Internet Engineering Task Force
INTERNET-DRAFT
draft-ietf-mmusic-sip-11.ps

MMUSIC WG
Handley/Schulzrinne/Schooler/Rosenberg
ISI/Columbia U./Caltech/Bell Labs.
December 3, 1998
Expires: May 1999

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. SIP 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

INTERNET-DRAFT draft-ietf-mmusic-sip-11.ps December 3, 1998

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 can 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 Public Switched Telephone Network (PSTN) parties can 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) [5] for advertising multimedia sessions via multicast and the session description protocol (SDP) (RFC 2327 [6]) for describing multimedia sessions. However, the functionality and operation of SIP does not depend on any of these protocols.

SIP can 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 can be reached via H.323 [7], obtain the H.245 [8] gateway and user address and then use H.225.0 [9] to establish the call. In another example, SIP might be used to determine that the callee is reachable via the 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 can convey to the invited system the information necessary to do this.

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 [10] 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 [11]). The terms and generic syntax of URI and URL are defined in RFC 2396 [12]. 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 multiparty conference unit (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 sends SIP 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 can 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 can be used to create a conference.

Downstream: Requests sent in the direction from the caller to the callee (i.e., user agent client to user agent server).

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 interprets, and, if necessary, rewrites 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 a 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: From the SDP specification: "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 [6]) (A session as defined for SDP can comprise one or more RTP sessions.) As defined, a callee can 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 6.17) 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 user agent server to the user agent client.

URL-encoded: A character string encoded according to RFC 1738, Section 2.2 [13].

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 accepts, rejects or redirects 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.

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	yes
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 Overview 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).

1.4.1 SIP Addressing

The "objects" addressed by SIP are users at hosts, identified by a SIP URL. The SIP URL takes a form similar to a mailto or telnet URL, i.e., *user@host*. The *user* part is a user name or a telephone number. The *host* part is either a domain name having a DNS SRV (RFC 2052 [14]), CNAME or A record (RFC 1035 [15]), or a numeric network address. See section 2 for a detailed discussion of SIP URL's.

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 their email address.

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, for example, `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

When a client wishes to send a request, the client either sends it to a locally configured SIP proxy server (as in HTTP), independent of the Request-URI, or sends it to the IP address and port corresponding to the Request-URI.

For the latter case, the client must determine the protocol, port and IP address of a server to send the request to. A client SHOULD follow the steps below to obtain this information. At each step, unless stated otherwise, the client SHOULD try to contact a server at the port number listed in the Request-URI. If no port number is present in the Request-URI, the client uses port 5060. If the Request-URI specifies a protocol (TCP or UDP), the client contacts the server using that protocol. If no protocol is specified, the client tries UDP (if UDP is supported), and if the attempt fails, then tries TCP (if TCP is supported).

The client tries to find one or more addresses for the server by querying DNS. There are several different ways it may query DNS, and these are listed in the steps below. If a step elicits no addresses, the client SHOULD continue to the next step. However if a step elicits one or more addresses, but no SIP server at any of those addresses responds, then the client concludes the server is down and SHOULD NOT continue on to the next step.

1. If the host portion of the Request-URI is an IP address, the client contacts the server at the given address. If the host portion of the Request-URI is not an IP address, the client proceeds to the next step.

2. An implementation MAY use the experimental procedure in Appendix D at this step. This step relies on DNS SRV records, which are currently experimental and may change before being published as a

standard. If this step is used, and no servers are returned in the DNS query, the client proceeds to the next step.

3. The client queries the name server for CNAME records with the QNAME=sip.host, where host is the host portion of the Request-URI, as described in RFC 2219 [16]. For each address in the answer, the client attempts to contact a server at that address. If, however, there were no CNAME records, the client goes to the next step.

4. The client queries the name server for A records with the QNAME=sip.host, where host is the host portion of the Request-URI, as described in RFC 2219 [16]. For each address in the answer, the client attempts to contact a server at that address. If, however, there were no A records, the client goes to the next step.

5. The client queries the name server for CNAME records for the host portion of the Request-URI. For each address in the answer, the client attempts to contact a server at that address. If, however, there were no CNAME records, the client goes to the next step.

6. The client queries the name server for A records for the host portion of the Request-URI. For each address in the answer, the client attempts to contact a server at that address. If there were no A records, the client stops, as it has been unable to locate a server.

A client SHOULD rely on explicit network notifications to determine that a server is not reachable, rather than relying on timeouts. In particular, a client SHOULD be prepared to receive ICMP port, protocol, or host unreachable messages as a form of explicit notifications, and MAY be prepared to process other messages. (For socket-based programs: For TCP, `connect()` returns ECONNREFUSED if the client could not connect to a server at that address. For UDP, the socket needs to be bound to the destination address using `connect()` rather than `sendto()` or similar so that a second `write()` fails with ECONNREFUSED if there is no server listening) If the client finds the server is not reachable at a particular address, it SHOULD behave as if it had received a 400-class error response to that request.

A client MAY cache a successful DNS query result. A successful query is one which contained records in the answer, and a server was contacted at one of the addresses from the answer. When the client wishes to send a request to the same host, it MUST start the search as if it had just received this answer from the name server. The client MUST expire the cache entry after the DNS time-to-live value has elapsed. If the client does not find a SIP server among the addresses listed in the cached answer, it MUST start the search at the beginning of the sequence described above.

For example, consider a client which wishes to send a SIP request. The Request-URI for the destination is `sip:user@company.com`. The client only supports UDP. It would follow these steps:

1. The host portion is not an IP address, so the client goes to step 2 above.

2. The client decides to implement the optional SRV record step. The client does a DNS query of QNAME="sip.udp.company.com", QCLASS=IN, QTYPE=SRV. Since it doesn't support TCP, it omits the TCP query. There were no addresses in the DNS response, so the client goes to the next step.

3. The client does a DNS query of QNAME="sip.company.com", QCLASS=IN, QTYPE=CNAME. There were no addresses in the DNS response. The client therefore goes to the next step.

4. The client does a DNS query of QNAME="sip.company.com", QCLASS=IN, QTYPE=A. There is one answer. The client tries that address using UDP at port 5060, and succeeds at delivering the request.

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. All responses to a request contain the same values in the Call-ID, CSeq, To, and From fields (with the possible addition of a tag in the To field (section 6.37)). This allows responses to be matched with requests. 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 use 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 [6]) 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 are allowed to 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). Note that an ACK can also be sent directly to the callee, bypassing the proxy. All requests and responses have the same Call-ID.

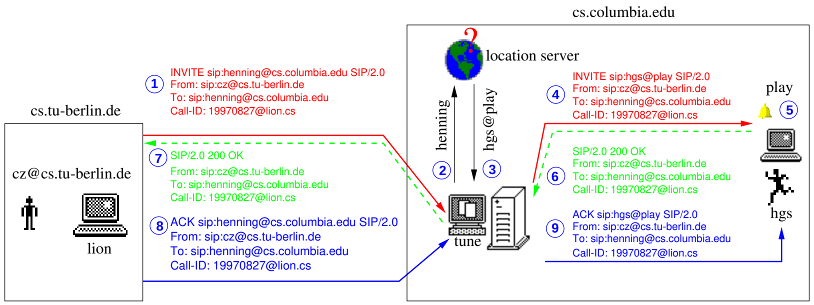

Figure 1: Example of SIP proxy server

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).

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 [17]), rwhois (RFC 2167 [18]), LDAP (RFC 1777 [19]), multicast-based protocols [20] 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.

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 Contact headers (Section 6.13). 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. A proxy server MUST check that it does not generate a request to a host listed in the Via sent-by, via-received or via-maddr parameters (Section 6.40). (Note: If a host has several names or network addresses, this does 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.,

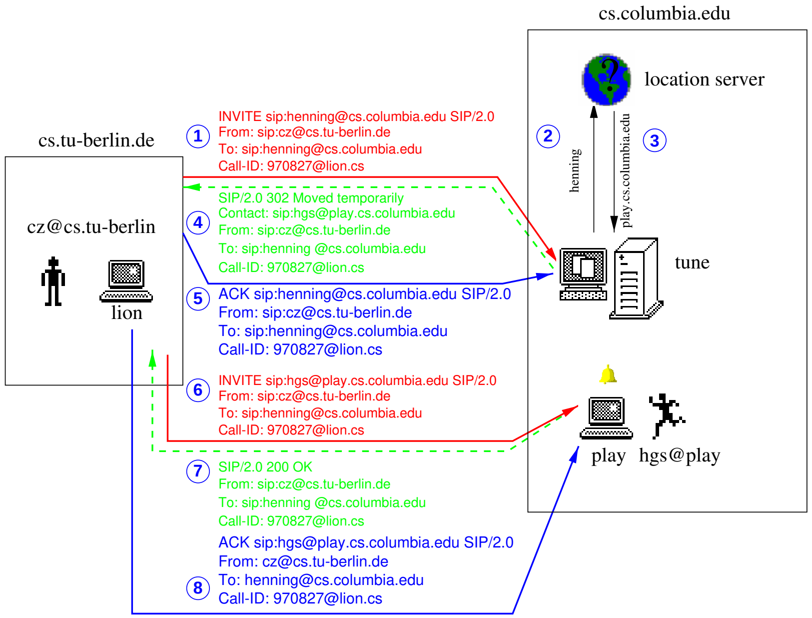

Figure 2: Example of SIP redirect server

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 same status response returned in the first response. Duplicate requests are not an error.

1.4.6 Changing an Existing Session

In some circumstances, it is desirable to change the parameters of an existing session. This is done by re-issuing the INVITE, using the same Call-ID, but a new or different body or header fields to convey the new information. This re INVITE MUST have a higher CSeq than any previous request from the client to the server.

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 can 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 involves 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 12. 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 can 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.

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, registrar, and redirect servers MUST implement both UDP and TCP transport.

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 (Contact). A SIP URL can also be embedded in web pages or other hyperlinks to indicate that a particular user or service can be called via SIP. When used as a hyperlink, the SIP URL indicates the use of the INVITE method.

The SIP URL scheme is defined to allow setting SIP request-header fields and the SIP message-body.

This corresponds to the use of mailto: URLs. It makes it possible, for example, to specify the subject, urgency or media types of calls initiated through a web page or as part of an email message.

A SIP URL follows the guidelines of RFC 2396 [12] and has the syntax shown in Fig. 3. The syntax is described using Augmented Backus-Naur Form (See Section C). Note that reserved characters have to be escaped and that the “set of characters reserved within any given URI component is defined by that component. In general, a character is reserved if the semantics of the URI changes if the character is replaced with its escaped US-ASCII encoding” [12].

```
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 | method-param
                 |  ttl-param | maddr-param | other-param
transport-param  =  ”transport=” ( ”udp” | ”tcp” )
user-param       =  ”user=” ( ”phone” | ”ip” )
method-param     =  ”method=” Method
ttl-param        =  ”ttl=” ttl
ttl              =  1*3DIGIT ; 0 to 255
maddr-param      =  ”maddr=” host
other-param      =  ( token | ( token ”=” ( token | quoted-string)))
headers          =  ”?” header *( ”&” header )
header           =  hname ”=” hvalue
hname            =  1*uric
hvalue           =  *uric
uric             =  reserved | unreserved | escaped
reserved         =  ”;” | ”/” | ”?” | ”:” | ”@” | ”&”
                 |  ”=” | ”+” | ”$” | ”,”
digits           =  1*DIGIT
```

Figure 3: SIP URL syntax

The URI character classes referenced above are described in Appendix C.


```
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

The components of the SIP URI have the following meanings.

user: If the host is an Internet telephony gateway, the user field MAY 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.

If a server handles SIP addresses for another domain, it MUST URL-encode the “@” character (%40). The ”;” character MUST be URL-encoded, as otherwise it is not possible to distinguish the case `host;parameter` and `user;moreuser@host` in one parsing pass.

password: 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.

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.

The issue of IPv6 literal addresses in URLs is being looked at elsewhere in the IETF. SIP implementers are advised to keep up to date on that activity.

port: The port number to send a request to. If not present, the procedures outlined in Section 1.4.2 are used to determine the port number to send a request to.

URL parameters: SIP URLs can define specific parameters of the request. URL parameters are added after the host component and are separated by semi-colons. The transport parameter determines the

the transport mechanism (UDP or TCP). UDP is to be assumed when no explicit transport parameter is included. The maddr parameter provides the server address to be contacted for this user, overriding the address supplied in the host field. This address is typically a multicast address, but could also be the address of a backup server. The ttl parameter determines the time-to-live value of the UDP multicast packet and MUST only be used if maddr is a multicast address and the transport protocol is UDP. The user parameter was described above. For example, to specify to call `j.doe@big.com` using 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, maddr, and ttl 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. hname and hvalue are encodings of a SIP header name and value, respectively. All URL reserved characters in the header names and values MUST be escaped.

Method: The method of the SIP request can be specified with the method parameter. This parameter MUST NOT be used in the From and To header fields and the Request-URI; they are ignored if present.

Table 2 summarizes where the components of the SIP URL can be used and what default values they assume if not present.

	default	Req.-URI	To	From	Contact	external
user	–	x	x	x	x	x
password	–	x	x		x	x
host	mandatory	x	x	x	x	x
port	5060	x	x	x	x	x
user-param	ip	x	x	x	x	x
method	INVITE				x	x
maddr-param	–				x	x
ttl-param	1				x	x
transp.-param	–				x	x
headers	–				x	x

Table 2: Use and default values of URL components for SIP headers, Request-URI and references

Examples of SIP URLs are:

```
sip:j.doe@big.com
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
sip:alice%40example.com@gateway.com
sip:alice@registrar.com;method=REGISTER
```

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.

SIP header fields MAY contain non-SIP URLs. As an example, if a call from a telephone is relayed to the Internet via SIP, the SIP From header field might contain a phone URL.

3 SIP Message Overview

SIP is a text-based protocol and uses the ISO 10646 character set in UTF-8 encoding (RFC 2279 [21]). Senders MUST terminate lines with a CRLF, but receivers MUST also interpret CR and LF by themselves as line terminators.

Except for the above difference in character sets, much of the message syntax is and header fields are identical to HTTP/1.1; rather than repeating the syntax and semantics here we use [HX.Y] to refer to Section X.Y of the current HTTP/1.1 specification (RFC 2068 [11]). In addition, we describe SIP in both prose and an augmented Backus-Naur form (ABNF). See section C for an overview of ABNF.

Note, however, that SIP is not an extension of HTTP.

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 be larger than the path maximum transmission unit (MTU) if the MTU is known, or 1500 bytes if the MTU is unknown.

The 1500 bytes accommodates encapsulation within the "typical" ethernet MTU without IP fragmentation. 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 headers 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 one or more CRLF, CR, or LFs, these characters MUST be ignored.


```
general-header   =  Accept               ; Section 6.7
                 |  Accept-Encoding      ; Section 6.8
                 |  Accept-Language      ; Section 6.9
                 |  Call-ID              ; Section 6.12
                 |  Contact              ; Section 6.13
                 |  CSeq                 ; Section 6.17
                 |  Date                 ; Section 6.18
                 |  Encryption           ; Section 6.19
                 |  Expires              ; Section 6.20
                 |  From                 ; Section 6.21
                 |  Record-Route         ; Section 6.29
                 |  Timestamp            ; Section 6.36
                 |  To                   ; Section 6.37
                 |  Via                  ; Section 6.40
entity-header    =  Content-Encoding     ; Section 6.14
                 |  Content-Length       ; Section 6.15
                 |  Content-Type         ; Section 6.16
request-header   =  Authorization        ; Section 6.11
                 |  Contact              ; Section 6.13
                 |  Hide                 ; 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
                 |  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 OPTIONS method and forwarded accordingly. Methods that are not supported by a user agent server or registrar cause a 501 (Not Implemented) response to be returned (Section 7). As in HTTP, the Method token is case-sensitive.

```
Method = "INVITE" | "ACK" | "OPTIONS" | "BYE"
       | "CANCEL" | "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 and possibly the media it is willing to send as well as their parameters such as network destination. A success response MUST indicate in its message body which media the callee wishes to receive and MAY indicate the media the callee is going to send.

Not all session description formats have the ability to indicate sending media.

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 leg 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. If there is a change, the user agent MUST update any internal state or information generated as a result of that header. 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 can 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.

A proxy server receiving an ACK request after having sent a 3xx, 4xx, 5xx, or 6xx response must make a determination about whether the ACK is for it, or for some user agent or proxy server further downstream. This determination is made by examining the tag in the To field. If the tag in the ACK To header field matches the tag in the To header field of the response, and the From, CSeq and Call-ID header fields in the response match those in the ACK, the ACK is meant for the proxy server. Otherwise, the ACK SHOULD be proxied downstream as any other request.

It is possible for a user agent client or proxy server to receive multiple 3xx, 4xx, 5xx, and 6xx responses to a request along a single branch. This can happen under various error conditions, typically when a forking proxy transitions from stateful to stateless before receiving all responses. The various responses will all be identical, except for the tag in the To field, which is different for each one. It can therefore be used as a means to disambiguate them.

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

4.2.3 OPTIONS

The server 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). Such a server SHOULD return an Allow header field indicating the methods that it supports. Proxy and redirect servers simply forward the request without indicating their capabilities.

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 release the call. A BYE request is forwarded like an INVITE request and MAY be issued by either caller or callee. A party to a call SHOULD issue a BYE request before releasing a call ("hanging up"). A party receiving a BYE request MUST cease transmitting media streams specifically directed at the party issuing the BYE request.

If the INVITE request contained a Contact header, the callee SHOULD send a 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 field 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, the numeric part of CSeq and From headers in the CANCEL request are identical to those in the original request. This allows a CANCEL request to be matched with the request it cancels. However, to allow the client to distinguish responses to the CANCEL from those to the original request, the CSeq Method component is set to CANCEL. 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 or user agent server receiving a CANCEL request responds with a status of 200 (OK) if the transaction exists and a status of 481 (Transaction Does Not Exist) 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 field with a SIP server.

A user agent MAY register with a local server on startup by sending a REGISTER request to the well-known "all SIP servers" multicast address "sip.mcast.net" (224.0.1.75). This request SHOULD be scoped to ensure it is not forwarded beyond the boundaries of the administrative system. This MAY be done with either TTL or administrative scopes[25], depending on what is implemented in the network. SIP user agents MAY listen to that address and use it to become aware of the location of other local users [20]; however, they do not respond to the request. A user agent MAY also be configured with the address of a registrar server to which it sends a REGISTER request upon startup.

Requests are processed in the order received. Clients SHOULD avoid sending a new registration (as opposed to a retransmission) until they have received the response from the server for the previous one.

Clients may register from different locations, by necessity using different Call-ID values. Thus, the CSeq value cannot be used to enforce ordering. Since registrations are additive, ordering is less of a problem than if each REGISTER request completely replaced all earlier ones.

The meaning of the REGISTER request-header fields is 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 field 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) might register under the Request-URI `sip:atlanta.hiayh.org`, with the former as the To header 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.

Call-ID: All registrations from a client SHOULD use the same Call-ID header value, at least within the same reboot cycle.

Cseq: Registrations with the same Call-ID MUST have increasing CSeq header values. However, the server does not reject out-of-order requests.

Contact: The request MAY contain a Contact header field; future non-REGISTER requests for the URI given in the To header field SHOULD be directed to the address(es) given in the Contact header.

If the request does not contain a Contact header, the registration remains unchanged.

This is useful to obtain the current list of registrations in the response.

Registrations using SIP URIs that differ in one or more of host, port, transport-param or maddr-param (see Figure 3) from an existing registration are added to the list of registrations. Other URI types are compared according to the standard URI equivalency rules for the URI schema. If the URIs are equivalent to that of an existing registration, the new registration replaces the old one if it has a higher q value or, for the same value of q, if the ttl value is higher. All current registrations MUST share the same action value. Registrations that have a different action than current registrations for the same user MUST be rejected with status of 409 (Conflict).

A proxy server ignores the q parameter when processing non-REGISTER requests, while a redirect server simply returns that parameter in its Contact response header field.

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 is supposed to 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". In the telephone network, directed pickup permits a user at a remote station who hears his own phone ringing to pick up at that station, dial an access code, and be connected to the calling user as if he had answered his own phone.

A server MAY choose any duration for the registration lifetime. Registrations not refreshed after this amount of time SHOULD be silently discarded. Responses to a registration SHOULD include an Expires header (Section 6.20) or expires Contact parameters (Section 6.13), indicating the time at which the server will drop the registration. If none is present, one hour is assumed. Clients MAY request a registration lifetime by indicating the time in an Expires header in the request. A server SHOULD NOT use a higher lifetime than the one requested, but MAY use a lower one. 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 Contact or the wildcard Contact designated by a "*" for all registrations. Registrations are matched based on the user, host, port and maddr parameters.

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

It is particularly important that REGISTER requests are authenticated since they allow to redirect future requests (see Section 13.2).

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.

Support of this method is RECOMMENDED.

4.3 Request-URI

The Request-URI 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 MAY be re-written by proxies.

When used as a Request-URI, a 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 Contact header field 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` is proxied to the particular person, e.g., `sip:bob@ny.acme.com`, with the To field remaining as `sip:sales@acme.com`. At `ny.acme.com`, Bob then designates Alice as the temporary substitute.

The host part of the Request-URI typically agrees with one of the host names of the receiving 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 can disagree in the case of a firewall proxy that handles outgoing calls. This mode of operation is 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 header field contains a scheme it does understand. This is because proxies are responsible for processing the Request-URI; the To field is of end-to-end significance.

4.3.1 SIP Version

Both request and response messages include the version of SIP in use, and follow [H3.1] (with HTTP replaced by SIP, and HTTP/1.1 replaced by SIP/2.0) regarding version ordering, compliance requirements, and upgrading of version numbers. To be 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 = token
```

See Section C for a definition of token. The creator of a new SIP option MUST either prefix the option with their 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". Individual organizations are then responsible for ensuring that option names don't collide. Options registered with IANA have the prefix "org.iana.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 MUST 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 consist of alphanum (See Figure 3) characters only;
- 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;
- Contact information (postal and email address);

This procedure has been borrowed from RTSP [4] and the RTP AVP [26].

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
```

SIP's structure of responses is similar to [H6], but is defined explicitly here.

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-UTF8, 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 needs to 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 cannot be fulfilled 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 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.


```
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

```
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 Entity Too Large
              |  ”414”  ; Request-URI Too Large
              |  ”415”  ; Unsupported Media Type
              |  ”420”  ; Bad Extension
              |  ”480”  ; Temporarily not available
              |  ”481”  ; Call Leg/Transaction Does Not Exist
              |  ”482”  ; Loop Detected
              |  ”483”  ; Too Many Hops
              |  ”484”  ; Address Incomplete
              |  ”485”  ; Ambiguous
              |  ”486”  ; Busy Here
```

Figure 7: Client error status codes

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

Figure 8: Server error status codes

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

Figure 9: Global failure status codes

6 Header Field Definitions

SIP header fields are similar to HTTP header fields in both syntax and semantics. In particular, SIP header fields follow the syntax for message-header as described in [H4.2]. The rules for extending header fields over multiple lines, and use of multiple message-header fields with the same field-name, described in [H4.2] also apply to SIP. The rules in [H4.2] regarding ordering of header fields apply to SIP, with the exception of Via fields, see below, whose order matters. Additionally, header fields which are hop-by-hop MUST appear before any header fields which are end-to-end. Proxies SHOULD NOT reorder header fields. Proxies add Via header fields and MAY add other hop-by-hop header fields. They can modify certain header fields, such as Max-Forwards 6.23 and "fix up" the Via header fields with "received" parameters as described in Section 6.40.1. Proxies MUST NOT alter any fields that are authenticated (see Section 13.2).

The header fields required, optional and not applicable for each method are listed in Table 4 and Table 5. 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. See sections 6.15, 6.16 and 8 for details.

The "where" column describes the request and response types with which the header field can be used. "R" refers to header fields that can be used in requests (that is, request and general header fields). "r" designates a response or general-header field as applicable to all responses, while a list of numeric values indicates the status codes with which the header field can be used. "g" and "e" designate general (Section 6.1) and entity header (Section 6.2) fields, respectively. If a header field is marked "c", it is copied from the request to the response.

The "enc." column describes whether this message header field 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.

The "e-e" column has a value of "e" for end-to-end and a value of "h" for hop-by-hop header fields.

Other header fields can be added as required; a server MUST ignore header fields not defined in this specification that it does not understand. A proxy MUST NOT remove or modify header fields not defined in this specification that it does not understand. 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.

	where	enc.	e-e	ACK	BYE	CAN	INV	OPT	REG
Accept	R		e	-	-	-	o	o	o
Accept	415		e	-	-	-	o	o	o
Accept-Encoding	R		e	-	-	-	o	o	o
Accept-Encoding	415		e	-	-	-	o	o	o
Accept-Language	R		e	-	o	o	o	o	o
Accept-Language	415		e	-	o	o	o	o	o
Allow	200		e	-	-	-	-	m	-
Allow	405		e	o	o	o	o	o	o
Authorization	R		e	o	o	o	o	o	o
Call-ID	gc	n	e	m	m	m	m	m	m
Contact	R		e	o	-	-	o	o	o
Contact	1xx		e	-	-	-	o	o	-
Contact	2xx		e	-	-	-	o	o	o
Contact	3xx		e	-	o	-	o	o	o
Contact	485		e	-	o	-	o	o	o
Content-Encoding	e		e	o	-	-	o	o	o
Content-Length	e		e	o	-	-	o	o	o
Content-Type	e		e	*	-	-	*	*	*
CSeq	gc	n	e	m	m	m	m	m	m
Date	g		e	o	o	o	o	o	o
Encryption	g	n	e	o	o	o	o	o	o
Expires	g		e	-	-	-	o	-	o
From	gc	n	e	m	m	m	m	m	m
Hide	R	n	h	o	o	o	o	o	o
Max-Forwards	R	n	e	o	o	o	o	o	o
Organization	g	c	h	-	-	-	o	o	o

Table 4: Summary of header fields, A–O

Table 6 in Appendix A lists those header fields that different client and server types MUST be able to parse.

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

	where	enc.	e-e	ACK	BYE	CAN	INV	OPT	REG
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		e	o	o	o	o	o	o
Retry-After	R	c	e	-	-	-	-	-	o
Retry-After	404,480,486	c	e	o	o	o	o	o	o
	503	c	e	o	o	o	o	o	o
	600,603	c	e	o	o	o	o	o	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	gc(1)	n	e	m	m	m	m	m	m
Unsupported	420		e	o	o	o	o	o	o
User-Agent	g	c	e	o	o	o	o	o	o
Via	gc(2)	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 5: Summary of header fields, P–Z; (1): copied with possible addition of tag; (2): UAS removes first Via header field

body can 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 MUST 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-UTF8
                   or combinations of token,
                   tspecials, and quoted-string>
```

The relative order of header fields with different field names is not significant. 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

The Accept header follows the syntax defined in [H14.1]. The semantics are also identical, with the exception that if no Accept header is present, the server SHOULD assume a default value of `application/sdp`.

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]. The syntax of this header is defined in [H14.3]. The semantics in SIP are identical to those defined in [H14.3].

6.9 Accept-Language

The Accept-Language header follows the syntax defined in [H14.4]. The rules for ordering the languages based on the q parameter apply to SIP as well. When used in SIP, the Accept-Language request-header field 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

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 and SHOULD be present in an OPTIONS response.

Allow = "Allow" ":" 1#Method

6.11 Authorization

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.

Section 13.2 overviews the use of the Authorization header, and section 15 describes the syntax and semantics when used with PGP based authentication.

6.12 Call-ID

The Call-ID general-header field uniquely identifies a particular invitation or all registrations of a particular client. Note that a single multimedia conference can 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 can 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 SHOULD use the same Call-ID, at least within the same boot cycle.

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 = 1*uric

"host" SHOULD be either a fully qualified domain name or a globally routable IP address. If this is the case, the "local-id" SHOULD be an identifier consisting of URI characters that is unique within "host". Use of cryptographically random identifiers [27] is RECOMMENDED. If, however, host is not an FQDN or globally routable IP address (such as a net 10 address), the local-id MUST be globally unique, as opposed to unique within host. These rules guarantee overall global uniqueness of the Call-ID. The value for Call-ID MUST NOT be reused for a different call. Call-IDs are case-sensitive.

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.

For systems which have tight bandwidth constraints, many of the mandatory SIP headers have a compact form, as discussed in Section 9. These are alternate names for the headers which occupy less space in the message. In the case of Call-ID, the compact form is i.

For example, both of the following are valid:

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

or

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

6.13 Contact

The Contact general-header field can appear in INVITE, ACK, and REGISTER requests, and in 1xx, 2xx, 3xx, and 485 responses. In general, it provides a URL where the user can be reached for further communications.

INVITE and ACK requests: INVITE and ACK requests MAY contain Contact headers indicating from which location the request is originating.

This allows the callee to send future requests, such as 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 Contact response header field indicating the SIP address under which it is reachable most directly for future SIP requests, such as ACK, within the same Call-ID. The Contact header field contains the address of the server itself or that of a proxy, e.g., if the host is behind a firewall. The value of this Contact header is copied into the Request-URI of subsequent requests for this call. If the response also contains a Record-Route header field, the address in the Contact header field is added as the last item in the Route header field. See Section 6.29 for details.

The Contact value SHOULD NOT be cached across calls, as it may not represent the most desirable location for a particular destination address.

INVITE 1xx responses: A UAS sending a provisional response (1xx) MAY insert a Contact response header. It has the same semantics in a 1xx response as a 2xx INVITE response. Note that CANCEL requests MUST NOT be sent to that address, but rather follow the same path as the original request.

REGISTER requests: REGISTER requests MAY contain a Contact header field indicating at which locations the user is reachable. The REGISTER request defines a wildcard Contact field, "*", which MUST only be used with Expires: 0 to remove all registrations for a particular user. An optional "expires" parameter indicates the desired expiration time of the registration. If a Contact entry does not have an "expires" parameter, the Expires header field is used as the default value. If neither of these mechanisms is used, SIP URIs are assumed to expire after one hour. Other URI schemes have no expiration times.

REGISTER 2xx responses: A REGISTER response MAY return all locations at which the user is currently reachable. An optional "expires" parameter indicates the expiration time of the registration. If a Contact entry does not have an "expires" parameter, the value of the Expires header field indicates the expiration time. If neither mechanism is used, the expiration time specified in the request, explicitly or by default, is used.

3xx and 485 responses: The Contact response-header field can be used with a 3xx or 485 (Ambiguous) response codes to indicate one or more alternate addresses to try. It can appear in responses to BYE, INVITE and OPTIONS methods. The Contact header field contains URIs giving the new locations or user names to try, or may simply specify additional transport parameters. A 300 (Multiple Choices), 301 (Moved Permanently), 302 (Moved Temporarily) or 485 (Ambiguous) response SHOULD contain a Contact field containing URIs of new addresses 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 different server or multicast address to try or a change of SIP transport from UDP to TCP or vice versa. The client copies the "user", "password", "host", "port" and "user-param" elements of the Contact URI into the Request-URI of the redirected request and directs the request to the address specified by the "maddr" and "port" parameters, using the transport protocol given in the "transport" parameter. If "maddr" is a multicast address, the value of "ttl" is used as the time-to-live value.

Note that the Contact header field 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 Contact response header field can contain any suitable URI indicating where the called party can be reached, not limited to SIP URLs. For example, it can contain a phone or fax, mailto: (RFC 2368, [28]) 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" parameter is used only when registering with the REGISTER request. It indicates whether the client wishes that the server proxy or redirect future requests intended for the client. If this parameter is not specified the action taken depends on server configuration. In its response, the registrar SHOULD indicate the mode used. This parameter is ignored for other requests.

expires: The "expires" parameter indicates how long the URI is valid. The parameter is either a number indicating seconds or a quoted string containing an SIP-date. If this parameter is not provided, the value of the Expires header field determines how long the URI is valid.

```
Contact            =  ( "Contact" | "m" ) ":"
                      ("*" | (1# ( name-addr | addr-spec )
                      [ *( ";" contact-params ) ] [ comment ] ))
name-addr          =  [ display-name ] "<" addr-spec ">"
addr-spec          =  SIP-URL | URI
display-name       =  *token | quoted-string
contact-params     =  "q"                  "="   qvalue
                   |  "action"             "="   "proxy" | "redirect"
                   |  "expires"            "="   delta-seconds | <"> SIP-date <">
                   |  extension-attribute
extension-attribute =  extension-name   [ "="   extension-value ]
```

Even if the "display-name" is empty, the "name-addr" form MUST be used if the "addr-spec" contains a comma, semicolon or question mark.

The Contact header field fulfills functionality similar to the Location header field in HTTP. However, the HTTP header only allows one address, unquoted. Since URIs can contain commas and semicolons as reserved characters, they can be mistaken for header or parameter delimiters, respectively. The current syntax corresponds to that for the To and From header, which also allows the use of display names.

Example:

```
Contact: "Mr. Watson" <sip:watson@worcester.bell-telephone.com>
   ;q=0.7; expires=3600,
   "Mr. Watson" <mailto:watson@bell-telephone.com> ;q=0.1
```

6.14 Content-Encoding

```
Content-Encoding  =  ( "Content-Encoding" | "e" ) ":"
                     1#content-coding
```

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 body to be compressed without losing the identity of its underlying media type.

If multiple encodings have been applied to an entity, the content codings MUST be listed in the order in which they were applied.

All content-coding values are case-insensitive. The Internet Assigned Numbers Authority (IANA) acts as a registry for content-coding value tokens. See [3.5] for a definition of the syntax for content-coding.

Clients MAY apply content encodings to the body in requests. If the server is not capable of decoding the body, or does not recognize any of the content-coding values, it MUST send a 415 "Unsupported Media Type" response, listing acceptable encodings in the Accept-Encoding header. A server MAY apply content encodings to the bodies in responses. The server MUST only use encodings listed in the Accept-Encoding header in the request.

6.15 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" | "l") ":" 1*DIGIT

An example is

```
Content-Length: 3495
```

Applications SHOULD 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 field MUST be set to zero. If a server receives a UDP request without Content-Length, it MUST assume that the request encompasses the remainder of the packet. If a server receives a UDP request with a Content-Length, but the value is larger than the size of the body sent in the request, the client SHOULD generate a 400 class response. If there is additional data in the UDP packet after the last byte of the body has been read, the server MUST treat the remaining data as a separate message. This allows several messages to be placed in a single UDP packet.

If a response does not contain a Content-Length, the client assumes that it encompasses the remainder of the UDP packet or the data until the TCP connection is closed, as applicable. Section 8 describes how to determine the length of the message body.

6.16 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" | "c") ":" media-type

Examples of this header field are

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


6.17 CSeq

Clients MUST add the CSeq (command sequence) general-header field to every request. A CSeq header field in a request contains the request method and 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 in the received CSeq header field, 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 BYE request with a CSeq that is not higher should cause a 400 response to be generated.

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, and then discard, 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, if the client desires. 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
```

6.18 Date

Date is a general-header field. Its syntax is:

SIP-date = rfc1123-date

See [H14.19] for a definition of rfc1123-date. Note that unlike HTTP/1.1, SIP only supports the most recent RFC1123 [29] formatting for dates.

The Date header field reflects the time when the request or response is first sent. Thus, retransmissions have the same Date header field value as the original.

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

6.19 Encryption

The Encryption general-header field specifies that the content has been encrypted. Section 13 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 based on the public key belonging to the entity named in the To header field. Responses are encrypted based on the public key conveyed in the Response-Key header field. Note that the public keys themselves may not be used for the encryption. This depends on the particular algorithms used.

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 recipient's 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 13.

The example in Figure 10 shows a message encrypted with ASCII-armored PGP that was generated by applying "pgp -ea" to the payload to be encrypted.

Since proxies can 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.

6.20 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. In a REGISTER request, the client indicates how long it wishes the registration to be valid. In the response, the server indicates the earliest expiration time of all registrations. The server MAY choose a shorter time interval than that requested by the client, but SHOULD NOT choose a longer one.

For INVITE requests, it is a request and response-header field. In a request, the caller can limit the validity of an invitation, for example, if a client wants to limit the time duration of a search or a conference


```
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+
```


Figure 10: PGP Encryption Example

invitation. 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 a SIP-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" ":" (SIP-date | delta-seconds)

Two examples of its use are

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

6.21 From

Requests and responses MUST contain a From general-header field, indicating the initiator of the request. The From field MAY contain the "tag" parameter. The server copies the From header field from the request to the response. The optional "display-name" is meant to be rendered by a human-user interface. A system SHOULD use the display name "Anonymous" if the identity of the client is to remain hidden.

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.

Even if the "display-name" is empty, the "name-addr" form MUST be used if the "addr-spec" contains a comma, question mark, or semicolon.

```
From         =  ( "From" | "f" ) ":" ( name-addr | addr-spec )
                *( ";" addr-params )
addr-params  =  tag-param
tag-param    =  "tag=" UUID
UUID         =  1*( hex | "-" )
```

Examples:

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

The "tag" MAY appear in the From field of a request. It MUST be present when it is possible that two instances of a user sharing a SIP address can make call invitations with the same Call-ID.

The "tag" value MUST be globally unique and cryptographically random with at least 32 bits of randomness. A single user maintains the same tag throughout the call identified by the Call-ID.

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

6.22 Hide

A client uses the Hide request header field to indicate that it wants the path comprised of the Via header fields (Section 6.40) to 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 field 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 is 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 13.

The Hide header field has the following syntax:

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

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 downstream 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.

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

The Max-Forwards value is a decimal integer indicating the remaining number of times this request message is allowed to 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 general-header field conveys the name of the organization to which the entity issuing the request or response belongs. It MAY also be inserted by proxies at the boundary of an organization.

The field MAY be used by client software to filter calls.

```
Organization  =  "Organization" ":" *TEXT-UTF8
```

6.25 Priority

The Priority request-header field indicates the urgency of the request as perceived by the client.

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

It is RECOMMENDED that the value of "emergency" 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 [30], 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.

Unlike its usage within HTTP, the Proxy-Authenticate header MUST be passed upstream in the response to tha UAC. In SIP, only UAC's can authenticate themselves to proxies.

The syntax for this header is defined in [H14.33]. See 14 for further details on its usage.

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.

Unlike Authorization, the Proxy-Authorization header field applies only to the next outbound proxy that demanded authentication using the Proxy- Authenticate field. When multiple proxies are used in a chain, the Proxy-Authorization header field is consumed by the first outbound proxy that was expecting to receive credentials. A proxy MAY relay the credentials from the client request to the next proxy if that is the mechanism by which the proxies cooperatively authenticate a given request.

See [H14.34] for a definition of the syntax, and section 14 for a discussion of its usage.

6.28 Proxy-Require

The Proxy-Require header field is used to indicate proxy-sensitive features that MUST be supported by the proxy. Any Proxy-Require header field 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 a request by any proxy that insists on being in the path of subsequent requests for the same call leg. 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 field 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 field of subsequent requests within the same call leg, *reversing* the order of requests, so that the first entry is closest to the user agent client. If the response contained a Contact 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 leg to the first Request-URI in the Route request header field and remove that entry.

The calling user agent MUST NOT use the Record-Route header field in requests that contain Route header fields.

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# name-addr
```

Proxy servers SHOULD use the "maddr" URL parameter containing their address to ensure that subsequent requests are guaranteed to reach exactly the same server.

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 field 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 MUST be tagged in the Proxy-Require field as well (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 13 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 server cannot encrypt for whatever reason, it MUST follow normal Require header field procedures and return a 420 (Bad Extension) response. If this Require header field is not present, a server SHOULD still encrypt if it can.

6.32 Retry-After

The Retry-After general-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 anticipates being available again. The value of this field can be either an SIP-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 "Contact: * ;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" ":" ( SIP-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, 01 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 field 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# name-addr
```

6.34 Server

The Server response-header field contains information about the software used by the user agent server to handle the request. The syntax for this field is defined in [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 does not have to use the same subject indication as the invitation.)

```
Subject  =  ( "Subject" | "s" ) ":" *TEXT-UTF8
```

Example:

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

6.36 Timestamp

The timestamp general-header field 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) ]
```

Note that there MUST NOT be any LWS between a DIGIT and the decimal point.

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 )
     *( ";" addr-params )
```

Requests and responses MUST contain a To general-header field, indicating the desired recipient of the request. The optional "display-name" is meant to be rendered by a human-user interface. The UAS or redirect server copies the To header field into its response, and MUST add a "tag" parameter if the request contained more than one Via header field.

> If there was more than one Via header field, the request was handled by at least one proxy server. Since the receiver cannot know whether any of the proxy servers forked the request, it is safest to assume that they might have.

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.

The "tag" parameter serves as a general mechanism to distinguish multiple instances of a user identified by a single SIP URL. As proxies can fork requests, the same request can reach multiple instances of a user (mobile and home phones, for example). As each can respond, there needs to be a means to distinguish the responses from each at the caller. The situation also arises with multicast requests. The tag in the To header field serves to distinguish responses at the UAC. It MUST be placed in the To field of the response by each instance when there is a possibility that the request was forked at an intermediate proxy. The "tag" MUST be added by UAS, registrars and redirect servers, but MUST NOT be inserted into responses forwarded upstream by proxies. The "tag" is added for all definitive responses for all methods, and MAY be added for informational responses from a UAS or redirect server. All subsequent transactions between two entities MUST include the "tag" parameter, as described in Section 11.

See Section 6.21 for details of the "tag" parameter.

The "tag" parameter in To headers is ignored when matching responses to requests that did not contain a "tag" in their To header.

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.

Even if the "display-name" is empty, the "name-addr" form MUST be used if the "addr-spec" contains a comma, question mark, or semicolon.

The following are examples of valid To headers:

```
To: The Operator <sip:operator@cs.columbia.edu>;tag=287447
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 multiple responses from a forked request. The "tag" is added to the To header field 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 field lists the features not supported by the server. See Section 6.30 for a usage example and motivation.

Syntax:

```
Unsupported = "Unsupported" ":" 1#option-tag
```

6.39 User-Agent

The User-Agent general-header field contains information about the client user agent originating the request. The syntax and semantics are defined in [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 can 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 proxied request.

A proxy SHOULD check the top-most Via header field 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 MUST 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. Use of the received attribute allows SIP requests to traverse NAT's which only modify the source IP address. NAT's which modify port numbers, called Network Address Port Translator's (NAPT) will not properly pass SIP when transported on UDP, in which case an application layer gateway is required. When run over TCP, SIP stands a better chance of traversing NAT's, since its port usage is identical to HTTP in this case.

A proxy sending a request to a multicast address MUST add the "maddr" parameter to its Via header field, and SHOULD add the "ttl" parameter. If a server receives a request which contained an "maddr" parameter in the topmost Via field, it SHOULD send the response to the multicast address listed in the "maddr" parameter.

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

A proxy server MUST NOT forward a request to a multicast group which already appears in any of the Via headers.

> 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 Header 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) changes the source address and port of the request (e.g., from net-10 to a globally routable address), in which case the Via header field cannot be relied on to route replies. To prevent this, a proxy SHOULD check the top-most Via header field 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 MUST add a "received" parameter to the Via header field inserted by the previous hop. Such a modified Via header field is known as a receiver-tagged Via header 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 added a parameter to the previous hop's Via header field, containing the address that the packet actually came from. (Note that the NAT `border.ieee.org` is not a SIP server.)

6.40.3 Responses

Via header fields in responses are processed by a proxy or UAC according to the following rules:

1. The first Via header field should indicate the proxy or client processing this response. If it does not, discard the message. Otherwise, remove this Via field.
2. If there is no second Via header field, this response is destined for this client. Otherwise, the processing depends on whether the Via field contains a "maddr" parameter or is a receiver-tagged field:
   - If the second Via header field contains a "maddr" parameter, send the response to the multicast address listed there, using the port indicated in "sent-by", or port 5060 if none is present. The response SHOULD be sent using the TTL indicated in the "ttl" parameter, or with a TTL of 1 if that parameter is not present. For robustness, responses MUST be sent to the address indicated in the "maddr" parameter even if it is not a multicast address.
   - If the second Via header field does not contain a "maddr" parameter and is a receiver-tagged field (Section 6.40.2), send the message to the address in the "received" parameter, using the port indicated in the "sent-by" value, or using port 5060 if none is present.
   - If neither of the previous cases apply, send the message to the address indicated by the "sent-by" value in the second Via header field.

6.40.4 User Agent and Redirect Servers

A UAS or redirect server sends a response based on one of the following rules:

- If the first Via header field in the request contains a "maddr" parameter, send the response to the multicast address listed there, using the port indicated in "sent-by", or port 5060 if none is present. The response SHOULD be sent using the TTL indicated in the "ttl" parameter, or with a TTL of 1 if that parameter is not present. For robustness, responses MUST be sent to the address indicated in the "maddr" parameter even if it is not a multicast address.

- If the address in the "sent-by" value of the first Via field differs from the source address of the packet, send the response to the actual packet source address, similar to the treatment for receiver-tagged Via header fields (Section 6.40.2).
- If neither of these conditions is true, send the response to the address contained in the "sent-by" value. If the request was sent using TCP, use the existing TCP connection if available.

6.40.5 Syntax

The format for a Via header field is shown in Fig. 11. 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 6.22). The "hidden" parameter is included if this header field was hidden by the upstream proxy (see 6.22). 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              =  ( "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 11: Syntax of Via header field

The "branch" parameter is included by every forking proxy. The token MUST be unique for each distinct request generated when a proxy forks. When a response arrives at the proxy it can use the branch value to figure out which branch the response corresponds to. A proxy which generates a single request (non-forking) MAY also insert the "branch" parameter. The identifier has to be unique only within a set of isomorphic requests.

```
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 can be based on any available knowledge, such as the location of the cache or user, the Accept-Language field in a request, or the Content-Language field in a response. The default language is i-default [31].

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 field 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.

The warn-code consists of three digits. A first digit of "3" indicates warnings specific to SIP.

This is a list of the currently-defined "warn-code"s, each with a recommended warn-text in English, and a description of its meaning. Note that these warnings describe failures induced by the session description.

Warnings 300 through 329 are reserved for indicating problems with keywords in the session description, 330 through 339 are warnings related to basic network services requested in the session description, 370 through 379 are warnings related to quantitative QoS parameters requested in the session description, and 390 through 399 are miscellaneous warnings that do not fall into one of the above categories.

300 Incompatible network protocol: One or more network protocols contained in the session description are not available.

301 Incompatible network address formats: One or more network address formats contained in the session description are not available.

302 Incompatible transport protocol: One or more transport protocols described in the session description are not available.

303 Incompatible bandwidth units: One or more bandwidth measurement units contained in the session description were not understood.

304 Media type not available: One or more media types contained in the session description are not available.

305 Incompatible media format: One or more media formats contained in the session description available.

306 Attribute not understood: One or more of the media attributes in the session description are not supported.

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

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

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

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

399 Miscellaneous warning: The warning text can 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.

Additional "warn-code"s, as in the example below, can be defined through IANA.
Examples:

```
Warning: 307 isi.edu "Session parameter 'foo' not understood"
Warning: 301 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] for a definition of the syntax, and section 14 for an overview of usage.

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, [32]), Section 15. 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. A server SHOULD send a 1xx response if it expects to take more than 200 ms to obtain a final response. A server MAY 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.

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 might 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 Contact 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 Contact fields (Section 6.13). Unlike HTTP, the SIP response MAY contain several Contact fields or a list of addresses in a Contact field. User agents MAY use the Contact header 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 status response 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 Contact header field (Section 6.13). 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 Contact header field (Section 6.13). The duration of the redirection can be indicated through an Expires (Section 6.20) header.

7.3.4 305 Use Proxy

The requested resource MUST be accessed through the proxy given by the Contact field. The Contact field gives the URI of the proxy. The recipient is expected to repeat this single request via the proxy. 305 responses MUST only be generated by user agent servers.

7.3.5 380 Alternative Service

The call was not successful, but alternative services are possible. The alternative services are described in the message body of the response. Formats for such bodies are not defined here, and may be the subject of future standardization.

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 might 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 field 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 13.2 and 14.

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

7.4.10 409 Conflict

The request could not be completed due to a conflict with the current state of the resource. This response is returned if the action parameter in a REGISTER request conflicts with existing registrations.

7.4.11 410 Gone

The requested resource is no longer available at the server and no forwarding address is known. This condition is expected to be considered permanent. If the server does not know, or has no facility to determine, whether or not the condition is permanent, the status code 404 (Not Found) SHOULD be used instead.

7.4.12 411 Length Required

The server refuses to accept the request without a defined Content-Length. The client MAY repeat the request if it adds a valid Content-Length header field containing the length of the message-body in the request message.

7.4.13 413 Request Entity Too Large

The server is refusing to process a request because the request entity is larger than the server is willing or able to process. The server MAY close the connection to prevent the client from continuing the request.

If the condition is temporary, the server SHOULD include a Retry-After header field to indicate that it is temporary and after what time the client MAY try again.

7.4.14 414 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.

7.4.15 415 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. The server SHOULD return a list of acceptable formats using the Accept, Accept-Encoding and Accept-Language header fields.

7.4.16 420 Bad Extension

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

7.4.17 480 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 could 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. Status 486 (Busy Here) MAY be used to more precisely indicate a particular reason for the call failure.

This status is also returned by a redirect server that recognizes the user identified by the Request-URI, but does not currently have a valid forwarding location for that user.

7.4.18 481 Call Leg/Transaction Does Not Exist

This status is returned under two conditions: The server received a BYE request that does not match any existing call leg or the server received a CANCEL request that does not match any existing transaction. (A server simply discards an ACK referring to an unknown transaction.)

7.4.19 482 Loop Detected

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

7.4.20 483 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.

7.4.21 484 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.

7.4.22 485 Ambiguous

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

Revealing alternatives can 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
Contact: Carol Lee <sip:carol.lee@example.com>
Contact: Ping Lee <sip:p.lee@example.com>
Contact: Lee M. Foote <sip:lee.foote@example.com>
```

> 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.

7.4.23 486 Busy Here

The callee's end system was contacted successfully but the callee is currently not willing or able to take additional calls. The response MAY indicate a better time to call in the Retry-After header. The user could also be available elsewhere, such as through a voice mail service, thus, this response does not terminate any searches. Status 600 (Busy Everywhere) SHOULD be used if the client knows that no other end system will be able to accept this call.

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. The client MAY display the specific error condition, and MAY retry the request after several seconds.

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 Time-out

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 MAY contain an entity describing why that version is not supported and what other protocols are supported by that server. The format for such an entity is not defined here and may be the subject of future standardization.

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 Everywhere

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 uses status code 603 (Decline) instead. This status response is returned only if the client knows that no other end point (such as a voice mail system) will answer the request. Otherwise, 486 (Busy Here) should be returned.

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 field 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

Requests MAY contain message bodies unless otherwise noted. Within this specification, the BYE request MUST NOT contain a message body. 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, the request method and the response status code determine the type and interpretation of any message body. All responses MAY include a body. Message bodies for 1xx responses contain advisory information about the progress of the request. 2xx responses to INVITE requests contain session descriptions. In 3xx responses, the message body MAY contain the description of alternative destinations or services, as described in Section 7.3. For responses with status 400 or greater, the message 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 SHOULD be given by the Content-Length header field. Section 6.15 describes the behavior in detail.

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 address this problem, a more compact form of SIP is also defined by using abbreviations for the common header fields listed below:

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

Thus, the message in section 16.2 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
```

Clients MAY mix short field names and long field names within the same request. Servers MUST accept both short and long field names for requests. Proxies MAY change header fields between their long and short forms, but this MUST NOT be done to fields following an Authorization header.

10 Behavior of SIP Clients and Servers

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 discard isomorphic requests, but first 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 MAY send a CANCEL on all branches where it has not yet received a final response.

When a user agent receives a request, it checks the Call-ID against those of in-progress calls. 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 field 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 field 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 request contained only one Via header field.

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 SHOULD 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 associated 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 SHOULD NOT be 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 SHOULD maintain state for at least 32 seconds after the receipt of the first definitive non-200 response, in order to handle retransmissions of the response.

> The 32 second window is given by the maximum retransmission duration of 200-class responses using the default timers, 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

Responses are returned to the address listed in the Via header field (Section 6.40), *not* the source address of the request.

Recall that responses are not generated by the next-hop stateless server, but generated by either a proxy server or the user agent server. Thus, the stateless proxy can only use the Via header field to forward the response.

10.2.2 Multicast UDP

Requests MAY be multicast; multicast requests likely feature a host-independent Request-URI. This request SHOULD be scoped to ensure it is not forwarded beyond the boundaries of the administrative system. This MAY be done with either TTL or administrative scopes[25], depending on what is implemented in the 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. The server delays its response by a random interval uniformly distributed between zero and one second. Servers MAY 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. A proxy or UAC SHOULD send a CANCEL on receiving the first 2xx or 6xx response to a multicast request.

Server response suppression is a MAY since it requires a server to violate some basic message processing rules. Lets say A sends a multicast request, and it is received by B,C, and D. B sends a 200 response. The topmost Via field in the response will contain the address of A. C will also receive this response, and could use it to suppress its own response. However, C would normally not examine this response, as the topmost Via is not its own. Normally, a response received with an incorrect topmost Via MUST be dropped, but not in this case. To distinguish this packet from a misrouted or multicast looped packet is fairly complex, and for this reason the procedure is a MAY. The CANCEL, instead, provides a simpler and more standard way to perform response suppression. It is for this reason that the use of CANCEL here is a SHOULD

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 can be generated.)

The client SHOULD keep the connection open at least until the first final response arrives. If the client closes or resets the TCP connection prior to receiving the first final response, the server treats this action as equivalent to a CANCEL request.

This behavior makes it less likely that malfunctioning clients cause a proxy server to keep connection state indefinitely.

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 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 with an exponential backoff, starting at a $T1$ second interval, doubling the interval for each packet, and capping off at a $T2$ second interval. This means that after the first packet is sent, the second is sent $T1$ seconds later, the next $2 * T1$ seconds after that, the next $4 * T1$ seconds after that, and so on, until the interval hits $T2$. Subsequent retransmissions are spaced by T2 seconds. If the client receives a provisional response, it continues to retransmit the request, but with an interval of $T2$ seconds. Retransmissions cease when the client has sent a total of eleven packets, or receives a definitive response. Default values for $T1$ and $T2$ are 500ms and 4s, respectively. Clients MAY use larger values, but SHOULD NOT use smaller ones. Servers retransmit the response upon receipt of a request retransmission. 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 $10 * T2$ seconds to avoid having to, for example, contact the user or location server again upon receiving a request retransmission.

Use of the exponential backoff is for congestion control purposes. However, the back-off must cap off, since request retransmissions are used to trigger response retransmissions at the server. Without a cap, the loss of a single response could significantly increase transaction latencies.

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 500ms. For congestion control purposes, the retransmission count has to be bounded. Given that most transactions are expected to 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.

Each server in a proxy chain generates its own final response to a CANCEL request. The server responds immediately upon receipt of the CANCEL request rather than waiting until it has received final responses from the CANCEL requests it generates.

BYE and OPTIONS final responses are generated by redirect and user agent servers; REGISTER final responses are generated by registrars. Note that in contrast to the reliability mechanism described in Section 10.5, responses to these requests are *not* retransmitted periodically and *not* acknowledged via ACK.

10.4.2 TCP

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

10.5 Reliability for INVITE Requests

Special considerations apply for the INVITE method.

1. After receiving an invitation, considerable time can elapse before the server can determine the outcome. For example, if the called party is "rung" or extensive searches are performed, 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 could 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 retransmission intervals 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".

10.5.1 UDP

For UDP, A SIP client SHOULD retransmit a SIP INVITE request with an interval that starts at $T1$ seconds, and doubles after each packet transmission. The client ceases retransmissions if it receives a provisional or definitive response, or once it has sent a total of 7 request packets.

A server which transmits a provisional response should retransmit it upon reception of a duplicate request. A server which transmits a final response should retransmit it with an interval that starts at $T1$ seconds, and doubles for each subsequent packet. Response retransmissions cease when any one of the following occurs:

1. An ACK request for the same transaction is received;

2. a BYE request for the same call leg is received;

3. a CANCEL request for the same call leg is received **and** the final response status was equal or greater to 300;

4. the response has been transmitted 7 times.

Only the user agent client generates an ACK for 2xx final responses, If the response contained a Contact header field, the ACK MAY be sent to the address listed in that Contact header field. If the response did not contain a Contact 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 same server that the original request was sent to, using the same Request-URI as the original request. Note, however, that the To header field in the ACK is copied from the response being acknowledged, not the request, and thus MAY additionally contain the tag parameter. Also note than unlike 2xx final responses, a proxy generates an ACK for non-2xx final responses.

The ACK request MUST NOT be acknowledged to prevent a response-ACK feedback loop. Fig. 12 and 13 show the client and server state diagram for invitations.

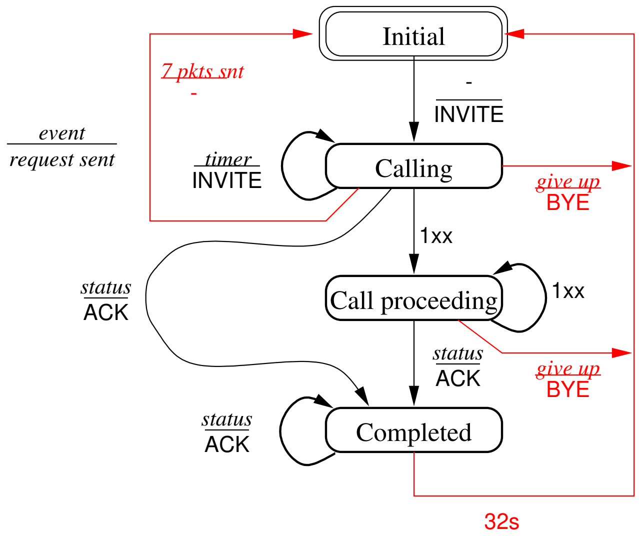

Figure 12: State transition diagram of client for INVITE method

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 with an exponential back-off helps ensure that the response is received, without placing an undue burden on the network.

10.5.2 TCP

A user agent using TCP MUST NOT retransmit requests, but uses the same algorithm as for UDP (Section 10.5.1) to retransmit responses until it receives an ACK.

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.

10.6 Reliability for ACK Requests

The ACK request does not generate responses. It is only generated when a response to an INVITE request arrives (see Section 10.5). This behavior is independent of the transport protocol. Note that the ACK request MAY take a different path than the original INVITE request, and MAY even cause a new TCP connection to be opened in order to send it.

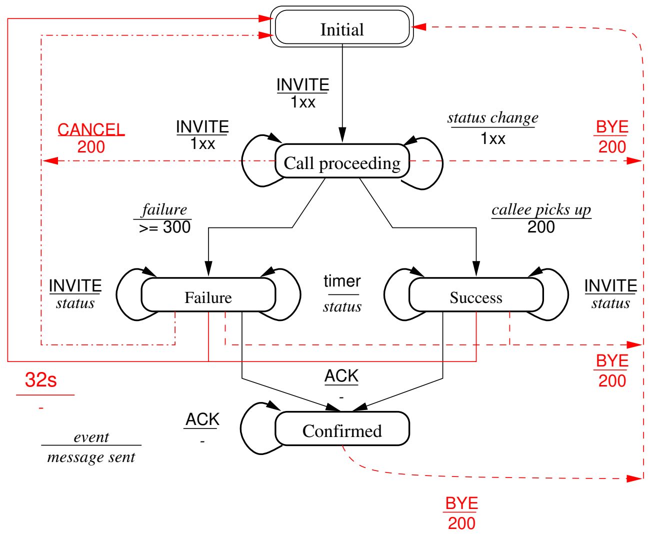

Figure 13: State transition diagram of server for INVITE method

10.7 ICMP Handling

Handling of ICMP messages in the case of UDP messages is straightforward. For requests, a host, network, port, or protocol unreachable error SHOULD be treated as if a 400-class response was received. For responses, these errors SHOULD cause the server to cease retransmitting the response.

Source quench ICMP messages SHOULD be ignored. TTL exceeded errors SHOULD be ignored. Parameter problem errors SHOULD be treated as if a 400-class response was received.

11 Behavior of SIP User Agents

This section describes the rules for user agent client and servers for generating and processing requests and responses.

11.1 Caller Issues Initial INVITE Request

When a user agent client desires to initiate a call, it formulates an INVITE request. The To field in the request contains the address of the callee. The Request-URI contains the same address. The From field contains the address of the caller. If the From address can appear in requests generated by other user agent clients for the same call, the caller MUST insert the tag parameter in the From field. A UAC MAY optionally

add a Contact header containing an address where it would like to be contacted for transactions from the callee back to the caller.

11.2 Callee Issues Response

When the initial INVITE request is received at the callee, the callee can accept, redirect, or reject the call. In all of these cases, it formulates a response. The response MUST copy the To, From, Call-ID, CSeq and Via fields from the request. Additionally, the responding UAS MUST add the tag parameter to the To field in the response if the request contained more than one Via header field. Since a request from a UAC may fork and arrive at multiple hosts, the tag parameter serves to distinguish, at the UAC, multiple responses from different UAS's. The UAS MAY add a Contact header field in the response. It contains an address where the callee would like to be contacted for subsequent transactions, including the ACK for the current INVITE. The UAS stores the values of the To and From field, including any tags. These become the local and remote addresses of the call leg, respectively.

11.3 Caller Receives Response to Initial Request

Multiple responses may arrive at the UAC for a single INVITE request, due to a forking proxy. Each response is distinguished by the "tag" parameter in the To header field, and each represents a distinct call leg. The caller MAY choose to acknowledge or terminate the call with each responding UAS. To acknowledge, it sends an ACK request, and to terminate it sends a BYE request. The To header field in the ACK or BYE MUST be the same as the To field in the 200 response, including any tag. The From header field MUST be the same as the From header field in the 200 (OK) response, including any tag. The Request-URI of the ACK or BYE request MAY be set to whatever address was found in the Contact header field in the 200 (OK) response, if present. Alternately, a UAC may copy the address from the To header field into the Request-URI. The UAC also notes the value of the To and From header fields in each response. For each call leg, the To header field becomes the remote address, and the From header field becomes the local address.

11.4 Caller or Callee Generate Subsequent Requests

Once the call has been established, either the caller or callee MAY generate INVITE or BYE requests to change or terminate the call. Regardless of whether the caller or callee is generating the new request, the header fields in the request are set as follows. For the desired call leg, the To header field is set to the remote address, and the From header field is set to the local address (both including any tags). The Contact header field MAY be different than the Contact header field sent in a previous response or request. The Request-URI MAY be set to the value of the Contact header field received in a previous request or response from the remote party, or to the value of the remote address.

11.5 Receiving Subsequent Requests

When a request is received subsequently, the following checks are made:

1. If the Call-ID is new, the request is for a new call, regardless of the values of the To and From header fields.

2. If the Call-ID exists, the request is for an existing call. If the To, From, Call-ID, and CSeq values exactly match (including tags) those of any requests received previously, the request is a retransmission.

3. If there was no match to the previous step, the To and From fields are compared against existing call leg local and remote addresses. If there is a match, and the CSeq in the request is higher than the last CSeq received on that leg, the request is a new transaction for an existing call leg.

12 Behavior of SIP Proxy and Redirect Servers

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

12.1 Redirect Server

A redirect server does not issue any SIP requests of its own. After receiving a request other than CANCEL, the server gathers the list of alternative locations and returns a final response of class 3xx or it refuses the request. For well-formed CANCEL requests, it SHOULD 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.

12.2 User Agent Server

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

12.3 Proxy Server

This section outlines processing rules for proxy servers. A proxy server can either be stateful or stateless. When stateful, a proxy remembers the incoming request which generated outgoing requests, and the outgoing requests. A stateless proxy forgets all information once an outgoing request is generated. A forking proxy SHOULD be stateful. Proxies that accept TCP connections MUST be stateful.

Otherwise, if the proxy were to lose a request, the TCP client would never retransmit it.

A stateful proxy SHOULD NOT become stateless until after it sends a definitive response upstream, and at least 32 seconds after it received a definitive response.

A stateful proxy acts as a virtual UAS/UAC. It implements the server state machine when receiving requests, and the client state machine for generating outgoing requests, with the exception of receiving a 2xx response to an INVITE. Instead of generating an ACK, the 2xx response is always forwarded upstream towards the caller. Furthermore, ACK's for 200 responses to INVITE's are always proxied downstream towards the UAS, as they would be for a stateless proxy.

A stateless proxy does not act as a virtual UAS/UAC (as this would require state). Rather, a stateless proxy forwards every request it receives downstream, and every response it receives upstream.

12.3.1 Proxying Requests

To prevent loops, a server MUST check if its own address is already contained in the Via header field of the incoming request.

The To, From, Call-ID, and Contact tags are copied exactly from the original request. The proxy SHOULD change the Request-URI to indicate the server where it intends to send the request.

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).

12.3.2 Proxying Responses

A proxy only processes a response if the topmost Via field matches one of its addresses. A response with a non-matching top Via field MUST be dropped.

12.3.3 Stateless Proxy: Proxying Responses

A stateless proxy removes its own Via field, and checks the address in the next Via field. In the case of UDP, the response is sent to the address listed in the "maddr" tag if present, otherwise to the "received" tag if present, and finally to the address in the "sent-by" field. A proxy MUST remain stateful when handling requests received via TCP.

A stateless proxy MUST NOT generate its own provisional responses.

12.3.4 Stateful Proxy: Receiving Requests

When a stateful proxy receives a request, it checks the To, From (including tags), Call-ID and CSeq against existing request records. If the tuple exists, the request is a retransmission. The provisional or final response sent previously is retransmitted, as per the server state machine. If the tuple does not exist, the request corresponds to a new transaction, and the request should be proxied.

A stateful proxy server MAY generate its own provisional (1xx) responses.

12.3.5 Stateful Proxy: Receiving ACKs

When an ACK request is received, it is either processed locally or proxied. To make this determination, the To, From, CSeq and Call-ID fields are compared against those in previous requests. If there is no match, the ACK request is proxied as if it were an INVITE request. If there is a match, and if the server had ever sent a 200 response upstream, the ACK is proxied. If the server had never sent any responses upstream, the ACK is also proxied. If the server had sent a 3xx, 4xx, 5xx or 6xx response, but no 2xx response, the ACK is processed locally if the tag in the To field of the ACK matches the tag sent by the proxy in the response.

12.3.6 Stateful Proxy: Receiving Responses

When a proxy server receives a response that has passed the Via checks, the proxy server checks the To (without the tag), From (including the tag), Call-ID and CSeq against values seen in previous requests. If there is no match, the response is forwarded upstream to the address listed in the Via field. If there is a match, the "branch" tag in the Via field is examined. If it matches a known branch identifier, the response is for the given branch, and processed by the virtual client for the given branch. Otherwise, the response is dropped.

A stateful proxy should obey the rules in Section 12.4 to determine if the response should be proxied upstream. If it is to be proxied, the same rules for stateless proxies above are followed, with the following addition for TCP. If a request was received via TCP (indicated by the protocol in the top Via header), the proxy checks to see if it has a connection currently open to that address. If so, the response is sent on that connection. Otherwise, a new TCP connection is opened to the address and port in the Via field, and the response is sent there. Note that this implies that a UAC or proxy MUST be prepared to receive responses on the incoming side of a TCP connection. Definitive non 200-class responses MUST be retransmitted by the proxy, even over a TCP connection.

12.3.7 Stateless, Non-Forking Proxy

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 12.4.

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.

12.4 Forking Proxy

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

Successful responses to an INVITE request MAY contain a Contact header field 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 C-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 can 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 {
    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: 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 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;
        }
      }
    }
    /* CANCEL: respond, fork and wait for responses */
    else if (class < 0) {
      best.status = 200;
      response(best);
      for (i = 0; i < N; i++) {
        if (!outgoing[i].done)
          request(CANCEL, address[i], outgoing[i].branch);
      }
      best.status = -1;
    }

    /* Send an ACK */
```

```
      if (class != 2) {
        if (R == INVITE) request(ACK, r.a, r.branch);
      }


      if (class == 2) {
        if (r.status < best.status) 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;
        }
      }
      else if (class == 4) {
        if (best.status >= 400) best = r;
      }
      else if (class == 5) {
        if (best.status >= 500) best = r;
      }
      else if (class == 6) {
        best = r;
        break;
      }
    }

    /* We haven't heard anything useful from anybody. */
    if (best.status == 1000) {
      best.status = 404;
    }
    if (best.status/100 != 3) loc = 0;
    response(best);
```



```
  }
```

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 MAY 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 could "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 Contact 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, responses can 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 MAY be terminated with CANCEL as described for 2xx responses.

A proxy server forwards any response for Call-IDs for which it does not have a pending transaction according to the response's Via header. User agent servers respond to BYE requests for unknown call legs with status code 481 (Transaction Does Not Exist); they drop ACK requests with unknown call legs silently.

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.

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.

13 Security Considerations

13.1 Confidentiality and Privacy: Encryption

13.1.1 End-to-End Encryption

SIP requests and responses can contain sensitive information about the communication patterns and communication content of individuals. 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 so that packet sniffers or other eavesdroppers cannot see who is calling whom. Hop-by-hop encryption can also encrypt requests and responses that have been end-to-end encrypted. Note that proxies can still see who is calling whom, and this information is also 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 can also provide useful information to an attacker. Section 6.22 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. All implementations SHOULD support PGP-based encryption [33] and MAY implement other schemes.

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 and 5 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 field 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 and Content-Type header fields will typically be encrypted. The Accept, Accept-Language, Date, Expires, Priority, Require, Call-ID, 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$
Call-ID: 187602141351@worcester.bell-telephone.com$
CSeq: 488$
$
```



```
*******************************************************
* 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$                 *
*************************************************
```

13.1.2 Privacy of SIP Responses

SIP requests can 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. Contact 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.

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

13.1.4 Hop-by-Hop Encryption

SIP requests and responses MAY also be protected by security mechanisms at the transport or network layer. No particular mechanism is defined or recommended here. Two possibilities are IPSEC [34] or TLS [35]. The use of a particular mechanism will generally need to be specified out of band, through manual configuration, for example.

13.1.5 Via field encryption

When Via header fields are to be hidden, a proxy that receives a request containing an appropriate "Hide: hop" header field (as specified in section 6.22) 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 header fields and the associated To header field, and replaces the Via header field with an index into the cache. On the reverse path, take the Via header 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.

  It is possible for replies to get directed to the wrong originator if the cache entry gets reused, so great care needs to 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 replay attacks and to ensure that no two requests from the same previous hop have the same encrypted Via field. This is the preferred solution.

13.2 Message Integrity and Access Control: Authentication

Protective measures need to be taken to prevent an active attacker from modifying and replaying SIP requests and responses. The same cryptographic measures that are used to ensure the authenticity of the SIP message also serve to authenticate the originator of the message. However, the "basic" and "digest" authentication mechanism offer authentication only, without message integrity.

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 field and their Proxy- counterparts to include cryptographically strong signatures. SIP also supports the HTTP "basic" and "digest" schemes (see Section 14) and other HTTP authentication schemes to be defined that offer a 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 is used in requests to authenticate the request originator end-to-end to proxies and the called user agent, and in responses to authenticate the called user agent or proxies returning their own failure codes. If required, hop-by-hop authentication can be provided, for example, by 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 15. As indicated above, SIP implementations MAY also use "basic" and "digest" authentication and other authentication mechanisms defined for HTTP. Note that "basic" authentication has severe security limitations. The following does not apply to these schemes.

To cryptographically sign a SIP request, the order of the SIP header fields is important. When an Authorization header field is present, it indicates that all header fields following the Authorization header field have been included in the signature. Therefore, hop-by-hop header fields which MUST or SHOULD be modified by proxies MUST precede the Authorization header field as they will generally be modified or added-to by proxy servers. Hop-by-hop header fields which MAY be modified by a proxy MAY appear before or after the Authorization header. When they appear before, they MAY be modified by a proxy. When they appear after, they MUST NOT be modified by a proxy. To sign a request, a client constructs a message from the request method (in upper case) followed, without LWS, by the SIP version number, followed, again without LWS, by the request headers to be signed and the message body. The message thus constructed is then 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
```

Clients wishing to authenticate requests MUST construct the portion of the message below the Authorization header using a canonical form. This allows a proxy to parse the message, take it apart, and reconstruct it, without causing an authentication failure due to extra white space, for example. Canonical form consists of the following rules:

- No short form header fields
- Header field names are capitalized as shown in this document
- No white space between the header name and the colon
- A single space after the colon
- Line termination with a CRLF
- No line folding
- No comma separated lists of header values; each must appear as a separate header
- Only a single SP between tokens, between tokens and quoted strings, and between quoted strings; no SP after last token or quoted string
- No LWS between tokens and separators, except as described above for after the colon in header fields

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 13.2.1.

13.2.1 Trusting responses

There is the possibility that an eavesdropper listens to requests and then injects unauthenticated responses that terminate, redirect or otherwise interfere with a call. (Even encrypted requests contain enough information to fake a response.)

Clients need to 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 Contact 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. 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, fake 6xx responses from a rogue proxy 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).

13.3 Callee Privacy

User location and SIP-initiated calls can 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.

13.4 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.

14 SIP Authentication using HTTP Basic and Digest Schemes

SIP implementations MAY use HTTP’s basic and digest authentication mechanisms to provide a rudimentary form of security. This section overviews usage of these mechanisms in SIP. The basic operation is almost completely identical to that for HTTP [36]. This section outlines this operation, pointing to [36] for details, and noting the differences when used in SIP.

14.1 Framework

The framework for SIP authentication parallels that for HTTP [36]. In particular, the BNF for auth-scheme, auth-param, challenge, realm, realm-value, and credentials is identical. The 401 response is used by user agent servers in SIP to challenge the authorization of a user agent client. Additionally, registrars and redirect servers MAY make use of 401 responses for authorization, but proxies MUST NOT, and instead MAY use the 407 response. The requirements for inclusion of the Proxy-Authenticate, Proxy-Authorization, WWW-Authenticate, and Authorization in the various messages is identical to [36].

Since SIP does not have the concept of a canonical root URL, the notion of protections spaces are interpreted differently for SIP. The realm is a protection domain for all SIP URIs with the same value for the userinfo, host and port part of the SIP Request-URI. For example:

```
INVITE sip:alice.wonderland@example.com SIP/2.0
WWW-Authenticate:  Basic realm="business"
```

and

```
INVITE sip:aw@example.com SIP/2.0
WWW-Authenticate: Basic realm="business"
```

define different protection realms according to this rule.

When a UAC resubmits a request with its credentials after receiving a 401 or 407 response, it MUST increment the CSeq header field as it would normally do when sending an updated request.

14.2 Basic Authentication

The rules for basic authentication follow those defined in [36, Section 2], but with the words "origin server" replaced with "user agent server, redirect server , or registrar".

Since SIP URIs are not hierarchical, the paragraph in [36, Section 2] that states that "all paths at or deeper than the depth of the last symbolic element in the path field of the Request-URI also are within the protection space specified by the Basic realm value of the current challenge" does not apply for SIP. SIP clients MAY preemptively send the corresponding Authorization header with requests for SIP URIs within the same protection realm (as defined above) without receipt of another challenge from the server.

14.3 Digest Authentication

The rules for digest authentication follow those defined in [36, Section 3], with "HTTP 1.1" replaced by "SIP/2.0" in addition to the following differences:

1. The URI included in the challenge has the following BNF:

   URI = SIP-URL

2. The BNF for digest-uri-value is:

   digest-uri-value = Request-URI ;as defined in Section 4.3

3. The example procedure for choosing a nonce based on Etag does not work for SIP.
4. The Authentication-Info and Proxy-Authentication-Info fields are not used in SIP.
5. The text in [36] regarding cache operation does not apply to SIP.
6. [36] requires that a server check that the URI in the request line, and the URI included in the Authorization header, point to the same resource. In a SIP context, these two URI's may actually refer to different users, due to forwarding at some proxy. Therefore, in SIP, a server MAY check that the request-uri in the Authorization header corresponds to a user that the server is willing to accept forwarded or direct calls for.

14.4 Proxy-Authentication

The use of the Proxy-Authentication and Proxy-Authorization parallel that as described in [36, Section 3.6], with one difference. Proxies MUST NOT add the Proxy-Authorization header. 407 responses MUST be forwarded upstream towards the client following the procedures for any other response. It is the client's responsibility to add the Proxy-Authorization header containing credentials for the proxy which has asked for authentication.

> If a proxy were to resubmit a request with a Proxy-Authorization header field, it would need to increment the CSeq in the new request. However, this would mean that the UAC which submitted the original request would discard a response from the UAS, as the CSeq value would be different.

See sections 6.26 and 6.27 for additional information on usage of these fields as they apply to SIP.

15 SIP Security Using PGP

15.1 PGP Authentication Scheme

The "pgp" authentication scheme is based on the model that the client authenticates 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.

15.1.1 The WWW-Authenticate Response Header

```
WWW-Authenticate  =  "WWW-Authenticate" ":" "pgp" pgp-challenge
pgp-challenge     =  * (";" pgp-params )
pgp-params        =  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 MAY additionally indicate the collection of users who might have access. An example might be "`Users with call-out privileges`".

pgp-algorithm: The value of this parameter indicates 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"
```

15.1.2 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    =  realm | pgp-version | pgp-signature | signed-by
pgp-signature   =  "signature" "=" quoted-string
signed-by       =  "signed-by" "=" <"> URI <">
```

The client MUST increment the CSeq header before resubmitting the request.

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 [33], 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, 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
  VAUGU4rhEBW972IPxFSOZ94L1qhCLInTPaqhHFw1cb31B01rA0RhpV4t5yCdUt
  SRYBSkOK29o5e1KlFeW23EzYPVUm2TlDAhbcjbMdfC+KLFX
  =aIrx"
```

15.2 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"
```

15.3 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 from the pgp key ring with the "pgp -kxa *user*".

Example:

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

16 Examples

In the following examples, we often omit the message body and the corresponding Content-Length and Content-Type headers for brevity.

16.1 Registration

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

```
C->S: REGISTER sip:bell-tel.com SIP/2.0
      Via: SIP/2.0/UDP saturn.bell-tel.com
      From: sip:watson@bell-tel.com
      To: sip:watson@bell-tel.com
      Call-ID: 70710@saturn.bell-tel.com
      CSeq: 1 REGISTER
      Contact: <sip:watson@saturn.bell-tel.com:3890;transport=udp>
      Expires: 7200
```

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 saturn.bell-tel.com
      From: sip:watson@bell-tel.com
      To: sip:watson@bell-tel.com
      Call-ID: 70710@saturn.bell-tel.com
      CSeq: 2 REGISTER
      Contact: *
      Expires: 0

C->S: REGISTER sip:bell-tel.com SIP/2.0
      Via: SIP/2.0/UDP saturn.bell-tel.com
      From: sip:watson@bell-tel.com
      To: sip:watson@bell-tel.com
      Call-ID: 70710@saturn.bell-tel.com
      CSeq: 3 REGISTER
      Contact: 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. For the server at example.com to reach Watson, he will need to send a REGISTER there, or inform the server of his current location through some other means.

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

```
C->S: REGISTER sip:bell-tel.com SIP/2.0
```



```
      Via: SIP/2.0/UDP pluto.bell-tel.com
      From: sip:jon.diligent@bell-tel.com
      To: sip:watson@bell-tel.com
      Call-ID: 17320@pluto.bell-tel.com
      CSeq: 1 REGISTER
      Contact: sip:tawatson@example.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.

16.2 Invitation to a 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 [6]) as the session description format.

16.2.1 Request

```
C->S: INVITE sip:schooler@cs.caltech.edu SIP/2.0
      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
      From: Mark Handley <sip:mjh@isi.edu>
      To: Eve Schooler <sip:schooler@caltech.edu>
      Call-ID: 2963313058@north.east.isi.edu
      CSeq: 1 INVITE
      Subject: SIP will be discussed, too
      Content-Type: application/sdp
      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 From request header above states that the request was initiated by mjh@isi.edu and addressed to schooler@caltech.edu (From header fields). The Via fields list the hosts along the path from invitation initiator (the last element of the list) towards the callee. 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 csvax.cs.caltech.edu. The second Via header field indicates that it was originally sent from the host north.east.isi.edu. The Request-URI indicates that the request is currently being being addressed to schooler@cs.caltech.edu, the local address that csvax looked up for the callee.

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.

16.2.2 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
      From: Mark Handley <sip:mjh@isi.edu>
      To: Eve Schooler <sip:schooler@caltech.edu> ;tag=9883472
      Call-ID: 2963313058@north.east.isi.edu
      CSeq: 1 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 Contact 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;ttl=16
      Via: SIP/2.0/UDP north.east.isi.edu
      From: Mark Handley <sip:mjh@isi.edu>
      To: Eve Schooler <sip:schooler@caltech.edu> ;tag=9883472
      Call-ID: 2963313058@north.east.isi.edu
      CSeq: 1 INVITE
      Contact: sip:es@jove.cs.caltech.edu
```

The caller confirms the invitation by sending an ACK request to the location named in the Contact header:

```
C->S: ACK sip:es@jove.cs.caltech.edu SIP/2.0
      Via: SIP/2.0/UDP north.east.isi.edu
      From: Mark Handley <sip:mjh@isi.edu>
      To: Eve Schooler <sip:schooler@caltech.edu> ;tag=9883472
```



```
      Call-ID: 2963313058@north.east.isi.edu
      CSeq: 1 ACK
```

16.3 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 kton.bell-tel.com
      From: A. Bell <sip:a.g.bell@bell-tel.com>
      To: T. Watson <sip:watson@bell-tel.com>
      Call-ID: 3298420296@kton.bell-tel.com
      CSeq: 1 INVITE
      Subject: Mr. Watson, come here.
      Content-Type: application/sdp
      Content-Length: ...

      v=0
      o=bell 53655765 2353687637 IN IP4 128.3.4.5
      s=Mr. Watson, come here.
      c=IN IP4 kton.bell-tel.com
      m=audio 3456 RTP/AVP 0 3 4 5

S->C: SIP/2.0 100 Trying
      Via: SIP/2.0/UDP kton.bell-tel.com
      From: A. Bell <sip:a.g.bell@bell-tel.com>
      To: T. Watson <sip:watson@bell-tel.com> ;tag=37462311
      Call-ID: 3298420296@kton.bell-tel.com
      CSeq: 1 INVITE
      Content-Length: 0

S->C: SIP/2.0 180 Ringing
      Via: SIP/2.0/UDP kton.bell-tel.com
      From: A. Bell <sip:a.g.bell@bell-tel.com>
      To: T. Watson <sip:watson@bell-tel.com> ;tag=37462311
      Call-ID: 3298420296@kton.bell-tel.com
      CSeq: 1 INVITE
      Content-Length: 0

S->C: SIP/2.0 182 Queued, 2 callers ahead
      Via: SIP/2.0/UDP kton.bell-tel.com
      From: A. Bell <sip:a.g.bell@bell-tel.com>
      To: T. Watson <sip:watson@bell-tel.com> ;tag=37462311
```

```
      Call-ID: 3298420296@kton.bell-tel.com
      CSeq: 1 INVITE
      Content-Length: 0

S->C: SIP/2.0 182 Queued, 1 caller ahead
      Via: SIP/2.0/UDP kton.bell-tel.com
      From: A. Bell <sip:a.g.bell@bell-tel.com>
      To: T. Watson <sip:watson@bell-tel.com> ;tag=37462311
      Call-ID: 3298420296@kton.bell-tel.com
      CSeq: 1 INVITE
      Content-Length: 0

S->C: SIP/2.0 200 OK
      Via: SIP/2.0/UDP kton.bell-tel.com
      From: A. Bell <sip:a.g.bell@bell-tel.com>
      To: <sip:watson@bell-tel.com> ;tag=37462311
      Call-ID: 3298420296@kton.bell-tel.com
      CSeq: 1 INVITE
      Contact: sip:watson@boston.bell-tel.com
      Content-Length: ...

      v=0
      o=watson 4858949 4858949 IN IP4 192.1.2.3
      s=I'm on my way
      c=IN IP4 boston.bell-tel.com
      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 c.bell-tel.com, Bell will send to port 5004 at boston.bell-tel.com.

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 agreed on the set of media, Bell 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 kton.bell-tel.com
      From: A. Bell <sip:a.g.bell@bell-tel.com>
      To: T. Watson <sip:watson@bell-tel.com> ;tag=37462311
      Call-ID: 3298420296@kton.bell-tel.com
      CSeq: 1 ACK
```


16.4 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
      Via: SIP/2.0/UDP kton.bell-tel.com
      From: A. Bell <sip:a.g.bell@bell-tel.com>
      To: T. A. Watson <sip:watson@bell-tel.com> ;tag=37462311
      Call-ID: 3298420296@kton.bell-tel.com
      CSeq: 2 BYE
```

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

16.5 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, `t.watson@x.bell-tel.com` (X) and `watson@y.bell-tel.com` (Y), and has registered with the IEEE proxy server (P) called `sip.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 and Via header fields.

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

```
C->P: INVITE sip:t.watson@ieee.org 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: 31415@c.bell-tel.com
      CSeq:    1 INVITE
```

The SIP server at `ieee.org` 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 sip.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: 31415@c.bell-tel.com
      CSeq:    1 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 sip.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>;tag=87454273
      Call-ID: 31415@c.bell-tel.com
      CSeq:    1 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 sip.ieee.org ;branch=1
      From:    A. Bell <sip:a.g.bell@bell-tel.com>
      To:      T. Watson <sip:t.watson@ieee.org>;tag=87454273
      Call-ID: 31415@c.bell-tel.com
      CSeq:    1 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 sip.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: 31415@c.bell-tel.com
      CSeq:    1 INVITE
```

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

```
P->X: INVITE sip:t.watson@x.bell-tel.com SIP/2.0
      Via:     SIP/2.0/UDP sip.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: 31415@c.bell-tel.com
      CSeq:    1 INVITE

P->Y: INVITE sip:watson@y.bell-tel.com SIP/2.0
      Via:     SIP/2.0/UDP sip.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: 31415@c.bell-tel.com
      CSeq:    1 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.

```
X->P: SIP/2.0 200 OK
      Via:      SIP/2.0/UDP sip.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> ;tag=192137601
      Call-ID:  31415@c.bell-tel.com
      CSeq:     1 INVITE
      Contact:  sip:t.watson@x.bell-tel.com

Y->P: SIP/2.0 200 OK
      Via:      SIP/2.0/UDP sip.ieee.org ;branch=4
      Via:      SIP/2.0/UDP c.bell-tel.com
      Contact:  sip:t.watson@y.bell-tel.com
      From:     A. Bell <sip:a.g.bell@bell-tel.com>
      To:       T. Watson <sip:t.watson@ieee.org> ;tag=35253448
      Call-ID:  31415@c.bell-tel.com
      CSeq:     1 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 sip.ieee.org ;branch=2
      From:    A. Bell <sip:a.g.bell@bell-tel.com>
      To:      T. Watson <sip:t.watson@ieee.org>
      Call-ID: 31415@c.bell-tel.com
      CSeq:    1 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 sip.ieee.org ;branch=2
      From:    A. Bell <sip:a.g.bell@bell-tel.com>
      To:      T. Watson <sip:t.watson@ieee.org>
      Call-ID: 31415@c.bell-tel.com
      CSeq:    1 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:t.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>;tag=192137601
      Call-ID:  31415@c.bell-tel.com
      CSeq:     1 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>;tag=35253448
      Call-ID:  31415@c.bell-tel.com
      CSeq:     1 ACK
```

After a brief discussion between Bell with X and Y, it becomes clear that Watson is at X. (Note that this is not a three-way call; only Bell can talk to X and Y, but X and Y cannot talk to each other.) 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>;tag=35253448
      Call-ID:  31415@c.bell-tel.com
      CSeq:     2 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>;tag=35253448
      Call-ID:  31415@c.bell-tel.com
      CSeq:     2 BYE
```

16.6 Redirects

Replies with status codes 301 (Moved Permanently) or 302 (Moved Temporarily) specify another location using the Contact field. Continuing our earlier example, the server P at `ieee.org` decides to redirect rather than proxy the request:

```
P->C: SIP/2.0 302 Moved temporarily
      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>;tag=72538263
      Call-ID: 31415@c.bell-tel.com
      CSeq:    1 INVITE
      Contact: sip:watson@h.bell-tel.com,
```



```
               sip:watson@acm.org, sip:t.watson@x.bell-tel.com,
               sip:watson@y.bell-tel.com
      CSeq: 1 INVITE
```

As another example, assume Alice (A) wants to delegate her calls to Bob (B) while she is on vacation until July 29th, 1998. Any calls meant for her will reach Bob with Alice's To field, indicating to him what role he is to play. Charlie (C) calls Alice (A), whose server returns:

```
A->C: SIP/2.0 302 Moved temporarily
      From: Charlie <sip:charlie@caller.com>
      To: Alice <sip:alice@anywhere.com> ;tag=2332462
      Call-ID: 27182@caller.com
      Contact: sip:bob@anywhere.com
      Expires: Wed, 29 Jul 1998 9:00:00 GMT
      CSeq: 1 INVITE
```

Charlie then sends the following request to the SIP server of the `anywhere.com` domain. Note that the server at `anywhere.com` forwards the request to Bob based on the Request-URI.

```
C->B: INVITE sip:bob@anywhere.com SIP/2.0
      From: sip:charlie@caller.com
      To: sip:alice@anywhere.com
      Call-ID: 27182@caller.com
      CSeq: 2 INVITE
```

In the third redirection example, we assume that all outgoing requests are directed through a local firewall F at `caller.com`, with Charlie again inviting Alice:

```
C->F: INVITE sip:alice@anywhere.com SIP/2.0
      From: sip:charlie@caller.com
      To: Alice <sip:alice@anywhere.com>
      Call-ID: 27182@caller.com
      CSeq: 1 INVITE
```

The local firewall at `caller.com` happens to be overloaded and thus redirects the call from Charlie to a secondary server S:

```
F->C: SIP/2.0 302 Moved temporarily
      From: sip:charlie@caller.com
      To: Alice <sip:alice@anywhere.com>
      Call-ID: 27182@caller.com
      CSeq: 1 INVITE
      Contact: <sip:alice@anywhere.com:5080;maddr=spare.caller.com>
```

Based on this response, Charlie directs the same invitation to the secondary server `spare.caller.com` at port 5080, but maintains the same Request-URI as before:

```
C->S: INVITE sip:alice@anywhere.com SIP/2.0
      From: sip:charlie@caller.com
      To: Alice <sip:alice@anywhere.com>
      Call-ID: 27182@caller.com
      CSeq: 2 INVITE
```

16.7 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> ;tag=7434264
      Call-ID: 14142@north.east.isi.edu
      CSeq: 1 INVITE
      Contact: sip:mjh@north.east.isi.edu
      Warning: 370 "Insufficient bandwidth (only have ISDN)",
        305 "Incompatible media format",
        330 "Multicast not available"
      Content-Type: application/sdp
      Content-Length: 50

      v=0
      s=Let's talk
      b=CT:128
      c=IN IP4 north.east.isi.edu
      m=audio 3456 RTP/AVP 5 0 7
      m=video 2232 RTP/AVP 31
```

In this example, the original request specified a bandwidth that was higher than the access link could support, requested multicast, and requested a set of media encodings. The response states that only 128 kb/s is available and that (only) 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.

16.8 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 encodings PCM Ulaw (payload type 0), 1016 (payload type 1), GSM (payload type 3), and SX7300/8000 (dynamic payload type 99), and video encodings H.261 (payload type 31) and H.263 (payload type 34).


```
C->S: OPTIONS sip:bob@example.com SIP/2.0
      From: Alice <sip:alice@anywhere.org>
      To: Bob <sip:bob@example.com>
      Call-ID: 6378@host.anywhere.org
      CSeq: 1 OPTIONS
      Accept: application/sdp

S->C: SIP/2.0 200 OK
      From: Alice <sip:alice@anywhere.org>
      To: Bob <sip:bob@example.com> ;tag=376364382
      Call-ID: 6378@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 31 34
      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, [6]). 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. In general, each capability listed below builds on the ones above it:

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 Contact header, but only the SIP-URL part, not the parameters.

Firewall-friendly: A firewall-friendly client understands the Route and Record-Route header fields and can be configured to use a local proxy for all outgoing requests.

Negotiation: A client MUST be able to request the OPTIONS method and understand the 380 (Alternative Service) status and the Contact 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 6 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).

The fields in the table have the following meaning. Type is as in Table 4 and 5. "-" 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 the Session Description Protocol (SDP)

This section describes the use of the Session Description Protocol (SDP) (RFC 2327 [6]).

B.1 Configuring Media Streams

The caller and callee align their media descriptions so that the nth media stream ("m=" line) in the caller's session description corresponds to the nth media stream in the callee's description.

All media descriptions SHOULD contain "a=rtpmap" mappings from RTP payload types to encodings.

This allows easier migration away from static payload types.

If the callee wants to neither send nor receive a stream offered by the caller, the callee sets the port number of that stream to zero in its media description.

There currently is no other way than port zero for the callee to refuse a bidirectional stream offered by the caller. Both caller and callee need to be aware what media tools are to be started.

For example, assume that the caller Alice has included the following description in her INVITE request. It includes an audio stream and two bidirectional video streams, using H.261 (payload type 31) and MPEG (payload type 32).

```
v=0
o=alice 2890844526 2890844526 IN IP4 host.anywhere.com
c=IN IP4 host.anywhere.com
```


	type	UAC	proxy	UAS	registrar
Accept	R	-	o	m	m
Accept-Encoding	R	-	-	m	m
Accept-Language	R	-	b	b	b
Allow	405	o	-	-	-
Authorization	R	a	o	a	a
Call-ID	g	m	m	m	m
Content-Encoding	g	m	-	m	m
Content-Length	g	m	m	m	m
Content-Type	g	m	-	m	m
CSeq	g	m	m	m	m
Encryption	g	e	-	e	e
Expires	g	-	o	o	m
From	g	m	o	m	m
Hide	R	-	m	-	-
Contact	R	-	-	-	m
Contact	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
Route	R	-	m	-	-
Timestamp	g	o	o	m	m
To	g	m	m	m	m
Unsupported	r	b	b	-	-
User-Agent	g	b	-	b	-
Via	g	m	m	m	m
WWW-Authenticate	401	a	-	-	-

Table 6: Header Field Processing Requirements

```
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000
m=video 51372 RTP/AVP 31
a=rtpmap:31 H261/90000
m=video 53000 RTP/AVP 32
a=rtpmap:32 MPV/90000
```

The callee, Bob, does not want to receive or send the first video stream, so it returns the media description below:

```
v=0
```

```
o=bob 2890844730 2890844730 IN IP4 host.example.com
c=IN IP4 host.example.com
m=audio 47920 RTP/AVP 0 1
a=rtpmap:0 PCMU/8000
a=rtpmap:1 1016/8000
m=video 0 RTP/AVP 31
m=video 53000 RTP/AVP 32
a=rtpmap:32 MPV/90000
```

B.2 Setting SDP Values for Unicast

If a session description from a caller contains a media stream which is listed as send (receive) only, it means that the caller is only willing to send (receive) this stream, not receive (send). The same is true for the callee.

For receive-only and send-or-receive streams, the port number and address in the session description indicate where the media stream should be sent to by the recipient of the session description, either caller or callee. For send-only streams, the address and port number have no significance and SHOULD be set to zero.

The list of payload types for each media stream conveys two pieces of information, namely the set of codecs that the caller or callee is capable of sending or receiving, and the RTP payload type numbers used to identify those codecs. For receive-only or send-and-receive media streams, a caller SHOULD list all of the codecs it is capable of supporting in the session description in an INVITE or ACK. For send-only streams, the caller SHOULD indicate only those it wishes to send for this session. For receive-only streams, the payload type numbers indicate the value of the payload type field in RTP packets the caller is expecting to receive for that codec type. For send-only streams, the payload type numbers indicate the value of the payload type field in RTP packets the caller is planning to send for that codec type. For send-and-receive streams, the payload type numbers indicate the value of the payload type field the caller expects to both send and receive.

If a media stream is listed as receive-only by the caller, the callee lists, in the response, those codecs it intends to use from among the ones listed in the request. If a media stream is listed as send-only by the caller, the callee lists, in the response, those codecs it is willing to receive among the ones listed in the the request. If the media stream is listed as both send and receive, the callee lists those codecs it is capable of sending or receiving among the ones listed by the caller in the INVITE. The actual payload type numbers in the callee's session description corresponding to a particular codec MUST be the same as the caller's session description.

If caller and callee have no media formats in common for a particular stream, the callee MUST return a session description containing the particular "m=" line, but with the port number set to zero, and no payload types listed.

If there are no media formats in common for all streams, the callee SHOULD return a 400 response, with a 304 Warning header field.

B.3 Multicast Operation

The interpretation of send-only and receive-only for multicast media sessions differs from that for unicast sessions. For multicast, send-only means that the *recipient* of the session description (caller or callee) SHOULD only send media streams to the address and port indicated. Receive-only means that the recipient of the session description SHOULD only receive media on the address and port indicated.

For multicast, receive and send multicast addresses are the same and all parties use the same port numbers to receive media data. If the session description provided by the caller is acceptable to the callee, the callee can choose not to include a session description or MAY echo the description in the response.

A callee MAY, in the response, return a session description with some of the payload types removed, or port numbers set to zero (but no other value). This indicates to the caller that the callee does not support the given stream or media types which were removed. A callee MUST NOT change whether a given stream is send-only, receive-only, or send-and-receive.

If a callee does not support multicast at all, it SHOULD return a 400 status response and include a 330 Warning.

B.4 Delayed Media Streams

In some cases, a caller may not know the set of media formats which it can support at the time it would like to issue an invitation. This is the case when the caller is actually a gateway to another protocol which performs media format negotiation after call setup. When this occurs, a caller MAY issue an INVITE with a session description that contains no media lines. The callee SHOULD interpret this to mean that the caller wishes to participate in a multimedia session described by the session description, but that the media streams are not yet known. The callee SHOULD return a session description indicating the streams and media formats it is willing to support, however. The caller MAY update the session description either in the ACK request or in a re-INVITE at a later time, once the streams are known.

B.5 Putting Media Streams on Hold

If a party in a call wants to put the other party "on hold", i.e., request that it temporarily stops sending one or more media streams, a party re-invites the other by sending an INVITE request with a modified session description. The session description is the same as in the original invitation (or response), but the "c" destination addresses for the media streams to be put on hold are set to zero (0.0.0.0).

B.6 Subject and SDP "s=" Line

The SDP "s=" line and the SIP Subject header field have different meanings when inviting to a multicast session. The session description line describes the subject of the multicast session, while the SIP Subject header field describes the reason for the invitation. The example in Section 16.2 illustrates this point. For invitations to two-party sessions, the SDP "s=" line MAY be left empty.

B.7 The SDP "o=" Line

The "o=" line is not strictly necessary for two-party sessions, but MUST be present to allow re-use of SDP-based tools.

C Summary of Augmented BNF

All of the mechanisms specified in this document are described in both prose and an augmented Backus-Naur Form (BNF) similar to that used by RFC 822 [9]. Implementors will need to be familiar with the notation in order to understand this specification. The augmented BNF includes the following constructs:

name = definition

The name of a rule is simply the name itself (without any enclosing "¡" and "¿") and is separated from its definition by the equal "=" character. White space is only significant in that indentation of continuation lines is used to indicate a rule definition that spans more than one line. Certain basic rules are in uppercase, such as SP, LWS, HT, CRLF, DIGIT, ALPHA, etc. Angle brackets are used within definitions whenever their presence will facilitate discerning the use of rule names.

```
"literal"
```

Quotation marks surround literal text. Unless stated otherwise, the text is case-insensitive.

```
rule1 | rule2
```

Elements separated by a bar ("—") are alternatives, e.g., "yes — no" will accept yes or no.

```
(rule1 rule2)
```

Elements enclosed in parentheses are treated as a single element. Thus, "(elem (foo — bar) elem)" allows the token sequences "elem foo elem" and "elem bar elem".

```
*rule
```

The character "*" preceding an element indicates repetition. The full form is "¡n¿*¡m¿element" indicating at least ¡n¿ and at most ¡m¿ occurrences of element. Default values are 0 and infinity so that "*(element)" allows any number, including zero; "1*element" requires at least one; and "1*2element" allows one or two.

```
[rule]
```

Square brackets enclose optional elements; "[foo bar]" is equivalent to "*1(foo bar)".

```
N rule
```

Specific repetition: "¡n¿(element)" is equivalent to "¡n¿*¡n¿(element)"; that is, exactly ¡n¿ occurrences of (element). Thus 2DIGIT is a 2-digit number, and 3ALPHA is a string of three alphabetic characters.

```
#rule
```

A construct "#" is defined, similar to "*", for defining lists of elements. The full form is "$< n >$#$< m >$ element" indicating at least $< n >$ and at most $< m >$ elements, each separated by one or more commas (",") and OPTIONAL linear white space (LWS). This makes the usual form of lists very easy; a rule such as

(*LWS element *(*LWS "," *LWS element))

can be shown as 1# element. Wherever this construct is used, null elements are allowed, but do not contribute to the count of elements present. That is, "(element), , (element)" is permitted, but counts as only two elements. Therefore, where at least one element is required, at least one non-null element MUST be present. Default values are 0 and infinity so that "#element" allows any number, including zero; "1#element" requires at least one; and "1#2element" allows one or two.

```
; comment
```

A semi-colon, set off some distance to the right of rule text, starts a comment that continues to the end of line. This is a simple way of including useful notes in parallel with the specifications.

```
implied *LWS
```

The grammar described by this specification is word-based. Except where noted otherwise, linear white space (LWS) can be included between any two adjacent words (token or quoted-string), and between adjacent tokens and separators, without changing the interpretation of a field. At least one delimiter (LWS and/or separators) MUST exist between any two tokens (for the definition of "token" below), since they would otherwise be interpreted as a single token.

C.1 Basic Rules

The following rules are used throughout this specification to describe basic parsing constructs. The US-ASCII coded character set is defined by ANSI X3.4-1986.

```
OCTET     =  <any 8-bit sequence of data>
CHAR      =  <any US-ASCII character (octets 0 - 127)>
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"
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"
alpha     =  lowalpha | upalpha
digit     =  "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
             "8" | "9"
alphanum  =  alpha | digit
CTL       =  <any US-ASCII control character
             (octets 0 – 31) and DEL (127)>
CR        =  %d13 ; US-ASCII CR, carriage return character
LF        =  %d10 ; US-ASCII LF, line feed character
SP        =  %d32 ; US-ASCII SP, space character
HT        =  %d09 ; US-ASCII HT, horizontal tab character
CRLF      =  CR LF ; typically the end of a line
```

The following are defined in RFC 2396 [12] for the SIP URI:

```
unreserved   =   alphanum | mark
mark         =   "-" | "_" | "." | "!" | "~" | "*" | "'"
             |   "(" | ")"
escaped      =   "%" hex hex
```

SIP header field values can be folded onto multiple lines if the continuation line begins with a space or horizontal tab. All linear white space, including folding, has the same semantics as SP. A recipient MAY replace any linear white space with a single SP before interpreting the field value or forwarding the message downstream.

```
LWS   =   [CRLF] 1*( SP | HT ) ; linear whitespace
```

The TEXT-UTF8 rule is only used for descriptive field contents and values that are not intended to be interpreted by the message parser. Words of *TEXT-UTF8 contain characters from the UTF-8 character set (RFC 2279 [21]). In this regard, SIP differs from HTTP, which uses the ISO 8859-1 character set.

```
TEXT-UTF8   =   <any UTF-8 character encoding,except CTLs,
                but including LWS>
```

A CRLF is allowed in the definition of TEXT-UTF8 only as part of a header field continuation. It is expected that the folding LWS will be replaced with a single SP before interpretation of the TEXT-UTF8 value.

Hexadecimal numeric characters are used in several protocol elements.

```
hex   =   "A" | "B" | "C" | "D" | "E" | "F"
          | "a" | "b" | "c" | "d" | "e" | "f" | digit
```

Many SIP header field values consist of words separated by LWS or special characters. These special characters MUST be in a quoted string to be used within a parameter value.

```
token        =   1*<any CHAR except CTL's or separators>
separators   =   "(" | ")" | "<" | ">" | "@" |
                 "," | ";" | ":" | "\" | <"> |
                 "/" | "[" | "]" | "?" | "=" |
                 "{" | "}" | SP | HT
```

Comments can be included in some SIP header fields by surrounding the comment text with parentheses. Comments are only allowed in fields containing "comment" as part of their field value definition. In all other fields, parentheses are considered part of the field value.

```
comment   =   "(" *(ctext | quoted-pair | comment) ")"
ctext     =   <any TEXT-UTF8 excluding "(" and ")">
```

A string of text is parsed as a single word if it is quoted using double-quote marks.

```
quoted-string   =   ( ¡"¿ *(qdtext | quoted-pair ) ¡"¿ )
qdtext          =   ¡any TEXT-UTF8 except ¡"¿¿
```

The backslash character ("\") MAY be used as a single-character quoting mechanism only within quoted-string and comment constructs.

```
quoted-pair   =   " \ " CHAR
```


D Using SRV DNS Records

The following procedure is experimental and relies on DNS SRV records (RFC 2052 [14]).

To determine the IP address of a SIP server to deliver a request to, the following steps are followed. The Request-URI is examined. If it does not specify a protocol (TCP or UDP), the client queries the name server for SRV records for both UDP (if supported by the client) and TCP (if supported by the client) SIP servers. The format of these queries is defined in RFC 2052 [14]. Otherwise, the client queries the name server for SIP servers available through the protocol specified in the Request-URI. If the client does not support the protocol specified in the Request-URI, it gives up. The results of the query or queries are merged together and ordered based on priority. Then, the searching technique outlined in RFC 2052 [14] is used to select servers in order. The user then attempts to contact each server in the order listed. If no port number is specified in the Request-URI, the client uses the port number returned in the DNS response, not port 5060.

E 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.

F Changes in Version -11

Since version -10, the following changes have been made. These changes reflect IESG comments.

- Respecting DNS TTL's for cacheing of queries changed from SHOULD to MUST.
- Default MTU for UDP requests changed from 1400 to 1500.
- Added note to status 480 indicating that a redirect server could use this as well to indicate that it doesn't know where the user is, for example, because all of the user's contact information has expired.
- Added description for status 305. (May need to consider whether we need both Location and Contact here, for compatibility with HTTP.)
- Use of CRLF for line termination is a SHOULD.
- Clarifications and minor changes in DNS procedures for finding a SIP server.
- Message retransmission timers exponentially back off (incorporated since -09).
- BNF fixed up, changed TEXT BNF to allow for UTF-8.
- More explicit rules on using HTTP basic and digest were added. Proxy-to-proxy authentication is disallowed due to incompatibilities with UDP transport.
- IANA registered options are prefixed by org.iana.sip instead of org.ietf.sip.
- Date, Expires field only supports RFC 1123 formatting, as opposed to HTTP, which also supports RFC 850 and asctime date formats.

- Changed the default reason phrase for 413 to align with the newest HTTP revision.
- Added requirement for CRLF line termination in canonicalization rules. Clarified canonicalization rules
- Default language for warn-text changed from English to i-default.
- Use of transport and network level security changed to MAY from RECOMMENDED.
- Mandated use of the name-addr form in the To field if addr-spec contains a semicolon, comma, or question mark. This is in line with From and Contact, but was omitted from the To section accidentally in previous versions
- Fixed deviation in BNF between ASCII and PostScript versions of spec.
- Made consistent the usage of the Contact expires parameter.
- Fixed inconsistency in usage of Contact header in 181 and 1xx responses.
- Fixed inconsistency regarding encryption of Call-ID. Call-ID should be in the clear.
- Rewrote SDP section for greater clarity; included discussion of behavior for delayed session description.
- Updated minimal requirements table, correcting inconsistencies with the text.

G Acknowledgments

We wish to thank the members of the IETF MMUSIC WG for their comments and suggestions. Detailed comments were provided by Anders Kristensen, Jim Buller, Dave Devanathan, Yaron Goland, Christian Huitema, Gadi Karmi, Jonathan Lennox, Keith Moore, Vern Paxson, Moshe J. Sambol, and Eric Tremblay.

This work is based, inter alia, on [37, 38].

H 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, "SAP: Session announcement protocol," Internet Draft, Internet Engineering Task Force, Nov. 1996. Work in progress.

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

[7] International Telecommunication Union, "Visual telephone systems and equipment for local area networks which provide a non-guaranteed quality of service," Recommendation H.323, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, May 1996.

[8] International Telecommunication Union, "Control protocol for multimedia communication," Recommendation H.245, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Feb. 1998.

[9] International Telecommunication Union, "Media stream packetization and synchronization on non-guaranteed quality of service LANs," Recommendation H.225.0, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Nov. 1996.

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

[11] 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.

[12] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform resource identifiers (URI): generic syntax," RFC 2396, Internet Engineering Task Force, Aug. 1998.

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

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

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

[16] M. Hamilton and R. Wright, "Use of DNS aliases for network services," RFC 2219, Internet Engineering Task Force, Oct. 1997.

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

[18] 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.

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

[20] 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.

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

[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] D. Meyer, "Administratively scoped IP multicast," RFC 2365, Internet Engineering Task Force, July 1998.

[26] H. Schulzrinne, "RTP profile for audio and video conferences with minimal control," RFC 1890, Internet Engineering Task Force, Jan. 1996.

[27] D. Eastlake, S. Crocker, and J. Schiller, "Randomness recommendations for security," RFC 1750, Internet Engineering Task Force, Dec. 1994.

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

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

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

[31] H. Alvestrand, "IETF policy on character sets and languages," RFC 2277, Internet Engineering Task Force, Jan. 1998.

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

[33] D. Atkins, W. Stallings, and P. Zimmermann, "PGP message exchange formats," RFC 1991, Internet Engineering Task Force, Aug. 1996.

[34] R. Atkinson, "Security architecture for the internet protocol," RFC 1825, Internet Engineering Task Force, Aug. 1995.

[35] C. Allen and T. Dierks, "The TLS protocol version 1.0," Internet Draft, Internet Engineering Task Force, Nov. 1997. Work in progress.

[36] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart, "HTTP authentication: Basic and digest access authentication," Internet Draft, Internet Engineering Task Force, Sept. 1998. Work in progress.

[37] 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.

[38] 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