Internet Engineering Task Force
INTERNET-DRAFT
draft-ietf-sip-rfc2543bis-04.ps

SIP WG
Handley/Schulzrinne/Schooler/Rosenberg
ACIRI/Columbia U./Caltech/dynamicsoft

July 20, 2001
Expires: February 2002

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.

Contents

1 Introduction

1.1 Overview of SIP Functionality

The Session Initiation Protocol (SIP) is an application-layer control protocol that can establish, modify and terminate multimedia sessions (conferences) or Internet telephony calls. SIP can invite participants to 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 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 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 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 and does not reserve network resources.

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 [7] and indicate requirement levels for compliant SIP implementations.

1.3 Overview of SIP Operation

This section explains the basic protocol functionality and operation. Terms are defined more precisely in Section 1.4. In SIP, protocol participants are identified by SIP URLs, described in Section 1.4.1. SIP is a request-response protocol, with requests sent by clients and received by servers. A single implementation typically combines both client and server functionality. SIP requests can be sent using any reliable or unreliable protocol, including UDP, SCTP [8] and TCP. Protocol operation is largely independent of the lower-layer transport protocol.

This specification defines six SIP request methods: INVITE (Section 5.1) initiates sessions, ACK (Section 5.1.1) confirms session establishment, OPTIONS (Section 8) requests information about capabilities, BYE (Section 6) terminates a sessions, CANCEL (Section 5.2) cancels a pending session and REGISTER

(Section 7) allows a client to bind a permanent SIP URL to a temporary SIP URL reflecting the current network location.

SIP requests and responses consists of a request (or status) line, a number of header lines and a message body (Section 3).

SIP requests can be sent directly from a user agent client to a user agent server, or they can traverse one or more proxy servers along the way. Often, proxy servers are associated with DNS domains, similar to SMTP MTAs.

User agents send requests either directly to the address indicated in the SIP URI or to a designated proxy ("outbound proxy"), independent of the destination address. The current destination address is carried in the Request-URI. Each proxy can forward the request based on local policy and information contained in the SIP request. The proxy MAY rewrite the request URI. A proxy MAY also forward the request to another designated proxy regardless of the request URI. For example, a departmental proxy could forward all authorized requests to a corporate-wide proxy which then forwards it to the proxy operated by the Internet service provider, which finally routes the request based on the request URI.

Proxies MAY modify any part of the SIP message that are not integrity-protected, except those needed to identify call legs. Proxies generally do not modify the session description, but MAY do so.

For example, if the user agent wants to contact the user `sip:alice@example.com`, it sends the request to the server handling the `example.com` domain (Section 1.4.2). If that host acts as a proxy server, it looks up whether it has a mapping from `alice@example.com` to another address. If so, it substitutes that address, say `alice@sales.example.com`, into the Request-URI and then sends the request to the server for the `sales.example.com` domain. Any server can also return a response indicating a different destination to be tried by the upstream client or indicating that the request cannot be forwarded.

Typically, only the first request within a call traverses all proxies, while subsequent requests are exchanged directly between user agents. However, a proxy can indicate that it wants to remain in the request path via a Record-Route (Section 10.34) header field.

1.4 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 2616 [9]). The terms and generic syntax of URI and URL are defined in RFC 2396 [10]. The following terms have special significance for SIP.

Back-To-Back User Agent: Also known as a B2BUA, this is a logical entity that receives an invitation, and acts as a UAS to process it. In order to determine how the request should be answered, it acts as a UAC and initiates a call outwards. Unlike a proxy server, it maintains complete call state and must participate in all requests for a call. Since it is purely a concatenation of other logical functions, no explicit definitions are needed for its behavior.

Call: A call consists of all participants in a session invited by a common source. A SIP call is identified by a globally unique call-id (Section 10.12), and is created when a user agent sends an INVITE request. This INVITE request may generate multiple acceptances, each of which are part of the same call (but different call legs). Furthermore, if a user is invited to the same multicast session by several people, each of these invitations will be a unique 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 a pairwise signaling relationship between two SIP user agents. A call leg is established when a call invitation results in a successful response. It is identified by the combination of the Call-ID header field, the local address of the participant, and the remote address of the other participant. For the caller, the local address is the From field of the INVITE, and the remote address is the To field of the 200 class response. For the callee, the local address is the To field of the 200 class response to the INVITE, and the remote address is the From field of the INVITE. SIP URIs are compared according to Section 2.1, non-SIP URIs according to Section 2.2. Within the same Call-ID, requests with From A and To value B belong to the same call leg as the requests in the opposite direction, i.e., From B and To A.

Call Stateful: A proxy is said to be *call stateful* when it retains state that persists for the duration of a call initiated through it. To properly manage that state, the proxy will normally need to receive the BYE requests that terminate the call.

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 session invitation. Note that the calling party does not have to be the same as the one creating the conference. A caller retains this role for the duration of a call.

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. A callee retains this role for the duration of a call.

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, isomorphic requests have to have the same Request-URI and the same top-most Via header.

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). Examples of sources of location information include SIP registrars, databases or mobility registration protocols. Location services are offered by location servers. Location servers MAY be part of a SIP server, but the manner in which a SIP server requests location services is beyond the scope of this document.

Outbound proxy: A *proxy* that is located near the originator of requests. It receives all outgoing requests **from** a particular UAC, including those requests whose Request-URLs identify a host other than the outbound proxy. The outbound proxy sends these requests, after any local processing, to the address indicated in the Request-URI. (All other proxy servers are simply referred as proxies, not *inbound* proxies.)

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.

Proxy servers are, for example, used to route requests, enforce policies, control firewalls.

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 make its information available through the location server.

Regular Transaction: A regular transaction is any transaction with a method other than INVITE, ACK, or CANCEL.

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 10.20) 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.

Spiral: A spiral is a SIP request which is routed to a proxy, forwarded onwards, and arrives once again at that proxy, but this time, with a Request-URI that differs from the previous arrival. A spiral is not an error condition, unlike a loop.

Stateless Proxy: A logical entity that does not maintain state for a SIP transaction. A stateless proxy forwards every request it receives downstream and every response it receives upstream.

Stateful Proxy: A logical entity that maintains state information for the duration of a SIP transaction. Also known as a transaction stateful proxy. The behavior of a stateful proxy is further defined in Section 17.3. A stateful proxy is not the same as a call stateful proxy.

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 [11].

User agent client (UAC): A user agent client is a logical entity that initiates a SIP transaction with a request. This role lasts only for the duration of that transaction. In other words, if a piece of software initiates a request, it acts as a UAC for the duration of that request. If it receives a request later on, it takes on the role of a User Agent Server for the processing of that transaction.

User agent server (UAS): A user agent server is a logical entity that responds to a SIP request, generally acting on behalf of some user. The response accepts, rejects or redirects the request. This role lasts only for the duration of that transaction. In other words, if a piece of software responds to a request, it acts as a UAS for the duration of that request. If it generates a request later on, it takes on the role of a User Agent Client for the processing of that transaction.

User agent (UA): A logical entity which acts as both a user agent client and user agent server for the duration of a call.

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 role of UAC and UAS as well as proxy and redirect servers are defined on a request-by-request basis. For example, the user agent initiating a call acts as a UAC when sending the initial INVITE request and as a UAS when receiving a BYE request from the callee. Similarly, the same software can act as a proxy server for one request and as a redirect server for the next request.

Proxy, redirect, location and registrar servers defined above are *logical* entities; implementations MAY combine them into a single application program. 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
inserts Via header	no	yes	no	no
accepts ACK	yes	yes	yes	no

Table 1: Properties of the different SIP server types

1.4.1 SIP Addressing

The "objects" addressed by SIP are users at hosts, identified by a SIP URL. The SIP URL takes 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 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

The Request-URI is determined according to the rules in Section 16 and can be derived from either the Route, Contact or To header fields.

When a client wishes to send a request, the client either sends it to a locally configured SIP proxy server, the so-called *outbound proxy*, independent of the Request-URI, or sends it to the IP address and port corresponding to the Request-URI. The outbound proxy can be configured by any mechanism, including DHCP [12] and can be specified either as a set of parameters such as network address or host name, protocol port and transport protocol, or as a SIP URI.

If the Request-URI is used, the client needs to determine the protocol, port and IP address of a server to which to send the request. A client SHOULD follow the steps below to obtain this information.

Clients MUST re-run the above selection algorithm, re-drawing any random numbers involved, once per transaction rather than for each request, i.e., requests within the same transaction MUST be sent to the same network address. Thus, the same address is used for the request, any retransmissions, any associated CANCEL requests and ACK requests for non-2xx responses. However, ACKs for 2xx responses use another iteration of the selection algorithm. (Indeed, in many cases, they may have different request URIs.)

A stateless proxy can accomplish this, for example, by using the modulo N of a hash of the Call-ID value or some other combination of transaction-identifying headers as the uniform random number described in the weighting algorithm of RFC 2782. Here, N is the sum of weights within the priority class.

A client SHOULD be able to interpret explicit network notifications (such as ICMP messages) which indicate that a server is not reachable, rather than relying solely on timeouts. (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()` or `send()` 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.

The client tries to find one or more addresses for the SIP server by querying DNS. If a step elicits no addresses, the client continues 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 does not continue on to the next step.

If the client is configured with the address of an outbound proxy, the parameters of the outbound proxy, including transport protocol and port, become the *destination* used below.

If there is no outbound proxy, the destination is the Request-URI. The destination address is the maddr parameter if it exists and the host element if not. The transport protocol is the transport parameter.

The service identifier for DNS SRV records [13] is "_sip".

1. If the destination address is a numeric IP address, the client contacts the server at the given address and the port number specified in the SIP-URI or, if not specified, the default port (5060).

   If the destination specifies a protocol, the client contacts the server using that protocol. If no protocol is specified, the client first tries UDP. If attempt fails, or if the client does not support UDP but supports other protocols, it tries those protocols in some implementation-defined order.

   The client then skips the remaining steps.

2. If the destination specifies no port number or port number 5060, the transport protocol determines the use of one of the following three rules:

   - If the destination does not specify a transport protocol, DNS SRV records are retrieved according to RFC 2782 [13]. The results of the query or queries are merged and ordered based on priority, keeping only records with transport protocols that the client supports. Then, the searching technique outlined in RFC 2782 [13] is used to select servers in order. Server selection across requests is independent of previous choices, except as noted above for stateless proxies. Message length or other request properties do not influence the server selection. The client attempts to contact each server in the order listed, at the port number specified in the SRV record. If none of the servers can be contacted, the client gives up. If there are no SRV records (with any transport protocol), DNS address records are used, as described below.
   - If a transport protocol is specified and this protocol is supported by the client, the procedure in the paragraph above is used, limited to DNS resource records with the transport protocol specified in the SIP-URI.
   - If the transport protocol specified is not supported by the client, the client gives up.

   If there are no SRV records, the next step applies.

3. If the destination specifies a port number other than 5060 or if there are no SRV records, the client queries the DNS server for address records for the destination address. Address records include A RR's, AAAA RR's, or other similar records, chosen according to the client's network protocol capabilities.

   If the DNS server returns no address records, the client gives up. If there are address records, the same rules as in step 2 apply.

Clients MUST NOT cache query results except according to the rules in RFC 1035 [14].

The results of the DNS lookup operation do not, in general, lead to a modification of the Request-URI.

A proxy is free to modify the Request-URI to any value desired, but the DNS lookups are usually based on the Request-URI obtained from a location server.

If the DNS time-to-live value exceeds a few minutes, servers generating a large number of requests are probably well advised to retry failed servers every few minutes.

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 10.43)). This allows responses to be matched with requests. The ACK request confirming the receipt of an INVITE response is *not* part of the transaction since it may traverse a different set of hosts.

If a reliable stream protocol is used, request and responses within a single SIP transaction are carried over the same connection (see Section 14). Several SIP requests from the same client to the same server MAY use the same connection or MAY use a new connection for each request.

If a client sends the request via a unicast datagram protocol such as UDP, the receiving user agent directs the response according to the information contained in the Via header fields (Section 10.46). Each proxy server in the forward path of the request forwards the response using these Via header fields, as described in detail in Sections 10.46.3 and 10.46.4. For datagram protocols, reliability is achieved using retransmission (Section 14).

1.4.4 Initiating a Session

A session is initiated with the INVITE request. A successful SIP invitation consists of two requests, INVITE followed by ACK. The INVITE (Section 5.1) 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 5.1.1) request.

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 use 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 use. 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.

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 alterna-

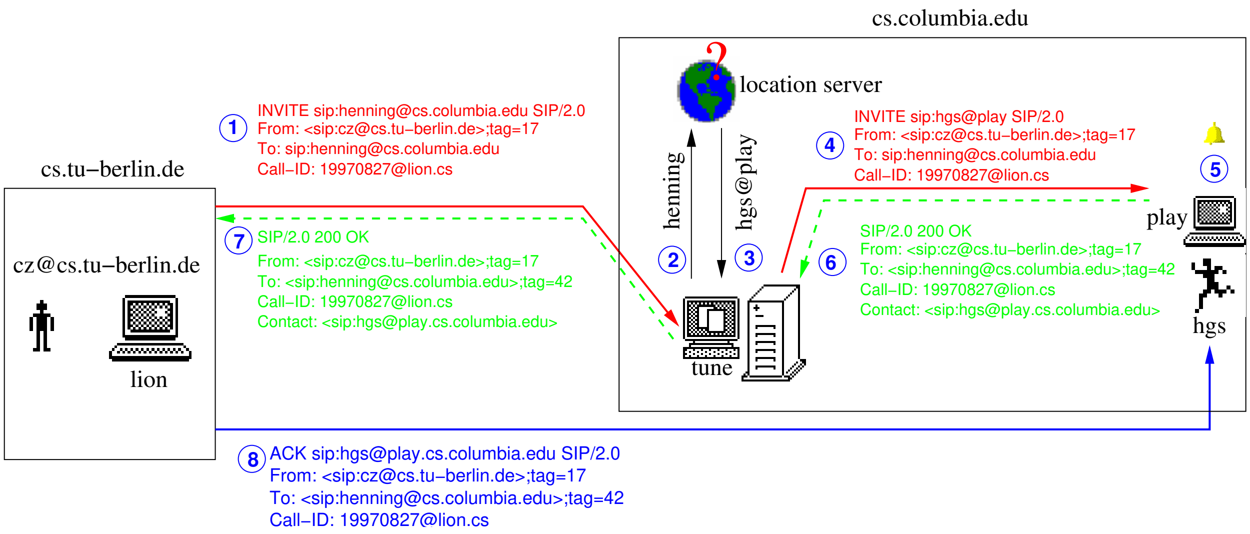

Figure 1: Example of SIP proxy server

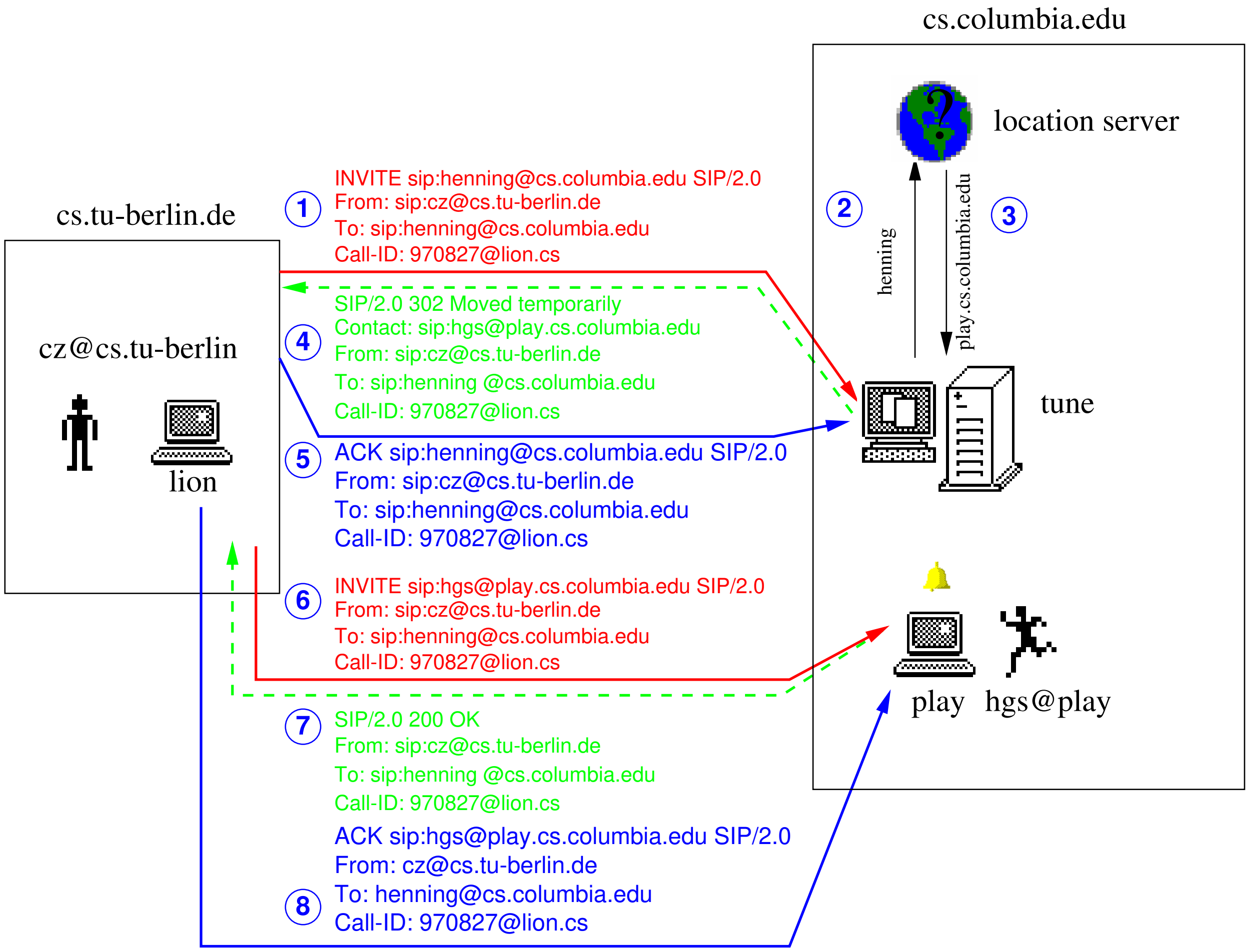

Figure 2: Example of SIP redirect server

tive.

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, 7). A location server MAY also use one or more other protocols, such as finger (RFC 1288 [15]), rwhois (RFC 2167 [16]), LDAP (RFC 1777 [17]), multicast-based protocols [18] 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 10.14). 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 beginning of the list of forwarders noted in the Via (Section 10.46) 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 SIP invitation may traverse more than one SIP proxy server. If one of these "forks" the request, i.e., issues more than one request in response to receiving the invitation request, it is possible that a client is reached, independently, by more than one copy of the invitation request. Each of these copies bears the same Call-ID, but a different topmost Via header branch parameter (see Section 10.46). The user agent MAY choose which final response to return for each such request, typically returning a 200 (OK) for only one of them.

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 within the same call leg, but within 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 17. 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. All entities MUST support UDP. User agents SHOULD implement TCP transport. Stateful proxy, registrar, and redirect servers MUST implement TCP transport. Other transports MAY be supported by any entity.

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 [10] and has the syntax shown in Fig. 3. The syntax is described using Augmented Backus-Naur Form (see Section C). Although the ABNF described in Section C uses implicit whitespace, unescaped whitespace MUST NOT be present within a SIP URL. Any 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" [10]. Excluded US-ASCII characters [10, Sec. 2.4.3], such as space and control characters and characters used as URL delimiters, also MUST be escaped. URLs MUST NOT contain unescaped space and control characters.

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

The components of the SIP URI have the following meanings.

user: The name of the user addressed. Note that this field MAY be empty where the destination host does not have a notion of users, e.g., for embedded devices.

telephone-subscriber: If the host is an Internet telephony gateway, a telephone-subscriber field MAY be used instead of a user field. The telephone-subscriber field uses the notation of RFC 2806 [19]. Any characters of the un-escaped "telephone-subscriber" that are not either in the set "unreserved" or "user-unreserved" MUST be escaped. The set of characters not reserved in the RFC 2806 description of telephone-subscriber contains a number of characters in various syntax elements that need to be escaped when used in SIP URLs, for example quotation marks (%22), hash (%23), colon (%3a), at-sign (%40) and the "unwise" characters, i.e., punctuation of %5b and above.

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 user parameter value "phone" indicates that the user part contains a telephone number. Even without this parameter, recipients of SIP URLs MAY interpret the pre-@ part as a telephone number if local restrictions on the name space for user name allow it.

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 host part SHOULD be a fully-qualified domain name or numeric IP address.

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.

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 to be used for sending SIP requests and responses. SIP can use any network transport protocol; parameter names are defined for UDP [20], TCP [21], TLS [22], and SCTP. UDP is to be assumed when no explicit transport parameter is included. The maddr parameter indicates the server address to be contacted for this user, overriding the address supplied in the host field. This address is typically, but not necessarily, a multicast address.

The maddr field can be used to force requests from traveling users to visit a home proxy even if an outbound proxy is used.

```
SIP-URL             =  "sip:" [ userinfo "@" ] hostport
                       url-parameters [ headers ]
userinfo            =  [ user | telephone-subscriber [ ":" password ]]
user                =  *( unreserved | escaped | user-unreserved )
user-unreserved     =  "&" | "=" | "+" | "$" | "," | ";" | "?" | "/"
password            =  *( unreserved | escaped |
                       "&" | "=" | "+" | "$" | "," )
hostport            =  host [ ":" port ]
host                =  hostname | IPv4address | IPv6reference
hostname            =  *( domainlabel "." ) toplabel [ "." ]
domainlabel         =  alphanum
                    |  alphanum *( alphanum | "-" ) alphanum
toplabel            =  alpha | alpha *( alphanum | "-" ) alphanum
IPv4address         =  1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
IPv6reference       =  "[" IPv6address "]"
IPv6address         =  hexpart [ ":" IPv4address ]
hexpart             =  hexseq | hexseq "::" [ hexseq ] | "::" [ hexseq ]
hexseq              =  hex4 *( ":" hex4)
hex4                =  1*4HEX
port                =  1*DIGIT
url-parameters      =  *( ";" url-parameter)
url-parameter       =  transport-param | user-param | method-param
                    |  ttl-param | maddr-param | other-param
transport-param     =  "transport="
                       ( "udp" | "tcp" | "sctp" | "tls" | other-transport)
other-transport     =  token
user-param          =  "user=" ( "phone" | "ip" | other-user)
other-user          =  token
method-param        =  "method=" Method
ttl-param           =  "ttl=" ttl
maddr-param         =  "maddr=" host
other-param         =  pname [ "=" pvalue ]
pname               =  1*paramchar
pvalue              =  1*paramchar
paramchar           =  param-unreserved | unreserved | escaped
param-unreserved    =  "[" | "]" | "/" | ":" | "&" | "+" | "$"
headers             =  "?" header *( "&" header )
header              =  hname "=" hvalue
hname               =  1*( hnv-unreserved | unreserved | escaped )
hvalue              =  *( hnv-unreserved | unreserved | escaped )
hnv-unreserved      =  "[" | "]" | "/" | "?" | ":" | "+" | "$"
```

Figure 3: SIP URL syntax

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; they are ignored if present.

For Request-URIs, these parameters are useful primarily for outbound proxies.

Receivers MUST silently ignore any URI parameters that they do not understand.

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. Entries marked "m" are mandatory, those marked "o" are optional, and those marked "-" are not allowed. For optional elements, the second column indicates the default value if the element is not present.

	default	Req.-URI	To	From	Contact	Rec.-Route	external
user	–	o	o	o	o	o	o
password	–	o	o	-	o	o	o
host	mandatory	m	m	m	m	m	m
port	5060	o	o	o	o	o	o
user-param	ip	o	o	o	o	o	o
method	INVITE	-	-	-	o	-	o
maddr-param	–	o	-	-	o	m	o
ttl-param	1	o	-	-	o	-	o
transp.-param	udp	o	-	-	o	-	o
other-param	–	o	o	o	o	o	o
headers	–	-	-	-	o	-	o

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
```

2.1 SIP URL Comparison

SIP URLs are compared for equality according to the following rules:

- Comparisons of scheme name ("sip"), domain names, parameter names and header names are case-insensitive, all other comparisons are case-sensitive.
- The ordering of parameters and headers is not significant in comparing SIP URLs.
- user or telephone-subscriber, password, host, port and any url-parameter parameters of the URI must match. If one of the components in the previous sentence is omitted, it matches based on its default value. (For example, otherwise equivalent URLs without a port specification and with port 5060 match.) URL parameters (which excludes telephone-subscriber, password, and port) not found in both URLs being compared, for which there is no default value, are ignored, and therefore not included in the comparison operation. Other URL components not found in both URLs being compared, for which there is no default value, are included in the comparison operation, and the result will be that the URLs do not match.
- Characters other than those in the "reserved" and "unsafe" sets (see RFC 2396 [10]) are equivalent to their ""%" HEX HEX" encoding.
- An IP address that is the result of a DNS lookup of a host name does **not** match that host name.
- URL parameters that have no default value are compared only if they are present in both URLs.

Thus, the following URLs are equivalent:

```
sip:juser@%65xample.com:5060
sip:juser@ExAmPlE.CoM;Transport=udp
```

while

```
SIP:JUSER@ExAmPlE.CoM;Transport=udp
sip:juser@ExAmPlE.CoM;Transport=UDP
```

are not.

The following URLs are also equivalent:

```
sip:user@domain.com;foo=baz
sip:user@domain.com
```

while

```
sip:user@domain.com
sip:domain.com
```

are not.

Header fields such as Contact, From and To are equal if and only if their URIs match under the rules above and if their header parameters (such as contact-param, from-param and to-param) match in name and parameter value, where parameter names and token parameter values are compared ignoring case and quoted-string parameter values are case-sensitive.

2.2 Non-SIP URLs

SIP header fields and the Request-URI MAY contain non-SIP URLs, with the exceptions noted below. As an example, if a call from a telephone is relayed to the Internet via SIP, the SIP From header field might contain a tel: URL [19].

In the following locations, only SIP URLs are allowed:

- Request-URI in a REGISTER request;
- Contact header field in INVITE and and 2xx responses to INVITE.

Implementations MAY compare non-SIP URLs by treating them as generic URIs [10] or, alternatively, compare them byte-by-byte.

3 SIP Message Overview

SIP is a text-based protocol and uses the ISO 10646 character set in UTF-8 encoding (RFC 2279 [23]). Senders MUST terminate lines with a CRLF, but receivers MUST also interpret CR and LF by themselves as line terminators. Only the combinations CR CR, LF LF and CRLF CRLF terminate the message header. Implementations MUST only send CRLF CRLF.

CR and LF instead of CRLF is for backwards-compatibility; their use is deprecated.

Except for the above difference in character sets and line termination, 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 2616 [9]). 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 or other unreliable datagram protocols. Each such datagram carries one request or response. 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. However, implementations MUST be able to handle messages up to the maximum datagram packet size. For UDP, this size is 65,535 bytes, including headers.

The MTU of 1500 bytes accommodates encapsulation within the “typical” ethernet MTU without IP fragmentation. Recent studies [24, 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 [25]). 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 9) messages use the generic-message format of RFC 822 [26] 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 9.1

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 10.6
                 |  Accept-Encoding        ; Section 10.7
                 |  Accept-Language        ; Section 10.8
                 |  Call-ID                ; Section 10.12
                 |  Call-Info              ; Section 10.13
                 |  Contact                ; Section 10.14
                 |  CSeq                   ; Section 10.20
                 |  Date                   ; Section 10.21
                 |  Encryption             ; Section 10.22
                 |  From                   ; Section 10.25
                 |  MIME-Version           ; Section 10.28
                 |  Organization           ; Section 10.29
                 |  Record-Route           ; Section 10.34
                 |  Require                ; Section 10.35
                 |  Supported              ; Section 10.41
                 |  Timestamp              ; Section 10.42
                 |  To                     ; Section 10.43
                 |  User-Agent             ; Section 10.45
                 |  Via                    ; Section 10.46
entity-header    =  Allow                  ; Section 10.10
                 |  Content-Disposition    ; Section 10.15
                 |  Content-Encoding       ; Section 10.16
                 |  Content-Language       ; Section 10.17
                 |  Content-Length         ; Section 10.18
                 |  Content-Type           ; Section 10.19
                 |  Expires                ; Section 10.24
request-header   =  Alert-Info             ; Section 10.9
                 |  Authorization          ; Section 10.11
                 |  In-Reply-To            ; Section 10.26
                 |  Max-Forwards           ; Section 10.27
                 |  Priority               ; Section 10.30
                 |  Proxy-Authorization    ; Section 10.32
                 |  Proxy-Require          ; Section 10.33
                 |  Route                  ; Section 10.38
                 |  Response-Key           ; Section 10.36
                 |  Subject                ; Section 10.40
response-header  =  Error-Info             ; Section 10.23
                 |  Proxy-Authenticate     ; Section 10.31
                 |  Retry-After            ; Section 10.37
                 |  Server                 ; Section 10.39
                 |  Unsupported            ; Section 10.44
                 |  Warning                ; Section 10.47
                 |  WWW-Authenticate       ; Section 10.48
```

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 12
```

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. No LWS is allowed in any of the elements. The Request-URI MUST NOT be enclosed in "<>". absoluteURI is defined in [H3.2.1].

```
Request-Line  =  Method SP Request-URI SP SIP-Version CRLF
Request-URI   =  SIP-URL | absoluteURI
SIP-Version   =  "SIP/2.0"
```

4.2 Methods

The methods are described in detail below: REGISTER 7 for registering contact information, INVITE, ACK and CANCEL (Section 5.1) for setting up sessions, BYE (Section 6) for terminating sessions and OPTIONS (Section 8) for querying servers about their capabilities. SIP extensions may define additional methods ("extension-method").

Proxy and redirect servers treat all methods other than INVITE and CANCEL, whether the method is defined in this specification or elsewhere, in the same way. Thus, no method-specific support is required in these servers for methods other than INVITE and CANCEL. Methods that are not supported by a user agent server or registrar cause a 501 (Not Implemented) response to be returned (Section 11). As in HTTP, the Method token is case-sensitive.

```
Method            =  "INVITE" | "ACK" | "OPTIONS" | "BYE"
                     | "CANCEL" | "REGISTER" | extension-method
extension-method  =  token
```

4.3 Request-URI

The Request-URI is a SIP URL as described in Section 2 or a general URI (RFC 2396 [10]). In particular, it MUST NOT contain unescaped spaces or control characters. 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.

As shown in Table 2, the Request-URI MAY contain the user-param parameter as well as transport-related parameters. A server that receives a SIP-URL with illegal elements removes them before further processing.

Transport-related parameters are needed when a UAC proxies all requests to a default proxy, which would then need this information to generate the appropriate request.

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.

SIP servers MAY support Request-URIs with schemes other than "sip", for example the "tel" URI scheme [19]. It MAY translate non-SIP URIs using any mechanism at its disposal, resulting in either a SIP URI or some other scheme.

If a SIP server receives a request with a URI indicating a scheme the 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, since proxies are responsible for processing the Request-URI. (The To field is only of interest to the UAS.)

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". The string is case-insensitive, but implementations MUST use upper-case.

Unlike HTTP/1.1, SIP treats the version number as a literal string. In practice, this should make no difference.

4.4 Option Tags

Option tags are unique identifiers used to designate new options in SIP. These tags are used in Require (Section 10.35), Supported (Section 10.41) and Unsupported (Section 10.44) header fields.
Syntax:

```
option-tag = token
```

See Section C for the 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).

An example of a reverse-domain-name option is "com.foo.mynewfeature", whose inventor can be reached at "foo.com". For these features, individual organizations are responsible for ensuring that option names do not collide within the same domain. The host name part of the option MUST use lower-case; the option name is case-sensitive.

Options registered with IANA do not contain periods and are globally unique. IANA option tags are case-sensitive.

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;
- A listing of any new SIP header fields, header parameter fields or parameter values defined by this option. A SIP option MUST NOT redefine header fields or parameters defined in either RFC 2543, any standards-track extensions to RFC 2543, or other extensions registered through IANA.
- 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).

Registrations should be sent to `iana@iana.org`.

This procedure has been borrowed from RTSP [4] and the RTP AVP [27].

5 INVITE, ACK and CANCEL

5.1 INVITE

The INVITE method indicates that the user or service is being invited to participate in a session. The message body MAY contain 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.

The caller MAY choose to omit the request body (i.e., not send a session description) or send a session description that does not list any media types. This indicates that the caller does not know its desired media characteristics until the call has been accepted. In this case, the UAS SHOULD still return a session description in its informational (1xx) or success (2xx) response, containing those media streams and codecs it supports.

If the INVITE request did not contain a complete session description, the caller MUST include one in the ACK request. A UAC MUST NOT send an updated session description in an ACK request if it had already sent a session description in the INVITE request. If the UAC wishes to modify the session after the call setup has begun, it MUST initiate another INVITE transaction after the current one has completed.

Delaying the session description until the ACK request is useful for gateways from H.323v1 to SIP, where the H.323 media characteristics are not known until the call is established.

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.

The behavior of UAS depend on whether they are Internet telephony gateways to the PSTN. A UAS not acting as a gateway which receives an INVITE with a Request-URI that does not correspond to one of its configured addresses, MUST respond with 404 (Not Found).

A UAS acting as a gateway translates the INVITE request into a telephony signaling message. If the INVITE has a Call-ID value that matches a recent call, the UAS compares the Request-URI with the Request-URI of the previous INVITE request for the same Call-ID. If the Request-URI contains additional digits in the "user" part, the UAS treats the INVITE as adding additional digits to the original dialed string. This is known as overlap dialing.

If the gateway knows that the telephone number is incomplete, it returns a 484 (Address Incomplete) status 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.)

If an INVITE request for an existing session fails, the session description agreed upon in the last successful INVITE transaction remains in force.

A UAC MUST NOT issue another INVITE request for the same call leg before the previous INVITE transaction has completed. A UAS that receives an INVITE before it sent the final response to an INVITE with a lower CSeq number on the same call leg MUST return a 400 (Bad Request) response and MUST include a Retry-After header field with a randomly chosen value of between 0 and 10 seconds.

If a UA A sends an INVITE request to B and receives an INVITE request from B before it has received the response to its request from B, A MAY return a 500 (Internal Server Error), which SHOULD include a Retry-After header field specifying when the request should be resubmitted.

In most cases, a UA can assume that the order of messages received corresponds to the order they were sent. In rare circumstances, the response from B and the request from B may be reordered on the wire.

In addition, if A or B change multicast addresses, strict transaction ordering is necessary so that both sides agree on the final result.

A UAC MUST be prepared to receive media data according to the session description as soon as it sends an INVITE (or re-INVITE) and can start sending media data when it receives a provisional or final response containing a session description.

The initial INVITE from the UAC SHOULD contain the Allow and Supported header fields, and MAY contain the Accept header field. A 200 (OK) response to the initial INVITE for a call SHOULD contain the Allow and Supported header fields, and MAY contain the Accept header field.

Including these header fields allows the UAC to determine the features and extensions supported by the UAS for the duration of the call, without probing.

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

5.1.1 ACK

The ACK request confirms that the client has received a final response to an INVITE request. (ACK is used *only* with INVITE requests.) Treatment of ACK for a 200 class response differs significantly from that of a non-200 class response. 2xx responses are acknowledged by client user agents, all other final responses by the first stateful 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 or UAC to receive a non-2xx final response. For a non-200 class response, the Via in the ACK that is constructed MUST be the same as the request being acknowledged. The ACK for a 200 class response will contain Route headers if Record-Route headers were present in the response. An ACK for a non-200 class response never contains Route headers. The ACK request for a 200 class response is forwarded as the corresponding INVITE request, based on its Request-URI or Route headers, and thus MAY take a different path than the original INVITE request, and MAY even cause a new transport connection to be opened in order to send it. The Request-URI for the ACK is set to the top entry in the route set for a 200 class response (see Section 16). For a non-200 class response, the Request-URI MUST be the same as the Request-URI in the request being acknowledged.

The ACK request does not generate responses for any transport protocol.

The ACK request for a 200 class response MAY contain a message body with the final session description to be used by the callee. See Section 5.1 for further details on the relationship between session descriptions in INVITE and ACK requests.

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. However, an ACK not destined for the proxy SHOULD NOT be retransmitted.

> 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 user agents.

5.2 CANCEL

The CANCEL request cancels a pending request with the same Call-ID, To, From, top Via header and Request-URI and CSeq (sequence number only) header field values, but does not affect a completed request or existing calls. (A request is considered completed if the server has returned a final status response.) The UAC can use a BYE request to terminate a call if the CANCEL arrived too late.

A user agent client or proxy client MAY issue a CANCEL request at any time. A proxy client SHOULD generate a CANCEL request for branches without a final response after it has forked a request and receives a 2xx or 6xx response from one of the branches. A UAC or proxy client SHOULD send a CANCEL if the time noted in the Expires header of the request has elapsed or no provisional or final response was received after a client-determined timeout interval. Finally, internal logic such as scripts, MAY trigger CANCEL requests.

A stateful proxy that receives a CANCEL request immediately responds with a 200 class response. It then generates a new CANCEL, and forwards the request to all destinations with pending requests. A stateless proxy, or a stateful proxy with no transaction state for the cancelled request, proxies the CANCEL request to the same set of destinations the original request was proxied to.

The Request-URI, topmost Via, Call-ID, To, the numeric part of CSeq and From header fields in the CANCEL request are identical to those in the original request being cancelled, including tags. 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.)

The behavior of the user agent or redirect server on receiving a CANCEL request depends on whether the server has already sent a final response for the original request. If it has, the CANCEL request has no effect on the original request, any call state and on the responses generated for the original request. If the server has not issued a final response for the original request, it immediately responds to the original request with a 487 (Request Terminated), following normal rules for response retransmissions defined in Section 14. For INVITE requests, the UAC as usual sends an ACK request to confirm receipt of any final response. The CANCEL request itself is answered with a 200 (OK) response in either case. If the UAS or redirect server has no record of the request being cancelled, the CANCEL is responded to with a 481.

A proxy client or UAC cannot rely on receiving a 487 (Request Terminated) response, as a RFC 2543-compliant UAS will not generate such a response. If there has been no final response after 32 seconds, the client MAY consider the original transaction to have been cancelled.

> 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 can be distinguished by the method in the Cseq header field.

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

6 BYE

The user agent client uses BYE to indicate to the server that it wishes to release the call leg. A BYE request is forwarded like an INVITE request and MAY be issued by either caller or callee. A BYE request SHOULD NOT be sent to terminate a pending call request which has not generated either a final response or a provisional response containing a To tag. 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.

A UAS receiving a BYE request MUST respond to any pending requests received for that call, including INVITE. It is RECOMMENDED that a 487 response is generated.

This method SHOULD be supported by user agent servers.

7 Registrars, Registrations and the REGISTER Method

A client uses the REGISTER method to bind the address listed in the To header field with a SIP server to one or more URIs where the client can be reached, contained in the Contact header fields. These URIs may use any URI scheme, not limited to SIP.

It is particularly important that REGISTER requests are authenticated since they allow to redirect future requests (see Section 18.2).

7.1 Where to Register

A user agent SHOULD attempt to register periodically according to the rules below. A UA is said to be "visiting" if its From address domain differs from the current network domain and is said to be "at home" if the two are the same.

Local server: If an outbound proxy is configured, the UA SHOULD send a REGISTER request to it. If the UA is visiting, it uses the From address consisting of the URL-escaped user identity at the visited domain, e.g., the user identified as `alice@wonderland.com` would register as `alice%40wonderland.com@exa` if she is visiting the `example.com` domain.

Multicast: If no local outbound proxy is configured, multicast registrations are addressed to the well-known "all SIP servers" multicast address "sip.mcast.net" (224.0.1.75). This request MUST 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 [28], 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 [18]; however, they do not respond to the request.

> Multicast registration may be inappropriate in some environments, for example, if multiple businesses share the same local area network.

Home server: If the UA is visiting, it SHOULD also send a registration to its home SIP server, identified by its home address. For example, `alice@wonderland.com` would send a registration to the SIP server for the domain `wonderland.com` when she is visiting another network. TBD: What Contact should be used?

A user agent SHOULD register with a local server on startup and periodically thereafter by sending a REGISTER request. The period is given by the expiration time indicated in the registration response. It is RECOMMENDED that the UA registers via multicast and send a registration to its "home" address, i.e., the server for the domain that it uses as its From address in outgoing requests.

7.2 REGISTER Header Fields

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 traveler `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 REGISTER request is no longer forwarded once it has 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.

7.3 Registering Contact Locations

REGISTER 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.

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. It is RECOMMENDED that registrars authorize whether the entity in the From field is allowed to register addresses for the address-of-record in the To field.

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, as described below.

If a SIP URI in a registration Contact header field differs from existing registrations according to the rules in Section 2.1, it is added to the list of registration. If it is equivalent, according to these rules, to an existing registration, all Contact header field parameters for this entry are updated accordingly. URIs other than SIP URIs are compared according to the standard URI equivalency rules for the URI schema.

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.

7.4 Registration Expiration

An optional "expires" Contact parameter indicates the desired expiration time of the registration. If a Contact entry does not have an "expires" parameter, the Expires request and response 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. Registrations not refreshed after this amount of time SHOULD be silently discarded.

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. If a registration updates an existing registration, the Expires value of the most recent registration is used, even if it is shorter than the earlier registration.

The registrar determines the expiration time; it may be longer or shorter than the one requested by the registrand. The REGISTER response contains the actual registration lifetime; the client MUST refresh at least as often and SHOULD NOT refresh more frequently. In general, the server SHOULD honor the expiration time offered by the user agent. A server MAY decide to lengthen the expiration interval if, for example, the refresh rate of a particular client exceeds a threshold.

This behavior is different from RFC 2543, which only allowed registrars to decrease, but not increase, the interval.

> Allowing the registrar to set the registration interval protects it against excessively frequent registration refreshes while limiting the state that it needs to maintain and decreasing the chance for stale registrations that require proxying effort.

Registration refreshes SHOULD be sent to the same address as the original registration, unless redirected.

7.5 List of Current Registrations

2xx REGISTER responses SHOULD list all current registration in the Contact header field. An "expires" parameter MUST indicate the expiration time of the registration.

7.6 Removing Registrations

Registrations expire as described above or may be removed explicitly by setting the expires parameter for an existing registration to zero or including an Expires: 0 header field. Registrations are matched based on the user, host, port and maddr parameters. A client can remove *all* registrations by including a single Contact header field with the wildcard address "*". This usage is only allowed in REGISTER requests when a Expires header with value of zero is present.

Support of this method is RECOMMENDED; registrars MUST support it.

8 OPTIONS

The OPTIONS method is used to query a server 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). A server SHOULD return Allow, Accept, Accept-Encoding, Accept-Language and Supported header fields. The response MAY contain a message body indicating the capabilities of the end system (rather than properties of any existing call).

The use of the Call-ID header field is discussed in Section 10.12. An OPTIONS requests for an existing call-id has no impact on that call.

This method MUST be supported by SIP user agents and registrars.

9 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 9.1
             *( general-header
             | response-header
             | entity-header )
             CRLF
             [ message-body ]       ; Section 12
```

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

9.1 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
```

9.1.1 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. 4
                |  Success                ;Fig. 4
                |  Redirection            ;Fig. 5
                |  Client-Error           ;Fig. 6
                |  Server-Error           ;Fig. 7
                |  Global-Failure         ;Fig. 8
                |  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 11. 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:

```
Informational  =  "100"  ; Trying
               |  "180"  ; Ringing
               |  "181"  ; Call Is Being Forwarded
               |  "182"  ; Queued
               |  "183"  ; Session Progress
Success        =  "200"  ; OK
```

Figure 4: Informational and success status codes

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

Figure 5: Redirection status codes

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 4 through 8 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. However, proxies SHOULD distinguish 100 from other 1xx responses. (The former SHOULD NOT be forwarded, while the latter MUST be. See Section 17.3.) 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.

```
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
              |  "487"  ; Request Terminated
              |  "488"  ; Not Acceptable Here
```

Figure 6: Client error status codes

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

Figure 7: Server error status codes

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

Figure 8: Global failure status codes

10 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.

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. "Optional" means that a UA MAY include the header field in a request or response, and a UA MAY ignore the header field if present in the request or response. A "mandatory" request header field MUST be present in a request, and MUST be understood by the UAS receiving the request. A mandatory response header field MUST be present in the response, and the header field MUST be understood by the UAC processing the response. "Not applicable" means for request header fields that the header field MUST NOT be present in a request. If one is placed in a request by mistake, it MUST be ignored by the UAS receiving the request. Similarly, a header field labeled "not applicable" for a response means that the UAS MUST NOT place the header in the response, and the UAC MUST ignore the header in the response. "m*" indicates a header that SHOULD be sent, but servers need to be prepared to receive messages without that header field. A "*" indicates that the header fields are required if the message body is not empty. See sections 10.18, 10.19 and 12 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 10.1) and entity header (Section 10.2) fields, respectively. If a header field is marked "c", it is copied from the request to the response.

The "proxy" column describes whether proxies can add comma-separated elements to headers ("c", for concatenate or comma), can modify the header ("m"), can add the header if not present ("a") or need to read the header ("r"). Headers that need to be read cannot be encrypted. Proxies MUST NOT alter any fields that are authenticated (see Section 18.2), but MAY add copies of fields that were authenticated by the UA if indicated in the table. Depending on local policy, proxies MAY inspect any non-encrypted header fields and MAY modify any non-authenticated header field, but proxies cannot rely on fields other than the ones indicated in the table to be readable or modifiable.

If authentication is used, the rules in Section 18.2 apply. Proxies SHOULD NOT re-order 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 13 for use over UDP when the request has to fit into a single packet and size is an issue.

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

10.1 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"

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG
Accept	R		-	o	o	o	o	o
Accept	415		-	o	o	o	o	o
Accept	2xx		-	-	-	o	o	o
Accept-Encoding	R		-	o	o	o	o	o
Accept-Encoding	2xx		-	-	-	o	o	o
Accept-Encoding	415		-	o	o	o	o	o
Accept-Language	R		-	o	o	o	o	o
Accept-Language	2xx		-	-	-	o	o	o
Accept-Language	415		-	o	o	o	o	o
Alert-Info	R	am	-	-	-	o	-	-
Allow	R		o	o	o	o	o	o
Allow	2xx		o	o	o	m*	m*	o
Allow	r		o	o	o	o	o	o
Allow	405		m	m	m	m	m	m
Authorization	R		o	o	o	o	o	o
Authorization	r		o	o	o	o	o	o
Call-ID	gc	r	m	m	m	m	m	m
Call-Info	g	am	-	-	-	o	o	o
Contact	R		o	-	-	m	o	o
Contact	1xx		-	-	-	o	o	-
Contact	2xx		-	-	-	m	o	o
Contact	3xx		-	o	-	o	o	o
Contact	485		-	o	-	o	o	o
Content-Disposition	e		o	o	-	o	o	o
Content-Encoding	e		o	o	-	o	o	o
Content-Language	e		o	o	-	o	o	o
Content-Length	e	r	m*	m*	m*	m*	m*	m*
Content-Type	e		*	*	-	*	*	*
CSeq	gc	r	m	m	m	m	m	m
Date	g	a	o	o	o	o	o	o
Encryption	g	r	o	o	o	o	o	o
Error-Info	R		o	o	o	o	o	o
Expires	g		-	-	-	o	-	o
From	gc	r	m	m	m	m	m	m
In-Reply-To	R		-	-	-	o	-	-
Max-Forwards	R	rm	o	o	o	o	o	o
MIME-Version	g		o	o	o	o	o	o
Organization	g	am	-	-	-	o	o	o

Table 4: Summary of header fields, A–O

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG
Priority	R	a	-	-	-	o	-	-
Proxy-Authenticate	401,407		o	o	o	o	o	o
Proxy-Authorization	R	r	o	o	o	o	o	o
Proxy-Require	R	r	o	o	o	o	o	o
Record-Route	R	amr	o	o	o	o	o	o
Record-Route	2xx,401,484		o	o	o	o	o	o
Require	g	acr	o	o	o	o	o	o
Response-Key	R		-	o	o	o	o	o
Retry-After	404,413,480,486		o	o	o	o	o	o
	500,503		o	o	o	o	o	o
	600,603		o	o	o	o	o	o
Route	R	r	o	o	o	o	o	o
Server	r		o	o	o	o	o	o
Subject	R		-	-	-	o	-	-
Supported	g		-	o	o	o	o	o
Timestamp	g		o	o	o	o	o	o
To	gc(1)	r	m	m	m	m	m	m
Unsupported	420		o	o	o	o	o	o
User-Agent	g		o	o	o	o	o	o
Via	gc	acmr	m	m	m	m	m	m
Warning	r		o	o	o	o	o	o
WWW-Authenticate	R		o	o	o	o	o	o
WWW-Authenticate	401		o	o	o	o	o	o

Table 5: Summary of header fields, P–Z; (1): copied with possible addition of tag

fields.

10.2 Entity Header Fields

The "entity-header" fields define meta-information about the message-body or, if no body is present, about the resource identified by the request. The term "entity header" is an HTTP 1.1 term where the response body 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.

10.3 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.

10.4 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.

10.5 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 [26]. 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,
                   separators, 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.

Unless otherwise stated, parameter names, parameter values and tokens are case-insensitive. Values expressed as quoted strings are case-sensitive.

The Contact, From and To header fields contain a URL. If the URL contains a comma, question mark or semicolon, the URL MUST be enclosed in angle brackets (< and >). Any URL parameters are contained within these brackets. If the URL is not enclosed in angle brackets, any semicolon-delimited parameters are header-parameters, not URL parameters.

10.6 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`.

As a request-header field, it is used only with those methods that take message bodies. In a 415 (Unsupported Media Type) response, it indicates which content types are acceptable in requests. In 200 (OK) responses for INVITE, it lists the content types acceptable for future requests in this call.

Example:

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

10.7 Accept-Encoding

The Accept-Encoding general-header field is similar to Accept, but restricts the content-codings [H3.5] 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].

> Note: An empty Accept-Encoding header field is permissible, even though the syntax in [H14.3] does not provide for it. It is equivalent to Accept-Encoding: identity, i.e., only the identity encoding, meaning no encoding, is permissible.

If no Accept-Encoding header field is present in a request, the server MUST use the "identity" encoding.

> HTTP/1.1 [H14.3] states that the server SHOULD use the "identity" encoding unless it has additional information about the capabilities of the client. This is needed for backwards-compatibility with old HTTP clients and does not affect SIP.

10.8 Accept-Language

The Accept-Language general-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 general-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
```

10.9 Alert-Info

The Alert-Info header field indicates that the content indicated in the URLs should be rendered instead of ring tone. A user SHOULD be able to disable this feature selectively to prevent unauthorized disruptions.

```
Alert-Info     =  "Alert-Info" ":" # ( "<" URI ">" *( ";" generic-param ))
generic-param  =  token [ "=" ( token | host | quoted-string ) ]
```

Example:

```
Alert-Info: <http://wwww.example.com/sounds/moo.wav>
```

10.10 Allow

The Allow header field lists the set of methods supported by the user agent generating the message. An Allow header field MUST be present in a 405 (Method Not Allowed) response to any request, and SHOULD be present in a 2xx OPTIONS response. In this usage, it indicates the set of methods that can be invoked on the resource identified by the Request-URI of the request sent by the UAC.

Allow SHOULD be present in an INVITE request (initial or re-INVITE), and SHOULD be present in any 2xx response to an INVITE. In this usage, it indicates what methods can be invoked within the call leg, on the user agent sending the message, for the duration of the call leg. For example, if a Allow was present in a 200 OK to an initial INVITE listing INFO, the caller would know that it is safe to send an INFO request on the call leg established by the 200 OK. An Allow MAY be sent in any 1xx responses for an INVITE; it carries the same meaning as if it appeared in a 2xx, but conveys this information to the UAC before the call is established. This is helpful if the UAC wishes to send additional requests on the call leg before it is accepted.

Allow MAY be present in provisional and final responses for methods besides INVITE and OPTIONS. It carries the same semantics in this case is it does in a 2xx to OPTIONS.

All methods, including ACK and CANCEL, understood by the UA MUST be included in the list of methods in the Allow header, when present. The absence of an Allow header MUST NOT be interpreted to mean that the UA sending the message supports no methods. Rather, it implies that the UA is not providing any information on what methods it supports.

> Supplying an Allow header in responses to methods other than OPTIONS cuts down on the number of messages needed.

```
Allow  =  "Allow" ":" 1#Method
```

10.11 Authorization

A user agent that wishes to authenticate itself with a UAS or registrar – usually, but not necessarily, after receiving a 401 response – MAY do so by including an Authorization 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 18.2 overviews the use of the Authorization header field, and Section 19 describes the syntax and semantics when used with HTTP Basic and Digest authentication.

10.12 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 (in addition to the CSeq 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. For these requests, it makes no difference whether the Call-ID value matches an existing call or not.

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.

```
callid   =  token [ "@" token ]
Call-ID  =  ( "Call-ID" | "i" ) ":" callid
```

The callid MUST be a globally unique identifier and MUST NOT be reused for later calls. Use of cryptographically random identifiers [29] is RECOMMENDED. Implementations MAY use the form "localid@host". Call-IDs are case-sensitive and are simply compared byte-by-byte.

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 13. 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
```

10.13 Call-Info

The Call-Info general header field provides additional information about the caller or callee, depending on whether it is found in a request or response. The purpose of the URI is described by the "purpose" parameter. "icon" designates an image suitable as an iconic representation of the caller or callee; "info" describes the caller or callee in general, e.g., through a web page; "card" provides a business card (e.g., in vCard [30] or LDIF [31] formats).

```
Call-Info   =  "Call-Info" ":" # ( "<" URI ">" *( ";" info-param) )
info-param  =  "purpose" "=" ( "icon" | "info" | "card" | token )
            |  generic-param
```

Example:

```
Call-Info: <http://wwww.example.com/alice/photo.jpg> ;purpose=icon,
  <http://www.example.com/alice/> ;purpose=info
```

10.14 Contact

Among the methods discussed in this specification, the Contact general-header field can appear in INVITE, OPTIONS, ACK, and REGISTER requests, and in 1xx, 2xx, 3xx, and 485 responses. Other methods defined elsewhere may allow or require the use of the Contact header field. This is generally necessary if the recipient of this method needs to send requests to the originator. In general, it provides a URL where the user can be reached for further communications.

In some of the cases below, the client uses information from the Contact header field in Request-URI of future requests. In these cases, the client copies all but the "method-param" and "header" elements of the addr-spec part of the Contact header field into the Request-URI of the request. It uses the "header" parameters to create headers for the request, replacing any default headers normally used. Unless the client is configured to use a default proxy for all outgoing requests, it then directs the request to the address and port 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.

INVITE, OPTIONS and ACK requests: INVITE requests MUST, and ACK requests MAY contain a single Contact header indicating a single URI from which location the request is originating. The URI SHOULD contain the address of the client itself (i.e., its IP address, or a FQDN for the host, or an SRV record with the highest priority entry beingan FQDN of that host). See Section 16 for usage of the Contact header for routing subsequent requests. For OPTIONS, Contact provides a hint where future SIP requests can be sent or the user can be contacted via non-SIP means.

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.

INVITE and OPTIONS 2xx responses: A user agent server sending a definitive, positive response (2xx) MUST insert a single Contact response header field indicating a single SIP URI under which it is reachable most directly for future SIP requests, such as ACK, within the same call leg. The URI SHOULD contain the address of the server itself (i.e., its IP address, or a FQDN for the host, or an SRV record with the highest priority entry beingan FQDN of that host). See Section 16 for usage of the Contact header for routing subsequent requests.

If a UA supports both UDP and TCP, it SHOULD NOT indicate a transport parameter in the URI.

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

REGISTER requests and responses: See Section 7. The Contact header value of "*" is only used in REGISTER requests.

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 information from the Contact header field into the Request-URI as described above.

4xx, 5xx and 6xx responses: The Contact response-header field can be used with a 4xx, 5xx or 6xx response to indicate the location where additional information about the error can be found.

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 could contain URL's for phones, fax, or irc (if they were defined) or a mailto: (RFC 2368, [32]) 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 to 1, with higher values indicating higher preference. The default value is 0.5.

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 a SIP-date. If this parameter is not provided, the value of the Expires header field determines how long the URI is valid. Implementations MAY treat values larger than 2**32-1 (4294967295 seconds or 136 years) as equivalent to 2**32-1.

```
Contact           =  ( "Contact" | "m" ) ":"
                     ("*" | (1# (( name-addr | addr-spec )
                     *( ";" contact-params ))))
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 <">
                  |  contact-extension
contact-extension =  generic-param
qvalue            =  ( "0" [ "." 0*3DIGIT ] )
                  |  ( "1" [ "." 0*3("0") ] )
```

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. Note that there may or may not be LWS between the display-name and the "<".

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
```

10.15 Content-Disposition

```
Content-Disposition    =  "Content-Disposition" ":"
                          disposition-type *( ";" disposition-param )
disposition-type       =  "render" | "session" | "icon" | "alert"
                       |  disp-extension-token
disposition-param      =  "handling" "="
                          ( "optional" | "required" | other-handling )
                       |  generic-param
other-handling         =  token
disp-extension-token   =  token
```

The Content-Disposition header field describes how the message body or, in the case of multipart messages, a message body part is to be interpreted by the UAC or UAS. The SIP header extends the MIME Content-Type (RFC 1806 [33]).

The value "session" indicates that the body part describes a session, for either calls or early (pre-call) media. The value "render" indicates that the body part should be displayed or otherwise rendered to the user. For backward-compatibility, if the Content-Disposition header is not missing, bodies of Content-Type `application/sdp` imply the disposition "session", while other content types imply "render".

The disposition type "icon" indicates that the body part contains an image suitable as an iconic representation of the caller or callee. The value "alert" indicates that the body part contains information, such as an audio clip, that should be rendered instead of ring tone.

The handling parameter, handling-parm, describes how the UAS should react if it receives a message body whose content type or disposition type it does not understand. If the parameter has the value "optional", the UAS MUST ignore the message body; if it has the value "required", the UAS MUST return 415 (Unsupported Media Type). If the handling parameter is missing, the value "required" is to be assumed.

If this header field is missing, the MIME type determines the default content disposition. If there is none, "render" is assumed.

10.16 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 [H3.5] 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.

10.17 Content-Language

See [H14.12].

10.18 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. (The size of the message-body does *not* include the CRLF separating headers and body.) 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 datagram request without Content-Length, it MUST assume that the request encompasses the remainder of the packet. If a server receives a datagram request with a Content-Length, but the value differs from the size of the body sent in the request, the server SHOULD return a 400 (Bad Request) response.

If a response does not contain a Content-Length, the client assumes that it encompasses the remainder of the datagram packet or the data until the stream connection is closed, as applicable. Section 12 describes how to determine the length of the message body.

> The ability to omit Content-Length simplifies the creation of cgi-like scripts that dynamically generate responses.

10.19 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]. The Content-Type header MUST be present if the body is not empty. If the body is empty, and a Content-Length header is present, it indicates that the body of the specific type has zero length (for example, if it is an emtpy audio file).

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
```

10.20 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. The sequence number MUST be expressible as a 32-bit unsigned integer. A server MUST echo the CSeq value from the request in its response. The CSeq header serves to order transactions within a call leg, and to provide a means to uniquely identify transactions.

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

For requests that are outside of a call leg, or for a request that initiates a session, the value of the sequence number is arbitrary, but MUST be less than 2**31. For requests which are subsequent ones within an existing call leg (such as a re-INVITE or BYE), the CSeq header MUST contain strictly monotonically increasing and contiguous (increasing-by-one) sequence numbers; sequence numbers do not wrap around. Retransmissions of the same request carry the same CSeq value.

For requests outside of a call leg, ordering is irrelevant, and so the value of the CSeq number in requests received by a UAS is not important. For requests within a call leg, ordering is important. Therefore, a UAS MUST remember the highest sequence number for any request received within a call leg. The server MUST reject, using a 400 class response, any request within a call leg with a lower sequence number. Any request that is received with a sequence number higher than the highest received so far (even it is higher by more than one), SHOULD be accepted.

If a client initiates a session, and receives multiple 200 class responses, each establishes a separate call leg. For subsequent requests within each of those call legs (each of which differs only by the tag in the To field), the CSeq numbers increment independently from the other call legs. Furthermore, the CSeq numbering space is unique in each direction. That is, the CSeq values in requests from A to B are independent of the values in requests from B to A.

The ACK request MUST contain the same CSeq numeric value as the INVITE request that it refers to, but with a Method of “ACK”. The CANCEL request MUST contain the same CSeq numeric value as the request it cancels, but with a Method of “CANCEL”.

The Method value allows the client to distinguish the response to a CANCEL request from that of the request it is cancelling. 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.

Example:

```
CSeq: 4711 INVITE
```

10.21 Date

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

```
Date      = "Date" ":" SIP-date
SIP-date  = rfc1123-date
```

See [H14.18] for a definition of rfc1123-date. Note that unlike HTTP/1.1, SIP only supports the most recent RFC 1123 [34] formatting for dates. As in [H3.3], SIP restricts the timezone in SIP-date to "GMT", while RFC 1123 allows any timezone.

The consistent use of GMT between Date, Expires and Retry-After headers allows implementation of simple clients that do not have a notion of absolute time.

Note that rfc1123-date is case-sensitive.

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.

Registrars MUST include this header in REGISTER responses if they use absolute expiration times and SHOULD include it for all responses.

The Date header field can be used by simple end systems without a battery-backed clock to acquire a notion of current time. However, in its GMT-form, it requires clients to know their offset from GMT.

10.22 Encryption

The Encryption general-header field specifies that the content has been encrypted. Section 18 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  = generic-param
```

The token indicates the form of encryption used; it is described in Section 18.

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.

10.23 Error-Info

The Error-Info response header provides a pointer to additional information about the error status response. This header field is only contained in 3xx, 4xx, 5xx and 6xx responses.

Error-Info = "Error-Info" ":" # ("<" URI ">" *(";" generic-param))

10.24 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, as described in Section 7, and INVITE methods.

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 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.

Note that the expiration time does *not* affect the duration of the actual session that may result from the invitation. Session description protocols may offer the ability to express time limits on the session duration, however.

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. Implementations MAY treat values larger than 2**32-1 (4294967295 or 136 years) as equivalent to 2**32-1.

Expires = "Expires" ":" (SIP-date | delta-seconds)

Two examples of its use are

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

10.25 From

Requests and responses MUST contain a From general-header field, indicating the initiator of the request. (Note that this may be different from the initiator of the call leg. Requests sent by the callee to the caller use the callee's address in the From header field.) The From field MUST contain the "tag" parameter. However, a server MUST be prepared to receive a request without a tag, in which case the tag is considered to effectively have a value of zero. This is to maintain backwards compatibility with RFC2543, which did not mandate From tags. . 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 ignores them.

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. Syntax issues are discussed in Section 10.5.

```
From        =  ( "From" | "f" ) ":" ( name-addr | addr-spec )
               *( ";" from-param )
from-param  =  tag-param | generic-param
tag-param   =  "tag" "=" token
```

Examples:

```
From: "A. G. Bell" <sip:agb@bell-telephone.com> ;tag=a48s
From: sip:+12125551212@server.phone2net.com;tag=887s
From: Anonymous <sip:c8oqz84zk7z@privacy.org>;tag=hyh8
```

The "tag" value MUST be globally unique and cryptographically random with at least 32 bits of randomness. It SHOULD differ for each call leg.

For the purpose of identifying call legs, two From or To header fields are equal if and only if:

- The addr-spec component is equal, according to the rules in Section 2.1.
- Any "tag" and "generic-param" parameters are equal, compared according to the case-sensitivity rules in Section 10. Only parameters that appear in both header fields are compared.

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 [26] header, but with a URI instead of just an email address.

10.26 In-Reply-To

The In-Reply-To request header field enumerates the call-IDs that this call references or returns.

This allows automatic call distribution systems to route return calls to the originator of the first call and allows callees to filter calls, so that only calls that return calls they have originated will be accepted. This field is not a substitute for request authentication.

```
In-Reply-To  =  "In-Reply-To" ":" 1# callid
```

Example:

```
In-Reply-To: 70710@saturn.bell-tel.com, 17320@saturn.bell-tel.com
```

10.27 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 and returns 483 (Too many hops). Instead, a server MAY act as a final recipient for OPTIONS requests. It is RECOMMENDED that the server include Supported, Server and Allow header fields in the response.

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
```

10.28 MIME-Version

See [H19.4.1].

10.29 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-TRIM

10.30 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" | other-priority
other-priority	=	token

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 [35], with the addition of "emergency".

10.31 Proxy-Authenticate

The Proxy-Authenticate response-header field MUST be included as part of a 407 (Proxy Authentication Required) response. It may also occur in a 401 (Unauthorized) response if the request was forked. 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 the UAC. In SIP, only UAC's can authenticate themselves to proxies.

The syntax for this header is defined in [H14.33]. See 19 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 407 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.

10.32 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 19 for a discussion of its usage.

10.33 Proxy-Require

The Proxy-Require header field is used to indicate proxy-sensitive features that MUST be supported by the proxy. If a proxy 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. A UAC SHOULD attempt to retry the request, without using the features listed in the Unsupported header.

See Section 10.35 for more details on the mechanics of this message and a usage example.

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

10.34 Record-Route

The Record-Route header field has the following syntax:

```
Record-Route  =  "Record-Route" ":" 1# ( name-addr *( ";" rr-param ))
rr-param      =  generic-param
```

Details of its use are described in Section 16.

10.35 Require

The Require general-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. A UAC SHOULD attempt to retry the request, without using the features listed 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 10.33).

10.36 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 = generic-param

The "key-scheme" gives the type of encryption to be used for the response. Section 18 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.

10.37 Retry-After

The Retry-After response-header field can be used with a 503 (Service Unavailable) response to indicate how long the service is expected to be unavailable to the requesting client and with a 404 (Not Found), 600 (Busy), or 603 (Decline) response to indicate when the called party 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.

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 ] *( ";" retry-param )
retry-param   =  "duration" "=" delta-seconds
              |  generic-param
```

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.

10.38 Route

The Route header field has the following syntax:

```
Route  =  "Route" ":" 1# ( name-addr *( ";" rr-param ))
```

Details of its use are described in Section 16.

10.39 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 [H14.38].

10.40 Subject

This header field provides a summary or indicates the nature of the call, allowing call filtering without having to parse the session description. (Note that the session description does not have to use the same subject indication as the invitation.)

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

Example:

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

10.41 Supported

The Supported general-header field enumerates all the capabilities of the client or server. This header field SHOULD be included in all requests (except ACK) and in all responses.

Including the header field in all responses greatly simplifies the use of extensions for call control in subsequent transactions with the same server.

Syntax:

```
Supported = ( "Supported" | "k" ) ":" 1#option-tag
```

10.42 Timestamp

The Timestamp general-header field describes when the client sent the request to the server. The client uses the current time value at the time of transmission, i.e., each retransmission of a request is likely to have a different timestamp value.

The value of the timestamp is of significance only to the client and it MAY use any timescale. The server MUST echo the exact same value in all provisional and final responses 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.

10.43 To

The To general-header field specifies the "logical" recipient of the request.

```
To       = ( "To" | "t" ) ":" ( name-addr | addr-spec )
           *( ";" to-param )
to-param = tag-param | generic-param
```

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.

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 user agent, registrar and redirect servers, but MUST NOT be inserted into responses forwarded upstream by proxies. However, responses generated locally by a proxy, and then sent upstream, MUST contain a tag.

A UAS or redirect server MUST add a "tag" parameter for all final responses for all transactions within a call leg. All such parameters have the same value within the same call leg. These servers SHOULD add the tag for informational responses during the initial INVITE transaction, but MUST add a tag to informational responses for all subsequent transactions.

See Section 10.25 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.

Section 15 describes when the "tag" parameter MUST appear in subsequent requests. Note that if a request already contained a tag, this tag MUST be mirrored in the response; a new tag MUST NOT be inserted.

Section 10.25 describes how To and From header fields are compared for the purpose of matching requests to call legs.

UAS SHOULD accept requests even if they do not recognize the URI scheme (e.g., a `tel:` URI) or if the To header does not address the user. Only Request-URI that do not match the recipient should cause requests to be rejected.

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. Note that LWS is common, but **not** mandatory between the display-name and the "<".

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.

10.44 Unsupported

The Unsupported response-header field lists the features not supported by the server. See Section 10.35 for a usage example and motivation.

Syntax:

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

10.45 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 [H14.43].

10.46 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.

10.46.1 Requests

The client originating the request MUST insert into the request a Via field containing the transport protocol used to send the message, the client's 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 client that sends a request to a multicast address MUST add the "maddr" parameter to its Via header field, and SHOULD add the "ttl" parameter. (In that case, the maddr parameter SHOULD contain the destination multicast address, although under exceptional circumstances it MAY contain a unicast address.) If a server receives a request which contained an "maddr" parameter in the topmost Via field, it SHOULD send the response to the address listed in the "maddr" parameter.

Loop detection is described in Section 17.3.1.

10.46.2 Receiver-tagged Via Header Fields

A proxy or UAS receiving a request SHOULD check the first Via header field to ensure that it contains the sender's correct network address, as seen from that proxy. If the Via header contains a domain name or if it contains an IP address that differs from the packet source address, the proxy or UAS SHOULD add a "received" attribute to that Via header field.

> A multi-homed host may not be able to insert a network address into the Via header field that can be reached by the next hop, for example because if one of the networks is private. The address placed into the Via header may differ from the interface actually used, as that interface is selected only at packet sending time by the IP layer. Similarly, a request traversing a network address translator (NAT) will also cause the sending address to differ from the address seen by the next hop. The mechanism described here is unlikely to be sufficient, however, for allowing packets to traverse a NAT in the reverse direction.

An example is:

```
Via: SIP/2.0/UDP erlang.bell-telephone.com:5060
Via: SIP/2.0/UDP 128.59.16.1:5060 ;received=128.59.19.3
```

In this example, the message originated from a multi-homed host with two addresses, 128.59.16.1 and 128.59.19.3. The sender guessed wrong as to which network interface would be used. Erlang.bell-telephone.com noticed the mismatch, and added a parameter to the previous hop's Via header field, containing the address that the packet actually came from.

10.46.3 Receiving Responses

Via header fields in responses received 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. Specifically, the sent-by value should equal the value inserted by the proxy or UAC. The recevied parameter MUST NOT be used by a proxy or UAC to determine if the response is for a request it sent. If the sent-by value is not equal to the value inserted by the proxy or UAC, 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, use this Via field as the destination, as described in Section 10.46.5.

10.46.4 Generating Responses

A UAS, proxy or redirect that server that generates a response copies the Via header fields from the request into the response, without changing their order, and uses the top (first) Via element as the destination, as described in the next section.

10.46.5 Sending Responses

Given a destination described by a Via header field, the response is sent according to the following rules:

- If the "sent-protocol" is a reliable transport protocol such as TCP, TLS or SCTP, send the response using the existing TCP connection to the source of the original request. If no connection is open, open a connection to the IP address in the received parameter, if present using the port in the sent-by value, or port 5060 if none is present. If the connection attempt fails, or if there was no received parameter, the server SHOULD attempt to open a connection to the address in the sent-by value, which may be a domain name. To do this, it constructs a SIP URL of the form "sip:¡sent-by¿;transport=¡sent-protocol¿" and then uses the procedures defined in [36] to determine the IP address and port to open the connection and send the response to.
- Otherwise, if the Via header field contains a "maddr" parameter, forward the response to the address listed there, using the port indicated in "sent-by", or port 5060 if none is present. If the address is a multicast address, 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.
- Otherwise, if it is a receiver-tagged field (Section 10.46.2), send the response to the address in the "received" parameter, using the port indicated in the "sent-by" value, or using port 5060 if none is specified explicitly. If this fails, e.g., elicits an ICMP "port unreachable" response, send the response to the address in the "sent-by" parameter. The address to send to is determined by constructing a SIP URL of the form "sip:¡sent-by¿", and then using the DNS procedures defined in [36] to send the response.
- Otherwise, if it is not receiver-tagged, send the response to the address indicated by the "sent-by" value.

Note that the response to an unreliable datagram request is **not** returned to the port from which the request came, but it is always returned to the source IP that that request came from.

10.46.6 Syntax

The format for a Via header field is shown in Fig. 9. 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 10.46.2).

The "branch" parameter is included by every proxy. The token MUST be unique for each distinct request. The precise format of the token is implementation-defined. In order to be able to both detect

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

Figure 9: Syntax of Via header field

loops and associate responses with the corresponding request, the parameter SHOULD consist of two parts separable by the implementation. One part, used for loop detection (Section 17.3.1), MAY be computed as a cryptographic hash of the To, From, Call-ID header fields, the Request-URI of the request received (before translation) and the sequence number from the CSeq header field. The hash SHOULD also include any other fields the proxy uses to make a routing decision on the request. This is to ensure that if the request is routed back to the proxy, and one of those fields changes, it is treated as a spiral, and not a loop. The algorithm used to compute the hash is implementation-dependent, but MD5 [37], expressed in hexadecimal, is a reasonable choice. (Note that base64 is not permissible for a token.) The other part, used for matching responses to requests, is a globally unique function of the branch taken, for example, a hash of a sequence number, local IP address and request-URI of the request sent on the branch.

For example: `7a83e5750418bce23d5106b4c06cc632.1`

The "branch" parameter MUST depend on the incoming request-URI, or any other headers used for routing, to distinguish looped requests from requests whose request-URI (or whatever headers are used for routing) is changed and which then reach a server visited earlier.

CANCEL and ACK requests MUST have the same branch value as the corresponding request they cancel or acknowledge. When a response arrives at the proxy it can use the branch value to figure out which branch the response corresponds to.

```
Via: SIP/2.0/UDP first.example.com:4000;ttl=16
  ;maddr=224.2.0.1 ;branch=a7c6a8dlze.1 (Acme server)
Via: SIP/2.0/UDP adk8%20.8x%fe%03 ;hidden
```

10.47 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
pseudonym	=	token

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 [38].

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 are not 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.

If the warning is caused by the session description, the status response SHOULD include a session description similar to that included in OPTIONS responses indicating the capabilities of the UAS.

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'"
```

10.48 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 [H14.47] for a definition of the syntax, and Section 19 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 URL in the destination (To header) and "realm" and attempt to re-use these values on the next request for that destination.

11 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.

11.1 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.

Informational (1xx) responses other than 100 (Trying) MAY contain message bodies, including session descriptions. If a 1xx response contains a session description, a UAC SHOULD cease generating local ringback tone. Session descriptions in 1xx responses are interpreted in the same manner as those in 2xx responses. In particular, the session description MUST be formatted in such a way that it would be valid in a 2xx response. Thus, the UAS can only include a session description in its provisional response if the UAC has included one in an earlier INVITE. (SIP extensions may specify additional circumstances where session descriptions may be included.) If a later provisional response or 2xx contains a different session description, this new description is treated as if it were the original response to the session description in the INVITE.

The UAS can remove the media stream by setting the port number to zero in a subsequent session description contained in a provisional response and thus restore normal ringback behavior. The UAS cannot add media streams beyond those offered by the UAC in the INVITE. A provisional response without a session description has no effect on any early media that have already been set up.

The media streams are assumed to be bidirectional unless marked as send-only or receive-only. For SDP, this is described in Section B. Client behavior when receiving several different session descriptions from different branches is undefined.

11.1.1 100 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.

11.1.2 180 Ringing

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

11.1.3 181 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.

11.1.4 182 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.

11.1.5 183 Session Progress

The 183 (Session Progress) response is used to convey information about the progress of the call which is not otherwise classified. The Reason-Phrase MAY be used to convey more details about the call progress.

11.2 Successful 2xx

The request was successful and MUST terminate a search.

11.2.1 200 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.

11.3 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.

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.

11.3.1 300 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 10.14). 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.

11.3.2 301 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 10.14). 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.

11.3.3 302 Moved Temporarily

The requesting client SHOULD retry the request at the new address(es) given by the Contact header field (Section 10.14). The Request-URI of the new request uses the value of the Contact header in the response. The new request can take two different forms. In the first approach, the To, From, Call-ID, and CSeq header fields in the new request are the same as in the original request, with a new branch identifier in the Via header field. Proxies MUST follow this behavior and UACs MAY. UAs MAY also use the Contact information for the To header field, as well as a new Call-ID value.

Reusing the CSeq value allows proxies to avoid forwarding the request to the same destination twice, as a proxy will consider it a retransmission.

The duration of the redirection can be indicated through an Expires (Section 10.24) header. If there is no explicit expiration time, the address is only valid for this call and MUST NOT be cached for future calls.

11.3.4 305 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.

11.3.5 380 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.

11.4 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.

11.4.1 400 Bad Request

The request could not be understood due to malformed syntax. The Reason-Phrase SHOULD identify the syntax problem in more detail, e.g., “Missing Call-ID header”.

11.4.2 401 Unauthorized

The request requires user authentication. This response is issued by user agent servers and registrars, while 407 (Proxy Authentication Required) is used by proxy servers.

11.4.3 402 Payment Required

Reserved for future use.

11.4.4 403 Forbidden

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

11.4.5 404 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.

11.4.6 405 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.

11.4.7 406 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.

11.4.8 407 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 10.31) 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 10.32). SIP access authentication is explained in section 18.2 and 19.

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

11.4.9 408 Request Timeout

The server could not produce a response within a suitable amount of time, for example, since it could not determine the location of the user in time. The amount of time may have been indicated in the Expires request-header field or may be set by the server. The client MAY repeat the request without modifications at any later time.

11.4.10 409 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.

11.4.11 410 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.

11.4.12 413 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.

11.4.13 414 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.

11.4.14 415 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 server for the requested method. The server SHOULD return a list of acceptable formats using the Accept, Accept-Encoding and Accept-Language header fields. The client SHOULD retry the request, this time omitting any bodies not supported by the server.

11.4.15 420 Bad Extension

The server did not understand the protocol extension specified in a Proxy-Require (Section 10.33) or Require (Section 10.35) header field. The server SHOULD include a list of the unsupported extensions in an Unsupported header in the response. The client SHOULD retry the request, this time without using those extensions listed in the Unsupported header in the response.

11.4.16 480 Temporarily Unavailable

The callee's end system was contacted successfully but the callee is currently unavailable (e.g., not logged in, logged in in such a manner as to preclude communication with the callee or activated the "do not disturb" feature). 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.

11.4.17 481 Call Leg/Transaction Does Not Exist

This status is returned under three conditions: The UAS received a BYE request that does not match any existing call leg, the server received a CANCEL request that does not match any existing transaction or the UAS received an INVITE with a To tag that does not match the local tag value. (A server simply discards an ACK referring to an unknown transaction.) A UAC receiving a 481 to a request sent for an existing call leg MUST consider that call leg terminated.

11.4.18 482 Loop Detected

The server received a request with a Via (Section 10.46) path containing itself.

11.4.19 483 Too Many Hops

The server received a request that contains a Max-Forwards (Section 10.27) header with the value zero.

11.4.20 484 Address Incomplete

The server received a request with a To (Section 10.43) 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.

11.4.21 485 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.

11.4.22 486 Busy Here

The callee's end system was contacted successfully but the callee is currently not willing or able to take additional calls at this end system. 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.

11.4.23 487 Request Terminated

The request was terminated by a BYE or CANCEL request. This response is never returned for a CANCEL request itself.

11.4.24 488 Not Acceptable Here

The response has the same meaning as 606 (Not Acceptable), but only applies to the specific entity addressed by the Request-URI and the request may succeed elsewhere.

11.5 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.

11.5.1 500 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.

If the condition is temporary, the server MAY indicate when the client may retry the request using the Retry-After header.

11.5.2 501 Not Implemented

The server does not support the functionality required to fulfill the request. This is the appropriate response when a UAS does not recognize the request method and is not capable of supporting it for any user. (Proxies forward all requests regardless of method.)

11.5.3 502 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.

11.5.4 503 Service Unavailable

The server is currently unable to handle the request due to a temporary overloading (i.e., congestion) 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.

A client (proxy or UAC) receiving a 503 SHOULD attempt to forward the request to an alternate server. It SHOULD NOT forward any other requests to that server for the duration specified in the Retry-After header, if present. A proxy which receives a 503 SHOULD NOT forward it upstream unless it can determine that any subsequent requests it might proxy will also generate a 503. In other words, forwarding a 503 means that the proxy knows it cannot service any requests, not just the one for the Request-URI in the request which generated the 503.

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.

11.5.5 504 Server Time-out

The server did not receive a timely response from the server (e.g., a location server) it accessed in attempting to process the request. Note that 408 (Request Timeout) should be used if there was no response within the period specified in the Expires header field from the upstream server.

11.5.6 505 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.

11.5.7 513 Message Too Large

The server was unable to process the request since the message length exceeded its capabilities.

11.6 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.

11.6.1 600 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.

11.6.2 603 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.

11.6.3 604 Does Not Exist Anywhere

The server has authoritative information that the user indicated in the Request-URI does not exist anywhere. Searching for the user elsewhere will not yield any results.

11.6.4 606 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 10.47. 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.

12 SIP Message Body

12.1 Body Inclusion

Requests MAY contain message bodies unless otherwise noted. In this specification, the CANCEL request MUST NOT contain a message body.

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. 1xx responses to INVITE requests MAY contain session descriptions. Their interpretation depends on the response status code, but generally informs the caller what kind of session the callee is likely to establish, subject to later modification in the 2xx response. Request methods not defined in this specification MAY also contain session descriptions. 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 11.3. 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`.

12.2 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.

The "multipart" MIME type [39] MAY be used within the body of the message. Clients that send requests containing multipart message bodies MUST be able to send a session description as a non-multipart message body if the server requests this through an Accept header field.

12.3 Message Body Length

The body length in bytes SHOULD be given by the Content-Length header field. Section 10.18 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.)

13 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	
k	Supported	from “know”
l	Content-Length	
m	Contact	from “moved”
s	Subject	
t	To	
v	Via	

Thus, the message in section 20.2 could also be written:

```
INVITE sip:bob@example.com SIP/2.0
v:SIP/2.0/UDP 131.215.131.131;maddr=239.128.16.254;ttl=16
v:SIP/2.0/UDP 216.112.6.38
f:sip:alice@wonderland.com
t:sip:bob@example.com
m:sip:alice@mouse.wonderland.com
i:62729-27@216.112.6.38
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
a=rtpmap:0 PCMU/8000
```

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.

14 Behavior of SIP Clients and Servers

14.1 Multicast Unreliable Transport Protocols

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 scope. This MAY be done with either TTL or administrative scopes [28], 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 MUST be returned to the address listed in the maddr parameter of the Via header field. (This parameter is REQUIRED.) Generally, this will be a multicast address. Such multicast responses are multicast with the same TTL as the request, where the TTL is derived from the ttl parameter in the Via header (Section 10.46).

To avoid response implosion, servers MUST NOT answer multicast requests with a status code other than 2xx, 401, 407, 484 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, 401, 407 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.

14.2 Reliable Transport Protocols

A single reliable transport connection such as TCP 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.

The server SHOULD NOT close the connection until it has sent its final response (and possibly received the ACK), 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. These requests can be for the same transaction or call, or for totally different transactions or calls. There is no requirement that a transaction must complete before a new one is initiated on an existing connection. As a result, a server MUST support receiving a request for a new transaction on an existing connection before the previous transaction on the same connection has completed.

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.

14.3 Reliability for Requests Other Than INVITE

14.3.1 Unreliable Transport Protocols

A SIP client using an unreliable transport protocol such as UDP SHOULD retransmit requests other than INVITE or ACK 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 reaches $T2$. 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 500 ms and 4 s, 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 500 ms. 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 10.42), 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 14.4, responses to these requests are *not* retransmitted periodically and *not* acknowledged via ACK.

14.3.2 Reliable Transport Protocol

Clients using a reliable transport protocol such as TCP, SCTP or TLS do *not* need to retransmit requests, but MAY give up after receiving no response for an extended period of time.

14.4 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".

14.4.1 Unreliable Transport Protocols

A SIP client using an unreliable transport protocol 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 seven request packets. If no response (final or provisional) is received after sending seven request packets, processing continues as if a 481 response was received for that request (no ACK is generated, however), and a CANCEL SHOULD NOT be sent.

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 until it reaches $T2$ seconds. Response retransmissions cease when an ACK request is received or the response has been transmitted seven times. The entity generating the ACK (a UAC for 2xx responses, UAC or proxy for non-2xx) retransmits the ACK on receipt of a response retransmission. The value of a final response is not changed by the arrival of a BYE or CANCEL request.

Servers SHOULD only send a single 401 or 407 status response upon receiving a request that is not authenticated at either the SIP, transport or network layer. (See Section 18.4)

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.

Fig. 10 and 11 show the client and server state diagram for INVITE transactions. The "terminated" event occurs if the server receives either a CANCEL or BYE request. Note that the state diagram only shows the behavior for the INVITE transaction; the responses for BYE and CANCEL are not shown and follow the rules laid in Section 14.3.

> The mechanism in Sec. 14.3 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.

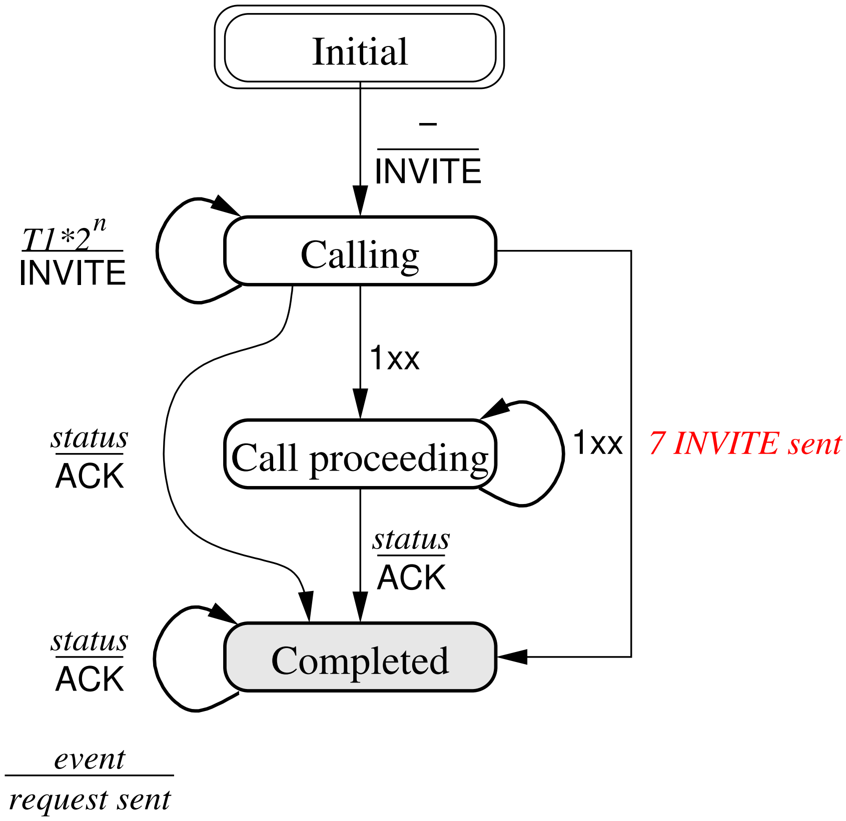

Figure 10: State transition diagram of client for INVITE method

14.4.2 Reliable Transport Protocol

A user agent using a reliable transport protocol such as TCP, SCTP or TLS MUST NOT retransmit requests, but uses the same algorithm as for unreliable transport protocols (Section 14.4.1) to retransmit responses until it receives an ACK. A client MAY give up on the request if there is no response within a client-defined timeout interval.

> It is necessary to retransmit 2xx responses as their reliability is assured end-to-end only. If the chain of proxies has an unreliable transport protocol 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.

14.5 ICMP Handling

Handling of ICMP messages in the case of unreliable transport protocol 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.

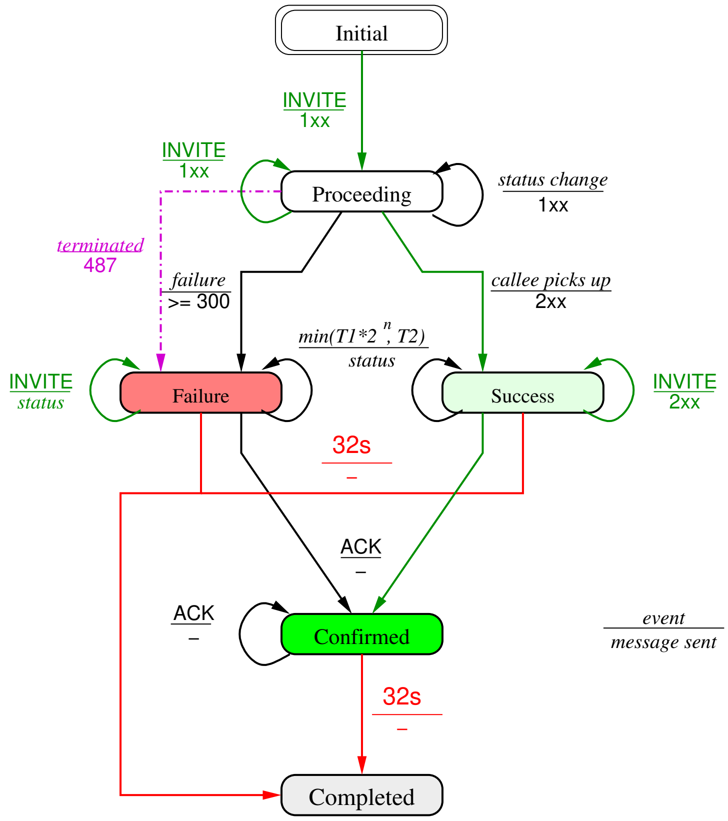

Figure 11: State transition diagram of server for INVITE method

15 Behavior of SIP User Agents

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

15.1 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, and remains unaltered as the request traverses proxies. The Request-URI contains the same address, but may be rewritten by proxies. The From field contains the address of the caller. It MUST contain a tag. A UAC MUST add a Contact header containing an address where it would like to be contacted for transactions from the callee back to the caller.

If the UAC desires to end the call before a response is received to the INVITE, it SHOULD send a CANCEL. This CANCEL will normally result in a 487 response to be returned to the INVITE, indicating successful cancellation. However, it is possible that the CANCEL and a 200 class response to the INVITE "pass on the wire". In this case, the UAC will receive a 2xx to the INVITE. It then terminates the call by following the procedures described in Section 15.4.

15.2 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. 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 MUST 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 tags. These become the local and remote addresses of the call leg, respectively.

If an ACK never arrives for the 2xx to an initial INVITE transaction, the UAS considers the call down and SHOULD send a BYE.

15.3 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, with a distinct call leg identifier. The call leg identifier is defined as the combination of the remote address, local address, and Call-ID. The local address is the value of the From field, including the tag, in the 2xx responses (they will all be the same). The remote address is the value of the To field, including the tag. Each 2xx response to the INVITE will differ in the value of the tag in the To field.

The UAC MUST generate an ACK request for each distinct call leg created by a 2xx. The Request-URI and Route headers for the ACK are constructed as described in Section 16.4. The To field in the ACK MUST contain the remote address for the call leg (which includes the tag). The From field in the ACK MUST contain the local address for the call leg. The Call-ID MUST contain the Call-ID for the call leg. The Via header in the ACK MUST be indentical to the one in the request being acknowledged, including any branch parameter. The CSeq number MUST be the same as the INVITE being acknowledged, but the CSeq method MUST be ACK. The ACK might possibly require a session description in the body. See Section B for guidelines.

If, after acknowledging any 2xx response to an INVITE, the caller does not want to continue with that call leg, then the caller MUST terminate the call-leg by sending a BYE request as described in Section 15.4.

15.4 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. It MAY initiate other requests as needed. A UA MUST NOT initiate a new INVITE transaction within a call leg while one is in progress. A UA MUST NOT initiate a new regular transaction while a regular transaction is in progress. However, a UA MAY initiate a regular transaction while an INVITE transaction on the same call leg is in progress.

If a UAS never receives an ACK for a 2xx response to a re-INVITE, it SHOULD generate a re-INVITE itself in order to synchronize the state of the session.

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 tags). A UAC copies the tag from the final response into the ACK, but it MUST NOT copy the tag into any subsequent requests unless the response was a 200-

class response to an INVITE request. The To field of CANCEL requests always contain exactly the same value as the request it is cancelling.

For an INVITE, the Contact header field MAY be different than the Contact header field sent in a previous response or request.

The callee's requests use the caller's To header field value as the From header value and the From header field value as the To header field value.

The network destination and Request-URI of requests is determined according to the following rules:

- If the response from the previous request contained a Record-Route header field, the UAC sends the request to the last entry in the list and removes that entry. As described in Section 10.34, the Request-URI is set to that value.
- Otherwise, if the response for the previous request contained a Contact header field, the request is directed to the host and port identified there. The Request-URI is set to the value of the Contact header. The request does not contain a Route header field in this case.
- Otherwise, the Request-URI contains the same URL as the To header.

If the UAC is configured with the address of an outbound proxy server, the UAC sends the request there, independent of the Request-URI. The outbound proxy is **NOT** named in the Request-URI. If there is no outbound proxy server, the Request-URI determines the network destination.

If a UAC does not support DNS resolution or the full Record-Route/Route mechanism, it MAY send all requests to a locally configured outbound proxy. In that case, that proxy behaves as described above. The UAC MUST, however, perform the mapping of Record-Route to Route header fields and MUST include *all* Route header fields, i.e., the UAC does not remove the first Route header field.

15.5 Receiving Subsequent Requests

When a request is received during a call, 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.

   It is possible that the To header in an INVITE request has a tag, but the UAS believes this to be a new call. This will occur if the UAS crashed and rebooted in the middle of a call, and the UAC has sent what it believes to be a re-INVITE. The UAS MAY either accept or reject the request. Accepting the request provides robustness, so that calls can persist even through crashes. UAs wishing to support this capability must choose monotonically increasing CSeq numbers even across reboots. This is because subsequent requests from the crashed-and-rebooted UA towards the other UA need to have a CSeq number higher than previous requests in that direction.

   Note also that the crashed-and-rebooted UA will have lost any Route headers which would need to be inserted into a subsequent request. Therefore, it is possible that the requests may not be properly forwarded by proxies.

   RTP media agents allowing restarts need to be robust by accepting out-of-range timestamps and sequence numbers.

If the UAS wishes to reject the INVITE, because it does not wish to recreate the call, it MUST respond to the request with a 481 status code. A UAC receiving a 481 response for any mid-call request (INVITE or otherwise) MUST consider that call terminated.

2. If the To, From, Call-ID, CSeq, Request-URI, and branch-ID in the topmost Via 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. It is possible for the CSeq header to be higher than the previous by more than one. This is not an error condition, and a UAS SHOULD be prepared to receive and process requests with CSeq values more than one higher than the previous received request. A request on an existing call leg with a lower CSeq MUST be rejected.

16 Routing of Requests

Record routing is the process whereby a proxy server can request to receive all messages between two UAs for a particular call leg. The proxy accomplishes this by inserting a header, called Record-Route into the initial request that begins the call leg. The user agents use these headers to construct a set of Route headers, that gets inserted into subsequent requests in the call leg. The Route headers contain a set of proxies that the request must visit on its way from one UA to another. Proxies use these to forward the requests, much like strict IP source routing.

The process of record-routing works for any SIP request that initiates some kind of session. For this specification, that includes only INVITE. Extensions MAY identify new requests as ones which initiate sessions, in which case the procedures defined here apply to processing of those requests.

All user agents MUST support the processing rules below which apply to them. As such, they MUST be able to parse and process both Record-Route and Route headers. Proxies MAY support the record routing procedures of Section 16.3, but they MUSTsupport the route header procedures of Sectionsec:rr:proxy2.

This is a change from RFC2543, where all record-route and route processing was optional for user agents.

The syntax for the Route header is described in Section 10.38. The syntax for the Record-Route header is described in Section 10.34.

16.1 UAC Processing for initial transaction

The UAC formulates its *initial* request for the session as defined in section 15.1. If the final response is a 200 class response, it may contain Record-Route headers, and may contain a Contact header.

Contact was not mandatory in RFC2543. Thus, if the UAC is talking to an older UAS, the UAS might not insert the Contact header. Thus, this text says that the Contact "may" be present in the response. Note also that this may is lower case; it is NOT saying that a UAS compliant to this specification can optionally insert the Contact header into the 200-class response.

The UAC MUST construct a *route set*, defined as a list of URIs, in the following manner:

1. The list of URIs present in the Record-Route headers in the 200 class response are taken, if present, including all URI parameters, and their order is reversed.

2. The URI in the Contact header from the 200 class response, if present, is taken, including all URI parameters, and appended to the end of the list from the previous step.

3. The list of URIs resulting from the above two operations is referred to as the *route set*.

The UAC MUST store the route set for the duration of the call leg. NOTE that it is possible for the route set to be empty. This will occur if neither Record-Route headers nor a Contact header were present in the 200 class response. Since there may be multiple 200 OK responses to an INVITE request, each response constitutes a separate call leg, and thus has a separate route set. The UAC MUST also remember whether the bottom-most entry in the route set was constructed from a Contact header or not. This is a boolean value, which we refer to as CONTACT_SET.

An ACK request for the 200 class response to an initial INVITE transaction MUST be formulated according to the rules of Section 15.3, as if it were a subsequent request within the call leg. That is, the ACK for a 200 class response contains Route headers.

Record-Route headers MAY be present in a provisional response to INVITE. In this case, the UAC can construct a route set for the call-leg associated with that provisional response, in the same way it would construct a route set for a 200 class response. This route set will only be needed if the UAC sends a request to the far end of the call leg before the initial INVITE transaction completes.

16.2 UAS Processing of initial transaction

When a UAS receives a request for a new call session, and it responds with a 200 class response, it MUST copy the contents of the Record-Route headers from the request to the response. This includes the URIs, URI parameters, and any Record-Route header parameters, whether they are known or unknown to the UAS.

Record-Route parameters are very useful for proxies. They allow the proxy to match the record-route entry in the response with the one in the request.

The order of the Record-Route headers MUST be preserved in the response.

The UAS MUST construct a *route set* in the following manner:

1. The list of URIs in the Record-Route headers in the initial request, if present, are taken, including any URI parameters.

2. The URI in the Contact header from the request, if present, is taken, including any URI parameters. The URI is appended to the bottom of the list of URIs from the previous step.

3. The resulting list of URIs is called the *route set*.

The UAS MUST store the route set for the duration of the call leg. It is possible for the route set to be empty. This will occur if neither Record-Route headers nor a Contact header were present in the initial request. The UAS MUST also remember whether the bottom-most entry in the route set was constructed from a Contact header or not. This is effectively a boolean value, which we refer to as CONTACT_SET.

If the UAS sends provisional responses before the session is accepted, it SHOULD copy the Record-Route headers from the request into the provisional responses in the same manner described above for the 200 class response.

16.3 Proxy procedures for record routing a transaction

Each proxy MAY independently decide to record-route a transaction that initiates a session. Amongst the methods defined in this specification, that includes only the INVITE transaction. However, extensions MAY designate new methods as ones that initiate a session of some sort. In that case, the procedures described here apply to those requests. Both the initial request that initiates the session, and any refreshes (such as a re-INVITE) MUST be record-routed if the the initial request was record routed. This means a proxy will often need to record-route requests that contain Route headers. Generally, the choice about whether to record-route or not is a tradeoff of features vs. performance. Faster request processing and higher scalability is achieved when proxies do not record route. However, provision of certain services may require a proxy to observe all messages for a call leg. It is RECOMMENDED that proxies do not automatically record route. They should do so only if specifically required.

To record-route, the proxy inserts a Record-Route header into the request before proxying it onwards. A forking proxy MAY insert a different Record-Route header into each forked request. The Record-Route header that it inserts MUST be inserted as the first Record-Route header, appearing before any existing ones in the request. The URL in the header MUST be a SIP URL. . The URL SHOULD NOTcontain the transport parameter, but can in private networks where it is known that elements on both sides of the proxy support the specific transport. The URL MUST have the property that when the processing described in [36] is followed for that URL, the result of the lookup is an address/port of the proxy server inserting the Record-Route. The URL and the proxy configuration SHOULD be such that if a request is received with this URL in the Request-URI, the proxy's normal request processing will cause it to be forwarded to one of the previous-hop servers that the request traversed, including the UAS.

> These two properties are important. The first one guarantees that subsequent requests from the called party are routed back to this actually proxy. The second property is there for robustness. It guarantees that the request URI always contain meaningful information, even if there are no Route headers that tell the proxy where to forward the request to next.

The URL placed into the Record-Route header MUST be unique for each unique Request-URI in the request, but must not equal the Request-URI of the request (they will not be equal if the proxy adds an maddr parameter). The proxy MAY insert Record-Route header parameters into the request. These will be returned to the proxy in any 200 class response to the INVITE, and are useful for pushing state into the message.

If a 200 class response arrives for the proxied request, the response will contain the entire list of Record-Route headers inserted by proxies along the request path. The proxy MAY modify the Record-Route value matching the one it inserted into the request. Like the URL in the request, it MUST be a SIP URL and MUST NOTcontain a transport parameter. The URL in this Record-Route header MUST have the property that when the processing described in [36] is followed for that URL, the result of the lookup is an address/port of the proxy server that inserted the Record-Route. The URL in this header, and the proxy configuration SHOULD be such that if a request is received with this URL in the Request-URI, the proxy's normal request processing will cause it to be forwarded to the same next hop server that the original request was forwarded to. The URL placed into the Record-Route header MUST be unique for each unique Request-URI in the request, but must not equal the Request-URI of the request (they will not be equal if the proxy adds an maddr parameter).

The four properties (two for the request, two for the response), can be satisfied in a number of ways. One way is that the URL inserted into the Record-Route in the request is nearly the same as the Contact header in the initial request (if present, else the From field), but with the maddr and port set to resolve to the proxy, and with a transaction identifier added to the user part of the request-URI (in order to meet the requirement that the URI in the Record-Route be different for each distinct Request-URI). Then, the proxy modifies the URL in the Record-Route header in the response, setting it to be the URL from the Request-URI of the initial request, but with the maddr and port set to resolve to the proxy.

As an example, consider a proxy at 10.0.1.1 listening on port 5061 which receives the following request (many headers are omitted for brevity):

```
INVITE sip:user@example.com SIP/2.0
Via:  SIP/2.0/UDP callerspc.univ.edu
Contact:  sip:caller@callerspc.univ.edu
```

The proxy forwards this request to `sip:j_user@div11.example.com`, and record-routes:

```
INVITE sip:j_user@div11.example.com SIP/2.0
Via: SIP/2.0/UDP 10.0.1.1:5061
Via: SIP/2.0/UDP callerspc.univ.edu
Record-Route: <sip:caller.8jjs0@callerspc.univ.edu:5061;maddr=10.0.1.1>
Contact:  sip:caller@callerspc.univ.edu
```

The 200 response received by the proxy will look like, in part:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 10.0.1.1:5061
Via: SIP/2.0/UDP callerspc.univ.edu
Record-Route: <sip:caller.8jjs@callerspc.univ.edu:5061;maddr=10.0.1.1>
Contact: sip:j_user@host32.div11.example.com
```

The proxy modifies its Record-Route header in the response, resulting in the new response forwarded upstream:

```
SIP/2.0 200 OK
Via:  SIP/2.0/UDP callerspc.univ.edu
Record-Route:  <sip:user@example.com:5061;maddr=10.0.1.1>
Contact:  sip:j_user@host32.div11.example.com
```

The route set computed by the UAS is:

```
sip:caller.8jjs@callerspc.univ.edu:5061;maddr=10.0.1.1
sip:caller@callerspc.univ.edu
```

and the route set computed by the UAC is:

```
sip:j_user@example.com:5061;maddr=10.0.1.1
sip:j_user@host32.div11.example.com
```

There are other ways to meet these requirements. The proxy could construct a URL for the request which encodes all of the needed information, by placing it in the user portion, for example. A call stateful proxy could insert a URL into the request with the form sip:proxy.example.com, and not modify it in the response. This URL clearly satisfies the first required property (of getting routed back to the proxy that inserted it). The second property can be maintained by being call stateful, and extracting the needed parameters from local storage.

When a proxy does decide to modify the Record-Route header in the response, one of the operations it must perform is to locate the Record-Route that it had inserted. If the request spiraled, and the proxy inserted a Record-Route in each iteration of the spiral, locating the correct header in the response (which must be the proper iteration in the reverse direction) is tricky. Note that the rules above dictate that a proxy insert a different URI into the Record-Route for each distinct Request-URI received. The two issues can be solved jointly. A RECOMMENDED mechanism is for the proxy to append a piece of data to the user portion of the URL. This piece of data is a hash of the transaction key for the incoming request, concatenated with a unique identifier for the proxy instance. Since the transaction key includes the Request-URI, this key will be unique for each distinct Request-URI. When the response arrives, the proxy modifies the first Record-Route whose identifier matches the proxy instance. The modification results in a URI without this piece of data appended to the user portion of the URI. Upon the next iteration, the same algorithm (find the topmost Record-Route header with the parameter) will correctly extract the next Record-Route header inserted by that proxy.

16.4 UA Processing of Subsequent Requests in a Call Leg

When a UA wishes to send another request for the call-leg (such as a BYE or INVITE), it follows the procedures defined in this subsection. The procedures here MUST also be followed for an ACKrequest for a 200 response to the initial INVITE for a call leg. The procedures here MUST NOTbe followed for a CANCELrequest or an ACKrequest for a non-200 class response. This implies that CANCELfor an initial INVITE never contains a Route header, nor does an ACKfor a non-200 response.

The request is constructed as specified in Section 15.4. The UA then takes the list of URI in the route set. The top URI is inserted into the request URI of the request, including all parameters. Any URI parameters not allowed in the request URI MUST then be stripped. Each of the remaining URIs (if any) from the route set, including all URI parameters, are placed into a Route header into the request, in order. The procedures of [36] are then applied to the URI in the request URI, the the request is forwarded to the resulting server.

If a UAS has a route set for a call leg, and receives a refresh for that call leg containing Record-Route headers (the only refresh defined in this specification is a re-INVITE), it MUST copy those headers into any 200 class response to that request. If the boolean variable CONTACT_SET is true, the Contact header in the request (if present) replaces the last entry in the route set. If the boolean variable CONTACT_SET is false, the UAS MUST add the URL in the Contact header in the re-INVITE to the bottom of the route set, and then set CONTACT_SET to true. If the request did not contain a Contact header, the route-set at the UAS remains unchanged.

Similarly, if a UAC has a route set for a call leg, and receives a 200 class response to a refresh it sent, the Contact header is examined. If not present, the route set remains unchanged. If the response had a Contact header, and the boolean variable CONTACT_SET is false, the URL in the Contact header in the response is added to the bottom of the route set, and CONTACT_SET is set to true. If the re-INVITE response had a Contact header, and CONTACT_SET is true, the URL in the Contact header of the re-INVITE response replaces the bottom value in the route set.

The above two paragraphs allow a UA to update its Contact address mid-call, but proxies cannot update their route address. Once on the route, a proxy remains on the route for the duration of the call leg.

Why the different treatment for Contact and Record-Route in a re-INVITE? It has to do with backwards compatibility. RFC2543 did not mandate that a proxy needs to refresh its record-route headers. As a result, the lack of a record-route in a re-INVITE cannot be interpreted to mean that the proxy does not want to be included on the route any longer. Updating of the Contact header mid-call is useful for mobility applications, and is also useful for forms of third party call control

.

16.4.1 Local outbound proxies

Special considerations exist for local outbound proxies.

A UA which uses a local outbound proxy will send all requests without Route headers to that proxy. Typically, this includes initial INVITE requests for a call.

If a local outbound proxy wishes to remain on the SIP messaging path for a call leg, it MUST record-route using the procedures above. .

A UA which uses a local outbound proxy, and attempts to send a subequent request in a call leg with a route set, SHOULD use the procedures in Section 16.4. However, in some instances, a UA may not be capable of DNS, and therefore may not be able to follow those procedures. In this case, the UA MAY send the request to its local outbound proxy. In this case, it MUST NOT remove the top Route header. It sets the Request-URI to the same value it used for the initial request, and sends it to its local outbound proxy.

16.5 Proxy routing procedures

The rules in this section MUST be followed by all proxies to determine the appropriate routing procedures to apply to a request. Proxies MAY respond to requests with Route headers for any reason (in order to perform proxy authorization, for example). But, if the request is to be proxied, the procedures here MUST be used to determine where the request is proxied to. The procedures in this section apply indepedent of the request method. They MUST be followed even if the request method is unknown to the proxy. Since CANCEL can never contain Route, these procedures never apply to CANCEL. However, they do apply to ACK requests for a "200 OK" response, which do contain Route headers.

When a proxy receives a request, it MUST check for the existence of a Route header. If one exists, it MUST pop that Route header, and place it (including all URI parameters) into the Request-URI. Any URL parameters present in the top Route header which are not allowed in the request URI MUST be removed from the Request-URI by the proxy.It SHOULD be sent using the procedures of [36]. If no Route header is present, the proxy examines the Request-URI. If it contains an maddr parameter, and the address in maddr is not an interface that the proxy is listening on, the proxy SHOULD forward the request using the procedures of [36] on the Request-URI. This will cause it to be forwarded to the address in the maddr. If the maddr is present, but is an interface the proxy is listening on, the proxy MUST strip the maddr, and then continue processing as if it were never there. If there was no maddr (or if it was stripped in the previous step), and the domain of the Request-URI is not a domain the proxy is managing, the procedures of [36] SHOULD be used to forward the request to that URI. If the domain is one the proxy is managing, the request is processed by whatever policies are desired by the administrator.

Stateless proxies are limited in the policies they may apply for routing. Specifically, stateless proxies MUST have routing policies that can be expressed as a combinatoric function of the fields of the incoming request and static configuration parameters. This explicitly excludes probabilistic routing, time-of-day routing, or other dynamic mechanisms. As a result of this restriction, a proxy that supports run-time configuration changes will need to process transactions statefully for some time if the configuration change affects routing.

16.6 Pre-Loaded Route Headers

Normally, a UA constructs the Route headers in a request from the route set learned through Record-Route headers. However, in some circumstances, it is useful for a UA to insert Route headers into an initial request. These headers may have been learnt by the UA through some out of bands means. When an initial request (initial as far as the UAC and UAS are concerned) contains Route headers, this is referred to as a "pre-loaded Route". It is equivalent to strict source routing in IP. Proxies will often not be able to distinguish this case from the case described in Section 16.3, and will therefore properly use these route headers to forward the request. If the proxies are interested in receiving subsequent messages for the call leg, their insertion of Record-Route as mandated by Section 16.3 will establish a correct route set at both UAC and UAS. This route set may end up being different from the pre-loaded Route used by the UAC. As such, a UAC that inserts a pre-loaded route set MUST follow the procedures of Section 16.1 in processing the response to this initial INVITE.

17 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.

17.1 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.

17.2 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.

17.3 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 MUST be stateful. Proxies that accept TCP connections MUST be stateful when handling the TCP connection. A proxy sending a request out to a multicast address 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 similar to a virtual UAS/UAC, but cannot be viewed as just a UAS and UAC glued together at the back. (In particular, it does not originate requests except ACK and CANCEL.) 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 forwards every request it receives downstream, and every response it receives upstream.

17.3.1 Proxying Requests

A proxy server MUST check for forwarding loops before proxying a request. A request has been looped if the server finds its own address in the Via header field **and** the hash computation over the fields enumerated in Section 10.46.6 yields the same value as the hash part of the "branch" parameter in the Via entry containing the proxy server's address.

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

The proxy server MUST copy all request header fields to the outgoing request. It MAY add other header fields.

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 10.46).

Proxies other than outbound proxies SHOULD change the Request-URI to indicate the server where it intends to send the request.

17.3.2 Proxying Responses

A proxy only processes a response if the topmost Via field matches one of its addresses (see Section 10.46). A response with a non-matching top Via field MUST be dropped.

17.3.3 Stateless Proxy: Proxying Responses

A stateless proxy MUST follow the procedures in Section 10.46 in order to determine where to forward the response to.

A stateless proxy MUST NOT generate its own provisional responses. It MUST forward all provisional responses, including 100, upstream.

17.3.4 Stateful Proxy: Receiving Requests

When a stateful proxy receives a request, it checks the To (including tags), From (including tags), Call-ID, CSeq, Request-URI and topmost Via 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.

17.3.5 Stateful Proxy: Receiving ACKs

When an ACK request is received, it is proxied unless the request's To (including the tag), From, CSeq and Call-ID header fields match those of a (non-2xx) response sent by the proxy. In that case, the request is processed locally and stops retransmissions of responses.

17.3.6 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. 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 17.4 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.

17.3.7 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 17.4.

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.

17.4 Forking Proxy

The server must respond to the request (other than ACK) immediately with a 100 (Trying) response if it expects to take more than 200 ms to obtain a final 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 10.34).

The following C-code describes the behavior of a proxy server issuing several requests in response to an incoming INVITE request with method R which is to be proxied to a list of N destination enumerated in '*address*', with an expiration time of '*expires*' seconds.

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[], int expires)
{
  struct {
    char *branch;         /* branch token */
    int branch_seq;       /* branch sequence number part */
    int done;             /* has responded */
  } outgoing[];
  char *location[];       /* list of locations */
  int heard = 0;          /* number of sites heard from */
  int class;              /* class of status code */
  int timeleft = expires; /* expiration 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 */
    char *branch;         /* branch token */
    int branch_seq;       /* branch sequence number */
  } 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 = "";
    outgoing[i].branch_seq = 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_seq == outgoing[i].branch_seq) {
          outgoing[i].done = 1;
          break;
        }
```

```
      }
    }
    /* CANCEL: respond, fork and wait for responses */
    /* terminate INVITE with 40
    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 = 408; /* request expired */
    }
    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 SHOULD send a CANCEL to all incomplete branches and return the best available final status to the client if not all responses have been received after 60 seconds or the expiration period specified in the Expires header field of the request. If no responses have been received, the proxy returns a 408 (Timeout) response to the client.

When forwarding responses, a proxy MUST forward the whole response, including *all* header fields of the selected response as well as the body.

1xx: The proxy MUST forward provisional responses greater than 100 upstream towards the client and SHOULD NOT forward 100 (Trying) responses.

2xx: If the request was an INVITE, the proxy MUST forward the response upstream towards the client, without sending an ACK downstream. For other requests, it should only forward the response upstream if it has not forwarded any other responses upstream.

After receiving a 2xx, the server SHOULD terminate all other pending requests by sending a CANCEL request. (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.)

If the request was not an INVITE, the proxy SHOULD drop 2xx responses if it had already forwarded a final response upstream.

3xx: For INVITE requests, the proxy MUST send an ACK. It MAY recurse on the listed Contact addresses. Otherwise, the lowest-numbered response is returned if there were no 2xx or 6xx 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: For INVITE requests, the proxy MUST send an ACK. It remembers the response if it has a lower status code class than any previous 4xx and 5xx response. On completion, a response with the lowest response class is returned if there were no 2xx, 3xx or 6xx responses. Within the set of responses from the lowest-numbered class, the proxy server may choose any response.

The proxy SHOULD collect all WWW-Authenticate and Proxy-Authenticate headers from all 401 and 407 responses and return all of them in the response if either 401 or 407 is the lowest-numbered response.

6xx: For INVITE requests, the proxy sends an ACK. It forwards the 6xx response unless a 2xx response has been received. Other pending requests SHOULD be terminated with CANCEL as described for 2xx responses. Unlike for 2xx responses, only one 6xx response is forwarded, since ACKs are generated locally.

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.

18 Security Considerations

18.1 Confidentiality and Privacy: Encryption

18.1.1 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 two 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;

The SIP request or response cannot be encrypted end-to-end as a whole 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.

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.

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 "r" in the "proxy" 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.

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>;tag=7abm$
  Encryption: PGP version=5.0$
  Content-Length: 224$
  Call-ID: 187602141351@worcester.bell-telephone.com$
  Content-Type: message/sip
  CSeq: 488$
  $
*******************************************************
* Subject: Mr. Watson, come here.$                    *
* Content-Type: application/sdp$                      *
* $                                                   *
* v=0$                                                *
* o=bell 53655765 2353687637 IN IP4 128.3.4.5$        *
* s=Mr. Watson, come here.$                           *
* t=0 0$                                              *
* 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.

A Content-Type indication of "message/sip" MAY be added, but will be overridden after receipt.

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>;tag=7abm$
   Encryption: PGP version=5.0$
   Content-Type: application/sdp$
   Content-Length: 107$
   Call-ID: 187602141351@worcester.bell-telephone.com$
   CSeq: 488$
   $
*************************************************
* $                                             *
* 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$                 *
*************************************************
```

18.1.2 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 behavior is explicit, it would not always be possible for the called user agent to infer what a reasonable behavior 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 (Section 10.36). 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.

18.1.3 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.

18.1.4 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 [40] or TLS [22]. The use of a particular mechanism will generally need to be specified out of band, through manual configuration, for example.

18.2 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 10.48) and Authorization (Section 10.11) header field and their Proxy- counterparts to include cryptographically strong signatures. SIP also supports the HTTP "basic" and "digest" schemes (see Section 19) and other HTTP authentication schemes to be defined that offer a rudimentary mechanism of ascertaining the identity of the caller.

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.

Generally, SIP authentication is for a specific request URI and realm, a protection domain. Thus, for basic and digest authentication, each such protection domain has its own set of user names and secrets. If a user agent does not care about different request URIs, it makes sense to establish a "global" user name, secret and realm that is the default challenge if a particular request URI does not have its own realm or set of user names. Similarly, SIP entities representing many users, such as PSTN gateways, MAY try a pre-configured global user name and secret when challenged, independent of the request URI.

SIP does not dictate which digital signature scheme is used for authentication. As indicated above, SIP implementations MAY also use "basic" and "digest" authentication and other authentication mechanisms defined for HTTP [41]. 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>;tag=7abm
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
s=Mr. Watson, come here.
t=0 0
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>;tag=7abm
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
s=Mr. Watson, come here.
t=0 0
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;
- The To and From header fields always include the < and > delimiters even if the display-name is empty.

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: signed-response request header field. The client indicates the desired authentication method via the WWW-Authenticate header.

The correct behavior in handling unauthenticated responses to a request that requires authenticated responses is described in section 18.2.1.

18.2.1 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 behavior 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).

18.3 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.

18.4 Denial of Service

Attackers can spoof a Via header field to direct responses to a third party, using a SIP UAS or proxy to generate traffic. This attack can be prevented by requiring an existing security association, such as TLS or IPsec. This may be an appropriate solution, e.g., between proxy servers that exchange significant amounts of signaling traffic or between a user agent and its outbound proxy.

If such a security association is not feasible, clients and proxies SHOULD respond to unauthenticated requests with only a *single* 401 (Unauthorized) or 407 (Proxy Authentication Required) instead of using the normal response retransmission algorithm. Retransmitting the 401 or 407 status response amplifies the problem of an attacker using a spoofed Via header address to to direct traffic to a third party.

18.5 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.

19 SIP Authentication using HTTP Basic and Digest Schemes

SIP implementations MAY use HTTP's basic and digest authentication mechanisms (RFC 2617 [41]) 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 [41]. This section outlines this operation, pointing to RFC 2617 [41] for details, and noting the differences when used in SIP. Since RFC 2543 is based on HTTP basic and digest as defined in RFC 2069 [42], SIP servers supporting RFC 2617 MUST ensure they are backwards compatible with RFC 2069. Procedures for this backwards compatibility are specified in RFC 2617.

19.1 Framework

The framework for SIP authentication parallels that for HTTP (RFC 2617 [41]). 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 RFC 2617 [41].

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.

19.2 Basic Authentication

The rules for basic authentication follow those defined in [41, 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 [41, 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.

Due to its weak security, the usage of basic authentication is NOT RECOMMENDED. However, servers SHOULD support it to handle older RFC 2543 clients that might still use it.

19.3 Digest Authentication

The rules for digest authentication follow those defined in [41, 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 in RFC 2617 has an error in that the URI is not enclosed in quotation marks. (The example in Section 3.5 is correct.) For SIP, the URI MUST be enclosed in quotation marks.

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

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

4. The example procedure for choosing a nonce based on Etag does not work for SIP.

5. The text in RFC 2617 [41] regarding cache operation does not apply to SIP.

6. RFC 2617 [41] 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.

RFC2543 did not allow usage of the Authentication-Info header (it effectively used RFC 2069). However, we now allow usage of this header, since it provides integrity checks over the bodies and provides mutual authentication. RFC2617 [41] defines mechanisms for backwards compatibility using the qop attribute in the request. These mechanisms MUST be used by a server to determine if the client supports the new mechanisms in RFC 2617 that were not specified in RFC 2069.

19.4 Proxy-Authentication

The use of the Proxy-Authentication and Proxy-Authorization parallel that as described in [41, Section 3.6], with one difference. Proxies MUST NOT add the Proxy-Authorization header. 407 (Proxy Authentication Required) 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 10.31 and 10.32 for additional information on usage of these fields as they apply to SIP.

It is also possible that a 401 (Unauthorized) response contains several challenges, from a mixture of proxies and user agent servers, if the request was forked.

20 Examples

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

20.1 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>;tag=19al
      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>;tag=19al
      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>;tag=19al
       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>;tag=7eff
       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 sent 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.

20.2 Invitation to a Multicast Conference

The first example invites `bob@example.com` to a multicast session. All examples use the Session Description Protocol (SDP) (RFC 2327 [6]) as the session description format.

20.2.1 Request

```
C->S:  INVITE sip:bob@one.example.com SIP/2.0
       Via: SIP/2.0/UDP sip.example.com;branch=7c337f30d7ce.1
         ;maddr=239.128.16.254;ttl=16
       Via: SIP/2.0/UDP mouse.wonderland.com
       From: Alice <sip:alice@wonderland.com>;tag=1ija
       To: Bob <sip:bob@example.com>
       Call-ID: 602214199@mouse.wonderland.com
       CSeq: 1 INVITE
       Contact: Alice <sip:alice@mouse.wonderland.com>
```

```
      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
      t=3149328700 0
      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
      a=rtpmap:0 PCMU/8000
```

The From request header above states that the request was initiated by alice@wonderland.com and addressed to bob@example.com (To 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 sip.example.com. The second Via header field indicates that it was originally sent from the outbound proxy mouse.wonderland.com. The Request-URI indicates that the request is currently being being addressed to bob@one.example.com, the local address that the SIP server for the example.com domain 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.

20.2.2 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 sip.example.com;branch=7c337f30d7ce.1
        ;maddr=239.128.16.254;ttl=16
      Via: SIP/2.0/UDP mouse.wonderland.com
      From: Alice <sip:alice@wonderland.com>;tag=1ija
      To: Bob <sip:bob@example.com> ;tag=3141593
      Call-ID: 602214199@mouse.wonderland.com
      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 sip.example.com;branch=7c337f30d7ce.1
        ;maddr=239.128.16.254;ttl=16
      Via: SIP/2.0/UDP mouse.wonderland.com
      From: Alice <sip:alice@wonderland.com>;tag=1ija
      To: Bob <sip:bob@example.com> ;tag=3141593
      Call-ID: 602214199@mouse.wonderland.com
      CSeq: 1 INVITE
      Contact: <sip:bob@one.example.com>
```

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

```
C->S: ACK sip:bob@one.example.com SIP/2.0
      Via: SIP/2.0/UDP mouse.wonderland.com
      From: Alice <sip:alice@wonderland.com>;tag=1ija
      To: Bob <sip:bob@example.com> ;tag=3141593
      Call-ID: 602214199@mouse.wonderland.com
      CSeq: 1 ACK
```

20.3 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>;tag=3pcc
      To: T. Watson <sip:watson@bell-tel.com>
      Call-ID: 662606876@kton.bell-tel.com
      CSeq: 1 INVITE
      Contact: <sip:a.g.bell@kton.bell-tel.com>
      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.
      e=a.g.bell@bell-tel.com
      t=3149328600 0
      c=IN IP4 kton.bell-tel.com
      m=audio 3456 RTP/AVP 0 3 4 5
      a=rtpmap:0 PCMU/8000
      a=rtpmap:3 GSM/8000
      a=rtpmap:4 G723/8000
      a=rtpmap:5 DVI4/8000

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>;tag=3pcc
      To: T. Watson <sip:watson@bell-tel.com> ;tag=37462311
      Call-ID: 662606876@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>;tag=3pcc
      To: T. Watson <sip:watson@bell-tel.com> ;tag=37462311
      Call-ID: 662606876@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>;tag=3pcc
      To: T. Watson <sip:watson@bell-tel.com> ;tag=37462311
      Call-ID: 662606876@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>;tag=3pcc
      To: T. Watson <sip:watson@bell-tel.com> ;tag=37462311
      Call-ID: 662606876@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>;tag=3pcc
```

```
      To: <sip:watson@bell-tel.com> ;tag=37462311
      Call-ID: 662606876@kton.bell-tel.com
      CSeq: 1 INVITE
      Contact: sip:watson@boston.bell-tel.com
      Content-Type: application/sdp
      Content-Length: ...

      v=0
      o=watson 4858949 4858949 IN IP4 192.1.2.3
      s=I'm on my way
      e=watson@bell-tel.com
      t=3149329600 0
      c=IN IP4 boston.bell-tel.com
      m=audio 5004 RTP/AVP 0 3
      a=rtpmap:0 PCMU/8000
      a=rtpmap:3 GSM/8000
```

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 use PCMU and GSM. 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>;tag=3pcc
      To: T. Watson <sip:watson@bell-tel.com> ;tag=37462311
      Call-ID: 662606876@kton.bell-tel.com
      CSeq: 1 ACK
```

20.4 Terminating a Call

To terminate a call, the caller can send a BYE request formatted as follows:

```
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>;tag=3pcc
      To: T. A. Watson <sip:watson@bell-tel.com> ;tag=37462311
      Call-ID: 3298420296@kton.bell-tel.com
      CSeq: 2 BYE
```

If the callee wishes to terminate the call, it sends a BYE request as well. However, this BYE request has the contents of the To field from the above message in the From field, and the contents of the From field in the above message in the To field.

20.5 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 message bodies containing the session descriptions.

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>;tag=3pcc
      To:      T. Watson <sip:t.watson@ieee.org>
      Call-ID: 31415@c.bell-tel.com
      CSeq:    1 INVITE
      Contact: a.g.bell@c.bell-tel.com
```

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=3d8a50dbf5a28d.1
      Via:     SIP/2.0/UDP c.bell-tel.com
      From:    A. Bell <sip:a.g.bell@bell-tel.com>;tag=3pcc
      To:      T. Watson <sip:t.watson@ieee.org>
      Call-ID: 31415@c.bell-tel.com
      CSeq:    1 INVITE
      Contact: a.g.bell@c.bell-tel.com
```

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=3d8a50dbf5a28d.1
      Via:     SIP/2.0/UDP c.bell-tel.com
      From:    A. Bell <sip:a.g.bell@bell-tel.com>;tag=3pcc
      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=3d8a50dbf5a28d.1
      From:    A. Bell <sip:a.g.bell@bell-tel.com>;tag=3pcc
      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=3d8a50dbf5a28d.2
      Via:     SIP/2.0/UDP c.bell-tel.com
      From:    A. Bell <sip:a.g.bell@bell-tel.com>;tag=3pcc
      To:      T. Watson <sip:t.watson@ieee.org>
      Call-ID: 31415@c.bell-tel.com
      CSeq:    1 INVITE
      Contact: a.g.bell@c.bell-tel.com
```

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=3d8a50dbf5a28d.3
      Via:     SIP/2.0/UDP c.bell-tel.com
      From:    A. Bell <sip:a.g.bell@bell-tel.com>;tag=3pcc
      To:      T. Watson <sip:t.watson@ieee.org>
      Call-ID: 31415@c.bell-tel.com
      CSeq:    1 INVITE
      Contact: a.g.bell@c.bell-tel.com

P->Y: INVITE sip:watson@y.bell-tel.com SIP/2.0
      Via:     SIP/2.0/UDP sip.ieee.org ;branch=3d8a50dbf5a28d.4
      Via:     SIP/2.0/UDP c.bell-tel.com
      From:    A. Bell <sip:a.g.bell@bell-tel.com>;tag=3pcc
      To:      T. Watson <sip:t.watson@ieee.org>
      Call-ID: 31415@c.bell-tel.com
      CSeq:    1 INVITE
      Contact: a.g.bell@c.bell-tel.com
```

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=3d8a50dbf5a28d.3
```

```
      Via:       SIP/2.0/UDP c.bell-tel.com
      From:      A. Bell <sip:a.g.bell@bell-tel.com>;tag=3pcc
      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=3d8a50dbf5a28d.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>;tag=3pcc
      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=3d8a50dbf5a28d.2
      From:     A. Bell <sip:a.g.bell@bell-tel.com>;tag=3pcc
      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=3d8a50dbf5a28d.2
      From:     A. Bell <sip:a.g.bell@bell-tel.com>;tag=3pcc
      To:       T. Watson <sip:t.watson@ieee.org>
      Call-ID: 31415@c.bell-tel.com
      CSeq:     1 CANCEL
```

In addition, A responds to the original INVITE request with a 487 (Request Terminated):

```
A->P: SIP/2.0 487 Request Terminated
      Via:      SIP/2.0/UDP sip.ieee.org ;branch=3d8a50dbf5a28d.2
      Via:      SIP/2.0/UDP c.bell-tel.com
      From:     A. Bell <sip:a.g.bell@bell-tel.com>;tag=3pcc
      To:       T. Watson <sip:t.watson@ieee.org>;tag=0303
      Call-ID: 31415@c.bell-tel.com
      CSeq:     1 INVITE
```

This response terminates at P. P generates an ACK for the 487:

```
P->A: ACK sip:watson@acm.org SIP/2.0
      Via:     SIP/2.0/UDP sip.ieee.org ;branch=3d8a50dbf5a28d.2
      From:    A. Bell <sip:a.g.bell@bell-tel.com>;tag=3pcc
      To:      T. Watson <sip:t.watson@ieee.org>;tag=0303
      Call-ID: 31415@c.bell-tel.com
      CSeq:    1 ACK
```

Bell gets the two 200 responses from X and Y in short order and sends and ACK to both directly. Bell can now keep both call legs or terminate one with a BYE request. Here, he temporarily keeps both to determine where the real Watson is located.

```
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>;tag=3pcc
      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>;tag=3pcc
      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>;tag=3pcc
      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>;tag=3pcc
      To:       T. Watson <sip:t.watson@ieee.org>;tag=35253448
      Call-ID:  31415@c.bell-tel.com
      CSeq:     2 BYE
```

20.6 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>;tag=3pcc
      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
```

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>;tag=5h7j
      To: Alice <sip:alice@wonderland.com> ;tag=2332462
      Call-ID: 27182@caller.com
      Contact: sip:bob@example.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 `example.com` domain. Note that the server at `example.com` forwards the request to Bob based on the Request-URI.

```
C->B: INVITE sip:bob@example.com SIP/2.0
      Via: SIP/2.0/UDP h.caller.com
      From: <sip:charlie@caller.com>;tag=5h7j
      To: sip:alice@wonderland.com
      Call-ID: 27182@caller.com
      CSeq: 2 INVITE
      Contact: sip:charlie@h.caller.com
```

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

```
C->F: INVITE sip:alice@wonderland.com SIP/2.0
      Via: SIP/2.0/UDP h.caller.com
      From: <sip:charlie@caller.com>;tag=5h7j
```

```
       To: Alice <sip:alice@wonderland.com>
       Call-ID: 27182@caller.com
       CSeq: 1 INVITE
       Contact: sip:charlie@h.caller.com
```

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
       Via: SIP/2.0/UDP h.caller.com
       From: <sip:charlie@caller.com>;tag=5h7j
       To: Alice <sip:alice@wonderland.com>
       Call-ID: 27182@caller.com
       CSeq: 1 INVITE
       Contact: <sip:alice@wonderland.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@wonderland.com SIP/2.0
       Via: SIP/2.0/UDP h.caller.com
       From: <sip:charlie@caller.com>;tag=5h7j
       To: Alice <sip:alice@wonderland.com>
       Call-ID: 27182@caller.com
       CSeq: 2 INVITE
       Contact: sip:charlie@h.caller.com
```

20.7 Negotiation

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

```
S->C: SIP/2.0 606 Not Acceptable
       Via: SIP/2.0/UDP c.bell-tel.com
       From: A. Bell <sip:a.g.bell@bell-tel.com>;tag=3ab6
       To: T. Watson <sip:t.watson@ieee.org> ;tag=7434264
       Call-ID: 14142@c.bell-tel.com
       CSeq: 1 INVITE
       Warning: 370 "Insufficient bandwidth (only have ISDN)",
         305 "Incompatible media format",
         330 "Multicast not available"
       Content-Type: application/sdp
       Content-Length: ...

       v=0
       o=c 3149329138 3149329165 IN IP4 38.245.76.2
```

```
      s=Let's talk
      e=t.watson@ieee.org
      t=3149328630 0
      b=CT:128
      c=IN IP4 x.bell-tel.com
      m=audio 3456 RTP/AVP 5 0 7
      a=rtpmap:5 DVI4/8000
      a=rtpmap:0 PCMU/8000
      a=rtpmap:7 LPC/8000
      m=video 2232 RTP/AVP 31
      a=rtpmap:31 H261/90000
```

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.

20.8 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 mu-law (RTP 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
      Via: SIP/2.0/UDP cat.wonderland.com
      From: Alice <sip:alice@wonderland.com>;tag=1gloo
      To: Bob <sip:bob@example.com>
      Call-ID: 6378@cat.wonderland.com
      CSeq: 1 OPTIONS
      Accept: application/sdp

S->C: SIP/2.0 200 OK
      From: Alice <sip:alice@wonderland.com>;tag=1gloo
      To: Bob <sip:bob@example.com> ;tag=376364382
      Call-ID: 6378@cat.wonderland.com
      Content-Length: 81
      Content-Type: application/sdp

      v=0
      o=alice 3149329138 3149329165 IN IP4 24.124.37.3
      s=Security problems
      e=bob@example.com
```

```
t=3149328650 0
c=IN IP4 24.124.37.3
m=audio 0 RTP/AVP 0 1 3 99
a=rtpmap:0 PCMU/8000
a=rtpmap:1 1016/8000
a=rtpmap:3 GSM/8000
a=rtpmap:99 SX7300/8000
m=video 0 RTP/AVP 31 34
a=rtpmap:31 H261/90000
a=rtpmap:34 H263/90000
```

A Minimal Implementation

A.1 Transport Protocol Support

All entities MUST support UDP. User agents SHOULD implement TCP transport. Stateful proxy, registrar, and redirect servers MUST implement TCP transport. Other transports MAY be supported by any entity.

A.2 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, Record-Route, Route 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. UAs MUST be able to use outbound proxies.

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 indicates 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.

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.3 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.4 Header Processing

Implementations SHOULD NOT have built-in limits for the number of header instances and header field lengths, beyond imposing an overall message length. Header fields that identify the transaction SHOULD appear first in the message so that implementations that cannot handle the full message can at least return a status response, e.g., 513 (Message Too Large).

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]). SDP is identified as Content-Type “application/sdp”. Each SIP message MUST contain zero or one SDP messages. Although the SDP specification allows for multiple session descriptions to be concatenated together into a large SDP message, an SDP message used with SIP MUST contain only a single session description.

Proxies generally do not modify the session description, but MAY do so if necessary, e.g., for network address translators, and if the session description is not protected by a cryptographic integrity mechanism.

B.1 General Methodology

The usage of SDP within SIP follows an “offer-answer” model. One side offers an SDP that provides their view of the session, and the other side answers the SDP with a matching one. The offer-answer model MAY occur in two ways. The offer MAY be placed in an INVITE, in which case the the answer MUST be in a 200 class response, and the ACK MUST NOT contain SDP. Or, the INVITE MAY contain no SDP, in which case the offer MUST be in the 200 class response, and the ACK MUST contain the answer.

If the offered session is not acceptable, it is rejected. If the offer is in the INVITE, rejection occurs by responding with a 488 or 606 response. If the offer occurs in the 200 class response, the UAC MUST send an ACK request, with a valid answer. It MAY send a BYE request to terminate the call, or MAY generate a re-INVITE, with the offer in the INVITE, to change the session parameters back to an acceptable form.

SDP processing does not depend on whether the offer comes in the INVITE or 200 class response.

	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
Contact	R	-	-	m	m
Contact	r	m	r	-	-
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
Max-Forwards	R	-	b	-	-
Proxy-Authenticate	407	a	-	-	-
Proxy-Authorization	R	-	a	-	-
Proxy-Require	R	-	m	-	-
Record-Route	R	m	-	m	m
Require	R	m	-	m	m
Response-Key	R	-	-	e	e
Route	R	m	m	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

B.2 Generating the initial offer

The offer (and answer) MUST be a valid SDP, as defined by RFC 2327 [6]. This means it MUST contain a v line, o line, s line and t line. Either an e line or p line MUST be present. However, it is RECOMMENDED that all implementations accept SDP without e, p, or s lines. The numeric value of the session id and version in the o line MUST be representable with a 64 bit signed integer.

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 20.2 illustrates this point. For invitations to two-party sessions, the SDP "s=" line MAY consist of a single space character (0x20).

Unfortunately, SDP does not allow to leave the "s=" line empty.

B.2.1 Unicast

The offer MUST contain zero or more media sections. Zero media sessions implies that the offerer wishes to communicate, but that the streams for the session will be added at a later time through re-INVITEs.

If a session description from an offerer contains a media stream which is listed as send (receive) only, it means that the offerer is only willing to send (receive) this stream, not receive (send). Media streams are marked as send-only or receive-only media streams using the SDP "a=sendonly" and "a=recvonly" attributes, respectively. If neither attribute is present, the default is both send and receive (which MAY be explicitly indicated with the "a=sendrecv" attribute).

For recvonly and sendrecv streams, the port number and address in the session description indicate where the media stream should be sent to. For send-only RTP streams, the address and port number indicate where RTCP reports are to be sent to. Specifically, RTCP reports are sent to the port number one higher than the number indicated. The IP address and port present in the offer indicate nothing about the source IP address and source port of RTP and RTCP packets that will be sent by the offerer. A port number of zero in the offer indicates that the stream is offered but should never be used. This has no useful semantics in an initial INVITE, but is allowed for reasons of completeness, since the response can contain a zero port indicating a rejected stream (Section B.3. Furthermore, existing streams can be terminated by setting the port to zero (Section B.4. In general, a port number of zero indicates that the media stream is not wanted.

The list of payload types for each media stream conveys two pieces of information, namely the set of codecs that the offerer is capable of sending and/or receiving (depending on the direction attributes), and the RTP payload type numbers used to identify those codecs. If multiple codecs are listed, it means that the offerer is capable of making use of any of those codecs during the call. In other words, the answerer MAY change codecs in the middle of the call, without sending a SIP message, to make use of any of those listed. For a send-only stream, the offer SHOULD indicate those codecs the offerer is willing to send for this stream. For a receive-only stream, the offer SHOULD indicate those codecs the offerer is willing to receive for this stream. For a send and receive stream, the offer SHOULD indicate those codecs that the offerer is willing to send and receive with.

For receive-only streams, the payload type numbers indicate the value of the payload type field in RTP packets the offerer is expecting to receive for that codec. For send-only streams, the payload type numbers indicate the value of the payload type field in RTP packets the offerer 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 offerer expects to both send and receive. This means that the payload type for a codec is the same in both directions.

All media descriptions SHOULD contain "a=rtpmap" mappings from RTP payload types to encodings. If there is no "a=rtpmap", the static payload type table from RFC 1890 [27] is to be used.

This allows easier migration away from static payload types.

In all cases, the codecs in the m line are listed in order of preference, with the first codec listed being preferred. In this case, preferred means that the recipient of the offer SHOULD use the codec with the highest preference that is acceptable to it.

If multiple media streams of different types are present, it means that the offerer wishes to use those streams at the same time. A typical case is an audio and video stream as part of a videoconference.

If multiple media streams of the same type are present, it means that the offerer wishes to send (and/or receive), multiple streams at the same time. When sending multiple streams of the same type, the source of the stream (such as the microphone or circuit on a gateway) is sent multiple times, once for each stream. Each stream MAY use different encodings. When receiving multiple streams of the same type, the streams MUST be mixed before playing them out. A typical usage example is a pre-paid calling card application, where the user can enter in a "long pound" at any time during a call to hangup and make a new call on the same card. This requires media from the user to the remote gateway, and to a system that looks for the long pound.

There are some codecs, such as the RTP payload format for DTMF tones and digits [43] and comfort noise codecs, which can only encode specific types of media content. When one of these codecs is present in an offered stream that is send-only or send-and-receive, it means that the offerer will send using that codec only when the content of the media stream is of a type that can be encoded with that codec. When the content of the media stream cannot be encoded with that codec, another codec indicated in the m line can be used. If there are no other codecs in the m line, nothing is sent. This is useful for handling the case where a UA would like to send DTMF only, using RFC 2833, to a remote server. This is indicated with a single media line containing only the DTMF codec.

B.2.2 Multicast

Construction of a session description for a multicast offer follows the procedures above, with a few exceptions.

The address listed in the c line MUST be a multicast address. It indicates the address that the offerer wishes to receive packets on.

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.

B.3 Generating the answer

The answer to an offered SDP is based on the offered SDP. If the answer is different in any way (different IP addresses, ports, etc.), the origin line MUST be different in the answer, since the answer is generated by a different entity. In that case, the version number in the o line of the answer is unrelated to the version number in the o line of the offer.

For each m line in the offer, there MUST be a corresponding m line in the answer. The answer MUST contain exactly the same number of m lines as the offer. This allows for streams to be matched up based on their order. This implies that if the offer contained zero m lines, the answer MUST contain zero m lines.

An offered stream MAY be rejected in the answer, for any reason. The definition of rejected is both neither offerer and answerer MUST NOT generate media (or RTCP packets) for that stream. To reject an offered stream, the port number in the corresponding stream in the answer is set to zero. Any media formats listed are ignored. At least one MUST be present, as specified by SDP.

B.3.1 Unicast

If a stream is offered as sendonly, the corresponding stream MUST be marked as recvonly in the answer. Furthermore, the corresponding stream in the answer MUST contain at least one codec the answerer is willing to receive with from amongst those listed in the offer. The stream MAY indicate additional codecs, not listed in the corresponding stream in the offer, that the answerer is willing to receive with. The connection address and port indicate the address where the answerer wishes to receive RTP (RTCP will be received on the port which is one higher).

If a media stream is listed as recvonly in the offer, the answer MUST be marked as sendonly. Furthermore, the corresponding stream in the answer MUST contain at least one codec the answerer is willing to send with from amongst those listed in the offer. The IP address and port indicate the address where the answerer wishes to receive RTCP (RTCP will be received on the port number one higher than the one listed in the SDP).

If an offered media stream is listed as sendrecv (or contains no direction attribute, in which case it is sendrecv by default), the corresponding stream in the answer MAY be marked as sendonly, recvonly, or sendrecv. The default is sendrecv. If the stream in the answer is marked as sendonly, it MUST contain at least one codec the answerer is willing to send with from amongst those listed in the offer. The IP address and port indicate the address where the answerer wishes to receive RTCP. If the stream in the answer is marked as recvonly, it MUST contain at least one codec the answerer is willing to receive with from amongst those listed in the offer. The stream MAY indicate additional codecs, not listed in the corresponding stream in the offer, that the answerer is willing to receive with. The connection address and port in the answer indicate the address where the answerer wishes to receive RTP and RTCP (RTCP will be received on the port number one higher than the one listed in the SDP). If the stream in the answer is marked as sendrecv, it MUST contain at least one codec the answerer is willing to both send and receive with, from amongst those listed in the offer. The stream MAY indicate additional codecs, not listed in the corresponding stream in the offer, that the answerer is willing to receive with. The connection address and port indicate the address where the answerer wishes to receive RTP (RTCP will be received on the port which is one higher).

The payload type numbers for a particular codec within a stream MUST be the same in offer and answer. In other words, a different dynamic payload type number for the same codec cannot be used in each direction.

In all cases, the codecs in the m line are listed in order of preference, with the first codec listed being preferred. In this case, preferred means that the recipient of the answer SHOULD use the codec with the highest preference that is acceptable to it.

Although the answerer MAY list the codecs in their desired order of preference, it is RECOMMENDED that unless there is a specific reason, the answer list codecs in the same relative order they were present in the offer. In other words, if a stream in the offer lists codecs 8, 22 and 48, in that order, and the answerer only supports codecs 8 and 48, it is RECOMMENDED that, if the answerer has no reason to change it, the ordering of codecs in the answer be 8, 48, and not 48, 8. This helps ensure that the same codec is used in both directions.

If the answerer has no media formats in common for a particular offered stream, the answerer MUST reject that media stream.

If there are no media formats in common for all streams, the entire offered session is rejected.

B.3.2 Multicast

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 offerer is acceptable to the answerer, the answerer can choose not to include a session description or MAY echo the description in the response.

An answerer MAY 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 offerer that the answerer does not support the given stream or media types which were removed. An answerer MUST NOT change whether a given stream is send-only, receive-only, or send-and-receive.

If an answerer does not support multicast at all, it SHOULD reject the session description.

B.4 Modifying the session

At any point during the call, either participant MAY issue a re-INVITE to modify characteristics of the session. It is fundamental to the operation of SIP that the exact same offer-answer procedure defined above is used for re-INVITE. This means that a re-INVITE MAY contain no SDP, so that the 200 OK to the re-INVITE contains the offer. In this case, the offerer SHOULD offer the same SDP it provided previously if it has no reason to change anything.

The offer in a re-INVITE MAY be identical to the last SDP provided to the other party (which may have been provided in an offer or an answer), or it MAY be different. We refer to the last SDP provided as the "previous SDP". If the offer is the same, the answer MAY be the same as the previous SDP from the answerer, or it MAY be different. If the offered SDP is different from the previous SDP, some constraints are placed on its construction, discussed below.

Nearly all aspects of the session can be modified. New streams can be added, existing streams can be deleted, and parameters of existing streams can change. When issuing an offer that modifies the session, the o line of the new SDP MUST be identical to that in the previous SDP, except that the version in the origin field MUST increment from the previous SDP. If the version in the origin line does not increment, the SDP MUST be identical to the SDP with that version number. The answerer MUST be prepared to receive an offer that contains SDP with a version that has not changed; this is effectively a no-op. However, the answerer MUST generate a valid answer (which MAY be the same as the previous SDP from the answerer, or MAY be different), according to the procedures defined in Section B.3.

If an SDP is offered which is different from the previous SDP, the new SDP MUST have a matching media section for each media section in the previous SDP. In other words, if the previous SDP had N media lines, the new SDP MUST have at least N media lines. The i^{th} media stream in the previous SDP, counting from the top, matches the i^{th} media stream in the new SDP, counting from the top. This matching is necessary in order for the answerer to determine which stream in the new SDP corresponds to a stream in the previous SDP. Because of these requirements, the number of m lines in a stream never decreases, but only increases. Deleted media streams from a previous SDP MUST NOT be removed from a new SDP.

B.4.1 Adding a media stream

New media streams are created by adding additional media descriptions below the existing ones. New media sections MUST appear below any existing media sections. The rules for formatting this media section are identical to those described in Section B.2.

When the answerer receives an SDP with more media descriptions than the previous SDP from the offerer, the answerer knows that new media streams are being added. These can be rejected or accepted by placing a matching media description in the answer. The procedures for constructing the new media description in the answer are described in Section B.3.

B.4.2 Removing a media stream

Existing media streams are removed by creating a new SDP with the port number for that stream set to zero. Otherwise, the media description SHOULD be formatted identically to the corresponding stream in the previous SDP.

A stream that is offered with a port of zero MUST be marked with port zero in the answer. Otherwise, the media description for the removed stream SHOULD be formatted identically to the corresponding stream in the previous SDP.

B.4.3 Modifying a media stream

Nearly all characteristics of a media stream can be modified.

The port number for a stream MAY be changed. To do this, the offerer creates a new media description, with the port number in the m line different from the corresponding stream in the previous SDP. If only the port number is to be changed, the rest of the media stream description SHOULD remain unchanged. The offerer MUST be prepared to receive media on both the old and new ports as soon as the offer is sent. The offerer MUST NOT cease listening for media on the old port until media arrives on the new port. At that time, it MAY cease listening for media on the old port.

The corresponding media stream in the answer MAY be the same as the stream in the previous SDP from the answerer, or MAY be different. If the updated stream is accepted by the answerer, the answerer SHOULD begin sending traffic for that stream to the new port immediately. If the answerer changes the port from the previous SDP, it MUST be prepared to receive media on both the old and new ports as soon as the answer is sent. The answerer MUST NOT cease listening for media on the old port until media arrives on the new port. At that time, it MAY cease listening for media on the old port.

To change the IP address where media is sent to, the same procedure is followed for changing the port number. The only difference is that the connection line is updated, not the port number.

The list of codecs used in the session MAY be changed. To do this, the offerer creates a new media description, with the list of media formats in the m line different from the corresponding stream in the previous SDP. This list MAY include new codecs, and MAY remove codecs present from the previous SDP. When a new codec is used with a dynamic payload type number, it MUST NOT reuse a dynamic payload type number used previously in the session.

The corresponding media stream in the answer is formulated as described in Section B.3. If the new list of codecs for a stream changes the choice of which codec is used, the new codec SHOULD be used immediately. That means the offerer MUST be prepared to receive media with a new codec as soon as it sends the offer, and the answerer MUST be prepared to receive media with a new codec as soon as it sends the answer.

The media type (audio, video, etc.) for a stream MAY be changed. This is particularly useful for changing between voice and fax in a single stream, which are both separate media types. To do this, the offerer creates a new media description, with a new media type, in place of the description in the previous SDP which is to be changed. The IP address and port for the stream MAY change, or MAY remain the same. However, the list of payload type numbers for the new codecs MUST be different than any used previously for this stream.

The corresponding media stream in the answer is formulated as described in Section B.3. Assuming the stream is acceptable, the answerer SHOULD begin sending with the new media type and codecs as soon as it receives the offer.

The transport for a stream MAY be changed. The process for doing this is identical to changing the port, excepting the transport is updated, not the port.

Any other attributes in a media description MAY be updated in an offer.

B.4.4 Putting a media stream 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 offers the other an updated SDP. This SDP has the connection address set to zero (0.0.0.0) for those streams that are to be put on hold. The treament of this is no different than for any other change in address, with the exception that the 0.0.0.0 address is effectively interpreted as “dev/null”, and no media is sent. Specifically, this means that a stream is placed “on hold” separately in each direction.

Each stream is placed "on hold" independently. The recipient of an offer for a stream on-hold SHOULD NOT automatically return an answer with the corresponding stream on hold. An SDP with all streams "on hold" is referred to as *held SDP*.

Certain third party call control scenarios do not work when a UA responds to held SDP with held SDP.

Typically, when a user "presses" hold, the UA will generate a re-INVITE with all streams in the SDP indicating an address of 0.0.0.0, and it will also locally mute, so that no media is sent to the far end.

B.5 Example

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

```
v=0
o=alice 2890844526 2890844526 IN IP4 host.anywhere.com
s=New board design
e=alice@foo.org
t=0 0
c=IN IP4 host.anywhere.com
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 as the answer:

```
v=0
o=bob 2890844730 2890844730 IN IP4 host.example.com
s=New board design
e=bob@bar.com
t=0 0
c=IN IP4 host.example.com
m=audio 47920 RTP/AVP 0 1
a=rtpmap:0 PCMU/8000
m=video 0 RTP/AVP 31
m=video 53000 RTP/AVP 32
a=rtpmap:32 MPV/90000
```

At some point later, Bob decides to change the port where he will receive the audio stream (from 47920 to 6400), and at the same time, add an additional audio stream as receive only, using the RTP payload format for events. Bob offers the following SDP in the INVITE:

```
v=0
o=bob 2890844730 2890844731 IN IP4 host.example.com
s=New board design
e=bob@bar.com
t=0 0
c=IN IP4 host.example.com
m=audio 6400 RTP/AVP 0 1
a=rtpmap:0 PCMU/8000
m=video 0 RTP/AVP 31
m=video 53000 RTP/AVP 32
a=rtpmap:32 MPV/90000
m=audio 8864 RTP/AVP 110
a=rtpmap:110 telephone-events
a=recvonly
```

Alice accepts the additional media stream, and so generates the following answer:

```
v=0
o=alice 2890844526 2890844527 IN IP4 host.anywhere.com
s=New board design
e=alice@foo.org
t=0 0
c=IN IP4 host.anywhere.com
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
m=audio 4520 RTP/AVP 110
a=rtpmap:110 telephone-events
a=sendonly
```

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 [26] and RFC 2234 [44]. 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. Note that URLs do **NOT** contain LWS.

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     = %x00-ff ; any 8-bit sequence of data
CHAR      = %x00-7f ; 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       = %x00-1f | %x7f ; (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 [10] 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  =  *( SP | HT ) [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 [23]). The TEXT-UTF8-TRIM rule is used for descriptive field contents that are *not* quoted strings, where leading and trailing LWS is not meaningful. In this regard, SIP differs from HTTP, which uses the ISO 8859-1 character set.

```
TEXT-UTF8        =  *(TEXT-UTF8char | LWS)
TEXT-UTF8-TRIM   =  *TEXT-UTF8char *(*LWS TEXT-UTF8char)
TEXT-UTF8char    =  %x21-7e
                 |  UTF8-NONASCII
UTF8-NONASCII    =  %xc0-df 1UTF8-CONT
                 |  %xe0-ef 2UTF8-CONT
                 |  %xf0-f7 3UTF8-CONT
                 |  %xf8-fb 4UTF8-CONT
                 |  %xfc-fd 5UTF8-CONT
UTF8-CONT        =  %x80-bf
```

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. Unless otherwise stated, tokens are case-insensitive. These special characters MUST be in a quoted string to be used within a parameter value.

```
token       =  1*(alphanum | "-" | "." | "!" | "%" | "*" | "_" | "+" | "`" | "'" | "~" )
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. In quoted strings, quotation marks (") and backslashes (\) need to be escaped.

```
quoted-string  =  ( <"> *(qdtext | quoted-pair ) <"> )
qdtext         =  LWS | %x21 | %x23-5b | %x5d-7e
               |  UTF8-NONASCII
```

The backslash character ("\") MAY be used as a single-character quoting mechanism only within quoted-string and comment constructs. Unlike HTTP/1.1, the characters CR and LF cannot be escaped by this mechanism to avoid conflict with line folding and header separation.

```
quoted-pair  =  "\" (%x00 - %x09 | %x0b | %x0c | %x0e - %x7f)
```

D IANA Considerations

Section 4.4 describes a name space and mechanism for registering SIP options. Section 10.47 describes the name space for registering SIP warn-codes.

SIP Header field names are registered with IANA. They do not require working group or working group chair review, but SHOULD be documented in an RFC or Internet draft. For Internet drafts, IANA is requested to make the draft available as part of the registration database.

By the time an RFC is published, colliding names may have already been implemented.

The following information needs to be provided to IANA in order to register a new header name:

- The name and email address of the individual performing the registration.
- The name of the header field being registered.
- A compact form version for that header field, if one is defined.
- The name of the draft or RFC where the header is defined.
- A copy of the draft or RFC where the header is defined.

SIP response status codes are registered with IANA. Because the status code space is limited, they do require working group or working group chair review, and MUST be documented in an RFC or Internet draft. For Internet drafts, IANA is requested to make the draft available as part of the registration database.

The following information needs to be provided to IANA in order to register a new response code:

- The name and email address of the individual performing the registration.
- The number of the response code being registered.
- The default reason phrase for that status code.
- The name of the draft or RFC where the status code is defined.
- A copy of the draft or RFC where the status code is defined.

When a registration for either a new header or new status code is created based on an I-D, and that I-D becomes an RFC, the person that performed the registration MUST notify IANA to change the registration to point to the RFC instead of the Internet Draft.

Headers SHOULD NOT use the X- prefix notation and MUST NOT duplicate the names of headers used by SMTP or HTTP unless the syntax is a compatible superset and the semantics are similar. Some common and widely used header fields MAY be assigned one-letter compact forms (Section 13). Compact forms can only be assigned after SIP working group review. In the absence of this working group, a designated expert reviews the request.

E Changes from RFC 2543

In addition to editorial clarifications, this document changes or adds the following features to SIP as specified in RFC 2543:

- Extensions developed by the IETF no longer use the org.ietf prefix.
- Tag syntax was generalized.
- Via header branch parameters were extended to allow "spirals", where two requests that differ only in the request URI are not treated as copies.
- New optional header fields, Alert-Info, Call-Info, In-Reply-To.

F Changes Made in Version 00

- In Sec. 14.4.1, indicated that UAC should send both CANCEL and BYE after a retransmission fails.
- Added semicolon and question mark to the list of unreserved characters for the user part of SIP URLs to handle tel: URLs properly.
- Uniform handling of if hop count Max-Forwards: return 483. Note that this differs from HTTP/1.1 behavior, where only OPTIONS and TRACE allow this header, but respond as the final recipient when the value reaches zero.
- Clarified that a forking proxy sends ACKs only for INVITE requests.
- Clarified wording of DNS caching. Added paragraph on "negative caching", i.e., what to do if one of the hosts failed. It is probably not a good idea to simply drop this host from the list if the DNS ttl value is more than a few minutes, since that would mean that load balancing may not work for quite a while after a server is brought back on line. This will be true in particular if a server group receives a large number of requests from a small number of upstream servers, as is likely to be the case for calls between major consumer ISPs. However, without getting into arbitrary and complicated retry rules, it seems hard to specify any general algorithm. Might it be worthwhile to simply limit the "black list" interval to a few minutes?
- Added optional Call-Info and Alert-Info header fields that describe the caller and information to be used in alerting. (Currently, avoided use of "purpose" qualification since it is not yet clear whether

rendering content without understanding its meaning is always appropriate. For example, if a UAS does not understand that this header is to replace ringing, it would mix both local ring tone and the indicated sound URL.) TBD!

- SDP "s=" lines can't be empty, unfortunately.
- Noted that maddr could also contain a unicast address, but SHOULD contain the multicast address if the request is sent via multicast (Section 10.46, 14.1).
- Clarified that responses are sent to port in Via sent-by value.
- Added "other-*" to the user URL parameter and the Hide and Content-Disposition headers.
- Clarified generation of timeout (408) responses in forking proxies and mention the Expires header. (Section 17.4)
- Clarified that CANCEL and INVITE are separate transactions (Fig. 11). Thus, the INVITE request generates a 487 (Request Terminated) if a CANCEL or BYE arrives.
- Clarified that Record-Route SHOULD be inserted in every request, but that the route, once established, persists. This provides robustness if the called UAS crashes.
- Emphasized that proxy, redirect, registrar and location servers are logical, not physical entities and that UAC and UAS roles are defined on a request-by-request basis. (Section 1.4)
- In Section 10.46, noted that the maddr and received parameters also need to be encrypted when doing Via hiding.
- Simplified Fig. 11 to only show INVITE transaction.
- Added definition of the use of Contact (Section 10.14) for OPTIONS.
- Added HTTP/RFC822 headers Content-Language and MIME-Version.
- Added note in Section A indicating that UAs need to support UDP.
- Added explanation in Section 15.5 explaining what a UA should do when receiving an initial INVITE with a tag.
- Clarified UA and proxy behavior for 302 responses (Section 11.3.3).
- Added details on what a UAS should do when receiving a tagged INVITE request for an unknown call leg. This could occur if the UAS had crashed and the UAC sends a re-INVITE or if the BYE got lost and the UAC still believes to be in the call.
- Added definition of Contact in 4xx, 5xx and 6xx to "redirect" to more error details.
- Added note to forking proxy description in Section 17.4 to gather *-Authenticate from responses. This allows several branches to be authenticated simultaneously.
- Changed URI syntax to use URL escaping instead of quotation marks.

- Changed SIP URL definition to reference RFC 2806 for telephone-subscriber part.
- Clarified that the To URI should basically be ignored by the receiving UAS except for matching requests to call legs. In particular, To headers with a scheme or name unknown to the callee should be accepted.
- Clarified in Section 10.46.1 that maddr is to be added by any client, either proxy or UAC.
- Added response code 488 to indicate that there was no common media at the particular destination. (606 indicates such failure globally.)
- In Section 10.24, noted that registration updates can shorten the validity period.
- Added note to Section 19.3 to enclose the URI in quotation marks. The BNF in RFC 2617 is in error.
- Clarified that registrars use Authorization and WWW-Authenticate, not proxy authentication.
- Added note in Section 10.14 that "headers" are copied from Contact into the new request.
- Changed URL syntax so that port specifications have to have at least one digit, in line with other URL formats such as "http". Previously, an empty port number was permissible.
- In Section B, added a section on how to add and delete streams in re-INVITEs.
- IETF-blessed extensions now have short names, without org.ietf. prefix.
- Cseq is unique within a call leg, not just within a call (Section 10.20).
- Added IPv6 literal addresses to the SIP URL definition in Section 2, according to RFC 2732 [45]. Modified the IPv4 address to limit segments to at most three digits.
- In Section 7, modify registration procedure so that it explicitly references the URL comparison. Updates with shorter expiration time are now allowed.
- For send-only media, SDP still must indicate the address and port, since these are needed as destinations for RTCP messages. (Section B)
- Changed references regarding DNS SRV records from RFC 2052 to RFC 2782, which is now a Proposed Standard. Integrated SRV into the search procedure in Section 1 and removed the SRV appendix. The only visible change is that protocol and service names are now prefixed by an underscore. Added wording that incorporates the precedence of maddr.
- Allow parameters in Record-Route and Route headers.
- In Table 2, list udp as the default value for the transport parameter in SIP URI.
- Removed sentence that From can be encrypted. It cannot, since the header is needed for call-leg identification.

- Added note that a UAC only copies a To tag into subsequent transactions if it arrives in a 200 OK to an INVITE in Section 15. This avoids the problem that occurs when requests get resubmitted after receiving, say, a 407 (or possibly 500, 503, 504, 305, 400, 411, 413, maybe even 408). Under the old rules, these requests would have a tag, which would force the called UAS to reject the request, since it doesn't have an entry for this tag.
- Loop detection has been modified to take the request-URI into account (Section 17.3 and 10.46.6). This allows the same request to visit the server twice, but with different request URIs ("spiral").
- Elaborated on URL comparison and comparison of From/To fields.
- Added np-queried user parameter.
- Changed tag syntax from UUID to token, since there's no reason to restrict it to hex.
- Added Content-Disposition header based on earlier discussions about labeling what to do with a message body (part).
- Clarification: proxies must insert To tags for locally generated responses.
- Clarification: multicast may be used for subsequent registrations.
- Feature: Added Supported header. Needed if client wants to indicate things the server can usefully return in the response.
- Bug: The From, To, and Via headers were missing extension parameters. The Encryption and Response-Key header fields now "officially" allow parameters consisting only of a token, rather than just "token = value".
- Bug: Allow was listed as optional in 405 responses in Table 4. It is mandatory.
- Added in Section 6: "A BYE request from either called or calling party terminates any pending INVITE, but the INVITE request transaction MUST be completed with a final response."
- Clarified in Section 5.1: "If an INVITE request for an existing session fails, the session description agreed upon in the last successful INVITE transaction remains in force."
- Clarified in Section 5.1 what happens if two INVITE requests meet each other on the wire, either traveling the same or in opposite directions:

  > A UAC MUST NOT issue another INVITE request for the same call leg before the previous transaction has completed. A UAS that receives an INVITE before it sent the final response to an INVITE with a lower CSeq number MUST return a 400 (Bad Request) response and MUST include a Retry-After header field with a randomly chosen value of between 0 and 10 seconds. A UA that receives an INVITE while it has an INVITE transaction pending, returns a 500 (Internal Server Error) and also includes a Retry-After header field.

- Expires header clarified: limits only duration of INVITE transaction, not the actual session. SDP does the latter.

- The In-Reply-To header was added (Section 10.26).
- There were two incompatible BNFs for WWW-Authenticate. One defined for PGP, and the other borrowed from HTTP. For basic or digest:

  ```
  WWW-Authenticate: basic realm="Wallyworld"
  ```

  and for pgp:

  ```
  WWW-Authenticate: pgp; realm="Wallyworld"
  ```

  The latter is incorrect and the semicolon has been removed.
- Added rules for Route construction from called to calling UA.
- We now allow Accept and Accept-Encoding in BYE and CANCEL requests. There is no particular reason not to allow them, as both requests could theoretically return responses, particularly when interworking with other signaling systems.
- PGP "pgp-pubalgorithm" allows server to request the desired public-key algorithm.
- ABNF rules now describe tokens explicitly rather than by subtraction; explicit character enumeration for CTL, etc.
- Registrars should be careful to check the Date header as the expiration time may well be in the past, as seen by the client.
- Content-Length is mandatory; Table 4 erroneously marked it as optional.
- User-Agent was classified in a syntax definition as a request header rather than a general header.
- Clarified ordering of items to be signed and include realm in list.
- Allow Record-Route in 401 and 484 responses.
- Hop-by-hop headers need to precede end-to-end headers only if authentication is used (Section 10).
- 1xx message bodies MAY now contain session descriptions.
- Changed references to HTTP/1.1 and authentication to point to the latest RFCs.
- Added 487 (Request terminated) status response. It is issued if the original request was terminated via CANCEL or BYE.
- The spec was not clear on the identification of a call leg. Section 1.3 says it's the combination of To, From, and Call-ID. However, requests from the callee to the caller have the To and From reversed, so this definition is not quite accurate. Additionally, the "tag" field should be included in the definition of call leg. The spec now says that a call leg is defined as the combination of local-address, remote-address, and call-id, where these addresses include tags.

  Text was added to Section 6.21 to emphasize that the From and To headers designate the originator of the request, not that of the call leg.

- All URI parameters, except method, are allowed in a Request-URI. Consequently, also updated the description of which parameters are copied from 3xx responses in Sec. 10.14.
- The use of CRLF, CR,or LF to terminate lines was confusing. Basically, each header line can be terminated by a CR, LF, or CRLF. Furthermore, the end of the headers is signified by a "double return". Simplified in Section 3 to require sending of CRLF, but require senders to receive CR and LF as well and only allow CR CR, LF LF in addition to double CRLF as a header-body separator.
- Round brackets in Contact header were part of the HTTP legacy, and very hard to implement. They are also not that useful and were removed.
- The spec said that a proxy is a back-to-back UAS/UAC. This is almost, but not quite, true. For example, a UAS should insert a tag into a provisional response, but a proxy should not. This was clarified.
- Section 6.13 in the RFC begins mid-paragraph after the BNF. The following text was misplaced in the conversion to ASCII:

  > 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.

G Changes Made in Version 01

- Uniform syntax specification for semicolon parameters:

```
Foo        =  "Foo" ":" something *( ";" foo-param )
foo-param  =  "bar" "=" token
           |  generic-param
```

- Removed np-queried user parameter since this is now part of a tel URL extension parameter.
- In Section B, noted that if the capabilities intersection is empty, a dummy format list still has to be returned due to SDP syntax constraints. Previously, the text had required that no formats be listed. (Brian Rosen)
- Reorganized tables 4 and 5 to show proxy interaction with headers rather than "end-to-end" or "hop-by-hop".

H Changes Made in Version 02

- Added "or UAS" in description of received headers in Section 10.46.1. This makes the response algorithm work even if the last IP address in the Via is incorrect.
- Tentatively removed restriction that CANCEL requests cannot have Route headers. (Billy Biggs)
- Tentatively added Also header for BYE requests, as it is widely implemented and a simple means to implement unsupervised call transfer. Subject to removal if there is protest. (Billy Biggs)

- If a proxy sends a request by UDP (TCP), the spec did not disallow placing TCP (UDP) in the transport parameter of the Via field, which it should. Added a note that the transport protocol actually used is included.
- No default value for the q parameter in Contact is defined. This is not strictly needed, but is useful for consistent behaviors at recursive proxies and at UAC's. Now 0.5.
- Clarified that To and From tag values should be different to simplify request matching when calling oneself.
- Removed ability to carry multiple requests in a single UDP packet (Section 10.18).
- Added note that Allow MAY be included in requests, to indicate requestor capabilities for the same call ID.
- Added note to Section 10.21 indicating that registrars MUST include the Date header to accomodate UAs that do not have a notion of absolute time.
- Added note to Section 7 emphasizing that non-SIP URIs are permissible.
- Rewrote the server lookup section to be more precise and more like pseudo-code, with nesting instead of "gotos".
- Removed note

  > Note that the two URLs example.com and example.com:5060, while considered equal, may not lead to the same server, as the former causes a DNS SRV lookup, while the latter only uses the A record.

  since that is no longer the case.
- Emphasized that proxies have to forward requests with unknown methods.
- Aligned definition of call leg with URI comparison rules.
- Required that second branch parameter be globally unique, so that a proxy can distinguish different branches in spiral scenarios similar to the following, with record-routing in place:

  ```
         B  ---> P1 -------> P2 ------------> P1 ----------------> A
  BYE B   B/1      P1/2,B/1    P2/3,P1/2,B/1    P1/4,P2/3,P1/2,B/1
  ```

  Here, A/1 denotes the Via entry with host A and branch parameter 1. Also, this requires updating the definition of isomorphic requests, since the Request-URI is the same for all BYE that are record-routed.
- Removed Via hiding from spec, for the following reasons:
  - complexity, particularly hidden "gotchas" that surface at various points (as in this instance);
  - interference with loop detection and debugging;

- Unlike HTTP, where via-hiding makes sense since all data is contained in the request or response, Via-hiding in SIP by itself does nothing to hide the caller or callee, as address information is revealed in a number of places:
  - Contact;
  - Route/Record-Route;
  - SDP, including the o= and c= lines;
  - possibly accidental leakage in User-Agent header and Call-ID headers.
- Unless this is implemented everywhere, the feature is not likely to be very useful, without the sender having any recourse such as "don't route this request unless you can hide". It appears that almost all existing proxies simply ignore the Hide header.

- Added Error-Info header field.

I Changes Made in Version 03

- Description of Route and Record-Route moved to separate section, Section 16, which is new. All UAs must now support this mechanism (Section A).

- Removed status code 411, since it cannot occur (Jonathan Rosenberg, James Jack).

- Rewrote Record-Route section to reflect new mechanism. In particular, requests from callee to caller now use the same path as in the opposite direction, without substituting the From header field values. The maddr parameter is now optional.

- Disallowed SIP URLs that only have a password, without a user name. The prototype from RFC 1738 also doesn't allow this.

- Allow registrar to set the expiration time.

- CSeq (Section 10.20) is counted within a call leg, not a call.

- Removed wording that connection closing is equivalent to CANCEL or 500. This does not work for connections that are used for multiple transactions and has other problems.

- Cleaned up CSeq section. Removed text about inserting CSeq method when it is absent. Clarified that CSeq increments for all requests, not just invite. Clarified that all out of order requests, not just out of order INVITE, are rejected with a 400 class response. Clarified the meaning of "initial" sequence number. Clarified that after a request forks, each 200 OK is a separate call leg, and thus, separate CSeq space. Clarified that CSeq numbers are independent for each direction of a call leg.

- Massive reorganization and cleanup of the SDP section. Introduced the concept of the offer-answer model. Clarified that set of codecs in m line are usable all at the same time. Inserted size restriction on representation of values in o line. Explicitly describe forked media. New media lines for adding streams appear at the bottom of the SDP (used to say append).

- Removed Also.

- Added text to Require and Proxy-Require sections, making it a SHOULD to retry the request without the unsupported extension.
- Added text to section on 415, saying that UAC SHOULD retry the request without the unsupported body.
- Added text to section on CANCEL and ACK, clarifying much of the behavior.
- Modified Content-Type to indicate that it can be present even if the body is empty.
- From tags mandatory
- Old text said that if you hang up before sending an ACK, you need not send the ACK. That is wrong. Text fixed so that an ACK is always sent.
- Old text said that if you never got a response to an INVITE, the UAC should send both an INVITE and CANCEL. This doesn't make sense. Rahter, it should do nothing and consider the call terminated.
- Added text that says pending requests are responded to with a 487 if a BYE is received.
- Updated section 2.2, so that its clear that Contact is not used with BYE.
- Clarified Via processing rules. Added text on handling loops when proxies route on headers besides the request URI. Added text on handling case when sent-by contains a domain name. Added text to 6.47 on opening TCP connections to send responses upstream.
- Clarified that a 1xx with an unknown xx is not the same as the 100 response.
- Removed usage of Retry-After in REGISTER.
- Clarified usage of persistent connections.
- Clarified that servers supporting HTTP basic or digest in rfc2617 MUST be backwards compatible with RFC 2069.
- Clarified that ACK contains the same branch ID as the request its acknowledging.
- Added definitions for spiral, B2BUA.
- Rephrased definitions for UAC, UAS, Call, call-leg, caller, callee, making them more concrete.
- URL comparison ignores parameters not present in both URLs only for unknown parameters.
- Clarified that * in Contact is used only in REGISTER with Expires header zero. Mentioned * case in section on Contact syntax.
- Removed text that says a UA can insert a Contact in 2xx that indicates the address of a proxy. Not likely to work in general.
- Removed SDP text about aligning media streams within a media type to handle certain crash and restart cases.
- Receiving a 481 to a mid-call request terminates that call leg. Agreed upon at IETF 49.

- Introduced definition of regular transaction - non-INVITE excepting ACK and CANCEL.
- Clarified rules for overlapping transactions.
- Forking proxies MUST be stateful (used to say SHOULD). Proxies that send requests on multicast MUST be stateful (used to say nothing)
- Text added recommending that registrars authorize that entity in From field can register address-of-record in the To field.
- Forwarding of non-100 provisionals upstream in a proxy changed from SHOULD to MUST.
- Removed PGP.

J Changes Made in Version 04

- Removed Unsupported as a request header from Table 5.
- Clarified SDP procedures for changing IP address and port. Specifically, spelled out the duration for which a UA needs to received media on the old port and address.
- Added text in the SDP session which recommends that the answerer use the same ordering of codecs as used on the offer, in order to help ensure symmetric codec operation under normal conditions.
- Fixed bug in the example in the SDP section, where the new media line was listed at the top. Should have been the bottom.
- Authorization credentials are cached based on the URL of the To header, not the entire To header as 10.48 implied.
- Section 10.31, on Proxy-Authenticate, indicated that a server responds with a 401 if the client guessed wrong. This is incorrect. It should be 407.
- Section 10.14, removed motivational text about Contact allowing an INVITE to be routed directly between end systems, since its confusing. Some have interpreted to mean that Record-Route is ignored when Contact is present.
- Added reference to SCTP RFC.
- Updated 2.2 to allow non-SIP URLs in OPTIONS and 2xx to OPTIONS.
- Fixed example in 20.5. Added ACK for 487, and added To tag to 487 response.
- Clarified further URL comparisons. Its only URL parameters without defaults that are ignored if not present in both URLs.
- Section 1.5.2, UDP mandatory for all. TCP is a SHOULD for UA, MUST for proxy, registrar, redirect servers.
- Brought syntax for Contact, Via, and the SIP URL into alignment between the text and postscript versions.

- Updated the text in section 6 which said that the ordering of header fields follows HTTP, with the exception of Via, where order matters. However, the HTTP spec says that order matters, so this sentence is redundant and confusing. The sentence was removed.
- Added e lines to SDP examples in the Examples section.
- Rewrote Allow discussion, more formally defining its semantics and usage cases.
- Updated text on 604 status, to indicate that its based on the Request-URI, not the To.
- Added response registrations to IANA considerations. Provided more details on registration process.
- Clarified that only a UAS rejects a request because the To tag doesn't match a local value.
- Clarified that stateless proxies need to route based on static criteria only.
- Proxy and UAC CANCEL generation upon 2xx, 6xx if it forked is now a SHOULD; used to be a MAY.
- Added text saying that a UAS SHOULD send a BYE if it never gets an ACK for a 2xx establishing a call leg.
- Added text saying that a UAS SHOULD send a re-INVITE if it never gets an ACK for a 2xx to a re-INVITE.
- Added text on 503 processing, indicating that a client should try a different server when receiving a 503, and that a proxy shouldn't forward a 503 upstream unless it can't service any other requests.
- Removed motivational text in Section 10.43 on Via headers since its not consistent with the text before it.
- Changed IPSec reference to RFC2401, from RFC1825.
- Updated retransmission defininition in 17.3.4 to be consistent with the rest of the spec.
- Softened the language for insertion of the transport param in the record-route. Specifically, it can be inserted in private networks where it is known apriori that the specific transport is supported.
- Updated definition of B2BUA.
- Added text to section on 420 processing, which mandates that the client retry the request without extensions listed in the Unsupported header in the response.
- Allow Authentication-Info header to be used for HTTP digest.

K Acknowledgments

We wish to thank the members of the IETF MMUSIC and SIP WGs for their comments and suggestions. Detailed comments were provided by Brian Bidulock, Jim Buller, Neil Deason, Dave Devanathan, Cdric Fluckiger, Yaron Goland, Bernie Hneisen, Phil Hoffer, Christian Huitema, Jean Jervis, Gadi Karmi, Peter Kjellerstedt, Anders Kristensen, Jonathan Lennox, Gethin Liddell, Keith Moore, Vern Paxson, Moshe J. Sambol, Chip Sharp, Igor Slepchin, Robert Sparks, Eric Tremblay., and Rick Workman.

This work is based, inter alia, on [46, 47].

L Authors' Addresses

Mark Handley
ACIRI
electronic mail: mjh@aciri.org

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
dynamicsoft
72 Eagle Rock Ave
East Hanover, NJ 07936
USA
electronic mail: jdrosen@dynamicsoft.com

References

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

[2] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation protocol (RSVP) – version 1 functional specification," Request for Comments 2205, Internet Engineering Task Force, Sept. 1997.

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

[4] H. Schulzrinne, A. Rao, and R. Lanphier, "Real time streaming protocol (RTSP)," Request for Comments 2326, Internet Engineering Task Force, Apr. 1998.

[5] M. Handley, C. Perkins, and E. Whelan, "Session announcement protocol," Request for Comments 2974, Internet Engineering Task Force, Oct. 2000.

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

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

[8] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson, "Stream control transmission protocol," Request for Comments 2960, Internet Engineering Task Force, Oct. 2000.

[9] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol – HTTP/1.1," Request for Comments 2616, Internet Engineering Task Force, June 1999.

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

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

[12] G. Nair and H. Schulzrinne, "DHCP option for SIP servers," Internet Draft, Internet Engineering Task Force, Mar. 2001. Work in progress.

[13] A. Gulbrandsen, P. Vixie, and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)," Request for Comments 2782, Internet Engineering Task Force, Feb. 2000.

[14] P. V. Mockapetris, "Domain names - implementation and specification," Request for Comments 1035, Internet Engineering Task Force, Nov. 1987.

[15] D. Zimmerman, "The finger user information protocol," Request for Comments 1288, Internet Engineering Task Force, Dec. 1991.

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

[17] W. Yeong, T. Howes, and S. Kille, "Lightweight directory access protocol," Request for Comments 1777, Internet Engineering Task Force, Mar. 1995.

[18] 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.

[19] A. Vaha-Sipila, "URLs for telephone calls," Request for Comments 2806, Internet Engineering Task Force, Apr. 2000.

[20] J. Postel, "User datagram protocol," Request for Comments 768, Internet Engineering Task Force, Aug. 1980.

[21] J. Postel, "DoD standard transmission control protocol," Request for Comments 761, Internet Engineering Task Force, Jan. 1980.

[22] T. Dierks and C. Allen, "The TLS protocol version 1.0," Request for Comments 2246, Internet Engineering Task Force, Jan. 1999.

[23] F. Yergeau, "UTF-8, a transformation format of ISO 10646," Request for Comments 2279, Internet Engineering Task Force, Jan. 1998.

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

[25] J. C. Mogul and S. E. Deering, "Path MTU discovery," Request for Comments 1191, Internet Engineering Task Force, Nov. 1990.

[26] D. Crocker, "Standard for the format of ARPA internet text messages," Request for Comments 822, Internet Engineering Task Force, Aug. 1982.

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

[28] D. Meyer, "Administratively scoped IP multicast," Request for Comments 2365, Internet Engineering Task Force, July 1998.

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

[30] F. Dawson and T. Howes, "vcard MIME directory profile," Request for Comments 2426, Internet Engineering Task Force, Sept. 1998.

[31] G. Good, "The LDAP data interchange format (LDIF) - technical specification," Request for Comments 2849, Internet Engineering Task Force, June 2000.

[32] P. Hoffman, L. Masinter, and J. Zawinski, "The mailto URL scheme," Request for Comments 2368, Internet Engineering Task Force, July 1998.

[33] R. Troost and S. Dorner, "Communicating presentation information in internet messages: The content-disposition header," Request for Comments 1806, Internet Engineering Task Force, June 1995.

[34] R. Braden and Ed, "Requirements for internet hosts - application and support," Request for Comments 1123, Internet Engineering Task Force, Oct. 1989.

[35] J. Palme, "Common internet message headers," Request for Comments 2076, Internet Engineering Task Force, Feb. 1997.

[36] H. Schulzrinne and J. Rosenberg, "SIP: Session initiation protocol – locating SIP servers," Internet Draft, Internet Engineering Task Force, Mar. 2001. Work in progress.

[37] R. Rivest, "The MD5 message-digest algorithm," Request for Comments 1321, Internet Engineering Task Force, Apr. 1992.

[38] H. Alvestrand, "IETF policy on character sets and languages," Request for Comments 2277, Internet Engineering Task Force, Jan. 1998.

[39] N. Freed and N. Borenstein, "Multipurpose internet mail extensions (MIME) part two: Media types," Request for Comments 2046, Internet Engineering Task Force, Nov. 1996.

[40] S. Kent and R. Atkinson, "Security architecture for the internet protocol," Request for Comments 2401, Internet Engineering Task Force, Nov. 1998.

[41] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart, "HTTP authentication: Basic and digest access authentication," Request for Comments 2617, Internet Engineering Task Force, June 1999.

[42] J. Franks, P. Hallam-Baker, J. Hostetler, P. Leach, A. Luotonen, E. Sink, and L. Stewart, "An extension to HTTP : Digest access authentication," Request for Comments 2069, Internet Engineering Task Force, Jan. 1997.

[43] H. Schulzrinne and S. Petrack, "RTP payload for DTMF digits, telephony tones and telephony signals," Request for Comments 2833, Internet Engineering Task Force, May 2000.

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

[45] R. Hinden, B. Carpenter, and L. Masinter, "Format for literal IPv6 addresses in URL's," Request for Comments 2732, Internet Engineering Task Force, Dec. 1999.

[46] 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.

[47] 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