.4 403 Forbidden

The server understood the request, but is refusing to fulfill it. Authorization will not help, and the request should not be repeated.

7.4.5 404 Not Found

The server has definitive information that the user does not exist at the domain specified in the Request-URI. This status is also returned if the domain in the Request-URI does not match any of the domains handled by the recipient of the request.

7.4.6 405 Method Not Allowed

The method specified in the Request-Line is not allowed for the address identified by the Request-URI. The response MUST include an Allow header containing a list of valid methods for the indicated address.

7.4.7 406 Not Acceptable

The resource identified by the request is only capable of generating response entities which have content characteristics not acceptable according to the accept headers sent in the request.

7.4.8 407 Proxy Authentication Required

This code is similar to 401 (Unauthorized), but indicates that the client MUST first authenticate itself with the proxy. The proxy MUST return a Proxy-Authenticate header field (section 6.26) containing a challenge applicable to the proxy for the requested resource. The client MAY repeat the request with a suitable Proxy-Authorization header field (section 6.27). SIP access authentication is explained in section 12.2 and [H11].

This status code should be used for applications where access to the communication channel (e.g., a telephony gateway) rather than the callee herself requires authentication.

7.4.9 408 Request Timeout

The server could not produce a response, e.g., a user location, within the time indicated in the Expires request-header field. The client MAY repeat the request without modifications at any later time.

7.4.10 414 Request-URI Too Long

The server is refusing to service the request because the Request-URI is longer than the server is willing to interpret.

7.4.11 415 Unsupported Media Type

The server is refusing to service the request because the message body of the request is in a format not supported by the requested resource for the requested method.

7.4.12 420 Bad Extension

The server did not understand the protocol extension specified in a Require (Section 6.30) header field.

7.4.13 480 Temporarily Unavailable

The callee's end system was contacted successfully but the callee is currently unavailable (e.g., not logged in or logged in in such a manner as to preclude communication with the callee). The response may indicate a better time to call in the Retry-After header. The user may also be available elsewhere (unbeknownst to this host), thus, this response does not terminate any searches. The reason phrase SHOULD indicate a more precise cause as to why the callee is unavailable. This value SHOULD be setable by the user agent.

7.4.14 481 Invalid Call-ID

The server received a BYE or CANCEL request with a Call-ID (Section 6.12) value it does not recognize. (A server simply discards an ACK with an invalid Call-ID.)

7.4.15 482 Loop Detected

The server received a request with a Via (Section 6.40) path containing itself.

7.4.16 483 Too Many Hops

The server received a request that contains more Via entries (hops) (Section 6.40) than allowed by the Max-Forwards (Section 6.23) header field.

7.4.17 484 Address Incomplete

The server received a request with a To (Section 6.37) address or Request-URI that was incomplete. Additional information should be provided.

> This status code allows overlapped dialing. With overlapped dialing, the client does not know the length of the dialing string. It sends strings of increasing lengths, prompting the user for more input, until it no longer receives a 484 status response.

7.4.18 485 Ambiguous

The callee address provided in the request was ambiguous. The response MAY contain a listing of possible unambiguous addresses in Location headers.

Revealing alternatives may infringe on privacy concerns of the user or the organization. It MUST be possible to configure a server to respond with status 404 (Not Found) or to suppress the listing of possible choices if the request address was ambiguous.

Example response to a request with the URL `lee@example.com`:

```
485 Ambiguous SIP/2.0
Location: sip:carol.lee@example.com (Carol Lee)
Location: sip:p.lee@example.com (Ping Lee)
Location: sip:lee.foote@example.com (Lee M. Foote)
```

> Some email and voice mail systems provide this functionality. A status code separate from 3xx is used since the semantics are different: for 300, it is assumed that the same person or service will be reached by the choices provided. While an automated choice or sequential search makes sense for a 3xx response, user intervention is required for a 485 response.

7.5 Server Failure 5xx

5xx responses are failure responses given when a server itself has erred. They are not definitive failures, and MUST NOT terminate a search if other possible locations remain untried.

7.5.1 500 Server Internal Error

The server encountered an unexpected condition that prevented it from fulfilling the request.

7.5.2 501 Not Implemented

The server does not support the functionality required to fulfill the request. This is the appropriate response when the server does not recognize the request method and is not capable of supporting it for any user.

7.5.3 502 Bad Gateway

The server, while acting as a gateway or proxy, received an invalid response from the downstream server it accessed in attempting to fulfill the request.

7.5.4 503 Service Unavailable

The server is currently unable to handle the request due to a temporary overloading or maintenance of the server. The implication is that this is a temporary condition which will be alleviated after some delay. If known, the length of the delay may be indicated in a Retry-After header. If no Retry-After is given, the client MUST handle the response as it would for a 500 response.

Note: The existence of the 503 status code does not imply that a server has to use it when becoming overloaded. Some servers may wish to simply refuse the connection.

7.5.5 504 Gateway Timeout

The server, while acting as a gateway, did not receive a timely response from the server (e.g., a location server) it accessed in attempting to complete the request.

7.5.6 505 Version Not Supported

The server does not support, or refuses to support, the SIP protocol version that was used in the request message. The server is indicating that it is unable or unwilling to complete the request using the same major version as the client, other than with this error message. The response SHOULD contain an entity describing why that version is not supported and what other protocols are supported by that server.

7.6 Global Failures 6xx

6xx responses indicate that a server has definitive information about a particular user, not just the particular instance indicated in the Request-URI. All further searches for this user are doomed to failure and pending searches SHOULD be terminated.

7.6.1 600 Busy

The callee's end system was contacted successfully but the callee is busy and does not wish to take the call at this time. The response may indicate a better time to call in the Retry-After header. If the callee does not wish to reveal the reason for declining the call, the callee should use status code 603 (Decline) instead.

7.6.2 603 Decline

The callee's machine was successfully contacted but the user explicitly does not wish to or cannot participate. The response may indicate a better time to call in the Retry-After header.

7.6.3 604 Does Not Exist Anywhere

The server has authoritative information that the user indicated in the To request field does not exist anywhere. Searching for the user elsewhere will not yield any results.

7.6.4 606 Not Acceptable

The user's agent was contacted successfully but some aspects of the session description such as the requested media, bandwidth, or addressing style were not acceptable.

A 606 (Not Acceptable) response means that the user wishes to communicate, but cannot adequately support the session described. The 606 (Not Acceptable) response MAY contain a list of reasons in a Warning header or headers describing why the session described cannot be supported. Reasons are listed in Section 6.41. It is hoped that negotiation will not frequently be needed, and when a new user is being invited to join an already existing conference, negotiation may not be possible. It is up to the invitation initiator to decide whether or not to act on a 606 (Not Acceptable) response.

8 SIP Message Body

8.1 Body Inclusion

For a request message, the presence of a body is signaled by the inclusion of a Content-Length header. Only ACK, INVITE, OPTIONS and REGISTER requests may contain message bodies. For ACK, INVITE and OPTIONS, the message body is always a session description. The use of message bodies for REGISTER requests is for further study.

For response messages, whether or not a body is included is dependent on both the request method and the response message's response code. All responses MAY include a body, although it may be of zero length. Message bodies for 1xx responses contain advisory information about the progress of the request. 2xx responses contain session descriptions. For responses with status 300 or greater, the messaage body MAY contain additional, human-readable information about the reasons for failure. It is RECOMMENDED that information in 1xx and 300 and greater responses be of type `text/plain` or `text/html`.

8.2 Message Body Type

The Internet media type of the message body MUST be given by the Content-Type header field, If the body has undergone any encoding (such as compression) then this MUST be indicated by the Content-Encoding header field, otherwise Content-Encoding MUST be omitted. If applicable, the character set of the message body is indicated as part of the Content-Type header-field value.

8.3 Message Body Length

The body length in bytes MUST be given by the Content-Length header field. If no body is present in a message, then the Content-Length header MUST set to zero. If a server receives a message without Content-Length, it MUST assume it to be zero.

The "chunked" transfer encoding of HTTP/1.1 MUST NOT be used for SIP. (Note: The chunked encoding modifies the body of a message in order to transfer it as a series of chunks, each with its own size indicator.)

9 Compact Form

When SIP is carried over UDP with authentication and a complex session description, it may be possible that the size of a request or response is larger than the MTU. To reduce this problem, a more compact form of SIP is also defined by using alternative names for common header fields. These short forms are NOT abbreviations, they are field names. No other header field abbreviations are allowed.

short field name	long field name	note
c	Content-Type	
e	Content-Encoding	
f	From	
i	Call-ID	
l	Content-Length	
m	Location	from "moved"
s	Subject	
t	To	
v	Via	

Thus, the header in section 14.2 could also be written:

```
INVITE sip:schooler@vlsi.caltech.edu SIP/2.0
v:SIP/2.0/UDP 131.215.131.131;maddr=239.128.16.254;ttl=16
v:SIP/2.0/UDP 128.16.64.19
f:sip:mjh@isi.edu
t:sip:schooler@cs.caltech.edu
i:62729-27@128.16.64.19
c:application/sdp
CSeq: 4711 INVITE
l:187

v=0
o=user1 53655765 2353687637 IN IP4 128.3.4.5
s=Mbone Audio
i=Discussion of Mbone Engineering Issues
```

```
e=mbone@somewhere.com
c=IN IP4 224.2.0.1/127
t=0 0
m=audio 3456 RTP/AVP 0
```

Mixing short field names and long field names is allowed, but not recommended. Servers MUST accept both short and long field names for requests. Proxies MUST NOT translate a request between short and long forms if authentication fields are present.

10 SIP Transport

10.1 General Remarks

SIP is defined so it can use either UDP (unicast or multicast) or TCP as a transport protocol; it provides its own reliability mechanism.

10.1.1 Requests

Servers ignore isomorphic requests, but retransmit the appropriate response. (SIP requests are said to be *idempotent*, i.e., receiving more than one copy of a request does not change the server state.)

After receiving a CANCEL request from an upstream client, a stateful proxy server SHOULD send a CANCEL on all branches where it has not yet received a final response.

If the To header user and host information does not match an address supported by the server, the server returns a 404 (Not Found) error response. Otherwise, it searches for the Call-ID value.

If the Call-ID was found, it compares the tag value of To with the user's tag and rejects the request if the two do not match. If the From header, including any tag value, matches the value for an existing call leg, the server compares the CSeq header value. If less than or equal to the current sequence number, the request is a retransmission. Otherwise, it is a new request. If the From header does not match an existing call leg, a new call leg is created.

If the Call-ID was not found, a new call leg is created, with entries for the To, From and Call-ID headers. In this case, the To header should not have contained a tag. The server returns a response containing the same To value, but with a unique tag added. The tag MAY be omitted if the To refers to a fully qualified host name.

10.1.2 Responses

A server MAY issue one or more provisional responses at any time before sending a final response. If a stateful proxy, user agent server, redirect server or registrar cannot respond to a request with a final response within 200 ms, it MUST issue a provisional (1xx) response as soon as possible. Stateless proxies MUST NOT issue provisional responses on their own.

Responses are mapped to requests by the matching To, From, Call-ID, CSeq headers and the branch parameter of the first Via header. Responses terminate request retransmissions even if they have Via headers that cause them to be delivered to an upstream client.

A stateful proxy may receive a response that it does not have state for, that is, where it has no a record of an isomorphic request. If the Via header field indicates that the upstream server used TCP, the proxy actively opens a TCP connection to that address. Thus, proxies have to be prepared to receive responses on the incoming side of passive TCP connections, even though most responses will arrive on the incoming side of an active connection. (An active connection is a TCP connection initiated by the proxy, a passive connection is one accepted by the proxy, but initiated by another entity.)

100 responses are not forwarded, other 1xx responses MAY be forwarded, possibly after the server eliminates responses with status codes that had already been sent earlier. 2xx responses are forwarded according to the Via header. Once a stateful proxy has received a 2xx response, it MUST NOT forward non-2xx final responses. Responses with status 300 and higher are retransmitted by each stateful proxy until the next upstream proxy sends an ACK (see below for timing details) or CANCEL.

A stateful proxy can remove state for a call attempt and close any connections 20 seconds after receiving the first final response.

> The 20 second window is given by the maximum retransmission duration of 200 responses (10 times $T4$), in case the ACK is lost somewhere on the way to the called user agent or the next stateful proxy.

10.2 Source Addresses, Destination Addresses and Connections

10.2.1 Unicast UDP

UDP packets MUST have a source address that is valid as a destination for requests and responses. Responses are returned to the address listed in the Via header field (Section 6.40), *not* the source address of the request.

10.2.2 Multicast UDP

Requests may be multicast; multicast requests likely feature a host-independent Request-URI. Multicast requests SHOULD have a time-to-live value of no greater than one, i.e., be restricted to the local network.

A client receiving a multicast query does not have to check whether the *host* part of the Request-URI matches its own host or domain name. If the request was received via multicast, the response is also returned via multicast. Responses to multicast requests are multicast with the same TTL as the request, where the TTL is derived from the ttl parameter in the Via header (Section 6.40).

To avoid response implosion, servers MUST NOT answer multicast requests with a status code other than 2xx or 6xx. Servers only return 6xx responses if the To represents a single individual rather than a group of people. The server delays its response by a random interval between zero and one second. Servers SHOULD suppress responses if they hear a lower-numbered or 6xx response from another group member prior to sending. Servers do not respond to CANCEL requests received via multicast to avoid request implosion.

10.3 TCP

A single TCP connection can serve one or more SIP transactions. A transaction contains zero or more provisional responses followed by one or more final responses. (Typically, transactions contain exactly one final response, but there are exceptional circumstances, where, for example, multiple 200 responses may be generated.)

The client MAY close the connection at any time, but SHOULD keep the connection open at least until the first final response arrives. The server SHOULD NOT close the TCP connection until it has sent its final response, at which point it MAY close the TCP connection if it wishes to. However, normally it is the client's responsibility to close the connection.

If the server leaves the connection open, and if the client so desires it may re-use the connection for further SIP requests or for requests from the same family of protocols (such as HTTP or stream control commands).

If a client closes a connection or the connection is reset (e.g., because the client has crashed and rebooted), the server treats this as equivalent to having received a CANCEL request.

If a server needs to return a response to a client and no longer has a connection open to that client, it MAY open a connection to the address listed in the Via header. Thus, a proxy or user agent MUST be prepared to receive both requests and responses on a "passive" connection.

10.4 Reliability for BYE, CANCEL, OPTIONS, REGISTER Requests

10.4.1 UDP

A SIP client *using UDP* SHOULD retransmit a BYE, CANCEL, OPTIONS, or REGISTER request periodically with timer $T1$ until it receives a response, or until it has reached a set limit on the number of retransmissions. If the response is provisional, the client continues to retransmit the request, albeit less frequently, using timer $T2$. The default values of timer $T1$ and $T2$ are 1 and 5 seconds, respectively. The default retransmit limit is 20 times. After the server sends a final response, it cannot be sure the client has received the response, and thus SHOULD cache the results for at least 100 seconds to avoid having to, for example, contact the user or location server again upon receiving a retransmission.

Each server in a proxy chain generates its own final response to a CANCEL request; BYE and OPTIONS final responses are generated by redirect and user agent servers; REGISTER final responses are generated by registrars. Note that responses to these commands are *not* acknowledged via ACK.

> The value of the initial retransmission timer is smaller than that that for TCP since it is expected that network paths suitable for interactive communications have round-trip times smaller than 1 second. To avoid flooding the network with packets every second even if the destination network is unreachable, the retransmission count has to be bounded. Given that most transactions should consist of one request and a few responses, round-trip time estimation is not likely to be very useful. If RTT estimation is desired to more quickly discover a missing final response, each request retransmission needs to be labeled with its own Timestamp (Section 6.36), returned in the response. The server caches the result until it can be sure that the client will not retransmit the same request again.

10.4.2 TCP

Clients using TCP do *not* need to retransmit requests.

10.5 Reliability for ACK Requests

The ACK request does not generate responses. It is only retransmitted when a response to an INVITE request arrives. This behavior is independent of the transport protocol.

10.6 Reliability for INVITE Requests

Special considerations apply for the INVITE method.

1. After receiving an invitation, considerable time may elapse before the server can determine the outcome. For example, the called party may be "rung" or extensive searches may be performed, so delays between the request and a definitive response can reach several tens of seconds. If either

caller or callee are automated servers not directly controlled by a human being, a call attempt may be unbounded in time.

2. If a telephony user interface is modeled or if we need to interface to the PSTN, the caller's user interface will provide "ringback", a signal that the callee is being alerted. (The status response 180 (Ringing) may be used to initiate ringback.) Once the callee picks up, the caller needs to know so that it can enable the voice path and stop ringback. The callee's response to the invitation could get lost. Unless the response is transmitted reliably, the caller will continue to hear ringback while the callee assumes that the call exists.

3. The client has to be able to terminate an on-going request, e.g., because it is no longer willing to wait for the connection or search to succeed. The server will have to wait several round-trip times to interpret the lack of request retransmissions as the end of a call. If the call succeeds shortly after the caller has given up, the callee will "pick up the phone" and not be "connected".

10.6.1 UDP

For UDP, A SIP client SHOULD retransmits a SIP INVITE request periodically with timer $T1$ until it receives a response. If the client receives no response, it ceases retransmission after 20 attempts. If the response is provisional, the client continues to retransmit the request, albeit less frequently, using timer $T3$. The default values of timer $T1$ and $T3$ are 1 and 30 seconds, respectively.

The value of $T3$ was chosen so that for most normal phone calls, only one INVITE request will be issued. Typically, a phone switches to an answering machine or voice mail after about 20–22 seconds. The number of retransmissions after receiving a provisional response is unlimited to allow call queueing. Clients may limit the number of invitations sent for each call attempt.

For 2xx final responses, only the user agent client generates an ACK. If the response contained a Location header, the ACK is sent to the address listed in that Location header field. If the response did not contain a Location header, the client uses the same To header field and Request-URI as for the INVITE request and sends the ACK to the same destination as the original INVITE request. ACKs for final responses other than 2xx are sent to the source of the response by each client.

The server retransmits the final response at intervals of $T4$ (default value of $T4$ = 2 seconds) until it receives an ACK request for the same Call-ID and CSeq from the same From source or until it has retransmitted the final response 10 times. The ACK request MUST NOT be acknowledged to prevent a response-ACK feedback loop.

Fig. 11 and 12 show the client and server state diagram for invitations.

The mechanism in Sec. 10.4 would not work well for INVITE because of the long delays between INVITE and a final response. If the 200 response were to get lost, the callee would believe the call to exist, but the voice path would be dead since the caller does not know that the callee has picked up. Thus, the INVITE retransmission interval would have to be on the order of a second or two to limit the duration of this state confusion. Retransmitting the response a fixed number of times increases the probability of success, but at the cost of significantly higher processing and network load.

10.6.2 TCP

A client using TCP MUST NOT retransmit requests, but uses the same algorithm as for UDP (Section 10.6.1) to retransmit responses until it receives an ACK. (An implementation can simply set $T1$ and $T3$ to infinity and otherwise maintain the same state diagram.)

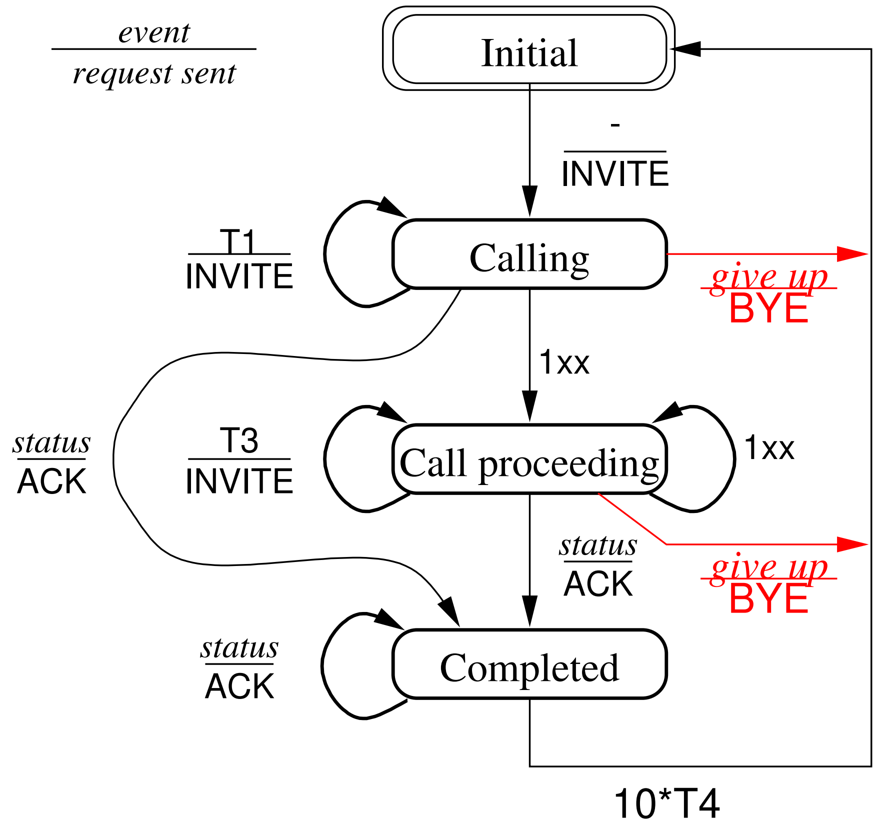

Figure 11: State transition diagram of client for INVITE method

It is necessary to retransmit 2xx responses as their reliability is assured end-to-end only. If the chain of proxies has a UDP link in the middle, it could lose the response, with no possibility of recovery. For simplicity, we also retransmit non-2xx responses, although that is not strictly necessary.

11 Behavior of SIP Servers

This section describes behavior of a SIP server in detail. Servers can operate in proxy or redirect mode. Proxy servers can "fork" connections, i.e., a single incoming request spawns several outgoing (client) requests.

A proxy server always inserts a Via header field containing its own address into those requests that are caused by an incoming request. Each proxy MUST insert a "branch" parameter (Section 6.40). To prevent loops, a server MUST check if its own address is already contained in the Via header of the incoming request.

A proxy server MAY convert a version-x SIP request or response to a version-y request or response, where x may be larger, smaller or equal to y.

This rule allows a proxy to serve as a go-between between two servers that have no version of the protocol in common.

11.1 Redirect Server

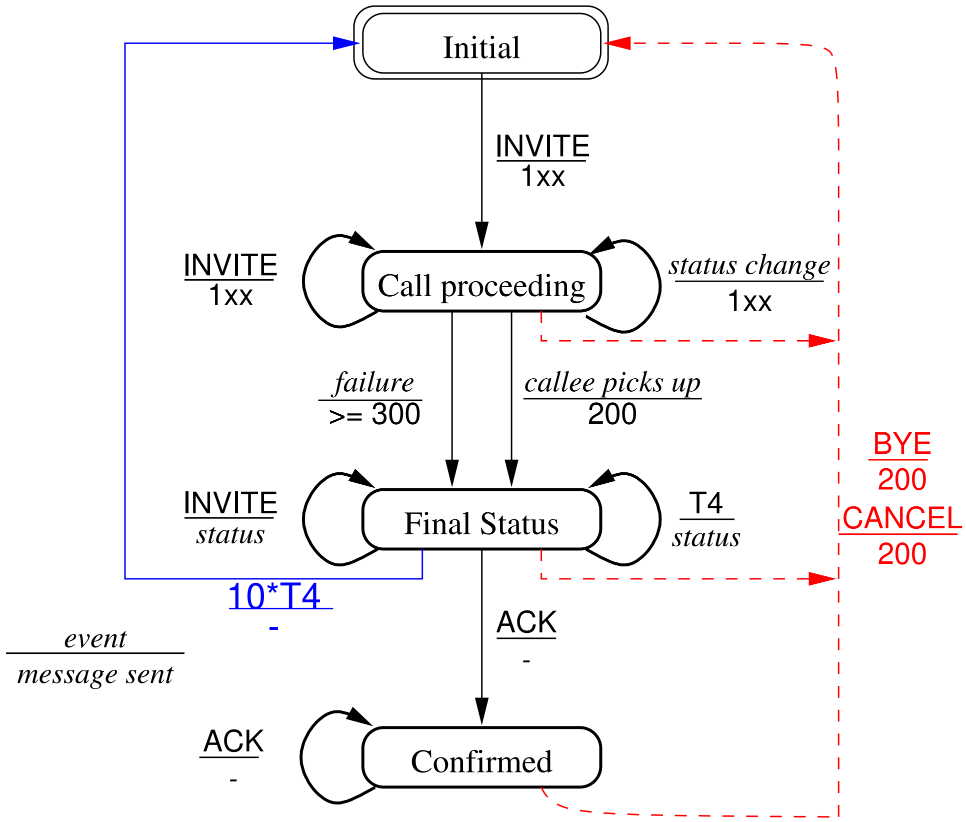

Figure 12: State transition diagram of server for INVITE method

A redirect server does not issue any SIP requests of its own. After receiving a request, the server gathers the list of alternative locations and returns a final response of class 3xx or it refuses the request. For CANCEL requests, it may also return a 2xx response. This response ends the SIP transaction. The redirect server maintains transaction state for the whole SIP transaction. It is up to the client to detect forwarding loops between redirect servers.

11.2 User Agent Server

User agent servers behave similarly to redirect servers, except that they may also accept requests and return a response of class 2xx.

11.3 Stateless Proxy: Proxy Servers Issuing Single Unicast Requests

Proxies in this category issue at most a single unicast request for each incoming SIP request, that is, they do not "fork" requests. However, servers may choose to always operate in a mode that allows issuing of several requests, as described in Section 11.4.

The server can forward the request and any responses. It does not have to maintain any state for the SIP transaction. Reliability is assured by the next redirect or stateful proxy server in the server chain.

A proxy server SHOULD cache the result of any address translations and the response to speed forwarding of retransmissions. After the cache entry has been expired, the server cannot tell whether an incoming request is actually a retransmission of an older request. The server will treat it as a new request and commence another search.

11.4 Proxy Server Issuing Several Requests

The server MUST respond to the request immediately with a 100 (Trying) response.

All outgoing requests carry the same Call-ID, To, From and CSeq headers as the request received. Each of the requests has a different (host-dependent) Request-URI.

Successful responses to an INVITE request SHOULD contain a Location header so that the following ACK or BYE bypasses the proxy search mechanism. If the proxy requires future requests to be routed through it, it adds a Record-Route header to the request (Section 6.29).

The following pseudo-code describes the behavior of a proxy server issuing several requests in response to an incoming INVITE request. The function `request(`r`, `a`, `b`)` sends a SIP request of type r to address a, with branch id b. `await_response()` waits until a response is received and returns the response. `close(`a`)` closes the TCP connection to client with address a. `response(`r`)` sends a response to the client. `ismulticast()` returns 1 if the location is a multicast address and zero otherwise. The variable `timeleft` indicates the amount of time left until the maximum response time has expired. The variable `recurse` indicates whether the server will recursively try addresses returned through a 3xx response. A server MAY decide to recursively try only certain addresses, e.g., those which are within the same domain as the proxy server. Thus, an initial multicast request may trigger additional unicast requests.

```
/* request type */
typedef enum {INVITE, ACK, BYE, OPTIONS, CANCEL, REGISTER} Method;

process_request(Method R, int N, address_t address[])
{
  struct {
    address_t address;  /* address */
    int branch;         /* branch id */
    int done;           /* has responded */
  } outgoing[];
  int done[];           /* address has responded */
  char *location[];     /* list of locations */
  int heard = 0;        /* number of sites heard from */
  int class;            /* class of status code */
  int timeleft = 120;   /* sample timeout value */
  int loc = 0;          /* number of locations */
  struct {              /* response */
    int status;         /* response: BYE=-2; CANCEL=-1 */
    int locations;      /* number of redirect locations */
    char *location[];   /* redirect locations */
    address_t a;        /* address of respondent */
    int branch;         /* branch identifier */
  } r, best;            /* response, best response */
  int i;
```

```
    best.status = 1000;
    for (i = 0; i < N; i++) {
      request(R, address[i], i);
      outgoing[i].done = 0;
      outgoing[i].branch = i;
    }

    while (timeleft > 0 && heard < N) {
      r = await_response();
      class = r.status / 100;

      if (r.status < 0) {
        break;
      }

      /* If final response, mark branch as done. */
      if (class >= 2) {
        heard++;
        for (i = 0; i < N; i++) {
          if (r.branch == outgoing[i].branch) {
            outgoing[i].done = 1;
            break;
          }
        }
      }

      if (class == 2) {
        best = r;
        break;
      }
      else if (class == 3) {
/* A server may optionally recurse.  The server MUST check
 * whether it has tried this location before and whether the
 * location is part of the Via path of the incoming request.
 * This check is omitted here for brevity.  Multicast locations
 * MUST NOT be returned to the client if the server is not
 * recursing.
         */
        if (recurse) {
          multicast = 0;
          N += r.locations;
          for (i = 0; i < r.locations; i++) {
            request(R, r.location[i]);
          }
```

```
        } else if (!ismulticast(r.location)) {
          best = r;
        }
        if (R == INVITE} request(ACK, r.a, r.branch);
      }
      else if (class == 4) {
        if (R == INVITE} request(ACK, r.a, r.branch);
        if (best.status >= 400) best = r;
      }
      else if (class == 5) {
        if (R == INVITE} request(ACK, r.a, r.branch);
        if (best.status >= 500) best = r;
      }
      else if (class == 6) {
        if (R == INVITE} request(ACK, r.a, r.branch);
        best = r;
        break;
      }
    }

    /* CANCEL */
    if (r.status == -1) {
      best.status = 200;
      response(best);
    }
    /* BYE */
    else if (r.status == -2) {
      for (i = 0; i < N; i++) {
        request(BYE, address[i], i);
      }
    }
    /* INVITE */
    else {
      /* We haven't heard anything useful from anybody. */
      if (best.status == 1000) {
        best.status = 404;
      }
      if (best.status/100 != 3) loc = 0;
      response(best);
    }

    /*
 * If complete or CANCELed, close the other pending transactions by
 * sending CANCEL.
     */
```

```
    if (r.status > 0 || r.status == -1) {
      for (i = 0; i < N; i++) {
        if (!outgoing[i].done) {
          request(CANCEL, address[i], outgoing[i].branch);
          if (tcp) close(a);
        }
      }
    }
  }
```

Responses are processed as follows. The process completes (and state can be freed) when all requests have been answered by final status responses (for unicast) or 60 seconds have elapsed (for multicast). A proxy MAY send a CANCEL to all branches and return a 408 (Timeout) to the client after 60 seconds or more.

1xx: The proxy MAY forward the response upstream towards the client.

2xx: The proxy MUST forward the response upstream towards the client, without sending an ACK downstream. After receiving a 2xx, the server SHOULD terminate all other pending requests by sending a CANCEL request and closing the TCP connection, if applicable. (Terminating pending requests is advisable as searches consume resources. Also, INVITE requests may "ring" on a number of workstations if the callee is currently logged in more than once.)

3xx: The proxy MUST send an ACK and MAY recurse on the listed Location addresses. Otherwise, the lowest-numbered response is returned if there were no 2xx responses.

> Location lists are not merged as that would prevent forwarding of authenticated responses. Also, some responses may have message bodies, so that merging is not feasible.

4xx, 5xx: The proxy MUST send an ACK and remember the response if it has a lower status code than any previous 4xx and 5xx responses. On completion, the lowest-numbered response is returned if there were no 2xx or 3xx responses.

6xx: The proxy MUST forward the response to the client and send an ACK. Other pending requests SHOULD be terminated with CANCEL as described for 2xx responses.

When operating in this mode, a proxy server MUST ignore any responses received for Call-IDs for which it does not have a pending transaction. (If server were to forward responses not belonging to a current transaction using the Via field, the requesting client would get confused if it has just issued another request using the same Call-ID.)

If a proxy server receives a BYE request for a pending search, the proxy MUST terminate all pending requests by sending a BYE request.

Special considerations apply for choosing forwarding destinations for ACK and BYE requests. In most cases, these requests will bypass proxies and reach the desired party directly, keeping proxies from having to make forwarding decisions.

User agent clients respond to ACK and BYE requests with unknown Call-ID with status code 481 (Invalid Call-ID).

A proxy MAY maintain call state for a period of its choosing. If a proxy still has list of destinations that it forwarded the last INVITE to, it SHOULD direct ACK requests only to those downstream servers. It SHOULD direct BYE to only those servers that had previously responded with 2xx or have not yet responded to the last INVITE. If the proxy has no call state for a particular Call-ID and To destination, it forks the request as it would for an INVITE request.

12 Security Considerations

12.1 Confidentiality and Privacy: Encryption

12.1.1 End-to-End Encryption

SIP requests and responses can contain sensitive information about the communication patterns and communication content of individuals and thus should be protected against eavesdropping. The SIP message body may also contain encryption keys for the session itself.

SIP supports three complementary forms of encryption to protect privacy:

- End-to-end encryption of the SIP message body and certain sensitive header fields;
- hop-by-hop encryption to prevent eavesdropping that tracks who is calling whom;
- hop-by-hop encryption of Via fields to hide the route a request has taken.

Not all of the SIP request or response can be encrypted end-to-end because header fields such as To and Via need to be visible to proxies so that the SIP request can be routed correctly. Hop-by-hop encryption encrypts the entire SIP request or response on the wire (the request may already have been end-to-end encrypted) so that packet sniffers or other eavesdroppers cannot see who is calling whom. Note that proxies can still see who is calling whom, and this information may also be deducible by performing a network traffic analysis, so this provides a very limited but still worthwhile degree of protection.

SIP Via fields are used to route a response back along the path taken by the request and to prevent infinite request loops. However, the information given by them may also provide useful information to an attacker. Section 6.21 describes how a sender can request that Via fields be encrypted by cooperating proxies without compromising the purpose of the Via field.

End-to-end encryption relies on keys shared by the two user agents involved in the request. Typically, the message is sent encrypted with the public key of the recipient, so that only that recipient can read the message. SIP does not limit the security mechanisms that may be used, but all implementations SHOULD support PGP-based encryption.

A SIP request (or response) is end-to-end encrypted by splitting the message to be sent into a part to be encrypted and a short header that will remain in the clear. Some parts of the SIP message, namely the request line, the response line and certain header fields marked with "n" in the "enc." column in Table 4 need to be read and returned by proxies and thus MUST NOT be encrypted end-to-end. Possibly sensitive information that needs to be made available as plaintext include destination address (To) and the forwarding path (Via) of the call. The Authorization header MUST remain in the clear if it contains a digital signature as the signature is generated after encryption, but MAY be encrypted if it contains "basic" or "digest" authentication. The From header field SHOULD normally remain in the clear, but MAY be encrypted if required, in which case some proxies MAY return a 401 (Unauthorized) status if they require a From field.

Other header fields MAY be encrypted or MAY travel in the clear as desired by the sender. The Subject, Allow, Call-ID, and Content-Type header fields will typically be encrypted. The Accept, Accept-Language, Date, Expires, Priority, Require, Cseq, and Timestamp header fields will remain in the clear.

All fields that will remain in the clear MUST precede those that will be encrypted. The message is encrypted starting with the first character of the first header field that will be encrypted and continuing through to the end of the message body. If no header fields are to be encrypted, encrypting starts with the second CRLF pair after the last header field, as shown below. Carriage return and line feed characters have been made visible as "$", and the encrypted part of the message is outlined.

```
  INVITE sip:watson@boston.bell-telephone.com SIP/2.0$
  Via: SIP/2.0/UDP 169.130.12.5$
  To: T. A. Watson <sip:watson@bell-telephone.com>$
  From: A. Bell <sip:a.g.bell@bell-telephone.com>$
  Encryption: PGP version=5.0$
  Content-Length: 224$
  CSeq: 488$
  $
*******************************************************
* Call-ID: 187602141351@worcester.bell-telephone.com$ *
* Subject: Mr. Watson, come here.$                    *
* Content-Type: application/sdp$                      *
* $                                                   *
* v=0$                                                *
* o=bell 53655765 2353687637 IN IP4 128.3.4.5$        *
* c=IN IP4 135.180.144.94$                            *
* m=audio 3456 RTP/AVP 0 3 4 5$                       *
*******************************************************
```

An Encryption header field MUST be added to indicate the encryption mechanism used. A Content-Length field is added that indicates the length of the encrypted body. The encrypted body is preceded by a blank line as a normal SIP message body would be.

Upon receipt by the called user agent possessing the correct decryption key, the message body as indicated by the Content-Length field is decrypted, and the now-decrypted body is appended to the clear-text header fields. There is no need for an additional Content-Length header field within the encrypted body because the length of the actual message body is unambiguous after decryption.

Had no SIP header fields required encryption, the message would have been as below. Note that the encrypted body must then include a blank line (start with CRLF) to disambiguate between any possible SIP header fields that might have been present and the SIP message body.

```
  INVITE sip:watson@boston.bell-telephone.com SIP/2.0$
  Via: SIP/2.0/UDP 169.130.12.5$
  To: T. A. Watson <sip:watson@bell-telephone.com>$
  From: A. Bell <a.g.bell@bell-telephone.com>$
  Encryption: PGP version=5.0$
```

```
  Content-Type: application/sdp$
  Content-Length: 107$
  $
*************************************************
* $                                             *
* v=0$                                          *
* o=bell 53655765 2353687637 IN IP4 128.3.4.5$  *
* c=IN IP4 135.180.144.94$                      *
* m=audio 3456 RTP/AVP 0 3 4 5$                 *
*************************************************
```

12.1.2 Privacy of SIP Responses

SIP requests may be sent securely using end-to-end encryption and authentication to a called user agent that sends an insecure response. This is allowed by the SIP security model, but is not a good idea. However, unless the correct behaviour is explicit, it would not always be possible for the called user agent to infer what a reasonable behaviour was. Thus when end-to-end encryption is used by the request originator, the encryption key to be used for the response SHOULD be specified in the request. If this were not done, it might be possible for the called user agent to incorrectly infer an appropriate key to use in the response. Thus, to prevent key-guessing becoming an acceptable strategy, we specify that a called user agent receiving a request that does not specify a key to be used for the response SHOULD send that response unencrypted.

Any SIP header fields that were encrypted in a request should also be encrypted in an encrypted response. Location response fields MAY be encrypted if the information they contain is sensitive, or MAY be left in the clear to permit proxies more scope for localized searches.

12.1.3 Encryption by Proxies

Normally, proxies are not allowed to alter end-to-end header fields and message bodies. Proxies MAY, however, encrypt an unsigned request or response with the key of the call recipient.

Proxies may need to encrypt a SIP request if the end system cannot perform encryption or to enforce organizational security policies.

12.1.4 Hop-by-Hop Encryption

It is RECOMMENDED that SIP requests and responses are also protected by security mechanisms at the transport and network layer.

12.1.5 Via field encryption

When Via fields are to be hidden, a proxy that receives a request containing an appropriate “Hide: hop” header field (as specified in section 6.21) SHOULD encrypt the header field. As only the proxy that encrypts the field will decrypt it, the algorithm chosen is entirely up to the proxy implementor. Two methods satisfy these requirements:

- The server keeps a cache of Via fields and the associated To field, and replaces the Via field with an index into the cache. On the reverse path, take the Via field from the cache rather than the message.

This is insufficient to prevent message looping, and so an additional ID must be added so that the proxy can detect loops. This should not normally be the address of the proxy as the goal is to hide the route, so instead a sufficiently large random number should be used by the proxy and maintained in the cache. Obtaining sufficiently much randomness to give sufficient protection against clashes may be hard.

It may also be possible for replies to get directed to the wrong originator if the cache entry gets reused, so great care must be taken to ensure this does not happen.

- The server may use a secret key to encrypt the Via field, a timestamp and an appropriate checksum in any such message with the same secret key. The checksum is needed to detect whether successful decoding has occurred, and the timestamp is required to prevent possible response attacks and to ensure that no two requests from the same previous hop have the same encrypted Via field.

The latter is the preferred solution, although proxy developers may devise other methods that might also satisfy the requirements.

12.2 Message Integrity and Access Control: Authentication

An active attacker may be able to modify and replay SIP requests and responses unless protective measures are taken. In practice, the same cryptographic measures that are used to ensure the authenticity of the SIP message also serve to authenticate the originator of the message.

Transport-layer or network-layer authentication may be used for hop-by-hop authentication. SIP also extends the HTTP WWW-Authenticate (Section 6.42) and Authorization (Section 6.11) header and their Proxy- counterparts to include cryptographically strong signatures. SIP also supports the HTTP "basic" authentication scheme that offers a very rudimentary mechanism of ascertaining the identity of the caller.

Since SIP requests are often sent to parties with which no prior communication relationship has existed, we do not specify authentication based on shared secrets.

SIP requests may be authenticated using the Authorization header field to include a digital signature of certain header fields, the request method and version number and the payload, none of which are modified between client and called user agent. The Authorization header field may be used in requests to end-to-end authenticate the request originator to proxies and the called user agent, and in responses to authenticate the called user agent or proxies returning their own failure codes. It does not provide hop-by-hop authentication, which may be provided if required using the IPSEC Authentication Header.

SIP does not dictate which digital signature scheme is used for authentication, but does define how to provide authentication using PGP in Section 13.

To sign a SIP request, the order of the SIP header fields is important. Via header fields MUST precede all other SIP header fields as these are modified in transit. When an Authorization header field is present, it indicates that all the header fields following the Authorization header field have been included in the signature. To sign a request, a client removes all of the SIP header from before where the Authorization field will be added. It then prepends the request method (in upper case) followed by the SIP version number field (in upper case) directly to the start of the message with no whitespace, CR or LF characters inserted. This extended message is what is signed.

For example, if the SIP request is to be:

```
INVITE sip:watson@boston.bell-telephone.com SIP/2.0
```

```
Via: SIP/2.0/UDP 169.130.12.5
Authorization: PGP version=5.0, signature=...
From: A. Bell <sip:a.g.bell@bell-telephone.com>
To: T. A. Watson <sip:watson@bell-telephone.com>
Call-ID: 187602141351@worcester.bell-telephone.com
Subject: Mr. Watson, come here.
Content-Type: application/sdp
Content-Length: ...

v=0
o=bell 53655765 2353687637 IN IP4 128.3.4.5
c=IN IP4 135.180.144.94
m=audio 3456 RTP/AVP 0 3 4 5
```

Then the data block that is signed is:

```
INVITESIP/2.0From: A. Bell <sip:a.g.bell@bell-telephone.com>
To: T. A. Watson <sip:watson@bell-telephone.com>
Call-ID: 187602141351@worcester.bell-telephone.com
Subject: Mr. Watson, come here.
Content-Type: application/sdp
Content-Length: ...

v=0
o=bell 53655765 2353687637 IN IP4 128.3.4.5
c=IN IP4 135.180.144.94
m=audio 3456 RTP/AVP 0 3 4 5
```

Note that if a message is encrypted and authenticated using a digital signature, when the message is generated encryption is performed before the digital signature is generated. On receipt, the digital signature is checked before decryption.

A client MAY require that a server sign its response by including a Require: org.ietf.sip.signed-response request header field. The client indicates the desired authentication method via the WWW-Authenticate header.

The correct behaviour in handling unauthenticated responses to a request that requires authenticated responses is described in section 12.2.1.

12.2.1 Trusting responses

It may be possible for an eavesdropper to listen to requests and to inject unauthenticated responses that would terminate, redirect or otherwise interfere with a call. (Even encrypted requests contain enough information to fake a response.)

Client should be particularly careful with 3xx redirection responses. Thus a client receiving, for example, a 301 (Moved Permanently) which was not authenticated when the public key of the called user agent is known to the client, and authentication was requested in the request SHOULD be treated as suspicious. The correct behaviour in such a case would be for the called-user to form a dated response containing the

Location field to be used, to sign it, and give this signed stub response to the proxy that will provide the redirection. Thus the response can be authenticated correctly. There may be circumstances where such unauthenticated responses are unavoidable, but a client SHOULD NOT automatically redirect such a request to the new location without alerting the user to the authentication failure before doing so.

Another problem might be responses such as 6xx failure responses which would simply terminate a search, or “4xx” and “5xx” response failures.

If TCP is being used, a proxy SHOULD treat 4xx and 5xx responses as valid, as they will not terminate a search. However, 6xx responses from a rogue proxy may terminate a search incorrectly. 6xx responses SHOULD be authenticated if requested by the client, and failure to do so SHOULD cause such a client to ignore the 6xx response and continue a search.

With UDP, the same problem with 6xx responses exists, but also an active eavesdropper can generate 4xx and 5xx responses that might cause a proxy or client to believe a failure occurred when in fact it did not. Typically 4xx and 5xx responses will not be signed by the called user agent, and so there is no simple way to detect these rogue responses. This problem is best prevented by using hop-by-hop encryption of the SIP request, which removes any additional problems that UDP might have over TCP.

These attacks are prevented by having the client require response authentication and dropping unauthenticated responses. A server user agent that cannot perform response authentication responds using the normal Require response of 420 (Bad Extension).

12.3 Callee Privacy

User location and SIP-initiated calls may violate a callee’s privacy. An implementation SHOULD be able to restrict, on a per-user basis, what kind of location and availability information is given out to certain classes of callers.

12.4 Known Security Problems

With either TCP or UDP, a denial of service attack exists by a rogue proxy sending 6xx responses. Although a client SHOULD choose to ignore such responses if it requested authentication, a proxy cannot do so. It is obliged to forward the 6xx response back to the client. The client can then ignore the response, but if it repeats the request it will probably reach the same rogue proxy again, and the process will repeat.

13 SIP Security Using PGP

13.1 PGP Authentication Scheme

The “pgp” authentication scheme is based on the model that the client must authenticate itself with a request signed with the client’s private key. The server can then ascertain the origin of the request if it has access to the public key, preferably signed by a trusted third party.

13.1.1 The WWW-Authenticate Response Header

WWW-Authenticate	=	"WWW-Authenticate" ":" "pgp" pgp-challenge
pgp-challenge	=	1# (realm | pgp-version | pgp-algorithm)
realm	=	"realm" "=" realm-value
realm-value	=	quoted-string
pgp-version	=	"version" "=" digit *("." digit) *letter
pgp-algorithm	=	"algorithm" "=" ("md5" | "sha1" | token)

The meanings of the values of the parameters used above are as follows:

realm: A string to be displayed to users so they know which identity to use. This string should contain at least the name of the host performing the authentication and might additionally indicate the collection of users who might have access. An example might be "`Users with call-out privileges`".

pgp-algorithm: A string indicating the PGP message integrity check (MIC) to be used to produce the signature. If this not present it is assumed to be "md5". The currently defined values are "md5" for the MD5 checksum, and "sha1" for the SHA.1 algorithm.

pgp-version: The version of PGP that the client MUST use. Common values are "2.6.2" and "5.0". The default is 5.0.

Example:

```
WWW-Authenticate: pgp version="5.0",
  realm="Your Startrek identity, please", algorithm="md5"
```

13.1.2 The Authorization Request Header

The client is expected to retry the request, passing an Authorization header line, which is defined as follows.

Authorization	=	"Authorization" ":" "pgp" pgp-response
pgp-response	=	1# (realm | pgp-version | pgp-signature | signed-by)
pgp-signature	=	"signature" "=" quoted-string
signed-by	=	"signed-by" "=" URI

The signature MUST correspond to the From header of the request unless the signed-by parameter is provided.

pgp-signature: The PGP ASCII-armored signature, as it appears between the "BEGIN PGP MESSAGE" and "END PGP MESSAGE" delimiters, without the version indication. The signature is included without any linebreaks.

The signature is computed across the request method, request version and header fields following the Authorization header and the message body, in the same order as they appear in the message. The request method and version are prepended to the header fields without any white space. The signature is computed across the headers as sent, including any folding and the terminating CRLF. The CRLF following the Authorization header is NOT included in the signature.

Using the ASCII-armored version is about 25% less space-efficient than including the binary signature, but it is significantly easier for the receiver to piece together. Versions of the PGP program always include the full (compressed) signed text in their output unless ASCII-armored mode (`-sta`) is specified. Typical signatures are about 200 bytes long. – The PGP signature mechanism allows the client to simply pass the request to an external PGP program. This relies on the requirement that proxy servers are not allowed to reorder or change header fields.

realm: The realm is copied from the corresponding WWW-Authenticate header field parameter.

signed-by: If and only if the request was not signed by the entity listed in the From header, the signed-by header indicates the name of the signing entity, expressed as a URI.

Receivers of signed SIP messages SHOULD discard any end-to-end header fields above the Authorization header, as they may have been maliciously added en route by a proxy.

Example:

```
Authorization: pgp version="5.0",
  realm="Your Startrek identity, please",
  signature="iQB1AwUBNNJiUaYBnHmiiQh1AQFYsgL/Wt3dk6TWK81/b0gcNDf
  VAUGU4rhEBW972IPxFSOZ94L1qhCLInTPaqhHFw1cb3lB01rA0RhpV4t5yCdUt
  SRYBSkOK29o5e1KlFeW23EzYPVUm2TlDAhbcjbMdfC+KLFX
  =aIrx"
```

13.2 PGP Encryption Scheme

The PGP encryption scheme uses the following syntax:

```
Encryption    =  "Encryption" ":" "pgp" pgp-eparams
pgp-eparams   =  1# ( pgp-version | pgp-encoding )
pgp-encoding  =  "encoding" "=" "ascii" | token
```

encoding: Describes the encoding or "armor" used by PGP. The value "ascii" refers to the standard PGP ASCII armor, without the lines containing "BEGIN PGP MESSAGE" and "END PGP MESSAGE" and without the version identifier. By default, the encrypted part is included as binary.

Example:

```
Encryption: pgp version="2.6.2", encoding="ascii"
```

13.3 Response-Key Header Field for PGP

```
Response-Key  =  "Response-Key" ":" "pgp" pgp-eparams
pgp-eparams   =  1# ( pgp-version | pgp-encoding | pgp-key)
pgp-key       =  "key" "=" quoted-string
```

If ASCII encoding has been requested via the encoding parameter, the key parameter contains the user's public key as extracted with the "pgp -kxa *user*".

Example:

```
Response-Key: pgp version="2.6.2", encoding="ascii",
  key="mQBtAzNWHNYAAAEDAL7QvAdK2utY05wuUG+ItYK5tCF8HNJM60sU4rLaV+eUnkMk
  mOmJWtc2wXcZx1XaXb2lkydTQOesrUR75IwNXBuZXPEIMThEa5WLsT7VLme7njnx
  sE86SgWmAZx5ookIdQAFEbQxSGVubmluZyBTY2h1bHpyaW5uZSA8c2NodWx6cmlu
  bmVAY3MuY29sdW1iaWEuZWR1Pg==
  =+y19"
```

14 Examples

14.1 Registration

A user at host `saturn.bell-tel.com` registers on start-up, via multicast, with the local SIP server named `sip.bell-tel.com`. In the example, the user agent on `saturn` expects to receive SIP requests on UDP port 3890.

```
C->S: REGISTER sip:@sip.bell-tel.com SIP/2.0
      Via: SIP/2.0/UDP 128.16.64.19
      From: sip:watson@bell-tel.com
      To: sip:watson@bell-tel.com
      Location: sip:saturn.bell-tel.com:3890;transport=udp
      Call-ID: 123@saturn.bell-tel.com
      Expires: 7200
      CSeq: 1 REGISTER
```

The registration expires after two hours. Any future invitations for `watson@bell-tel.com` arriving at `sip.bell-tel.com` will now be redirected to `watson@saturn.bell-tel.com`, UDP port 3890.

If Watson wants to be reached elsewhere, say, an on-line service he uses while traveling, he updates his reservation after first cancelling any existing locations:

```
C->S: REGISTER sip:@bell-tel.com SIP/2.0
      Via: SIP/2.0/UDP 128.16.64.19
      From: sip:watson@bell-tel.com
      To: sip:watson@bell-tel.com
      Call-ID: 1234@saturn.bell-tel.com
      Expire: 0
      Location: *

C->S: REGISTER sip:@bell-tel.com SIP/2.0
      Via: SIP/2.0/UDP 128.16.64.19
      From: sip:watson@bell-tel.com
      To: sip:watson@bell-tel.com
      Call-ID: 1235@saturn.bell-tel.com
      Location: sip:tawatson@example.com
```

Now, the server will forward any request for Watson to the server at `example.com`, using the Request-URI `tawatson@example.com`.

It is possible to use third-party registration. Here, the secretary `jon.diligent` registers his boss:

```
C->S: REGISTER sip:@bell-tel.com SIP/2.0
      Via: SIP/2.0/UDP 128.16.64.19
      From: sip:jon.diligent@bell-tel.com
      To: sip:watson@bell-tel.com
      Location: sip:tawatson@example.com
      Call-ID: 1236@saturn.bell-tel.com
```

The request could be send to either the registrar at `bell-tel.com` or the server at `example.com`. In the latter case, the server at `example.com` would proxy the request to the address indicated in the Request-URI. Then, Max-Forwards header could be used to restrict the registration to that server.

14.2 Invitation to Multicast Conference

The first example invites `schooler@vlsi.cs.caltech.edu` to a multicast session. All examples use the Session Description Protocol (SDP) (RFC 2327 [5]) as the session description format.

14.2.1 Request

```
C->S: INVITE sip:schooler@vlsi.cs.caltech.edu SIP/2.0
      Via: SIP/2.0/UDP 131.215.131.131;maddr=239.128.16.254;ttl=16
      Via: SIP/2.0/UDP 128.16.64.19
      From: Mark Handley <sip:mjh@isi.edu>
      To: Eve Schooler <sip:schooler@caltech.edu>
      Subject: SIP will be discussed, too
      Call-ID: 42100bb8-1dd2-11b2-8d70-c91e31477491@oregon.isi.edu
      Content-Type: application/sdp
      CSeq: 4711 INVITE
      Content-Length: 187

      v=0
      o=user1 53655765 2353687637 IN IP4 128.3.4.5
      s=Mbone Audio
      i=Discussion of Mbone Engineering Issues
      e=mbone@somewhere.com
      c=IN IP4 224.2.0.1/127
      t=0 0
      m=audio 3456 RTP/AVP 0
```

The Via fields list the hosts along the path from invitation initiator (the last element of the list) towards the invitee. In the example above, the message was last multicast to the administratively scoped group `239.128.16.254` with a ttl of `16` from the host `131.215.131.131`.

The request header above states that the request was initiated by `mjh@isi.edu`. The Via header indicates that is was initiated from the host `128.16.64.19`. `schooler@caltech.edu` is being invited; the message is currently being routed to `schooler@vlsi.cs.caltech.edu`.

In this case, the session description is using the Session Description Protocol (SDP), as stated in the Content-Type header.

The header is terminated by an empty line and is followed by a message body containing the session description.

14.2.2 Response

The called user agent, directly or indirectly through proxy servers, indicates that it is alerting ("ringing") the called party:

```
S->C: SIP/2.0 180 Ringing
      Via: SIP/2.0/UDP csvax.cs.caltech.edu;branch=8348;
        ;maddr=239.128.16.254;ttl=16
      Via: SIP/2.0/UDP north.east.isi.edu
      To: Eve Schooler <sip:schooler@caltech.edu>
      From: Mark Handley <sip:mjh@isi.edu>
      Call-ID: 42100bb8-1dd2-11b2-8d70-c91e31477491@north.east.isi.edu
      Location: sip:es@jove.cs.caltech.edu
      CSeq: 4711 INVITE
```

A sample response to the invitation is given below. The first line of the response states the SIP version number, that it is a 200 (OK) response, which means the request was successful. The Via headers are taken from the request, and entries are removed hop by hop as the response retraces the path of the request. A new authentication field MAY be added by the invited user's agent if required. The Call-ID is taken directly from the original request, along with the remaining fields of the request message. The original sense of From field is preserved (i.e., it is the session initiator).

In addition, the Location header gives details of the host where the user was located, or alternatively the relevant proxy contact point which should be reachable from the caller's host.

```
S->C: SIP/2.0 200 OK
      Via: SIP/2.0/UDP csvax.cs.caltech.edu;branch=8348
        maddr=239.128.16.254 16;ttl=16
      Via: SIP/2.0/UDP north.east.isi.edu
      From: sip:mjh@isi.edu
      To: sip:schooler@cs.caltech.edu
      Call-ID: 42100bb8-1dd2-11b2-8d70-c91e31477491@oregon.isi.edu
      Location: sip:es@jove.cs.caltech.edu
      CSeq: 4711 INVITE
```

The caller confirms the invitation by sending a request to the location named in the Location header:

```
C->S: ACK sip:es@jove.cs.caltech.edu SIP/2.0
```

```
      From: sip:mjh@isi.edu
      To: sip:schooler@cs.caltech.edu
      Call-ID: 42100bb8-1dd2-11b2-8d70-c91e31477491@oregon.isi.edu
      CSeq: 4711 ACK
```

14.3 Two-party Call

For two-party Internet phone calls, the response must contain a description of where to send the data. In the example below, Bell calls Watson. Bell indicates that he can receive RTP audio codings 0 (PCMU), 3 (GSM), 4 (G.723) and 5 (DVI4).

```
C->S: INVITE sip:watson@boston.bell-tel.com SIP/2.0
      Via: SIP/2.0/UDP 169.130.12.5
      From: A. Bell <sip:a.g.bell@bell-tel.com>
      To: T. Watson <sip:watson@bell-tel.com>
      Call-ID: 2d978243-b270-33dc-a261-d1fe3e2aa05a@kton.bell-tel.com
      Subject: Mr. Watson, come here.
      CSeq: 17 INVITE
      Content-Type: application/sdp
      Content-Length: ...

      v=0
      o=bell 53655765 2353687637 IN IP4 128.3.4.5
      c=IN IP4 135.180.144.94
      m=audio 3456 RTP/AVP 0 3 4 5

S->C: SIP/2.0 100 Trying
      From: A. Bell <sip:a.g.bell@bell-tel.com>
      To: T. Watson <sip:watson@bell-tel.com>
      Call-ID: 2d978243-b270-33dc-a261-d1fe3e2aa05a@kton.bell-tel.com
      Content-Length: 0

S->C: SIP/2.0 180 Ringing
      From: A. Bell <sip:a.g.bell@bell-tel.com>
      To: T. Watson <sip:watson@bell-tel.com>
      Call-ID: 2d978243-b270-33dc-a261-d1fe3e2aa05a@kton.bell-tel.com
      Content-Length: 0

S->C: SIP/2.0 182 Queued, 2 callers ahead
      From: A. Bell <sip:a.g.bell@bell-tel.com>
      To: T. Watson <sip:watson@bell-tel.com>
      Call-ID: 2d978243-b270-33dc-a261-d1fe3e2aa05a@kton.bell-tel.com
      Content-Length: 0

S->C: SIP/2.0 182 Queued, 1 caller ahead
```

```
       From: A. Bell <sip:a.g.bell@bell-tel.com>
       To: T. Watson <sip:watson@bell-tel.com>
       Call-ID: 2d978243-b270-33dc-a261-d1fe3e2aa05a@kton.bell-tel.com
       Content-Length: 0

S->C: SIP/2.0 200 OK
       From: A. Bell <sip:a.g.bell@bell-tel.com>
       To: sip:watson@bell-tel.com
       Call-ID: 2d978243-b270-33dc-a261-d1fe3e2aa05a@kton.bell-tel.com
       CSeq: 17 INVITE
       Location: sip:watson@boston.bell-tel.com
       Content-Length: ...

       v=0
       o=watson 4858949 4858949 IN IP4 192.1.2.3
       c=IN IP4 135.180.161.25
       m=audio 5004 RTP/AVP 0 3
```

The example illustrates the use of informational status responses. Here, the reception of the call is confirmed immediately (100), then, possibly after some database mapping delay, the call rings (180) and is then queued, with periodic status updates.

Watson can only receive PCMU and GSM. Note that Watson's list of codecs may or may not be a subset of the one offered by Bell, as each party indicates the data types it is willing to receive. Watson will send audio data to port 3456 at 135.180.144.94, Bell will send to port 5004 at 135.180.161.25.

By default, the media session is one RTP session. Watson will receive RTCP packets on port 5005, while Bell will receive them on port 3457.

Since the two sides have agree on the set of media, Watson confirms the call without enclosing another session description:

```
C->S: ACK sip:watson@boston.bell-tel.com SIP/2.0
       Via: SIP/2.0/UDP 169.130.12.5
       From: A. Bell <sip:a.g.bell@bell-tel.com>
       To: T. Watson <sip:watson@bell-tel.com>
       Call-ID: 2d978243-b270-33dc-a261-d1fe3e2aa05a@kton.bell-tel.com
       CSeq: 17 ACK
       Content-Length: 0
```

14.4 Terminating a Call

To terminate a call, caller or callee can send a BYE request:

```
C->S: BYE sip:watson@boston.bell-tel.com SIP/2.0
       From: A. Bell <sip:a.g.bell@bell-tel.com>
       To: T. A. Watson <sip:watson@bell-tel.com>
       Call-ID: 1985853074@kton.bell-tel.com
```

```
CSeq: 18 BYE
```

If the callee wants to abort the call, it simply reverses the To and From fields. Note that it is unlikely that an BYE from the callee will traverse the same proxies as the original INVITE.

14.5 Forking Proxy

In this example, Bell (a.g.bell@bell-tel.com) (C), currently seated at host c.bell-tel.com wants to call Watson (t.watson@ieee.org). At the time of the call, Watson is logged in at two workstations, watson@x.bell-tel.com (X) and watson@y.bell-tel.com (Y), and has registered with the IEEE proxy server (P) called proxy.ieee.org. The IEEE server also has a registration for the home machine of Watson, at watson@h.bell-tel.com (H), as well as a permanent registration at watson@acm.org (A). For brevity, the examples omit the session description.

Watson's user agent sends the invitation to the SIP server for the ieee.org domain:

```
C->P: INVITE sip:watson@ieee.org SIP/2.0
      From:    A. Bell <sip:a.g.bell@bell-tel.com>
      To:      T. Watson <sip:t.watson@ieee.org>
      Call-ID: 88323b2a-0a09-3888-88b4-2f93ee7808ea@kton.bell-tel.com
      CSeq:    19 INVITE
      Via:     SIP/2.0/UDP c.bell-tel.com
```

The SIP server tries the four addresses in parallel. It sends the following message to the home machine:

```
P->H: INVITE sip:watson@h.bell-tel.com SIP/2.0
      Via:     SIP/2.0/UDP proxy.ieee.org ;branch=1
      Via:     SIP/2.0/UDP c.bell-tel.com
      From:    A. Bell <sip:a.g.bell@bell-tel.com>
      To:      T. Watson <sip:t.watson@ieee.org>
      Call-ID: 88323b2a-0a09-3888-88b4-2f93ee7808ea@kton.bell-tel.com
      CSeq:    19 INVITE
```

This request immediately yields a 404 (Not Found) response, since Watson is not currently logged in at home:

```
H->P: SIP/2.0 404 Not Found
      Via:     SIP/2.0/UDP proxy.ieee.org ;branch=1
      Via:     SIP/2.0/UDP c.bell-tel.com
      From:    A. Bell <sip:a.g.bell@bell-tel.com>
      To:      T. Watson <sip:t.watson@ieee.org>
      Call-ID: 88323b2a-0a09-3888-88b4-2f93ee7808ea@c.bell-tel.com
      CSeq:    19 INVITE
```

The proxy ACKs the response so that host H can stop retransmitting it:

```
P->H: ACK sip:watson@h.bell-tel.com SIP/2.0
      Via:       SIP/2.0/UDP proxy.ieee.org ;branch=1
      From:      A. Bell <sip:a.g.bell@bell-tel.com>
      To:        T. Watson <sip:t.watson@ieee.org>
      Call-ID:   88323b2a-0a09-3888-88b4-2f93ee7808ea@c.bell-tel.com
      CSeq:      19 ACK
```

Also, P attempts to reach Watson through the ACM server:

```
P->A: INVITE sip:watson@acm.org SIP/2.0
      Via:       SIP/2.0/UDP proxy.ieee.org ;branch=2
      Via:       SIP/2.0/UDP c.bell-tel.com
      From:      A. Bell <sip:a.g.bell@bell-tel.com>
      To:        T. Watson <sip:t.watson@ieee.org>
      Call-ID:   88323b2a-0a09-3888-88b4-2f93ee7808ea@c.bell-tel.com
      CSeq:      19 INVITE
```

In parallel, the next attempt proceeds, with an INVITE to X and Y:

```
P->X: INVITE sip:watson@x.bell-tel.com SIP/2.0
      Via:      SIP/2.0/UDP proxy.ieee.org ;branch=3
      Via:      SIP/2.0/UDP c.bell-tel.com
      From:     A. Bell <sip:a.g.bell@bell-tel.com>
      To:       T. Watson <sip:t.watson@ieee.org>
      Call-ID: 88323b2a-0a09-3888-88b4-2f93ee7808ea@c.bell-tel.com
      CSeq:     19 INVITE

P->Y: INVITE sip:watson@y.bell-tel.com SIP/2.0
      Via:      SIP/2.0/UDP proxy.ieee.org ;branch=4
      Via:      SIP/2.0/UDP c.bell-tel.com
      From:     A. Bell <sip:a.g.bell@bell-tel.com>
      To:       T. Watson <sip:t.watson@ieee.org>
      Call-ID: 88323b2a-0a09-3888-88b4-2f93ee7808ea@c.bell-tel.com
      CSeq:     19 INVITE
```

As it happens, both Watson at X and a colleague in the other lab at host Y hear the phones ringing and pick up. Both X and Y return 200s via the proxy to Bell. The tag URI parameter is not strictly necessary here, since the Location header is unambiguous.

```
X->P: SIP/2.0 200 OK
      Via:       SIP/2.0/UDP proxy.ieee.org ;branch=3
      Via:       SIP/2.0/UDP c.bell-tel.com
      Location: sip:t.watson@x.bell-tel.com;tag=1620
      From:      A. Bell <sip:a.g.bell@bell-tel.com>
      To:        T. Watson <sip:t.watson@ieee.org>
```

```
      Call-ID:  88323b2a-0a09-3888-88b4-2f93ee7808ea@c.bell-tel.com
      CSeq:     19 INVITE

Y->P: SIP/2.0 200 OK
      Via:      SIP/2.0/UDP proxy.ieee.org ;branch=4
      Via:      SIP/2.0/UDP c.bell-tel.com
      Location: sip:t.watson@y.bell-tel.com;tag=2016
      From:     A. Bell <sip:a.g.bell@bell-tel.com>
      To:       T. Watson <sip:t.watson@ieee.org>
      Call-ID:  88323b2a-0a09-3888-88b4-2f93ee7808ea@c.bell-tel.com
      CSeq:     19 INVITE
```

Both responses are forwarded to Bell, using the Via information. At this point, the ACM server is still searching its database. P can now cancel this attempt:

```
P->A: CANCEL sip:watson@acm.org SIP/2.0
      Via:     SIP/2.0/UDP proxy.ieee.org ;branch=2
      From:    A. Bell <sip:a.g.bell@bell-tel.com>
      To:      T. Watson <sip:t.watson@ieee.org>
      Call-ID: 88323b2a-0a09-3888-88b4-2f93ee7808ea@c.bell-tel.com
      CSeq:    19 CANCEL
```

The ACM server gladly stops its neural-network database search and responds with a 200. The 200 will not travel any further, since P is the last Via stop.

```
A->P: SIP/2.0 200 OK
      Via:     SIP/2.0/UDP proxy.ieee.org ;branch=3
      From:    A. Bell <sip:a.g.bell@bell-tel.com>
      To:      T. Watson <sip:t.watson@ieee.org>
      Call-ID: 88323b2a-0a09-3888-88b4-2f93ee7808ea@c.bell-tel.com
      CSeq:    19 CANCEL
```

Bell gets the two 200 responses from X and Y in short order. Bell's reaction now depends on his software. He can either send an ACK to both if human intelligence is needed to determine who he wants to talk to or he can automatically reject one of the two calls. Here, he acknowledges both, separately and directly to the final destination:

```
C->X: ACK sip:watson@x.bell-tel.com SIP/2.0
      Via:      SIP/2.0/UDP c.bell-tel.com
      From:     A. Bell <sip:a.g.bell@bell-tel.com>
      To:       T. Watson <sip:t.watson@ieee.org>
      Call-ID:  88323b2a-0a09-3888-88b4-2f93ee7808ea@c.bell-tel.com
      CSeq:     19 ACK

C->Y: ACK sip:watson@y.bell-tel.com SIP/2.0
```

```
      Via:       SIP/2.0/UDP c.bell-tel.com
      From:      A. Bell <sip:a.g.bell@bell-tel.com>
      To:        T. Watson <sip:t.watson@ieee.org>
      Call-ID:   88323b2a-0a09-3888-88b4-2f93ee7808ea@c.bell-tel.com
      CSeq:      19 ACK
```

After a brief discussion between the three, it becomes clear that Watson is at X, thus Bell sends a BYE to Y, which is replied to:

```
C->Y: BYE sip:watson@y.bell-tel.com SIP/2.0
      Via:       SIP/2.0/UDP c.bell-tel.com
      From:      A. Bell <sip:a.g.bell@bell-tel.com>
      To:        T. Watson <sip:t.watson@ieee.org>
      Call-ID:   88323b2a-0a09-3888-88b4-2f93ee7808ea@c.bell-tel.com
      CSeq:      20 BYE

Y->C: SIP/2.0 200 OK
      Via:       SIP/2.0/UDP c.bell-tel.com
      From:      A. Bell <sip:a.g.bell@bell-tel.com>
      To:        T. Watson <sip:t.watson@ieee.org>
      Call-ID:   88323b2a-0a09-3888-88b4-2f93ee7808ea@c.bell-tel.com
      CSeq:      20 BYE
```

14.6 Redirects

Replies with status codes 301 (Moved Permanently) or 302 (Moved Temporarily) specify another location using the Location field:

```
S->C: SIP/2.0 302 Moved temporarily
      Via: SIP/2.0/UDP csvax.cs.caltech.edu ;branch=8348
      Via: SIP/2.0/UDP 128.16.64.19
      From: sip:mjh@isi.edu
      To: sip:schooler@cs.caltech.edu
      Call-ID: 46842902-e7b0-3583-ae6a-bee550833c34@oregon.isi.edu
      Location: sip:@239.128.16.254;ttl=16;transport=udp
      CSeq: 19 INVITE
      Content-Length: 0
```

In this example, the proxy located at `csvax.cs.caltech.edu` is being advised to contact the multicast group `239.128.16.254` with a ttl of `16` and UDP transport. In normal situations, a server would not suggest a redirect to a local multicast group unless, as in the above situation, it knows that the previous proxy or client is within the scope of the local group. If the request is redirected to a multicast group, a proxy server SHOULD query the multicast address itself rather than sending the redirect back towards the client as multicast may be scoped; this allows a proxy within the appropriate scope regions to make the query.

14.7 Alternative Services

An example of a 350 (Alternative Service) response is:

```
S->C: SIP/2.0 350 Alternative Service
      Via: SIP/2.0/UDP 131.215.131.131
      Via: SIP/2.0/UDP 128.16.64.19
      From: sip:mjh@isi.edu
      To: sip:schooler@cs.caltech.edu
      Call-ID: 46842902-e7b0-3583-ae6a-bee550833c34oregon.isi.edu
      Location: sip:recorder@131.215.131.131
      CSeq: 19 INVITE
      Content-Type: application/sdp
      Content-Length: 146

      v=0
      o=mm-server 2523535 0 IN IP4 131.215.131.131
      s=Answering Machine
      i=Leave an audio message
      c=IN IP4 131.215.131.131
      t=0 0
      m=audio 12345 RTP/AVP 0
```

In this case, the answering server provides a session description that describes an "answering machine". If the invitation initiator decides to take advantage of this service, it should send an invitation request to the answering machine at `131.215.131.131` with the session description provided (modified as appropriate for a unicast session to contain the appropriate local address and port for the invitation initiator). This request SHOULD contain a different Call-ID from the one in the original request. An example would be:

```
C->S: INVITE sip:recorder@131.215.131.131 SIP/2.0
      Via: SIP/2.0/UDP 128.16.64.19
      From: sip:mjh@isi.edu
      To: sip:schooler@cs.caltech.edu
      Call-ID: 9469f230-70e0-3216-8482-fe1a2a150386@128.16.64.19
      CSeq: 20 INVITE
      Content-Type: application/sdp
      Content-Length: 146

      v=0
      o=mm-server 2523535 0 IN IP4 131.215.131.131
      s=Answering Machine
      i=Leave an audio message
      c=IN IP4 128.16.64.19
      t=0 0
      m=audio 26472 RTP/AVP 0
```

Invitation initiators MAY choose to treat a 350 (Alternative Service) response as a failure if they wish to do so.

14.8 Negotiation

An example of a 606 (Not Acceptable) response is:

```
S->C: SIP/2.0 606 Not Acceptable
      From: sip:mjh@isi.edu
      To: sip:schooler@cs.caltech.edu
      Call-ID: 9469f230-70e0-3216-8482-fe1a2a150386@128.16.64.19
      Location: sip:mjh@131.215.131.131
      Warning: 606.1 Insufficient bandwidth (only have ISDN),
        606.3 Incompatible format,
        606.4 Multicast not available
      Content-Type: application/sdp
      Content-Length: 50

      v=0
      s=Lets talk
      b=CT:128
      c=IN IP4 131.215.131.131
      m=audio 3456 RTP/AVP 7 0 13
      m=video 2232 RTP/AVP 31
```

In this example, the original request specified 256 kb/s total bandwidth, and the response states that only 128 kb/s is available. The original request specified GSM audio, H.261 video, and WB whiteboard. The audio coding and whiteboard are not available, but the response states that DVI, PCM or LPC audio could be supported in order of preference. The response also states that multicast is not available. In such a case, it might be appropriate to set up a transcoding gateway and re-invite the user.

14.9 OPTIONS Request

A caller Alice can use an OPTIONS request to find out the capabilities of a potential callee Bob, without "ringing" the designated address. Bob returns a description indicating that he is capable of receiving audio and video, with a list of supported encodings.

```
C->S: OPTIONS sip:bob@example.com SIP/2.0
      From: Alice <sip:alice@anywhere.org>
      To: Bob <sip:bob@example.com>
      Call-ID: 45869@host.anywhere.org
      Accept: application/sdp

S->C: SIP/2.0 200 OK
      From: Alice <sip:alice@anywhere.org>
```

```
To: Bob <sip:bob@example.com>
Call-ID: 45869@host.anywhere.org
Content-Length: 81
Content-Type: application/sdp

v=0
m=audio 0 RTP/AVP 0 1 3 99
m=video 0 RTP/AVP 29 30
a=rtpmap:99 SX7300/8000
```

A Minimal Implementation

A.1 Client

All clients MUST be able to generate the INVITE and ACK requests. Clients MUST generate and parse the Call-ID, Content-Length, Content-Type, CSeq, From and To headers. Clients MUST also parse the Require header. A minimal implementation MUST understand SDP (RFC 2327, [5]). It MUST be able to recognize the status code classes 1 through 6 and act accordingly.

The following capability sets build on top of the minimal implementation described in the previous paragraph:

Basic: A basic implementation adds support for the BYE method to allow the interruption of a pending call attempt. It includes a User-Agent header in its requests and indicate its preferred language in the Accept-Language header.

Redirection: To support call forwarding, a client needs to be able to understand the Location header, but only the SIP-URL part, not the parameters.

Negotiation: A client MUST be able to request the OPTIONS method and understand the 380 (Alternative Service) status and the Location parameters to participate in terminal and media negotiation. It SHOULD be able to parse the Warning response header to provide useful feedback to the caller.

Authentication: If a client wishes to invite callees that require caller authentication, it must be able to recognize the 401 (Unauthorized) status code, must be able to generate the Authorization request header and MUST understand the WWW-Authenticate response header.

If a client wishes to use proxies that require caller authentication, it MUST be able to recognize the 407 (Proxy Authentication Required) status code, MUST be able to generate the Proxy-Authorization request header and understand the Proxy-Authenticate response header.

A.2 Server

A minimally compliant server implementation MUST understand the INVITE, ACK, OPTIONS and BYE requests. A proxy server MUST also understand CANCEL. It MUST parse and generate, as appropriate, the Call-ID, Content-Length, Content-Type, CSeq, Expires, From, Max-Forwards, Require, To and Via headers. It MUST echo the CSeq and Timestamp headers in the response. It SHOULD include the Server header in its responses.

A.3 Header Processing

Table 5 lists the headers that different implementations support. UAC refers to a user-agent client (calling user agent), UAS to a user-agent server (called user-agent).

	type	UAC	proxy	UAS	registrar
Accept	R	-	o	o	-
Accept-Language	R	-	b	b	b
Allow	405	o	-	-	-
Authorization	R	a	o	a	a
Call-ID	g	m	m	m	m
Content-Length	g	m	m	m	m
Content-Type	g	m	-	m	m
CSeq	g	o	m	m	m
Encryption	g	e	-	e	e
Expires	g	-	o	o	m
From	R	m	o	m	m
Location	R	-	-	-	m
Location	r	r	r	-	-
Max-Forwards	R	-	b	-	-
Proxy-Authenticate	407	a	-	-	-
Proxy-Authorization	R	-	a	-	-
Proxy-Require	R	-	m	-	-
Require	R	m	-	m	m
Response-Key	R	-	-	e	e
Timestamp	g	o	o	m	m
To	g	m	m	m	m
Unsupported	r	b	b	-	-
Via	g	-	m	m	m
WWW-Authenticate	401	a	-	-	-

Table 5: This table indicates which systems should be able to parse which response header fields. Type is as in Table 4. "-" indicates the field is not meaningful to this system (although it might be generated by it). "m" indicates the field MUST be understood. "b" indicates the field SHOULD be understood by a Basic implementation. "r" indicates the field SHOULD be understood if the system claims to understand redirection. "a" indicates the field SHOULD be understood if the system claims to support authentication. "e" indicates the field SHOULD be understood if the system claims to support encryption. "o" indicates support of the field is purely optional. Headers whose support is optional for all implementations are not shown.

B Usage of SDP

By default, the nth media session in a unicast INVITE request will become a single RTP session with the nth media session in the response. Thus, the callee should be careful to order media descriptions appropriately.

It is assumed that if caller or callee include a particular media type, they want to both send and receive media data. If the callee does not want to *send* a particular media type, it should mark the media entry as *recvonly*. If the callee does not want to *receive* a particular media type, it may mark it as *sendonly*. If the callee wants to neither receive nor send a particular media type, it should set the port to zero. (RTCP ports are not needed in this case.)

The caller should include all media types that it is willing to send so that the receiver can provide matching media descriptions.

The callee should set the port to zero if callee and caller only want to receive a media type.

C Summary of Augmented BNF

In this specification we use the Augmented Backus-Naur Form notation described in RFC 2234 [21]. For quick reference, the following is a brief summary of the main features of this ABNF.

"abc"
: The case-insensitive string of characters "abc" (or "Abc", "aBC", etc.);

%d32
: The character with ASCII code decimal 32 (space);

*term
: zero of more instances of term;

3*term
: three or more instances of term;

2*4term
: two, three or four instances of term;

[term]
: term is optional;

{ term1 term2 term3 }
: set notation: term1, term2 and term3 must all appear but their order is unimportant;

term1 | term2
: either term1 or term2 may appear but not both;

#term
: a comma separated list of term;

2#term
: a comma separated list of term containing at least 2 items;

2#4term
: a comma separated list of term containing 2 to 4 items.

Common Tokens

Certain tokens are used frequently in the BNF of this document, and not defined elsewhere. Their meaning is well understood but we include it here for completeness.

```
CR     =  %d13           ; carriage return character
LF     =  %d10           ; line feed character
CRLF   =  CR LF          ; typically the end of a line
SP     =  %d32           ; space character
TAB    =  %d09           ; tab character
LWS    =  *( SP | TAB)   ; linear whitespace
DIGIT  =  ”0” .. ”9”     ; a single decimal digit

unreserved  =  alphanum | mark
mark        =  ”-” | ”_” | ”.” | ”!” | ”~” | ”*” | ”’”
            |  ”(” | ”)”
escaped     =  ”%” hex hex
hex         =  digit | ”A” | ”B” | ”C” | ”D” | ”E” | ”F” |
               ”a” | ”b” | ”c” | ”d” | ”e” | ”f”
alphanum    =  alpha | digit
alpha       =  lowalpha | upalpha
lowalpha    =  ”a” | ”b” | ”c” | ”d” | ”e” | ”f” | ”g” | ”h” | ”i” |
               ”j” | ”k” | ”l” | ”m” | ”n” | ”o” | ”p” | ”q” | ”r” |
               ”s” | ”t” | ”u” | ”v” | ”w” | ”x” | ”y” | ”z”
upalpha     =  ”A” | ”B” | ”C” | ”D” | ”E” | ”F” | ”G” | ”H” | ”I” |
               ”J” | ”K” | ”L” | ”M” | ”N” | ”O” | ”P” | ”Q” | ”R” |
               ”S” | ”T” | ”U” | ”V” | ”W” | ”X” | ”Y” | ”Z”
digit       =  ”0” | ”1” | ”2” | ”3” | ”4” | ”5” | ”6” | ”7” |
               ”8” | ”9”
```

D IANA Considerations

Section 4.4 describes a name space and mechanism for registering SIP options.
Section 6.41 describes the name space for registering SIP warn-codes.

E Changes in Version -07

Since version -06, the following changes have been made.

- Removed references to Internet Drafts.
- Expanded URI definition to be independent of I-Ds.
- Clarified redirect behavior for BYE.
- Call-ID mandatory for all requests to allow to match requests with responses.

- Clarified that INVITE retransmit limit only applies if there has been no provisional response. Otherwise, call queueing is not possible.
- Removed open issues list.
- Removed REGISTER as special case of reliability mechanism.
- Split out large syntax diagrams into figures to avoid empty space.
- Abstract rewritten to reflect current protocol functionality.
- Reorganized "SIP Transport" chapter to more clearly reflect behavior for UDP and TCP.
- Modified syntax for Via to include multicast address as a parameter.
- "Ambiguous" status code moved from 381 to 485, to give precedence to other, more definitive 3xx responses. Also, 381 was the only 3xx response that a proxy could not automatically recurse on.
- Response merging made stricter, to avoid difficulties with merging bodies and non-standard headers of 3xx responses.
- REGISTER MUST have Location header.
- Responses SHOULD add a tag to the To header to allow requests (e.g., BYE) from several instances to be distinguished.
- REGISTER examples were missing Via headers.

F Acknowledgments

We wish to thank the members of the IETF MMUSIC WG for their comments and suggestions. Detailed comments were provided by Dave Devanathan, Yaron Goland, Christian Huitema, Jonathan Lennox, Moshe J. Sambol, and Eric Tremblay.

This work is based, inter alia, on [29, 30].

G Authors' Addresses

Mark Handley
USC Information Sciences Institute
c/o MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139
USA
electronic mail: mjh@isi.edu

Henning Schulzrinne
Dept. of Computer Science
Columbia University

1214 Amsterdam Avenue
New York, NY 10027
USA
electronic mail: schulzrinne@cs.columbia.edu

Eve Schooler
Computer Science Department 256-80
California Institute of Technology
Pasadena, CA 91125
USA
electronic mail: schooler@cs.caltech.edu

Jonathan Rosenberg
Lucent Technologies, Bell Laboratories
Rm. 4C-526
101 Crawfords Corner Road
Holmdel, NJ 07733
USA
electronic mail: jdrosen@bell-labs.com

References

[1] R. Pandya, “Emerging mobile and personal communication systems,” *IEEE Communications Magazine*, vol. 33, pp. 44–52, June 1995.

[2] B. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, “Resource ReSerVation protocol (RSVP) – version 1 functional specification,” RFC 2205, Internet Engineering Task Force, Oct. 1997.

[3] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: a transport protocol for real-time applications,” RFC 1889, Internet Engineering Task Force, Jan. 1996.

[4] H. Schulzrinne, R. Lanphier, and A. Rao, “Real time streaming protocol (RTSP),” RFC 2326, Internet Engineering Task Force, Apr. 1998.

[5] M. Handley and V. Jacobson, “SDP: session description protocol,” RFC 2327, Internet Engineering Task Force, Apr. 1998.

[6] S. Bradner, “Key words for use in RFCs to indicate requirement levels,” RFC 2119, Internet Engineering Task Force, Mar. 1997.

[7] R. Fielding, J. Gettys, J. Mogul, H. Nielsen, and T. Berners-Lee, “Hypertext transfer protocol – HTTP/1.1,” RFC 2068, Internet Engineering Task Force, Jan. 1997.

[8] T. Berners-Lee, L. Masinter, and M. McCahill, “Uniform resource locators (URL),” RFC 1738, Internet Engineering Task Force, Dec. 1994.

[9] A. Gulbrandsen and P. Vixie, “A DNS RR for specifying the location of services (DNS SRV),” RFC 2052, Internet Engineering Task Force, Oct. 1996.

[10] C. Partridge, "Mail routing and the domain system," RFC STD 14, 974, Internet Engineering Task Force, Jan. 1986.

[11] P. Mockapetris, "Domain names - implementation and specification," RFC STD 13, 1035, Internet Engineering Task Force, Nov. 1987.

[12] B. Braden, "Requirements for internet hosts - application and support," RFC STD 3, 1123, Internet Engineering Task Force, Oct. 1989.

[13] D. Zimmerman, "The finger user information protocol," RFC 1288, Internet Engineering Task Force, Dec. 1991.

[14] S. Williamson, M. Kosters, D. Blacka, J. Singh, and K. Zeilstra, "Referral whois (rwhois) protocol V1.5," RFC 2167, Internet Engineering Task Force, June 1997.

[15] W. Yeong, T. Howes, and S. Kille, "Lightweight directory access protocol," RFC 1777, Internet Engineering Task Force, Mar. 1995.

[16] E. M. Schooler, "A multicast user directory service for synchronous rendezvous," Master's Thesis CS-TR-96-18, Department of Computer Science, California Institute of Technology, Pasadena, California, Aug. 1996.

[17] T. Berners-Lee, "Universal resource identifiers in WWW: a unifying syntax for the expression of names and addresses of objects on the network as used in the world-wide web," RFC 1630, Internet Engineering Task Force, June 1994.

[18] T. Berners-Lee, L. Masinter, and R. Fielding, "Uniform resource identifiers (URI): generic syntax," Internet Draft, Internet Engineering Task Force, Mar. 1998. Work in progress.

[19] P. Leach and R. Salz, "UUIDs and GUIDs," Internet Draft, Internet Engineering Task Force, Feb. 1998. Work in progress.

[20] F. Yergeau, "UTF-8, a transformation format of ISO 10646," RFC 2279, Internet Engineering Task Force, Jan. 1998.

[21] D. Crocker and P. Overell, "Augmented BNF for syntax specifications: ABNF," RFC 2234, Internet Engineering Task Force, Nov. 1997.

[22] W. R. Stevens, *TCP/IP illustrated: the protocols*, vol. 1. Reading, Massachusetts: Addison-Wesley, 1994.

[23] J. Mogul and S. Deering, "Path MTU discovery," RFC 1191, Internet Engineering Task Force, Nov. 1990.

[24] D. Crocker, "Standard for the format of ARPA internet text messages," RFC STD 11, 822, Internet Engineering Task Force, Aug. 1982.

[25] P. Hoffman, L. Masinter, and J. Zawinski, "The mailto URL scheme," RFC 2368, Internet Engineering Task Force, July 1998.

[26] J. Palme, "Common internet message headers," RFC 2076, Internet Engineering Task Force, Feb. 1997.

[27] J. Mogul, T. Berners-Lee, L. Masinter, P. Leach, R. Fielding, H. Nielsen, and J. Gettys, "Hypertext transfer protocol – HTTP/1.1," Internet Draft, Internet Engineering Task Force, Mar. 1998. Work in progress.

[28] M. Elkins, "MIME security with pretty good privacy (PGP)," RFC 2015, Internet Engineering Task Force, Oct. 1996.

[29] E. M. Schooler, "Case study: multimedia conference control in a packet-switched teleconferencing system," *Journal of Internetworking: Research and Experience*, vol. 4, pp. 99–120, June 1993. ISI reprint series ISI/RS-93-359.

[30] H. Schulzrinne, "Personal mobility for multimedia services in the Internet," in *European Workshop on Interactive Distributed Multimedia Systems and Services (IDMS)*, (Berlin, Germany), Mar. 1996.

Full Copyright Statement