Internet Engineering Task Force SIP WG
INTERNET-DRAFT
Rosenberg,Schulzrinne,Camarillo,Johnston,Peterson,Sparks,Handley,Schooler
draft-ietf-sip-rfc2543bis-05.ps
Various places
October 26, 2001
Expires: April 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 telephone calls, multimedia distribution and multimedia conferences.

SIP invitations used to create sessions carry session descriptions which allow participants to agree on a set of compatible media types. SIP makes use of elements called proxy servers to help route requests to the users current location, assist in firewall traversal, and provide features to users. SIP also provides a registration function that allows them to upload their current location for use by proxy servers. SIP runs ontop of several different transport protocols.

Contents

1 Introduction

There are many applications of the Internet that require the creation and management of a session, where a session is considered an exchange of data between an association of participants. The implementation of these services is complicated by the practices of participants; users may move between endpoints, they may be addressable by multiple names, and they may communicate in several different media - sometimes simultaneously. Numerous protocols have been authored that carry various forms of real-time multimedia session data such as voice, video, or text messages. SIP works in concert with these protocols by enabling Internet endpoints (called "user agents") to discover one another and to agree on a characterization of a session they would like to share. For locating prospective session participants, SIP relies on an infrastructure of network hosts (called "proxy servers") to which user agents can send registrations, invitations to sessions and other requests. SIP is an agile, general-purpose tool for creating, modifying and terminating sessions that works independently of underlying transport protocols and without dependency on the type of session that is being established.

2 Overview of SIP Functionality

The Session Initiation Protocol (SIP) is an application-layer control protocol that can establish, modify and terminate multimedia sessions (conferences) such as Internet telephony calls. SIP can also invite participants to already existing sessions. A SIP entity issuing an invitation for an already existing session does not necessarily have to be a member of the session to which it is inviting. Media can be added to (and removed from) an existing session. SIP transparently supports name mapping and redirection services, which supports *personal mobility* [1, p. 44] - users can maintain a single externally visible identifier (SIP URI) regardless of their network location.

SIP supports five facets of establishing and terminating multimedia communications:

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

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

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

Session setup: "ringing", establishment of session parameters at both called and calling party;

Session handling: including transfer and termination of sessions, modifying session parameters, and invoking services.

SIP is not a vertically integrated communications system. SIP is rather a component of the overall IETF multimedia data and control architecture which incorporates 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. Therefore, SIP should be used in conjunction with other protocols in order to provide complete services to the users. However, the basic functionality and operation of SIP does not depend on any of these protocols.

SIP does not provide services. SIP rather provides primitives that can be used to implement different services. For example, SIP can locate a user and deliver an opaque object to his current location. If this primitive is used to deliver a session description written in SDP, for instance, the parameters of a session can be agreed between endpoints. If the same primitive is used to deliver a photo of the caller as well as the session description, a "caller ID" service can be easily implemented. As this example shows, a single primitive is typically used to provide several different services. Consequently, generality is more important than efficiency when designing SIP primitives.

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 initiate a session that uses some other conference control protocol. SIP does not allocate multicast addresses and does not reserve network resources.

3 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.

4 Overview of Operation

This section will introduce the basic operations of the SIP protocol using simple examples. Note that this section is tutorial in nature and does not contain any normative statements.

The first example will show the basic functions of SIP: location of an end point, signaling a desire to communicate, negotiation of session parameters to establish the session, and teardown of the session once established.

Figure 1 shows a typical example of a SIP message exchange between two users, Alice and Bob. (Each message is labeled with the letter "F" and a number for reference by the text.) In this example, Alice uses a SIP application on her PC (referred to as a softphone) to call Bob on his SIP phone over the Internet. Also shown are two SIP proxy servers which act on behalf of Alice and Bob to facilitate the session establishment. This typical arrangement is often referred to as the "SIP trapezoid" as shown by the geometric shape of the dashed lines in Figure 1.

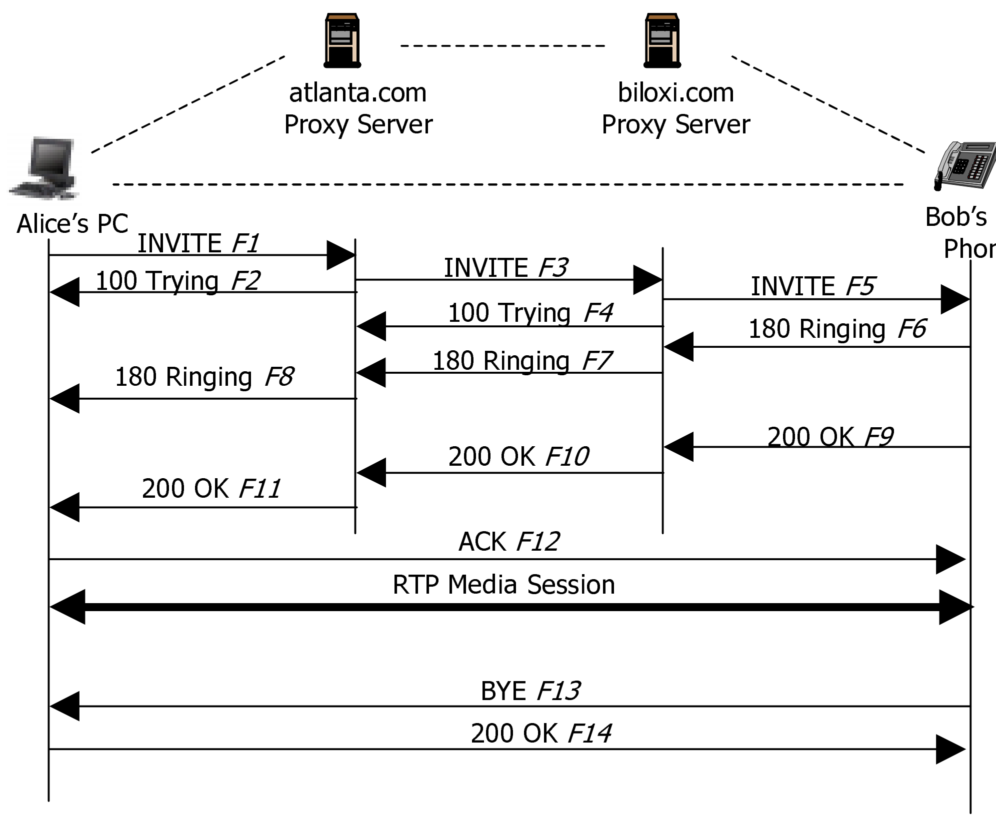

Figure 1: SIP session setup example with SIP trapezoid

Alice "calls" Bob using his SIP identity, a type of Uniform Resource Identifier (URI) called a SIP URI and defined in Section 21.1. It has a similar form to an email address, typically containing a username and a host name. In this case it is sip:bob@biloxi.com, where biloxi.com is the domain of Bob's SIP service provider (which can be an enterprise, retail provider, etc). Alice also has a SIP URI of sip:alice@atlanta.com. Alice might have typed in Bob's URI or perhaps clicked on a hyperlink or an entry in an address book.

SIP is based on an HTTP-like request/response transacton model. Each transaction consists of a request that invokes a particular "Method", or function, on the server, and at least one response. In this example, the transaction begins with Alice's softphone sending an INVITE request addressed to Bob's SIP URI. INVITE is an example of a SIP method which specifies the action that the requestor (Alice) wants the server (Bob) to take. The INVITE request contains a number of header fields. Header fields are additional named attributes which provide additional information about a message. The ones present in an INVITE include a unique identifier for the call, the destination address, Alice's address, and information about the type of session that Alice wishes to establish with Bob. The INVITE (message F1 in Figure 1) might look like this:

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP 10.1.3.3:5060
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@10.1.3.3
CSeq: 314159 INVITE
Contact: <sip:alice@10.1.3.3>
Content-Type: application/sdp
Contact-Length: 142

(Alice's SDP not shown)
```

The first line of the text-encoded message contains the method name (INVITE). The lines which follow are a list of header fields. This example contains a minimum required set. The headers are briefly described below:

Via contains the IP address (10.1.3.3), port number (5060), and transport protocol (UDP) on which Alice is expecting to receive responses to this request.

To contains a display name (Bob) and a SIP URI (sip:bob@biloxi.com) that the request was originally directed towards.

From also contains a display name (Alice) and a SIP URI (sip:alice@atlanta.com) that indicate the originator of the request. This header field also has a tag parameter which contains a pseudorandom string (1928301774) which was added to the URI by the softphone. It is used for identification purposes.

Call-ID contains a globally unique identifier for this call, generated by the combination of a pseudorandom string and the softphone's IP address. The combination of the To, From, and Call-ID completely define a peer-to-peer SIP relationship betwee Alice and Bob, and is referred to as a "dialog".

CSeq or Command Sequence contains an integer and a method name. The CSeq number is incremented for each new request, and is a traditional sequence number.

Contact contains a SIP URI which represents a direct route to reach or contact Alice, usually composed of a username at an IP address. While the Via header field is used to tell other elements where to send the response, the Contact header field tells other elements where to send future requests for this dialog.

Content-Type contains a description of the message body (not shown).

Content-Length contains an octet (byte) count of the message body.

The complete set of SIP header fields is defined in Section 22.

The details of the session, type of media, codec, sampling rate, etc. are not described using SIP. Rather, the body of a SIP message contains a description of the session, encoded in some other protocol format. One such format is Session Description Protocol (SDP) [6]. This SDP message (not shown in the example) is carried by the SIP message in an analogous way that a document attachment is carried by an email message, or a web page is carried in an HTTP message.

Since the softphone has no knowledge of Bob's exact location, or how to locate the SIP server in the biloxi.com domain, the softphone sends the INVITE to the SIP server that serves Alice's domain, atlanta.com. The IP address of the atlanta.com SIP server could have been configured in Alice's softphone, or it could have been discovered by DHCP, for example.

The atlanta.com SIP server is a type of SIP server known as a proxy server. A proxy server receives SIP requests and forwards them on behalf of the requestor. In this example, the proxy server receives the INVITE request and generates a 100 Trying response which is sent back to Alice's softphone. The 100 Trying response indicates that the INVITE has been received and that the proxy is working on her behalf to try to route the INVITE to the destination. Responses in SIP use a numerical three digit code followed by a descriptive phrase. This response contains the same To, From, Call-ID, and CSeq as the INVITE, which allows Alice's softphone to correlate this response to the sent INVITE. The atlanta.com proxy server locates the proxy server at biloxi.com, possibly by performing a DNS (Domain Name Service) lookup to find the SIP server which serves the biloxi.com domain. This is described in Section 24. As a result, it obtains the IP address of the biloxi.com proxy server and forwards, or proxies, the INVITE request there. Before forwarding the request, the atlanta.com proxy server adds an additional Via header field which contains its own IP address (the INVITE already contains Alice's IP address in the first Via). The biloxi.com proxy server receives the INVITE and responds with a 100 Trying response back to the Atlanta.com proxy server to indicate that it has received the INVITE and is processing the request. The proxy server consults a database, generically called a location service, which contains the current IP address of Bob. (We shall see in the next section how this database can be populated.) The biloxi.com proxy server adds another Via header with its own IP address to the INVITE and proxies it to Bob's SIP phone.

Bob's SIP phone receives the INVITE and begins to alert Bob to the incoming call from Alice so that Bob can decide whether or not to answer the call - i.e. Bob's phone rings. Bob's SIP phone sends an indication of this in a 180 Ringing response. This response is routed back thorough the two proxies in the reverse direction. Each proxy uses the Via header to figure out where to send the response, and removes its own address from the top. As a result, although DNS and location service lookups were required to route the initial INVITE, the 180 Ringing response can be returned to the caller without lookups, or without state being maintained in the proxies. This also has the desirable property that each proxy that sees the INVITE will also see all responses to the INVITE.

When Alice's softphone receives the 180 Ringing response, it passes this information to Alice, perhaps using an audio ringback tone, or just by displaying or flashing a message on Alice's screen.

In this example, Bob decides to answer the call. When he picks up the handset his SIP phone sends a 200 OK response to indicate that the call has been answered. The 200 OK contains a message body containing the SDP media description of the type of session that Bob is willing to establish with Alice. As a result, there is a two-phase exchange of SDP messages; Alice sent one to Bob, and Bob sent one back to Alice. This two-phase exchange provides basic negotiation capabilities, and is based on a simple offer/answer model, If Bob did not wish to answer the call, or was busy on another call, an error response would have been sent instead of the 200 OK, which would have resulted in no media session being established. The complete list of SIP response codes is in Section 23. The 200 OK (message F9 in Figure 1) might look like this:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 10.2.1.1:5060;branch=4b43c2ff8.1
Via: SIP/2.0/UDP 10.1.1.1:5060;branch=77ef4c2312983.1
Via: SIP/2.0/UDP 10.1.3.3:5060
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@10.1.3.3
```

```
CSeq: 314159 INVITE
Contact: <sip:bob@10.4.1.4>
Content-Type: application/sdp
Contact-Length: 131

(Bob’s SDP not shown)
```

The first line of the response contains the response code (200) and the reason phrase (OK). The remaining lines contain header fields. The Via header fields, To, From, Call- ID, and CSeq are all copied from the INVITE request. (Note that there are three Via headers - one added by Alice’s SIP phone, one added by the atlanta.com proxy, and one added by the biloxi.com proxy.) Also note that Bob’s SIP phone has added a tag parameter to the To header field. This tag will be incorporated by both User Agents into the dialog and will be included in all future requests and responses in this call. The Contact header field contains a URI at which Bob can be directly reached at his SIP phone. The Content-Type and Content-Length refer to the not shown message body which contains Bob’s SDP media information.

In additon to DNS and location service lookups shown in this example, proxy servers can make arbitrarily complex “routing decisions” in order to decide where to send a request. For example, if Bob’s SIP phone returned a 486 Busy Here response, the biloxi.com proxy server could proxy the INVITE to Bob’s voicemail server. A proxy server can also send an INVITE to a number of locations at the same time. This type of parallel search is known as “forking”.

In this case, the 200 OK is routed back through the two proxies and is received by Alice’s softphone which then stops the ringback tone and indicates that the call has been answered. Finally, an acknowledgement message, ACK, is sent by Alice to Bob to confirm the reception of the final response (200 OK). Note that in this example, the ACK is sent directly from Alice to Bob, bypassing the two proxies. This is due to the fact that through the INVITE/200 OK exchange, the two SIP user agents have learned each other’s IP address through the Contact header fields, something which was not known when the initial INVITE was sent. The lookups performed by the two proxies are no longer needed, so they drop put of the call flow. This completes the INVITE/200/ACK three way handshake used to establish SIP sessions, and is the end of the transaction. Full details on session setup is in Section 13.

Alice and Bob’s media session has now begun, and they begin sending media packets using the format agreed to in the exchange of SDP. In general, the end-to-end media packets will take a different path from the SIP signaling messages.

During the session, either Alice or Bob may decide to change the characteristics of the media session. This is accomplished by sending a re-INVITE containing a new media description. If the change is acceptable to the other party, a 200 OK is sent which is itself responded to with an ACK. This re-INVITE will reference the existing dialog so the other party knows that it is to modify an existing session instead of establishing a new session. If the change is not acceptable, an error response, such as a 406 Not Acceptable response is sent, which also receives an ACK. However, the failure of the re-INVITE does not cause the existing call to fail - the session continues using the previously negotiated characteristics. Full details on session modification is in Section 14.

At the end of the call, Bob disconnects (hangs up) first, and generates a BYE message. This BYE is routed directly to Alice’s softphone, again bypassing the proxies. Alice confirms receipt of the BYE with a 200 OK response, which terminates the session and the BYE transaction. Note that no ACK is sent - an ACK is only sent in response to a response to an INVITE request. The reasons for this special handling for INVITE will be discussed later, but relate to the reliability mechanisms in SIP, the length of time it can take for a ringing phone to be answered, and forking. For this reason, request handling in SIP is often classified as either INVITE or non- INVITE, referring to all other methods besides INVITE. Full details on session termination is in Section 15.

Full details of all the messages shown in the example of Figure 1 are shown in Section 25.2.

In some cases, it may be useful for proxies in the SIP signaling path see all the messaging between the two endpoints for the duration of the session. For example, if the biloxi.com proxy server wished to remain in the SIP messaging path beyond the initial INVITE, it would add to the INVITE a required routing header field known as Record-Route containing a URI which resolves to the proxy. This information would be received by both Bob's SIP phone and (due to the Record-Route header field being passed back in the 200 OK) Alice's softphone and stored for the duration of the dialog. This would then result in the ACK, BYE, and 200 OK to the BYE being received and proxied by the biloxi.com proxy server. Each proxy can independently decide to receive subsequent messaging, and that messaging will go through all proxies that elected to receive it. A common use of this capability is in firewall traversal or mid-call feature implementation.

Registration is another common operation in SIP. Registration is one way in which the biloxi.com server can learn the current location of Bob. Upon initialization, and at periodic intervals, Bob's SIP phone sends REGISTER messages a server in the biloxi.com domain known as a SIP registrar. The REGISTER messages associate Bob's SIP URL (sip:bob@biloxi.com) with the machine he is currently logged in at (conveyed as a SIP URL in the Contact header). The registrar writes this association, also called a binding, to a database, called the *location service*, where it can be used by the proxy in the biloxi.com domain. Often, a registrar server for a domain is co-located with the proxy for that domain. It is an important concept that the distinction between types of SIP servers are logical, not physical.

Bob is not limited to registering from a single device. For example, both his SIP phone at home and the one in the office could send in registrations. This information is stored together in the location service, and allows a proxy to perform various types of searches to locate Bob. Similarly, more than one user can be registered on a single device at the same time.

The location service is just an abstract concept. It generally contains information that allows a proxy to input a URI and get back a translated URI that tells the proxy where to send the request. Registrations are one way to create this information, but not the only way. Arbitrarily complex mapping functions can be programmed, at the discretion of the administrator.

Finally, it is important to note that in SIP, registration is used for routing incoming SIP requests and has no role in authorizing outgoing requests. Authorization and authentication are handled in SIP either on a request by request, challenge/response mechanism, or using a lower layer scheme as discussed in Section 20.

The complete set of SIP message details for this registration example is in Section 25.2.

Additional operations in SIP include querying for the capabilities of a SIP server or client using OPTIONS, and canceling a pending request using CANCEL will be introduced in later sections.

5 Structure of the Protocol

The SIP protocol is structured as a layered protocol, which means that its behavior is described in terms of a set of fairly independent processing stages, with only a loose coupling between each stage. The structuring of the protocols into layers is for the purpose of presentation and conciseness; it allows the grouping of functions common across elements into a single place. It does not dictate an implementation in any way. When we say that an element "contains" a layer, that means it is compliant to the set of rules defined by that layer.

Not every element specified by the protocol contains every layer. Furthermore, the elements specified by SIP are logical elements, not physical ones. A physical realization can choose to act as different logical elements, perhaps even on a transaction by transaction basis.

The lowest layer of the SIP protocol is its syntax and encoding. Its encoding is specified using a BNF. The complete BNF is specified in Section 26. However, a basic overview of the structure of a SIP message can be found in Section 7. This section introduces enough of an understanding of the format of a SIP message to facilitate understanding the remainder of the protocol.

The next higher layer is the transport layer. This layer defines how a client takes a request, and physically sends it over the network, and how a response is sent by a server, and then received by a client. All SIP elements contain a transport layer. The transport layer is described in Section 19.

The next higher layer is the transaction layer. Transactions are a fundamental component of SIP. A transaction is a request, sent by a client transaction (using the transport layer), to a server transaction, along with all responses to that request sent from the server transaction back to the client. The transaction layer handles retransmissions, matching of responses to requests, and timeouts. Any task that a UAC wishes to accomplish takes place using a series of transactions. Discussion of transactions can be found in Section 17. User agents contain a transaction layer, as do stateful proxies. Stateless proxies do not contain a transaction layer.

The transaction layer has a client component (referred to as a client transaction), and a server component (referred to as a server transaction), each of which are represented by an FSM that is constructed to process a particular request. The layer on top of the transaction layer is called the transaction user (TU), of which there are several types. When a TU wishes to send a request, it creates a client transaction instance and passes it the request, along with the destination IP address, port, and transport to send the request to.

SIP provides the ability for a transaction to be canceled by the client which initiated it. When a client cancels a transaction, it requests that the server give up on further processing, revert to the state that existed before the transaction was initiated, and generate a specific error response to that transaction. This is done with a CANCEL request, which constitutes its own transaction, but references the transaction to be cancelled. Cancellation is described in Section 9.

There are several different types of transaction users. A UAC contains a UAC core, a UAS contains a UAS core, and a proxy contains a proxy core. The behavior of the UAC and UAS cores depend largely on the method. However, there are some common rules for all methods. These rules are captured in Section 8. The primarily deal with construction of a request, in the case of a UAC, and processing of that request, and generation of a response, in the case of a UAS.

UAC and UAS core behavior for the REGISTER method is described in Section 10. Registrations play an important role in SIP. In fact, a UAS that handles a REGISTER is given a special name - a registrar - and it is described in that section.

UAC and UAS core behavior for the OPTIONS method, used for determining the capabilities of a UAC, are described in Section 11.

Certain other requests are sent within a *dialog*. A dialog is a peer-to-peer SIP relationship between a two user agents that persists for some time. The dialog facilitates sequencing of messages between the user agents, and proper routing of requests between both them. One way to setup a dialog is with the INVITE method. When a UAC sends a request that is within the context of a dialog, it follows the common UAC rules as discussed in Section 8, but also the rules for mid-dialog requests. Section 12 discusses dialogs, and presents the procedures for their construction, and maintenance, in addition to construction of requests within a dialog.

The most important method in SIP is the INVITE method, which is used to establish a session between participants. A session is a collection of participants, and streams of media between them, for the purposes of communication. Section 13 discusses how sessions are initiated, resulting in one or more SIP dialogs. Section 14 discusses how characteristics of that session are modified, through the use of an INVITE request within a dialog. Finally, section 15 discusses how a session is terminated.

The procedures of Sections 8, 10, 11, 12, 13, 14, and 15 deal entirely with the UA core. Section 16 discusses the proxy element, which facilitates routing of messages between user agents.

6 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 [8]). The terms and generic syntax of URI and URL are defined in RFC 2396 [9]. The following terms have special significance for SIP.

Back-to-Back user agent: A back-to-back user agent (B2BUA) is a logical entity that receives a request, and processes it as a UAS. In order to determine how the request should be answered, it acts as a UAC and generates requests. Unlike a proxy server, it maintains dialog state, and must participate in all requests sent on the dialogs it has established. Since it is a concatenation of a UAC and UAS, no explicit definitions are needed for its behavior.

Call: A call is an informal term that refers to a dialog between peers, generally set up for the purposes of a multimedia conversation.

Call leg: Another name for a dialog.

Call stateful: A proxy is call stateful if it retains state for a dialog from the initiating INVITE to the terminating BYE request. A call stateful proxy is always stateful, but the converse is not true.

Client: A client is any network element that sends SIP requests, and receives SIP responses. Clients may or may not interact directly with a human user. *User agent clients* and *proxies* are clients.

Conference: A multimedia session (see below) that contains multiple participants.

Dialog: A dialog is a peer-to-peer SIP relationship between a UAC and UAS that persists for some time. A dialog is established by SIP messages, such as a 2xx response to an INVITE request. A dialog is identified by a call identifier, local address, and remote address. A dialog was formerly known as a call leg in RFC 2543.

Downstream: A direction of message forwarding within a transaction which refers to the direction that requests flow from the 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.

Informational Response: Same as a provisional response.

Initiator, calling party, caller: The party initiating a session with an INVITE request. A caller retains this role from the time it sends the INVITE until the termination of any dialogs established by the INVITE.

Invitation: An INVITE request.

Invitee, invited user, called party, callee: The party that receives an INVITE request for the purposes of establishing a new session. A callee retains this role from the time it receives the INVITE until the termination of the dialog established by that INVITE.

Isomorphic request or response: Two requests are defined to be *isomorphic* for the purposes of this document if they have the same values for the Call-ID, To, From, CSeq, Request-URI and the top-most Via header. Two responses are isomorphic if they have the same values for the Call-ID, To, From, CSeq and top Via header. A message which is isomorphic to another is also known as a retransmission.

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). It is an abstract database, sometimes referred to as a location server. The contents of the database can be populated in many ways, including being written by registrars.

Loop: A request that arrives at a proxy, is forwarded, and later arrives back at the same proxy. When it arrives the second time, its Request-URI is identical to the first time, and other headers that affect proxy operation are unchanged, so that the proxy would make the same processing decision on the request it made the first time around. Looped requests are errors, and the procedures for detecting them and handling them are described by the protocol.

Method: The method is the primary function that a request is meant to invoke on a server. The method is carried in the request message itself. Example methods are INVITE and BYE.

Outbound proxy: A *proxy* that receives all requests from a client, even though it is not the server resolved by the Request-URI. The outbound proxy sends these requests, after any local processing, to the address indicated in the Request-URI, or to another outbound proxy.

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 entity that acts as both a server and a client for the purpose of making requests on behalf of other clients. A proxy server primarily plays to role of routing, which means its job is to ensure that a request is passed on to another entity that can further process the request. Proxies are also useful for enforcing policy and for firewall traversal. A proxy interprets, and, if necessary, rewrites parts of a request message before forwarding it.

Registrar: A registrar is a server that accepts REGISTER requests, and places the information it receives in those requests into the location service for the domain it handles.

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 party's application indicating that a called party is being alerted (ringing).

Server: A server is a network element that receives requests in order to service them, and sends back responses to those requests. Examples of servers are proxies, user agent servers, redirect servers, and registrars.

Sequential search: In a sequential search, a proxy server attempts each contact address in sequence, proceeding to the next one only after the previous has generated a non-2xx final response.

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, and the ACK for the response in the case the response was a 2xx. The ACK for a 2xx response is a separate transaction.

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, differs in a way which will result in a different processing decision than the original request. Typically, this means that it has 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 the client or server transaction state machines defined in this specification when it processes requests. A stateless proxy forwards every request it receives downstream and every response it receives upstream.

Stateful proxy: A logical entity that maintains the client and server transaction state machines defined by this specification during the processing of a request. Also known as a transaction stateful proxy. The behavior of a stateful proxy is further defined in Section 16. A stateful proxy is not the same as a call stateful proxy.

Transaction User (TU): The layer of protocol processing that resides above the transaction layer. Transaction users include the UAC core, UAS core, and proxy core.

Upstream: A direction of message forwarding within a transaction which refers to the direction that responses flow from the user agent server to user agent client.

URL-encoded: A character string encoded according to RFC 1738, Section 2.2 [10].

User agent client (UAC): A user agent client is a logical entity that creates a new request, and then uses the client transaction state machinery to send it. The role of UAC 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 transaction. If it receives a request later on, it takes on the role of a User Agent Server for the processing of that transaction.

UAC Core: The set of processing functions required of a UAC that reside above the transaction and transport layers.

User agent server (UAS): A user agent server is a logical entity that generates a response to a SIP request. 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 transaction. If it generates a request later on, it takes on the role of a User agent client for the processing of that transaction.

UAS Core: The set of processing functions required at a UAS that reside above the transaction and transport layers.

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

The role of UAC and UAS as well as proxy and redirect servers are defined on a transaction-by-transaction 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, location and registrar servers defined above are *logical* entities; implementations MAY combine them into a single application program.

7 SIP Messages

SIP is a text-based protocol and uses the ISO 10646 character set in UTF-8 encoding (RFC 2279 [11]).

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

Both Request (section 7.1) and Response (section 7.2) messages use the generic-message format of RFC 822 [12]. Both types of messages consist of a start-line, one or more header fields (also known as "headers"), an empty line indicating the end of the header fields, and an optional message-body.

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

The start-line, each message-header line, and the empty line MUST be terminated by a carriage-return line-feed sequence (CRLF). Note that the empty line MUST be present even if the message-body is not.

Except for the above difference in character sets, much of SIP's message and header field syntax is 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 [8]).

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

7.1 Requests

SIP Requests are distinguished by having a Request-Line for a start-line. A 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 end-of-line CRLF sequence. No LWS is allowed in any of the elements.

Method Request-URI SIP-Version

- Method

  This specification defines six methods : REGISTER for registering contact information, INVITE, ACK and CANCEL for setting up sessions, BYE for terminating sessions and OPTIONS for querying servers about their capabilities. SIP extensions may define additional methods.

- Request-URI

  The Request-URI is a SIP URL as described in Section 21.1 or a general URI (RFC 2396 [9]). It indicates the user or service to which this request is being addressed. The Request-URI MUST NOT contain unescaped spaces or control characters and MUST NOT be enclosed in ”<>”.

  SIP servers MAY support Request-URIs with schemes other than “sip”, for example the “tel” URI scheme of RFC 2806 [13]. It MAY translate non-SIP URIs using any mechanism at its disposal, resulting in either a SIP URI or some other scheme.

- 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 send upper-case.

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

7.2 Responses

SIP Responses are distinguished by having a Status-Line for a start-line. A Status-Line, consists of the protocol version 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.

SIP-version Status-Code Reason-Phrase

The Status-Code is a 3-digit integer result code that indicates the outcome of an attempt to understand and satisfy a 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. A client is not required to examine or display the Reason-Phrase.

The first digit of the Status-Code defines the class of response. The last two digits do not have any categorization role. For this reason, any response with a status code between 100 and 199 is referred to as a “1xx response”, any response with a status code between 200 and 299 as a “2xx response”, and so on. SIP/2.0 allows 6 values for the first digit:

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

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

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

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

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

6xx: Global Failure – the request cannot be fulfilled at any server.

Full definitions of these classes and each registered code appear in Section 23.

7.3 Header Fields

SIP header fields are similar to HTTP header fields in both syntax and semantics. In particular, SIP header fields follow the [H4.2] definitions of syntax for message-header, the rules for extending header fields over multiple lines, the use of multiple message-header fields with the same field-name, and the rules regarding ordering of header fields.

7.3.1 Header Field Format

Header fields follow the same generic header format as that given in Section 3.1 of RFC 822 [12]. Each header field consists of a field name followed by a colon (":") and the field value.

field-name: field-value

Note that the formal grammar for a message-header specified in Section 26 allow for an arbitrary amount of whitespace on either side of the colon. No space before the colon and a single space (SP) between the colon and the field-value is preferred. That is,

```
Subject:            lunch
Subject      :      lunch
Subject            :lunch
Subject: lunch
```

are all valid, and equivalent, but the last is the preferred form.

Header fields can be extended over multiple lines by preceding each extra line with at least one SP or horizontal tab (HT). The line break and the whitespace at the beginning of the next line are treated as a single SP character. Thus the following are equivalent:

```
Subject: I know you're there, pick up the phone and talk to me!
Subject: I know you're there,
         pick up the phone
         and talk to me!
```

The relative order of header fields with different field names is not significant. The relative order of those with the same field name is important. 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.

Implementations MUST be able to process multiple header fields with the same name in any combination of the single-value-per-line or comma-separated value forms.

The following blocks of headers are valid and equivalent:

```
Route: sip:alice@atlanta.com
Subject: Lunch
Route: sip:bob@biloxi.com
Route: sip:carol@chicago.com

Route: sip:alice@atlanta.com, sip:bob@biloxi.com
Route: sip:carol@chicago.com
Subject: Lunch

Subject: Lunch
Route: sip:alice@atlanta.com, sip:bob@biloxi.com, sip:carol@chicago.com
```

Each of the following blocks is valid but not equivalent to the others:

```
Route: sip:alice@atlanta.com
Route: sip:bob@biloxi.com
Route: sip:carol@chicago.com

Route: sip:bob@biloxi.com
Route: sip:alice@atlanta.com
Route: sip:carol@chicago.com

Route: sip:alice@atlanta.com,sip:carol@chicago.com,sip:bob@biloxi.com
```

The format of a header field-value is defined per header-name. It will always be either an opaque sequence of TEXT-UTF8 octets, or a combination of whitespace, tokens, separators, and quoted strings. Many of them will adhere to the general form of a value followed by a semi-colon separated sequence of parameter-name, parameter-value pairs:

field-name: field-value *(;parameter-name=parameter-value)

When comparing headers, field names are always case-insensitive. Unless otherwise stated in the definition of a particular header field, field values, parameter names, and parameter values (tokens in general) are case-insensitive. Unless specified otherwise, values expressed as quoted strings are case-sensitive.

The following are equivalent:

```
Contact: <sip:alice@atlanta.com>;expires=3600
CONTACT: <sip:alice@atlanta.com>;ExPiReS=3600

Contact-Disposition: session;handling=optional
contact-disposition: Session;HANDLING=OPTIONAL
```

The following are not equivalent;

```
Warning: 370 devnull "Choose a bigger pipe"
Warning: 370 devnull "CHOOSE A BIGGER PIPE"
```

7.3.2 Header Field Classification

Some header fields only make sense in requests or responses. These are called Request Header Fields and Response Header fields respectively. Those header fields that can appear in either a request or response are called General Header Fields. If a header appears in a message not matching its category (such as a request header in a response), it MUST be ignored. Section 22 defines the classification of each header.

7.3.3 Compact Form

SIP provides a mechanism to represent common header fields in an abbreviated form. This may be useful when messages would otherwise become to large to be carried on the transport available to it (exceeding the MTU when using UDP for example). These compact forms are defined in Section 22. A compact form MAY be substituted for the longer form of a header name at any time without changing the semantics of a the message. Multiple header fields in a message with the same header name MAY appear with an arbitrary mix of its long and short field name form. Implementations MUST accept both the long and short forms of each header name.

7.4 Bodies

Requests, including new requests defined in extensions to this specification, MAY contain message bodies unless otherwise noted.

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.

7.4.1 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 defined in RFC 2046 [14] MAY be used within the body of the message. Implementations that send requests containing multipart message bodies MUST be able to send a session description as a non-multipart message body if the remote implementation requests this through an Accept header field.

7.4.2 Message Body Length

The body length in bytes is provided by the Content-Length header field. Section 22.14 describes the necessary contents of this header 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.)

7.5 Framing SIP messages

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 [15, 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 [16]). Thus, another reasonable value would be a message size of 950 bytes, to accommodate packet headers within the SLIP MTU without fragmentation.

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.

Likewise, Implementations processing SIP messages over stream oriented transports MUST ignore noise between messages.

8 General User Agent Behavior

A user agent represents an end system. It contains a User Agent Client (UAC), which generates requests, and a User Agent Server (UAS) which responds to them. A UAC is capable of generating a request based on some external stimulus (the user clicking a button, or a signal on a PSTN line), and processing a response. A UAS is capable of receiving a request, and generating response, based on user input, external stimulus, the result of a program execution, or some other mechanism.

When a UAC sends a request, it will pass through some number of proxy servers, which forward the request towards the UAS. When the UAS generates a response, the response is forwarded towards the UAC.

UAC and UAS procedures depend strongly on two factors. First, whether the request or response is inside or outside of a dialog, and second, based on the method of a request. Dialogs are discussed thoroughly in Section 12; they represent a peer-to-peer relationship between user agents, and are established by specific SIP methods, such as INVITE.

In this section, we discuss the method independent rules for UAC and UAS behavior when processing of requests that are outside of a dialog. This includes, of course, the requests which themselves establish a dialog.

8.1 UAC Behavior

8.1.1 Generating the Request

A valid SIP request formulated by a UAC MUST at a minimum contain the following headers: To, From, CSeq, Call-ID, and Via; all of these headers are mandatory in all SIP messages. These five headers are the fundamental building blocks of a SIP message, as they jointly provide for most of the critical message routing services including the addressing of messages, the routing of responses, ordering of messages, and the unique identification of transactions.

Examples of requests send outside of a dialog include an INVITE to establish a session (Section 13) and an OPTIONS to query for capabilities (Section 11).

8.1.1.1 To The To general-header field first and foremost specifies the desired "logical" recipient of the request, or the address of record of the user or resource that is the target of this request. This may or may not be the ultimate recipient of the request. The To header MAY contain a SIP URI, but it may also make use of other URI schemes (for example as the tel URL [13]) when appropriate. The To header field allows for a display name; this is meant to contain a descriptive version of the URI, and is intended to be displayed to a user interface.

A UAC may learn how to populate the To header field for a particular request in a number of ways. Usually the user will suggest the To header field through a human interface, perhaps inputting the URI manually or selecting it from some sort of address book.

A request outside of a dialog MUST NOT contain a tag; the tag in the To field of a request identifies the peer of the dialog. Since no dialog is established, no tag is present.

For further information on the To header see Section 22.37.

The following is an example of valid To header:

```
To: Carol <sip:carol@chicago.com>
```

8.1.1.2 From The From general-header field indicates the logical identity of the initiator of the request, possibly the user's address of record. Like the To field, it contains a URI and optionally a display name. It is used by SIP elements to determine processing rules to apply to a request (for example, automatic call rejection). As such, it is very important that the URI not contain IP addresses or host names, since these are not logical names.

The From header field allows for a display name; this is meant to contain a descriptive version of the URI, and is intended to be displayed to a user interface. A UAC SHOULD use the display name "Anonymous" if the identity of the client is to remain hidden.

Usually the value that populates the From header field in requests generated by a particular user agent is pre-provisioned by the user or by the administrators of the user's local domain. If a particular user agent is used by multiple users, it might have switchable profiles that include a URI corresponding to the identity of the profiled user. Recipients of requests can authenticate the originator of a request in order to ascertain that they are who their From header field claims they are (see Section 20.2 for more on authentication).

The From field MUST contain a new "tag" parameter, chosen by the UAC. See Section 21.3 for details on choosing a tag.

For further information on the From header see Section 22.20.

Examples:

```
From: "Bob" <sip:bob@biloxi.com> ;tag=a48s
From: sip:+12125551212@server.phone2net.com;tag=887s
From: Anonymous <sip:c8oqz84zk7z@privacy.org>;tag=hyh8
```

8.1.1.3 Call-ID The Call-ID general-header field acts as a unique identifier to group together series of messages. It is always the same for all requests and responses sent by either UA in a dialog. It is also the same in each registration from a UA within a single boot cycle.

In a new request created by a UAC outside of any dialog, unless overridden by method specific behavior, it MUST be selected by the UAC as a a globally unique identifier over space and time; all SIP user agents must have a means to guarantee that the Call-ID headers they produce will not be inadvertently generated by any other user agent.

Use of cryptographically random identifiers [17] in the generation of Call-IDs 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, and reduces the likelihood of unintentional Call-ID collisions.

No provisioning or human interface is required for the selection of the Call-ID header field value for a request.

For further information on the Call-ID header see Section 22.8.

Example:

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

8.1.1.4 CSeq The Cseq header serves as a way to identify and order transactions. It consists of a sequence number and a method. The method MUST match that of the request. The sequence number value is arbitrary, but MUST be expressible as a 32-bit unsigned integer and MUST be less than 2**31.

As long as it follows the above guidelines, a client may use any mechanism it would like to select CSeq header field values.

For further information on the CSeq header see Section 22.16.

Example:

```
CSeq: 4711 INVITE
```

8.1.1.5 Via The Via header is used to determine the transport to use for sending a request, and for identifying the IP address and port where the response is to be sent. Rules for setting and using the values in this header are described in Section 19.

For further information on the Via header see Section 22.40.

8.1.1.6 Contact The Contact header provides a SIP URI that can be used to contact that specific instance of the user agent for subsequent requests. The Contact header MUST be present in any request that can result in the establishment of a dialog. For the methods defined in this specification, that includes only the INVITE request. For these requests, the scope of the Contact is the dialog. That is, the Contact header refers to the URL that the UA would like to receive requests at, for requests that are part of that dialog only. Only a single URI MUST be present.

For further information on the Contact header, see Section 22.10.

8.1.1.7 Request-URI The initial Request-URI of the message SHOULD be set to the value of the URI in the To field. One notable exception is the REGISTER method; behavior for setting the Request-URI of register is given in Section 10. Another exception is the case of pre-existing Route headers; in that case, the procedures of Section 12.2.1.1 as they pertain to the Request- URI are followed, even though there is no dialog.

8.1.1.8 Supported and Require If the UAC supports extensions to SIP that can be applied by the server to the response, the UAC SHOULD include a Supported header in the request listing the option tags for those extensions.

The option-tags listed MUST only refer to extensions defined in standards track RFCs. This is to prevent servers from insisting that clients implement non-standard, vendor defined features in order to receive service. Extensions defined by experimental and informational RFCs are explicitly excluded from usage with the Supported header in a request, since they too are often used to document vendor defined extensions.

If the UAC wishes to insist that a UAS understand an extension that the UAC will apply to the request in order to process the request, it MUST insert a Require header into the request listing the option tag for that extension. If the UAC wishes to apply an extension to the request and insist that a proxy understand that extension, it MUST insert a Proxy-Require header into the request listing the option tag for that extension.

8.1.1.9 Additional Message Components After a new request has been created, the headers described above have been properly constructed, any additional optional headers are added, as are any headers specific to the method.

SIP requests MAY contain a MIME-encoded message-body. Regardless of the type of body that a request contains, certain headers must be formulated to characterize the contents of the body. For further information on these headers see Section 7.4.

8.1.2 Sending the Request

The destination for the request is then computed. This can be a preconfigured IP address, port and transport of an outbound proxy, or it can be determined through DNS procedures applied to the Request-URI. These procedures are described in Section 24, which yield an ordered set of address, port and transports to attempt. The UAC SHOULD follow the procedures defined there for stateful elements, trying each address until a server is contacted. Each try constitutes a new transaction, and therefore a new client transaction MUST be constructed for each.

8.1.3 Processing Responses

Responses are first processed by the transport layer, and then passed up to the transaction layer. The transaction layer performs its processing, and then passes it up to the TU. The majority of response processing in the TU is method specific. However, there are some general behaviors independent of the method.

8.1.3.1 Unrecognized Responses A UAC MUST treat any response they do not recognize as being equivalent to the x00 response code of that class, and MUST be able to process the x00 response code for all classes. For example, if a UAC 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.

8.1.3.2 Vias If more than one Via header field is present in a response, the UAC SHOULD discard the message.

> The presence of additional Via header fields that precede the originator of the request suggests that the message was misrouted or possibly corrupted.

8.1.3.3 Processing 3xx responses Upon receipt of a redirection response (e.g. a 3xx response status code), clients SHOULD use the URI(s) in the Contact header field to formulate a new request.

To do that, 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.

In all other respects, requests sent upon receipt of a redirect response SHOULD re-use the headers and bodies of the original request.

The Contact values present in redirection responses SHOULD NOT be cached across calls, as they may not represent the most desirable location for a particular destination address.

8.1.3.4 Processing 4xx responses Certain 4xx response codes require specific UA processing, independent of the method.

If a 401 or 407 response is received, the UAC SHOULD follow the authorization procedures of Section 20.2.2 and Section 20.2.3 to retry the request with credentials.

If a 413 response is received (Section 23.4.11), it means that the request contained a body that was longer than the UAS was willing to accept. If possible, the UAC SHOULD retry the request, either omitting the body or using one of a smaller length.

If a 415 response is received (Section 23.4.13), it means the request contained media types not supported by the UAS. The UAC SHOULD retry sending the request, this time only using content with types listed in the Accept header in the response, with encodings listed in the Accept-Encoding header in the response, and with languages listed in the Accept-Language in the response.

If a 420 response is received (Section 23.4.14), it means the request contained a Require or Proxy-Require header listing an option-tag for a feature not supported by a proxy or UAS. The UAC SHOULD retry the request, this time omitting any extensions listed in the Unsupported header in the response.

In all of the above cases, retrying the request is accomplished by creating a new request with the appropriate modifications. This new request SHOULD have the same value of the Call-ID, To, and From of the previous request, but the CSeq should contain a new sequence number that is one higher than the previous.

With other 4xx responses, a retry may or may not be possible depending on the method and the use case.

8.2 UAS Behavior

When a request outside of a dialog is processed by a UAS, there are a set of processing rules which are followed, independent of the method. Section 12 gives guidance on how a UAS can tell whether a request is inside or outside of a dialog.

8.2.1 Authentication/Authorization

A UAS MAY authenticate the originator of a request, and this process may require the server to issue a challenge for credentials. The required behavior is independent of the method of the request, and is detailed in Section 20.2.

8.2.2 Method Inspection

Once a request is authenticated (or no authentication was desired), the UAS MUST inspect the method of the request. If the UAS does not support the method of a request it MUST generate a 405 (Method Not Allowed) response. Procedures for generation of responses are described in Section 8.2.7. The UAS MUST also add an Allow header to the 405 response. The Allow header field MUST list the set of methods supported by the UAS generating the message.

The Allow header is presented in Section 22.5.

If the method is one supported by the server, processing continues.

8.2.3 Header Inspection

If a UAS does not understand a header field in a request (i.e. the header is not defined in this specification or in any supported extension), the server MUST ignore that header and continue processing the message. A UAS SHOULD ignore any malformed headers which are not necessary for processing requests.

8.2.3.1 To and Request-URI The To header field identifies the original recipient of the request designated by the user identified in the From field. The original recipient may or may not be the UAS processing the request, do to call forwarding or other proxy operations. A UAS MAY apply any policy it wishes in determination of whether to accept requests when the To field is not the identity of the UAS. However, it is RECOMMENDED that a UAS accept requests even if they do not recognize the URI scheme (e.g., a `tel:` URI) in the To header, or if the To header does not address a known or current user of this UAS. If, on the other hand, the UAS decides to reject the request, it SHOULD generate a response with a 403 status code and send it to the server transaction for transmission.

However, the Request-URI identifies the UAS that is to process the request. If the Request-URI does not identify an address that the UAS is willing to accept requests for, it SHOULD reject the request with a 404 (Not Found) response. If the Request-URI does not provide sufficient information for the UAS to determine whether it is willing to process the request, it SHOULD return a 485 (Ambiguous) response. This response SHOULD contain a Contact header field containing URIs of new addresses to be tried. Typically, a UA which uses the REGISTER method to bind its address of record to a specific contact address, will see requests whose Request-URI equals those contact addresses.

8.2.3.2 Require Assuming the UAS decides that it is the proper element to process the request, it examines the Require header field, if present.

The Require general-header field is used by UAC to tell UAS about SIP extensions that the UAC expects the UAS to support in order to properly process the request. If a UAS does not understand an option listed in a Require header field, it MUST respond by generating a response with status code 420 (Bad Extension). The UAS MUST add a Unsupported, and list in it those options it does not understand amongst those in the Require header of the request. Upon receipt of the 420 the client SHOULD retry the request, this time without using those extensions listed in the Unsupported header in the response.

Example:

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

UASS->UAC:   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.

8.2.4 Content Processing

Assuming the UAS understands any extensions required by the client, the UAS examines the body of the message, and the headers that describe it. If there are any bodies whose type (indicated by the Content-Type), language (indicated by the Content-Language) or encoding (indicated by the Content-Encoding) are not understood, and that body part is not optional (as indicated by the Content-Disposition) header, the UAS MUST reject the request with a 415 (Unsupported Media Type) response. The response MUST contain a Accept header listing the types of all bodies it understands, in the event the request contained bodies of types not supported by the UAS. If the request contained content encodings not understood by the UAS, the response MUST contain an Accept-Encoding header listing the encodings understood by the UAS. If the request contained content with languages not understood by the UAS, the response MUST contain an Accept-Language header indicating the languages understood by the UAS.

Beyond these checks, body handling is method and type specific.

For further information on the processing of Content-specific headers see Section 7.4.

8.2.5 Applying Extensions

A UAS that wishes to apply some extension when generating the response MUST only do so if support for that extension is indicated in the Supported header in the request. If the desired extension is not supported, the server SHOULD rely only on baseline SIP and any other extensions supported by the client. To ensure that the SHOULD can be fulfilled, any specification of a new extension MUST include discussion of how to gracefully return to baseline SIP when the extension is not present. In rare circumstances, where the server cannot process the request without the extension, the server MAY send a 421 (Extension Required) response. This response indicates that the proper response cannot be generated without support of a specific extension. The needed extension(s) MUST be included in a Require header in the response. This behavior is NOT RECOMMENDED, as it will generally break interoperability.

Any extensions applied to a non-421 response MUST be listed in a Require header included in the response. Of course, the server MUST NOT apply extensions not listed in the Supported header in the request. As a result of this, the Require header in a response will only ever contain option tags defined in standards track RFCs.

8.2.6 Processing the Request

Assuming all of the checks in the previous subsections are passed, the UAS processing becomes method specific. Section 10 deals with the REGISTER request, section 11 deals with the OPTIONS request, section 13 deals with the INVITE request, and section 15 deals with the BYE request.

8.2.7 Generating the Response

When a UAS wishes to construct a response to a request, it follows these procedures. Additional procedures may be needed depending on the status code of the response and the circumstances of its construction. These additional procedures are documented elsewhere.

The From field of the response MUST equal the From field of the request. The Call-ID field of the response MUST equal the Call-ID field of the request. The Cseq field of the response MUST equal the Cseq field of the request. The Via headers in the response MUST equal the Via headers in the request, and MUST maintain the same ordering.

If a request contained a To tag in the request, the To field in the response MUST equal that of the request. However, if the To field in the request did not contain a tag, the URI in the To field in the response MUST equal the URI in the To field in the request. Additionally, the UAS MUST add a tag to the To field in the response. This serves to identify the UAS that is responding, possibly resulting in a component of a dialog ID. The same tag MUST be used for all responses to that request, both provisional and final. Procedures for generation of tags are defined in Section 21.3.

8.3 Redirect Servers

In some architectures it may be desirable to reduce the processing load on proxy servers that are responsible for routing requests by relying on redirection. Redirection allows servers to push routing information for a request back in a response to the client, thereby taking themselves out of the loop of further messaging for this transaction while still aiding in locating the target of the request. When the originator of the request receives the redirection it will send a new request based on the routing information it has received. By propagating routing information from the core of the network to its edges, redirection allows for considerable network scalability.

A redirect server is logically constituted of a server transaction layer and a transaction user that has access to a location service of some kind (see Section 10 for more on registrars and location services). This location service is effectively a database containing mappings between a single URI and a set of one or more alternative locations at which the target of that URI can be found.

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 from the location service and either 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 an entire SIP transaction. It is the responsibility of clients to detect forwarding loops between redirect servers.

When a redirect server returns a 3xx response to a request, it populates the list of (one or more) alternative locations into Contact headers. An "expires" parameter to the Contact header may also be supplied to indicate the lifetime of the Contact data.

The Contact header field contains URIs giving the new locations or user names to try, or may simply specify additional transport parameters. A 301 or 302 response may also give the same location and username that was targeted by the initial request 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.

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 informational 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 URIs. For example, it could contain URL's for phones, fax, or irc (if they were defined) or a mailto: (RFC 2368, [18]) URL.

The "expires" parameter of the Contact header field 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.

Redirect servers MUST ignore features that are not understood (including unrecognized headers, Required extensions, or even method names) and proceed with the redirection of the session in question. If a particular extension requires that intermediate devices support it, the extension MUST be tagged in the Proxy-Require field as well (see Section 22.28).

9 Canceling a Request

The previous section has discussed general UA behavior for generating requests, and processing responses, for requests of all methods. In this section, we discuss a general purpose method, called CANCEL.

The CANCEL request, as the name implies, is used to cancel a previous request sent by a client. Specifically, it asks the user agent server to cease processing the request, and generate an error response to that request. CANCEL has no effect on a request that has already been responded to. Because of this, it is most useful to CANCEL requests which can take a long time to respond to. For this reason, CANCEL is most useful for INVITE requests, which can take a long time to generate a response. In that usage, a UAS that receives a CANCEL request for an INVITE, but has not yet sent a response, would "stop ringing", and then respond to the INVITE with a specific error response (a 487).

Cancel requests can be constructed and sent by any type of client, including both proxies and user agent servers. Section 15 discusses under what conditions a UAC would CANCEL an INVITE request, and Section 16 discusses proxy usage of INVITE.

Because a stateful proxy can generate its own CANCEL, a stateful proxy also responds to a CANCEL, rather than simply forwarding a response it would receive from a downstream element. For that reason, CANCEL is referred to as a "hop-by-hop" request, since it is responded to at each stateful proxy hop.

9.1 Client Behavior

The following procedures are used to construct a CANCEL request. The Request-URI, Call-ID, To, the numeric part of CSeq and From header fields in the CANCEL request MUST be identical to those in the request being cancelled, including tags. A CANCEL constructed by a client MUST have only a single Via header, whose value matches the top Via in the request being cancelled. Using the same values for these headers allows the CANCEL to be matched with the request it cancels (Section 9.2 indicates how such matching occurs). However, the method part of the Cseq header MUST have a value of CANCEL. This allows it to be identified and processed as a transaction in its own right (See Section 17).

Once the CANCEL is constructed, the client SHOULD check whether any response (provisional or final) has been received for the request being cancelled (herein referred to as the "original request"). The CANCEL request MUST NOT be sent if no provisional response has been received, rather, the client MUST wait for the arrival of a provisional response before sending the request. If the original request has generated a final response, the CANCEL SHOULD NOT be sent, as it is an effective no-op, since CANCEL has no effect on requests which have already generated a final response. When the client decides to send the CANCEL, it creates a client transaction for the CANCEL, and passes it the CANCEL request along with the destination address, port and transport. The destination address, port, and transport for the CANCEL MUST be identical to those used to send the original request.

> If it was allowed to send the CANCEL before receiving a response for the previous request the server could receive the CANCEL before the original request.

Note that both the transaction corresponding to the original request and the CANCEL transaction will complete independently. However, a UAC canceling a request cannot rely on receiving a 487 (Request Terminated) response for the original request, as an RFC 2543-compliant UAS will not generate such a response. If there is no final response for the original request in 64*T1 seconds for an INVITE transaction, and T3 seconds for a non-INVITE transaction, the client SHOULD then consider the original transaction cancelled and SHOULD destroy the client transaction handling the original request.

9.2 Server Behavior

The CANCEL method requests that the TU at the server side cancel a pending request with the same Call-ID, To, From, top Via header and Request-URI and CSeq (sequence number only) header field values.

The processing of a CANCEL request at a server depends on the type of server. A stateless proxy will forward it, a stateful proxy might respond to it and generate some CANCEL requests of its own, and a UAS will respond to it. See Section 16.8 for proxy treatment of CANCEL.

When a UAS receives a CANCEL, it looks for any server transactions which were created by requests with the same To, From, Call-ID, Cseq numeric value, Request-URI and top Via header. If no matching transactions are found, the CANCEL is responded to with a 481 (Call Leg/Transaction Does Not Exist). If the transaction for the original request still exists, the behavior of the UAS on receiving a CANCEL request depends on whether it has already sent a final response for original request. If it has, the CANCEL request has no effect on the processing of the original request, no effect on any session state, and no effect on the responses generated for the original request. If the UAS has not issued a final response for the original request, it immediately responds to the original request with a 487 (Request Terminated).

The CANCEL request itself is answered with a 200 (OK) response in either case. Once the response is constructed it is passed to the server transaction for the CANCEL request.

10 Registrations

10.1 Overview of Usage

SIP is a protocol that offers a discovery capability. For one user to initiate a session with another, SIP must discover the current host(s) that the called user is reachable at. This discovery process is accomplished by SIP proxy servers, which are responsible for receiving a request, determining where to send it based on knowledge of the location of the user, and then sending it there. To do this, proxies consult an abstract service known as a *location service*, which provides address bindings for a particular domain. These address bindings map an incoming SIP URL, `sip:bob@Biloxi.com`, for example, to one or more SIP URLs which are somehow “closer” to the desired user, `sip:bob@engineering.Biloxi.com`, for example. Ultimately, a proxy will consult a location service which maps a received URL to the current host(s) that a user is logged in to.

There are many ways by which the contents of the location service can be established. One way is administratively. In the above example, Bob is known to be a member of the engineering department through access to a corporate database. SIP provides a mechanism, however, for a user agent to explicitly create a binding in the location service of a proxy. This mechanism is known as registration.

The process of registration entails sending a REGISTER message to a special type of UAS known as a registrar. The registrar acts as a front end to the location service for a domain, reading and writing mappings based on the contents of the REGISTER messages. This location service will then be consulted by a proxy server that is responsible for routing requests for that domain.

SIP does not mandate a particular mechanism for implementing the location service. The only requirement is that a registrar for some domain MUST be capable of reading and writing data to the location service, and a proxy for that domain MUST be capable of reading that same data. A registrar MAY be co-located with a particular SIP proxy server for the same domain, allowing usage of an in memory database for the location service. Usage of a shared database is another implementation choice. The choice depends entirely on the architectural requirements (redundancy, scalability, etc) of a particular deployment.

Registration creates bindings in a location service for a particular domain that associate an "address of record" URI with one or more "contact addresses". This means that when a proxy for that domain receives a request whose request URI matches the address of record, the proxy will forward the request to the contact addresses registered to that address of record. Generally, it only makes sense to register an address of record at a location service for a domain when requests for that address of record would be routed to that domain. In most cases, this means that the domain of the registration will need to match the domain in the URI of the address of record.

The most important usage of the registration mechanism is to inform a proxy of the mapping between the address of record and the current host on which the UA resides. However, the registration process is a general mechanism for establishing bindings, and can be used for other purposes (for example, to set up call forwarding).

```
                                              bob
                                            +----+
                                            | UA |
                                            |    |
                                            +----+
                                               |
                                               |3)INVITE
                                               |   carol@chicago.com
      chicago.com        +--------+            V
      +---------+ 2)Store|Location|4)Query +-----+
      |Registrar|=======>| Service|<=======|Proxy|sip.chicago.com
      +---------+        +--------+=======>+-----+
            A                      5)Resp      |
            |                                  |
            |                                  |
  1)REGISTER|                                  |
            |                                  |
         +----+                                |
         | UA |<-------------------------------+
cube2214a|    |                           6)INVITE
         +----+                    carol@cube2214a.chicago.com
          carol
```

Figure 2: REGISTER example

10.2 Construction of the REGISTER request

Several operations can be performed with a REGISTER method with respect to a registrar. One of these is the basic registration operation that is described above, which provides a new binding between an address of record and one or more contact addresses. Registration on behalf of a particular address of record may be performed by a third party if they are authorized to do so. A client may also remove previous bindings, or query to determine which bindings are currently in place for an address of record.

Aside from the exceptions noted in this and the following sections, the construction of the REGISTER method, and behavior of clients sending a REGISTER is identical to the general UAC behavior described in Section 8.1 and Section 17.1. Regardless of the operation that is performed by a REGISTER, the following header fields MUST be formulated as follows:

Request-URI: The Request-URI names the domain of the location service that the registration is meant for (e.g. “chicago.com”). The user name MUST be empty.

To: The To header field contains the address of record whose registration is to be created or modified. Note that the initial To header field and the Request-URI field SHOULD therefore be different in a REGISTER message.

From: The From header field contains the address of record of the person responsible for the registration, which MAY be identical to the value of the To header field. For third-party registrations the From header field and To header field are different.

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

> If different Call-IDs were used for overlapping REGISTER messages coming from the same client, the registrar might have trouble determining their ordering.

Contact: REGISTER requests MAY contain one or more Contact header fields. Contact addresses are presented in the Contact header fields of REGISTER requests.

Note that user agents MUST NOT send a new registration (containing new Contact header fields, as opposed to a retransmission) until they have received a response from the registrar for the previous one.

The following optional Contact header parameters also contain behavior specific to the registration process.

action: The “action” parameter has been deprecated. UACs SHOULDNOT use the “action” parameter.

expires: The “expires” parameter indicates how long the UAC would like the binding to be 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 binding is valid. Implementations MAY treat values larger than 2**32-1 (4294967295 seconds or 136 years) as equivalent to 2**32-1.

10.2.1 Adding Bindings with REGISTER

For a simple registration, a REGISTER request sent to a registrar includes contact addresses to which requests should be forward for the originating user’s address of record. The address of record itself (i.e.

'sip:carol@chicago.com') MUST populate the To header of the REGISTER. The Contact header fields of the request typically contain SIP URIs that identify particular SIP endpoints (i.e. 'sip:carol@cube2214a.chicago.com'), but they MAY use any URI scheme; this way a SIP UA can choose to register telephone numbers (with the tel URL, [13]) or email addresses (with a mailto URL, [18]) as Contacts for an address of record.

For example, if Carol, whose address of record is 'sip:carol@chicago.com', needed to register, she would typically want to register with the registrar associated with the location service of chicago.com. This location service would then be accessed by a proxy server that receives requests targeting users in the chicago.com domain, and hence new requests for Carol's address of record will be routed to her SIP endpoint.

Once a client has established bindings at a registrar, it MAY send subsequent registrations containing new bindings or modifications to pre-existing bindings as necessary. The 2xx response to the REGISTER message will contain (in Contact header fields) a complete list of bindings that have been registered for this address of record at this registrar.

10.2.1.1 Setting the Expiration Interval of Contact Addresses When a client sends a REGISTER request, it MAY suggest an expiration interval that indicates how long the client would like the registration to be valid (although as is detailed in Section 10.3, the registrar has the ultimate say).

There are two ways in which a client can suggest an expiration interval for a binding: through an Expires header, or an "expires" Contact header parameter. The latter allows expiration intervals to be suggested on a per-binding basis when more than one binding is given in a single REGISTER, whereas the former suggests an expiration interval for all Contact header fields that do not contain the "expires" parameter.

If neither mechanism for expressing a suggested expiration time is present in a REGISTER, a default suggestion of one hour is assumed.

10.2.1.2 Setting Preference among Contact Addresses If more than one Contact is sent in a REGISTER, then the registering UA intends to associate all of the URIs given in these Contact headers with the address of record present in the To field. This list can be prioritized with the "q" mechanism.

q: The "q" parameter indicates a relative preference for the particular Contact header field compared to other bindings present in this REGISTER message or existing within the location service of the registrar. For an example of how a proxy server uses "q" values, see Section 16.5.

10.2.2 Removing Bindings with REGISTER

Registrations are removed from the registrar through an expiration process; registrations are soft state and need to be refreshed periodically. A client may attempt to influence the expiration intervals selected by the registrar as described in Section 10.2.1.

A registering user agent requests the immediate removal of a binding by specifying an expiration interval of "0" for that contact address in a REGISTER. It is RECOMMENDED that user agents support this mechanism so that bindings can be removed (for whatever reason) before their expiration interval has passed.

The REGISTER-specific Contact header field value of "*" applies to all registrations, but it MUST only be used when the Expires header is present with a value of "0".

> Use of the "*" Contact header field value allows a registering user agent to remove all of its bindings expediently.

10.2.3 Fetching Bindings with REGISTER

If no Contact headers are present in a REGISTER, then the UA is not in fact registering any new bindings, and the list of bindings is therefore left unchanged. As noted above, in a successful response to this REGISTER message, the complete list of existing bindings is returned, and thus a REGISTER without Contact headers serves as a fetch operation.

10.2.4 Refreshing Registrations

When a 2xx response has been received by the client for a REGISTER request, the client MUST determine when each of the bindings enumerated in the response needs to be refreshed. This may include bindings that were registered in previous REGISTER transactions.

Since the list of bindings returned in the response to a REGISTER may contain bindings that were not included in this REGISTER transaction, the client must correlate Contact header fields in the response with the Contact header fields it sent in the request in order to establish proper expiration timers. This correlation should be performed in accordance with the URI comparison rules given in Section 21.1.4.

The registering UA MUST re-register each contact address at least as often as the mandated expiration interval. A REGISTER that refreshes a binding SHOULD have the same Call-ID as the request which created the binding. The CSeq header SHOULD have a numeric sequence number that is one higher than the value sent in the last request with the same Call-ID.

Note that a UA MUST must update its expiration timers for refreshing each binding every time it receives a response to a registration request.

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

10.2.5 Discovering a Registrar

Depending on the policy of their administrative domain, SIP UAs can be configured with the address of a local registrar. Some UAs may be equipped with protocol tools (outside the scope of SIP) that allow them to discover their local registrar dynamically.

Note that as an alternate means of discovering a registrar if no local registrar is configured in the user agent, clients MAY register via multicast. 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 (see [19]), 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 (see [20]); 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.

If a SIP UA knows of an appropriate registrar it SHOULD attempt to register with this server periodically - management of registration intervals is detailed below.

10.3 Processing of REGISTER at the Registrar

A registrar is a UAS that responds to a REGISTER request, and stores the information gathered from that request in a location service that is in turn accessible to proxy servers within its administrative domain. A registrar handles requests as a UAS (in conformity with Section 8.2 and Section 17.2) but it accepts only the

REGISTER method and generates only the responses detailed in this section. Note that the REGISTER method also does not support the Record-Route or Route header, and that proxy servers MUST NOT add Record-Route headers to REGISTER requests.

A registrar must know (through provisioning or some other mechanism) the set if administrative domain(s) for which its associated location service(s) are responsible. REGISTER requests MUST be processed by a registrar in the order that they are received.

Upon the arrival of a REGISTER message, the registrar MUST inspect the Request-URI to determine whether it has access to a location service responsible for the domain to which this request is addressed. If this message is for some other administrative domain, then if the registrar can act as a proxy server, it SHOULD forward the request to the addressed domain (following the general behavior for proxying messages described in Section 16).

When a registrar receives a REGISTER message, it is RECOMMENDED that the registrar authenticate the user agent client. Mechanisms for the authentication of SIP user agents are described in Section 20.2; registration behavior in no way overrides the generic authentication framework for SIP. If no authentication mechanism is available, the registrar MAY take the From address as the asserted identity of the originator of the request.

Once the identity of the registering user has been ascertained, it is RECOMMENDED that the registrar determine if the authenticated user agent is authorized to request and/or modify registrations for this address of record. For example, a registrar might consult a authorization database (directly or through an appropriate protocol) that maps credentials or other tokens of identity resulting from authentication to one or more addresses of record for which this identity is responsible.

> Note that in architectures that support third-party registration, one entity may be responsible for updating the registrations associated with multiple addresses of record.

When the registrar has determined that the client is permitted to make the request, the registrar MUST extract the address of record from the To header field of the REGISTER. Note that the registrar MUST extract the entire To header field URI in order to use it as an index in the location service.

Next, the registrar MUST query its location service (the repository of previously registered bindings) for the set of bindings associated with this address of record. If the address of record is not valid for this administrative domain (for example, because the username is not assigned), then the registration attempt fails (see below). A full URI comparison (as described in Section 21.1.4) MUST be performed to determine whether a given binding matches this address of record.

The registrar now MUST extract all the Contact header fields from the REGISTER message (note that there may be no Contact header field).

Each contact address in a REGISTER MUST now be compared to all existing registrations at this location service according to the rules in Section 21.1.4. Note that URIs other than SIP URIs in contact addresses MUST be compared according to the standard URI equivalency rules for the URI schema in question.

If a match is found among pre-existing registrations, the registrar MUST copy all parameters associated with the current Contact header field from the REGISTER message into the pre-existing binding in its location service (overwriting with changed values any existing parameters as necessary, with the exception of “expires”). Expiration intervals for this contact address MUST also be reset, based on any suggested expiration in the REGISTER (remember that this can be “0”).

If no match is found among the set of pre-existing registrations, the registrar MUST create a new binding in its location service between the address of record and the current Contact header field. All Contact header field parameters are copied verbatim into this new binding (again with the exception of “expires”). An expiration interval MUST be selected by the registrar, taking into account any suggested expiration for this contact address in the REGISTER.

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 likelihood of registrations going stale.

The expiration interval mandated by the registrar may be either longer or shorter than the interval suggested by the sender of the REGISTER, though the registrar SHOULD abide by the registering client's suggestion.

A server MAY decide to lengthen the expiration interval if the refresh rate of a particular client exceeds a threshold, for example.

After the expiration interval selected by the registrar for a binding has passed, if the binding has not been refreshed (increasing the expiration interval), the registrar SHOULD silently discard the binding.

Once all bindings in the location service have been updated to reflect any changes present to contact addresses in the REGISTER message, the registrar MUST remove any bindings that expire immediately.

The REGISTER might have set the expiration interval for some bindings to “0” to remove them before their expiration interval passes.

Finally, the registrar must generate a response. If the address of record given in the To header field of the REGISTER method is valid for its administrative domain, then a 200 response MUST be sent, which MUST contain a complete list (within Contact header fields) of the currently valid bindings in the location service associated with the address of record contained in the To field of the REGISTER request. This list MAY be empty (in which case the 200 would not contain any Contact headers).

In a successful response to a REGISTER, wherein the bindings for this address of record are enumerated as described above, the registrar MUST supply an expiration interval for each contact address in either an “expires” parameter of a Contact header or an Expires header. This interval specifies the expiration interval that has been mandated by the registrar (taking into account the registering UA's suggestion).

If the registration failed because the address of record contained in the To field of the REGISTER is not valid for this domain, then a 404 MUST be sent.

11 Querying for Capabilities

The SIP method OPTIONS allows a client to query another client or server as to its capabilities. This allows a client to discover information about the methods, content types, extensions, codecs etc. supported without actually ”ringing” the other party. For example, before a client inserts a Require header field into an INVITE listing an option that it is not certain the destination UAS supports, the client can query the destination UAS with an OPTIONS to see if this option is returned in a Supported header field.

The target of the OPTIONS request is identified by the Request-URI, which could identify another User Agent or a SIP Server. Alternatively, a server receiving an OPTIONS request with a Max-Forwards header value of 0 MAY respond to the request regardless of the Request-URI.

This behavior is common with HTTP/1.1.

An OPTIONS request sent as part of an established dialog does not have any impact on the dialog.

11.1 Construction of OPTIONS Request

An OPTIONS request is constructed using the standard rules for a SIP request as discussed Section 8.1.1.

A Contact header field MAY be present in an OPTIONS.

OPEN ISSUE #197: What is the semantic of this Contact

An Accept header field SHOULD be included to indicate the type of message body the UAC wishes to receive in the response.

Example OPTIONS request:

```
OPTIONS sip:carol@chicago.com SIP/2.0
Via: SIP/2.0/UDP 10.1.1.1:5060;branch=23411513a6
Via: SIP/2.0/UDP 10.1.3.3:5060
To: <sip:carol@chicago.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@10.1.3.3
CSeq: 63104 OPTIONS
Contact: <sip:alice@10.1.3.3>
Accept: application/sdp
Contact-Length: 0
```

11.2 Processing of OPTIONS Request

The response to an OPTIONS is constructed using the standard rules for a SIP response as discussed in Section 8.2.7. The response code chosen is the same that would have been chosen had the request been an INVITE. That is, a 200 (OK) would be returned if the UAS is ready to accept a call, a 486 (Busy Here) would be returned if the UAS is busy, etc. This allows an OPTIONS request to be used to determine the basic state of a UAS, which can be an indication of whether the UAC will accept an INVITE request.

Note that this use of OPTIONS has limitations due the differences in proxy handling of OPTIONS and INVITE requests. While a forked INVITE can result in multiple 200 OK responses being returned, a forked OPTIONS will only result in a single 200 OK response, since it is treated by proxies using the non-INVITE handling. See Section 13.2.1 for the normative details.

Allow, Accept, Accept-Encoding, Accept-Language, and Supported header fields SHOULD be present in a 200 OK response to an OPTIONS request.

A Contact header field MAY be present in a 200 OK response.

A Warning header field MAY be present.

A message body MAY be sent, the type of which is determined by the Accept header in the OPTIONS request.

Example OPTIONS response (corresponding to the request in Section 11.1):

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 10.1.1.1:5060;branch=23411513a6
Via: SIP/2.0/UDP 10.1.3.3:5060
To: <sip:carol@chicago.com>;tag=93810874
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@10.1.3.3
CSeq: 63104 OPTIONS
Contact: <sip:carol@10.3.6.6>
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE
Accept: application/sdp
```

```
Accept-Encoding: gzip
Accept-Language: en
Supported: foo
Content-Type: application/sdp
Contact-Length: 274

v=0
o=carol 28908764872 28908764872 IN IP4 10.3.6.6
s=-
t=0 0
c=IN IP4 10.3.6.6
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
```

12 Dialogs

A key concept for a user agent is that of a dialog. A dialog represents a peer- to-peer SIP relationship between a two user agents that persists for some time. The dialog facilitates sequencing of messages between the user agents, and proper routing of requests between both them. The dialog represents a context in which to interpret SIP messages. The previous section discussed method independent UA processing for requests and responses outside of a dialog. This section discusses how those requests and responses are used to construct a dialog, and then how subsequent requests and responses are sent within a dialog.

A dialog is identified at each UA with a dialog ID, which consists of a Call-ID value, a local URI and local tag (together called the local address), and a remote URI and remote tag (together called the remote address). The dialog ID at each UA involved in the dialog is not the same. Specifically, the local URI and local tag at one UA are identical to the remote URI and remote tag at the peer UA. The tags are opaque tokens that facilitate the generation of unique dialog IDs.

A dialog ID is also associated with all responses, and with any request that contains a tag in the To field. The rules for computing the dialog ID of a message depend on whether the entity is a UAC or UAS. For a UAC, the Call-ID value of the dialog ID is set to the Call-ID of the message, the remote address is set to the To field of the message, and the local address is set to the From field of the message (these rules apply to both requests and responses). As one would expect, for a UAS, the Call-ID value of the dialog ID is set to the Call-ID of the message, the remote address is set to the From field of the message, and the local address is set to the To field of the message.

A dialog contains certain pieces of state needed for further message transmissions within the dialog. This state consists of the Call-ID, a local sequence number (used to order requests from the UA to its peer), a remote sequence number (used to order requests from its peer to the UA), and a route set, which is an ordered list of URIs. The route set is the set of servers that need to be traversed to send a request to the peer. A dialog can also be in the “early” state, which occurs when it is created with a provisional response, and then transition to the “established” state when the final response comes.

12.1 Creation of a Dialog

Dialogs are created through the generation of non-failure responses to requests with specific methods. Within this specification, only the 2xx and 1xx responses to INVITE establish a dialog. A dialog established by a non-final response to a request is called an early dialog. Extensions MAY define other means for creating dialogs. Section 13 gives more details that are specific to the INVITE method. Here, we describe the process for creation of dialog state that is not dependent on the method.

12.1.0.1 UAS When a UAS responds to a request with a response that establishes a dialog (such as a 2xx to INVITE), the UAS MUST copy all Record-Route headers from the request into the response, and MUST maintain the order of those headers. This includes the URIs, URI parameters, and any Record-Route header parameters, whether they are known or unknown to the UAS. The UAS MUST add a Contact header field to the response. The Contact header field contains an address where the UAS would like to be contacted for subsequent requests in the dialog (which includes the ACK for a 2xx response in the case of an INVITE). Generally, the host portion of this URI is the IP address of the host, or its FQDN. The URI provided in the Contact header MUST be a SIP URL.

The UAS then constructs the state of the dialog. This state MUST be maintained for the duration of the dialog. First, the route set MUST be computed by following these steps:

1. The list of URIs in the Record-Route headers in the 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.

   Contact was not mandatory in RFC2543. Thus, if the UAS is talking to an older UAC, the UAC might not have inserted the Contact header.

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

These rules clearly imply that a UA MUST be able to parse and process Record-Route header fields. This is a change from RFC2543, where all record-route and route processing was optional for user agents.

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 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. This is needed in order for the UA to determine whether the bottom most value can be updated from subsequent requests; if it was constructed from a Contact, it can be updated.

The remote sequence number sequence number MUST be set to the value of the sequence number in the Cseq header of the request. The local sequence number MUST be empty. The call identifier component of the dialog ID MUST be set to the value of the Call-ID in the request. The local address component of the dialog ID MUST be set to the To field in the response to the request (which therefore includes the tag), and the remote address component of the dialog ID MUST be set to the From field in the request. A UAS MUST be prepared to receive a request without a tag in the From field, in which case the tag is considered to effectively have a value of null.

This is to maintain backwards compatibility with RFC2543, which did not mandate From tags.

12.1.0.2 UAC When a UAC receives a response that establishes a dialog, it constructs the state of the dialog. This state MUST be maintained for the duration of the dialog. First, the route set MUST be computed by following these steps:

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

2. The URI in the Contact header from the 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*.

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 response. The UAC 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. This is needed in order for the UA to determine whether the bottom most value can be updated from subsequent requests; if it was constructed from a Contact, it can be updated.

The local sequence number sequence number MUST be set to the value of the sequence number in the Cseq header of the request. The remote sequence number MUST be empty (it is established when the UA sends a request within the dialog). The call identifier component of the dialog ID MUST be set to the value of the Call-ID in the request. The local address component of the dialog ID MUST be set to the From field in the request, and the remote address component of the dialog ID MUST be set to the To field of the response. A UAC MUST be prepared to receive a response without a tag in the To field, in which case the tag is considered to effectively have a value of null.

> This is to maintain backwards compatibility with RFC2543, which did not mandate To tags.

12.2 Requests within a Dialog

Once a dialog has been established between two UAs either of them MAY initiate new transactions as needed within the dialog. However, a dialog imposes some restrictions on the use of simultaneous transactions.

A TU MUST NOT initiate a new regular transaction within a dialog while a regular transaction is in progress (in either direction) within that dialog.

> OPEN ISSUE #113: Should we relax the constraint on non-overlapping regular transactions?

A refresh request sent within a dialog is defined as a request that can modify the *route set* of the dialog. For dialogs that have been established with an INVITE, the only refresh request defined is re-INVITE (see Section 14). Other extensions may define different refresh requests for dialogs established in other ways.

> Note that an ACK is *NOT* a refresh request.

12.2.1 UAC Behavior

12.2.1.1 Generating the Request A request within a dialog is constructed by using many of the components of the state stored as part of the dialog.

The To header field of the request MUST be set to the remote address, and the From header field MUST be set to the local address (both including tags, assuming the tags are not null).

The Call-ID of the request MUST be set to the Call-ID of the dialog. Requests within a dialog MUST contain strictly monotonically increasing and contiguous CSeq sequence numbers (increasing-by-one) in each direction. Therefore, if the local sequence number is not empty, the value of the local sequence number MUST be incremented by one, and this value MUST placed into the Cseq header. If the local sequence number is empty, an initial value MUST be chosen using the guidelines of Section 8.1.1.4. The method field in the Cseq header MUST match the method of the request.

> With a length of 32 bits, a client 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 clients to use a time-based initial sequence number. A client could, for example, choose the 31 most significant bits of a 32-bit second clock as an initial sequence number.

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

The UAC takes the list of URI in the *route set*. The top URI MUST be inserted into the request URI of the request, including all URI 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, MUST be placed into a Route header field into the request, in order.

A TU SHOULD follow the rules just mentioned to build the Request-URI of the request, regardless of whether the UA uses an outbound proxy server or not. However, in some instances, a UA may not be willing or capable of sending the request to the top element in the *route set*. One example is a UA that is not capable of DNS, and therefore may not be able to follow those procedures. In these cases, the UA MAY send the request to a local outbound server. In this case, it MUST NOT remove the top Route header.

> In dialogs created by an INVITE, if the UA is the caller, it sets the Request-URI to the same value it used for the initial request, and sends it to its local outbound server.
>
> Bug#161: Which Request-URI does the callee use?

A UAC SHOULD include a Contact header in any refresh requests within a dialog, and unless there is a need to change it, the URI SHOULD be the same as used in previous requests within the dialog. As discussed in Section 12.2.2, a Contact header in a refresh request updates the route set. This allows a UA to provide a new contact address, should its address change during the duration of the dialog.

However, requests that are not refresh requests do not affect the *route set* for the dialog.

Once the request has been constructed, the address of the server is computed and the request is sent, using the same procedures for requests outside of a dialog (Section 8.1.1).

12.2.1.2 Processing the Responses The UAC will receives responses to the request from the transaction layer.

The behavior of a UAC that receives a 3xx response for a request sent within a dialog is the same as if the request would have been sent outside a dialog. This behavior is described in Section 13.2.2.

> Note however that when the UAC tries alternative locations it still uses the *route set* for the dialog to build the Route header of the request.

If a UAC has a *route set* for a dialog, and receives a 2xx response to a refresh it sent, the Contact header field of the response is examined. If not present, the *route set* remains unchanged. If the response had a Contact header field, and the boolean variable CONTACT_SET is false, the URL in the Contact header field in the response is added to the bottom of the *route set*, and CONTACT_SET is set to true. If the refresh request response had a Contact header field, and CONTACT_SET is true, the URL in the Contact header field of the response to the refresh request replaces the bottom value in the *route set*. If a refresh request is responded with a non-2xx final response the *route set* remains unchanged as if no refresh request had been issued.

If the response for the a request within a dialog is a 481 (Call/Transaction Does Not Exist) or a 408 (Request Timeout) the UAC SHOULD terminate the dialog.

> For INVITE initiated dialogs terminating the dialog consists of sending a BYE.

12.2.2 UAS behavior

The UAS will receive the request from the transaction layer. If the request has a tag in the To header field, the UAS core computes the dialog identifier corresponding to the request and compares it with existing dialogs. If there is a match, this is a mid-dialog request. In that case, the same processing rules for requests outside of a dialog, discussed in Section 8.2, are applied by the UAS once the request is received from the transaction layer.

Requests that do not change in any way the state of a dialog may be received within a dialog (e.g., an OPTIONS request). They are processed as if they had been received outside the dialog.

Requests within a dialog MAY contain Record-Route and Contact header fields. However, requests that are not refresh requests do not update the *route set* for the dialog. This specification only defines one refresh request: re-INVITE (see Section 14).

Special rules apply when updated Record-Route or Contact header fields are received inside a refresh request. If a UAS has a *route set* for a dialog, and receives a refresh for that dialog containing Record-Route header fields, it MUST copy those header fields into any 2xx response to that request. If the boolean variable CONTACT_SET is true, the Contact header field 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 field 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 field, the route-set at the UAS remains unchanged.

If the remote sequence number is empty, it MUST be set to the value of the sequence number in the Cseq header in the request. If the remote sequence number was not empty, but the sequence number of the request is lower than the remote sequence number, the request is out of order and MUST be rejected with a 500 response. If the remote sequence number was not empty, and the sequence number of the request is greater than the remote sequence number, the request is in order. It is possible for the CSeq header to be higher than the remote sequence number 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. The UAS MUST then set the remote sequence number to the value of the sequence number in the Cseq header in the request.

12.3 Termination of a Dialog

Dialogs can end in several different ways, depending on the method. When a dialog is established with INVITE, it is terminated with a BYE. No other means to terminate a dialog are described in this specification, but extensions can define other ways.

13 Initiating a Session

13.1 Overview

When a user agent client desires to initiate a session (for example, audio, video, or a game), it formulates an INVITE request. The INVITE request asks a server to establish a session. This request is forwarded by proxies, eventually arriving at one or more UAS which can potentially accept the invitation. These UAS's will frequently need to query the user about whether to accept the invitation. After some time, those UAS can accept the invitation (meaning the session is to be established) by sending a 2xx response. If the invitation is not accepted, a 3xx,4xx,5xx or 6xx response is sent, depending on the reason for the rejection. Before sending a final response, the UAS can also send a provisional response (1xx) to advise the UAC of progress in contacting the called user.

After possibly receiving one or more provisional responses, the UA will get one or more 2xx responses or one non-2xx final response. Because of the protracted amount of time it can take to receive final responses to INVITE, the reliability mechanisms for INVITE transactions differ from those of other requests (like OPTIONS). Once it receives a final response, the UAC needs send an ACK for every final response it receives. The procedure for sending this ACK depends on the type of response. For final responses between 300 and 699, the ACK processing is done in the transaction layer, and follows one set of rules (See Section 17). For 2xx responses, the ACK is generated by the UAC core.

A 2xx response to an INVITE establishes a session, and it also creates a dialog between the UA that issued the INVITE and the UA that generated the 2xx response. Therefore, when multiple 2xx responses are received from different remote UAs (because the INVITE forked), each 2xx establishes a different dialog. All these dialogs are part of the same call.

This section provides details on the establishment of a session using INVITE.

13.2 Caller Processing

13.2.1 Creating the Initial INVITE

Since the initial INVITE represents a request outside of a dialog, its construction follows the procedures of Section 8.1.1. Additional processing is required for the specific case of INVITE.

An Allow header field (Section 22.5) SHOULD be present in the INVITE. It indicates what methods can be invoked within a dialog, on the UA sending the INVITE, for the duration of the dialog. For example, a UA capable of receiving INFO requests within a dialog [21] SHOULD include an Allow header listing the INFO method.

A Supported header field (Section 22.35) SHOULD be present in the INVITE. It enumerates all the extensions understood by the UAC.

An Accept (Section 22.1) header field MAY be present in the INVITE. It indicates which content-types are acceptable to the UA, in both the response received by it, and in any subsequent requests sent to it within dialogs established by the INVITE. The Accept header is especially useful for indicating support of various session description formats.

The UA MAY add an Expires header field (Section 22.19) to limit the validity of the invitation. If the time indicated in the Expires header field is reached and no final answer for the INVITE has been received the UAC core SHOULD generate a CANCEL request for the original INVITE.

A UAC MAY also find useful to add, among others, Subject (Section 22.34), Organization (Section 22.24) and User-Agent (Section 22.39) header fields. They all contain useful information related to the INVITE.

The UAC MAY choose to add a message body to the INVITE. Section 8.1.1.9 deals with how to construct the header fields- Content-Type among others- needed to describe the message body.

There are special rules for message bodies that contain a session description - their corresponding Content-Disposition is “session”. SIP uses an offer/answer model where one UA sends a session description, called the offer, which contains a proposed description of the session. The offer indicates the desired communications means (audio, video, games), parameters of those means (such as codec types) and addresses for receiving media from the offerer. The other UA responds with another session description, called the answer, which indicates which communications means are accepted, the parameters which apply to those means, and addresses for receiving media from the answerer. The offer/answer model can be mapped into the INVITE transaction in two ways. The first, which is the most intuitive, is that the INVITE contains the offer, the 2xx response contains the answer, and no session description is provided in the ACK. In this model, the UAC is the offerer, and the UAS is the answerer. A second model is that the INVITE contains no session description, the 2xx response contains the offer, and the ACK contains the answer. In this model, the UAS is the offerer, and the UAC is the answerer. The second model is useful for gateways from H.323v1 to SIP, where the H.323 media characteristics are not known until the call is established. This is also useful for sessions that use third-party call control. As a result of these models, if the INVITE contains a session description, the ACK MUST NOT contain one. Conversely, if the caller chooses to omit the session description in the INVITE, the ACK MUST contain one (if a 2xx response is received). 2xx responses to an INVITE MUST always contain a session description. All user agents that support INVITE MUST support both models.

The Session Description Protocol (SDP) [6] MUST be supported by all user agents as a means to describe sessions, and its usage for construction offers and answers MUST follow the procedures defined in [22].

Note that the restrictions of the offer-answer model (session description only in the INVITE OR in the ACK, but not in both) just described only apply to bodies whose Content-Disposition header field is "session". Therefore, it is possible that both the INVITE and the ACK contain a body message (e.g., the INVITE carries a photo (Content-Disposition: render) and the ACK a session description (Content-Disposition: session)).

If the Content-Disposition header field is missing, bodies of Content-Type `application/sdp` imply the disposition "session", while other content types imply "render".

Once the INVITE has been created, the UAC follows the procedures defined for sending requests outside of a dialog (Section 8). This results in the construction of a client transaction that will ultimately send the request and deliver responses to the UAC.

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.

13.2.2 Processing INVITE Responses

Once the INVITE has been passed to the INVITE client trasaction, the UAC waits for responses for the INVITE. Responses are matched to their corresponding INVITE because they have the same Call-ID, the same From header field, the same To header field, excluding the tag, and the same CSeq. Rules for comparisons of these headers are described in Section 22.

13.2.2.1 1xx responses Zero, one or multiple provisional responses may arrive before one or more final responses are received. Provisional responses for an INVITE request can create "early dialogs". If a provisional response has a tag in the To field, and if the dialog ID of the response does not match an existing dialog, one is constructed using the procedures defined in Section 12.1.0.2.

The early dialog will only be needed if the UAC needs to send a request to its peer within the dialog before the initial INVITE transaction completes. Header fields present in a provisional response are applicable for the duration of the early dialog (e.g., an Allow header field in a provisional response contains the methods that can be used in the early dialog).

13.2.2.2 3xx responses A 3xx response may contain a Contact header field providing new addresses where the callee might be reachable. Depending on the status code of the 3xx response (see Section 23.3) the UAC MAY choose to try those new addresses.

13.2.2.3 4xx, 5xx and 6xx responses A single non-2xx final response may be received for the INVITE. 4xx, 5xx and 6xx responses may contain a Contact header field indicating the location where additional information about the error can be found.

All early dialogs are considered terminated upon reception of the non-2xx final response.

After having received the non-2xx final response the UAC core considers the INVITE transaction completed. The INVITE client transaction handles generation of ACKs for the response (see Section 17).

13.2.2.4 2xx responses Multiple 2xx 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 dialog, with a distinct dialog identifier.

If the dialog identifier in the 2xx response matches the dialog identifier of an existing dialog, the dialog MUST be transitioned to the "established", and the route set for the dialog MUST be recomputed based on the 2xx response using the procedures of Section 12.1.0.2. Otherwise, a new established dialog is constructed in the same fashion.

> The route set only is recomputed for backwards compatibility. RFC 2543 did not mandate mirroring of Record-Route headers in a 1xx, only 2xx. However, we cannot update the entire state of the dialog, since mid-dialog requests may have been sent within the early call leg, modifying the sequence numbers, for example.

The UAC core MUST generate an ACK request for each 2xx received from the transaction layer. The header fields of the ACK are constructed in the same way as for any request sent within a dialog (see Section 12) with the exception of the CSeq. The sequence number of the CSeq header field MUST be the same as the INVITE being acknowledged, but the CSeq method MUST be ACK. If the INVITE did not contain an offer, the 2xx will contain one, and therefore the ACK MUST carry an answer in its body.

Once the ACK has been constructed, the procedures of Section 24 are used to send it. However, the request is passed to the transport layer directly for transmission, rather than a client transaction. This is because the UAC core handles retransmissions of the ACK, not the transaction layer. The ACK MUST be passed to the client transport every time a retransmission of the 2xx final response that triggered the ACK arrives.

The UAC core considers the INVITE transaction completed 62*T1 seconds after the reception of the first 2xx response. At this point all the early dialogs that have not transitioned to established dialogs are terminated. Once the INVITE transaction is considered completed by the UAC core, no more new 2xx responses are expected to arrive.

If, after acknowledging any 2xx response to an INVITE, the caller does not want to continue with that dialog, then the caller MUST terminate the dialog by sending a BYE request as described in Section 15.

13.3 Callee Processing

13.3.1 Processing of the INVITE

The UAS core will receive INVITE requests from the transaction layer. It first performs the request processing procedures of Section 8.2, which are applied for both requests inside and outside of a dialog.

Assuming these processing states complete without generating a response, the UAS core performs the additional processing steps:

1. If the request is an INVITE that contains an Expires header field the UAS core inspects this header field. If the INVITE has already expired a 487 response is generated.

2. If the request has no tag in the To the UAS core checks ongoing transactions. If the To, From, Call-ID, CSeq exactly match (including tags) those of any request received previously, but the branch-ID in the topmost Via is different from those received previously, the UAS core SHOULD generate a 482 (Loop detected) response and pass it to the server transaction.

   The same request that was generated by the UAC has arrived to the UAS more than once following different paths. The UAS processes the request that was received first and responds with 482 (Loop detected) to the rest of them.

   If no match is found, the request does not belong to any existing dialog. If the request is an INVITE the UAS core follows the procedures described in this section.

3. If the request is a mid-dialog request, the method-independent processing described in Section 12.2.2 is first applied. It might also modify the session; Section 14 provides details.

4. If the request has a tag in the To header field but the dialog identifier does not match any of the existing dialogs, the UAS may have crashed and restarted, or may have received a request for a different (possibly failed) UAS. The UAS MAY either accept or reject the request. Accepting the request provides robustness, so that dialogs can persist even through crashes. UAs wishing to support this capability must choose monotonically increasing CSeq sequence numbers even across reboots. This is because subsequent requests from the crashed-and-rebooted UA towards the other UA need to have a CSeq sequence 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 request, because it does not wish to recreate the dialog, it MUST respond to the request with a 481 (Call/Transaction Does Not exist) status code and pass that to the server transaction.

Processing from here forward assumes that the INVITE is outside of a dialog, and is thus for the purposes of establishing a new session.

The INVITE may contain a session description, in which case the UAS is being presented with an offer for that session. It is possible that the user is already a participant in that session, even though the INVITE is outside of a dialog. This can happen when a user is invited to the same multicast conference by multiple other participants. If desired, the UAS 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. If the user is already a member of the session and the session parameters contained in the session description have not changed, the UAS MAY silently accept the INVITE

The INVITE may not contain a session description at all, in which case the UAS is being asked to participate in a session, but the UAC has asked that the UAS provide the offer of the session.

The callee can indicate progress, accept, redirect, or reject the invitation. In all of these cases, it formulates a response using the procedures described in Section 8.2.7.

13.3.1.1 Progess The UAS may not be able to answer the invitation immediately, and might choose to indicate some kind of progress to the caller (for example, an indication that a phone is ringing). This is accomplished with a provisional response between 101 and 199. These provisional responses establish early dialogs and therefore follow the procedures of Section 12.1.0.1 in addition to those of Section 8.2.7. A UAS MAY send as many provisional responses as it likes. Each of these MUST indicate the same dialog ID. SIP, however, does not guarantee that these provisional responses are reliably delivered to the UAC.

13.3.1.2 The INVITE is redirected If the UAS decides to redirect the call, a 3xx response is sent. A 300 (Multiple Choices), 301 (Moved Permanently) or 302 (Moved Temporarily) response SHOULD contain a Contact header field containing URIs of new addresses to be tried. The response is passed to the INVITE server transaction, which will deal with its retransmissions.

13.3.1.3 The INVITE is rejected A common scenario occurs when the callee is currently not willing or able to take additional calls at this end system. A 486 (Busy Here) SHOULD be returned in such scenario. If the UAS knows that no other end system will be able to accept this call a 600 (Busy Everywhere) response SHOULD be sent instead. However, it is unlikely that a UAS will be able to know this in general, and thus this response will not usually be used. The response is passed to the INVITE server transaction, which will deal with its retransmissions.

13.3.1.4 The INVITE is accepted The UAS core generates a 2xx response. This response establishes a dialog, and therefore follows the procedures of Section 12.1.0.1 in addition to those of Section 8.2.7.

A 2xx response to an INVITE SHOULD contain the Allow header field and the Supported header field, 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.

If the INVITE request contained an offer, the 2xx MUST contain an answer. If the INVITE did not contain an offer, the 2xx MUST contain an offer.

Once the response has been constructed it is passed to the INVITE server transaction. Note, however, that the INVITE server transaction does not retransmit 2xx responses to an INVITE. Therefore, it is necessary to pass periodically the response to the server transaction until the ACK arrives. The 2xx response is resubmitted to the server transaction with an interval that starts at T1 seconds and doubles for each retransmission until it reaches T2 seconds (T1 and T2 are defined in Section 17). Response retransmissions cease when an ACK request is received with the same dialog ID as the response. This is independent of whatever transport protocols are used to send the response.

> Since 2xx is retransmitted end-to-end, there may be hops between UAS and UAC which are UDP. To ensure reliable delivery across these hops, the response is retransmitted periodically even if the transport at the UAS is reliable.

If the server retransmits the 2xx response for 64*T1 seconds without receiving an ACK, it considers the dialog completed, the session terminated, and therefore it SHOULD send a BYE.

14 Modifying an Existing Session

A successful INVITE request (see Section 13) establishes both a dialog between two user agents and a session (using the offer/answer model). Section 12 explains how to modify an existing dialog using a refresh request (e.g., changing the *route set* of the dialog). This section describes how to modify the actual session. This modification can involve changing addresses or ports, adding a media stream, deleting a media stream, and so on. This is accomplished by sending a new INVITE request within the same dialog that established the session. An INVITE request sent within an existing dialog is known as a re-INVITE.

> Note that a single re-INVITE can modify at the same time the dialog and the parameters of the session.

Either the caller or callee can modify an existing session.

14.1 UAC Behavior

The same offer-answer model that applies to session descriptions in INVITEs (Section 13.2.1) applies to re-INVITEs. As a result, a UAC that wants to add a media stream, for example, will create a new offer that contains this media stream, and send that in an INVITE request to its peer. It is important to note that the full description of the session, not just the change, is sent. This maintains the idempotency of SIP, supports stateless session processing in various elements, and supports failover and recovery capabilities. Of course, a UAC MAY send a re-INVITE with no session description, in which case the response to the re-INVITE will contain the offer.

If the session description format has the capability for version numbers, the offerer SHOULD indicate that the version of the session description has changed.

The To, From, Call-ID, CSeq, and Request-URI of a re-INVITE are set following the same rules as for regular requests within an existing dialog, described in Section 12.

Note that, as opposed to initial INVITEs (see Section 13), re-INVITEs contain tags in the To header field and are sent using the *route set* for the dialog. Therefore, a single final (2xx or non-2xx) response is received for re-INVITEs.

Note that a UAC MUST NOT initiate a new INVITE transaction within a dialog while another transaction (INVITE or non-INVITE) is in progress. However, a UA MAY initiate a regular transaction within an early dialog - while an INVITE transaction is in progress.

If a re-INVITE is responded with a non-2xx final response the session parameters MUST remain unchanged, as if no re-INVITE had been issued.

The rules for transmitting a re-INVITE and for generating an ACK for a 2xx response to re-INVITE are the same as for an INVITE (Section 13.2.1).

14.2 UAS Behavior

Section 13.3.1 describes the steps to follow in order to distinguish incoming re-INVITEs from incoming initial INVITEs. This Section describes the procedures to follow upon reception of a re-INVITE for an existing dialog.

A UAS that receives a second INVITE before it sent the final response to a first INVITE with a lower CSeq sequence number on the same dialog MUST return a 500 response to the second INVITE and MUST include a Retry-After header field with a randomly chosen value of between 0 and 10 seconds. Similarly, a UAS the receives an INVITE on a dialog while an INVITE it had sent on that dialog is in progress MUST return a 500 response to the received INVITE and MUST include a Retry-After header field with a randomly chosen value of between 0 and 10 seconds.

If a user agent receives a re-INVITE for an existing dialog 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. 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 a UAS generates a 2xx response and never receives an ACK, it SHOULD generate a re-INVITE itself with an offer equal to the last session description sent to the peer. The purpose of this is to ensure that both caller and callee have a consistent view of the session parameters.

A UAS providing an offer in a 2xx (because the INVITE did not contain an offer) MUST offer the same session description as last provided to the peer, with the exception of being able to change the IP address/port if so desired.

> Under error conditions (e.g., the UAS has crashed and restarted) the session description in the 2xx response for an empty re-INVITE may be different than the one in use at that moment. If the new session description is not acceptable for the UAC it SHOULD then send a BYE (after ACKing the 2xx response).

15 Terminating a Session

Terminating a session is done either with the BYE request, or the CANCEL request, depending on the state of the dialog. Either caller or callee can terminate, and may do so for any reason. Sections 13 and 12 document some cases where call termination is normative behavior. As a general rule, if a UA decides that the session is to be terminated, it MUST follow the procedures here to initiate signaling action to convey that.

> Note that both the session and the dialog between both user agents will be terminated.

When a UAC sends an INVITE request to create a session, if a 1xx response with a tag in the To field is received, an early dialog is created. When a 2xx response is received, the dialog becomes established. For either state of the dialog, if the UAC desires to terminate the session, the UAC SHOULD follow the procedures described in Section 15.1.1 to terminate the session. If the callee for a new session wishes to terminate the dialog, it uses the procedures of Section 15.1.1, but MUST NOT do so until it has receive an ACK or until the server transaction times out.

> This does not mean a user can't hang up right away; it just means that the software in their phone needs to maintain state for a short while in order to properly clean up.

> OPEN ISSUE #202: Is this the right solution.

If the UAC desires to end the session before any type of dialog has been created, it SHOULD send a CANCEL for the INVITE request that requested establishment of the session that is to be terminated. The UAC constructs and sends the CANCEL following the procedures described in Section 9. 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 2xx response to the INVITE "pass on the wire". In this case, the UAC will receive a 2xx to the INVITE. It SHOULD then terminate the call by following the procedures described in Section 15.1.1.

15.1 Terminating a Dialog with a BYE

15.1.1 UAC Behavior

A user agent client uses BYE request, sent within a dialog, to indicate to the server that it wishes to terminate the session. This will also terminate the dialog. A BYE request MAY be issued by either caller or callee. A BYE request SHOULD NOT be sent before the creation of a dialog (either early or established). In that case the UAC SHOULD follow the procedures described in Section 9 instead.

> Proxies ensure that a CANCEL request is routed in the same way as the INVITE was. However, a proxy performing load balancing may route a BYE without a Route header field in a different way than the INVITE, since both requests have different CSeq sequence numbers.

The To, From, Call-ID, CSeq, and Request-URI of a BYE are set following the same rules as for regular requests sent within a dialog, described in Section 12.

Once the BYE is constructed, it creates a new non-INVITE client transaction, and passes it the BYE request. The user agent SHOULD stop sending media as soon as the BYE request is passed to the client transaction.

15.1.2 UAS Behavior

A UAS core receiving a BYE request checks to see if it matches an existing dialog. If the BYE does not match an existing dialog, the UAS core SHOULD generate a 481 response and pass that to the server transaction.

A UAS core receiving a BYE request for an existing dialog MUST follow the procedures of Section 12.2.2 to process the request. Once done, the UAS MUST cease transmitting media streams for the session being terminated. The UAS core MUST generate a 2xx response to the BYE, and MUST pass that to the server transaction for transmission.

The UAS MUST still respond to any pending requests received for that dialog, (which can only be an INVITE). It is RECOMMENDED that a 487 (Request Terminated) response is generated to those pending requests.

16 Proxy Behavior

16.1 Overview

SIP proxies are elements that route SIP requests to user agent servers and SIP responses to user agent clients. A request may traverse several proxies on its way to a UAS. Each will make routing decisions, modifying the request before forwarding it to the next element. Responses will route through the same set of proxies traversed by the request in the reverse order.

It is important to note that being a proxy is a logical role for a SIP element. When a request arrives, an element that can play the role of a proxy must first decide if it needs to respond to the request on its own. For instance, the request could be malformed or the element may need credentials from the client before acting as a proxy. The element MAY respond with any appropriate error code. When responding directly to a request, the element is playing the role of a UAS and MUST behave as described in Section 8.2.

A proxy can operate in either a stateful or stateless mode for each new request.

When stateless, a proxy acts as a simple forwarding element. It forwards each request downstream to a single element determined by making a routing decision based on the request. It simply forwards every response it receives upstream. A stateless proxy discards information about a message once it has been forwarded.

On the other hand, a stateful proxy remembers information (specifically, transaction state) about each incoming request and any requests it sends as a result of processing the incoming request. It uses this information to affect the processing of future messages associated with that request. A stateful proxy MAY chose to “fork” a request, routing it to multiple destinations. Any request that is forwarded to more than one location MUST be handled statefully. Any request processed using TCP (or any other mechanism that is inherently stateful), MUST be handled statefully.

Much of the processing involved when acting statelessly or statefully for a request is identical. The next several subsections are written from the point of view of a stateful proxy. The last section calls out those places where a stateless proxy behaves differently.

16.2 Stateful Proxy

When stateful, a proxy is purely a SIP transaction processing engine. Its behavior is modeled here in terms of the Server and Client Transactions defined in Section 17.A stateful proxy has a server transaction associated with one or more client transactions by a higher layer proxy processing component (see figure 3), known as a proxy core. An incoming request is processed by a server transaction. Requests from the server transaction are passed to a proxy core. The proxy core determines where to route the request, choosing one or more next-hop locations. An outgoing request for each next-hop location is processed by its own associated client transaction. The proxy core collects the responses from the client transactions and uses them to send responses to the server transaction.

A stateful proxy creates a new server transaction for each new request received. Any retransmissions of the request will then be handled by that server transaction per Section 17.

Note that this is a model of proxy behavior, not of software. An implementation is free to take any approach that replicates the external behavior this model defines.

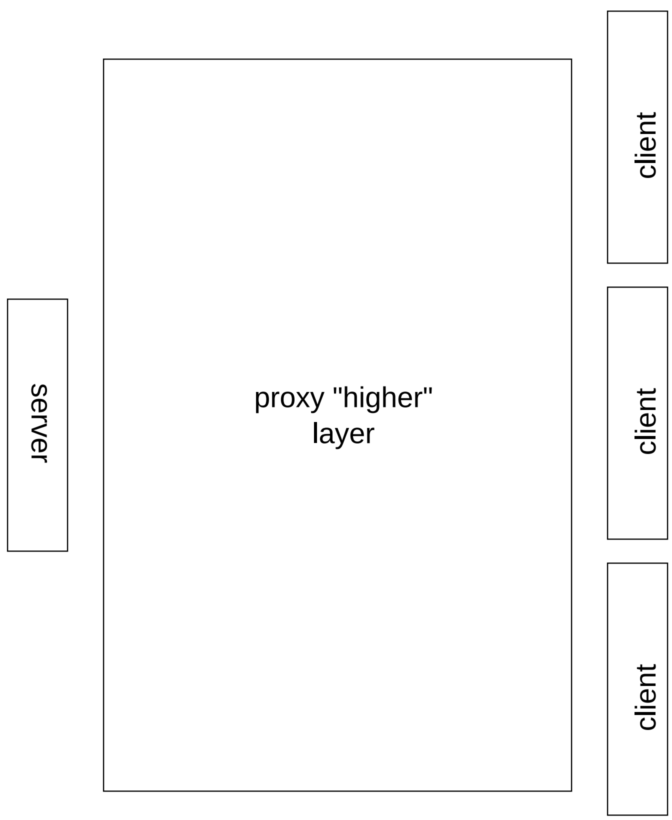

Figure 3: Stateful Proxy Model

For all new requests, including any with unknown methods, an element intending to proxy the request MUST:

1. Validate the request (Section 16.3)

2. Make a routing decision (Section 16.4)

3. Forward the request to each chosen destination (Section 16.5)

4. Process all responses (Section 16.6)

16.3 Request Validation

Before an element can proxy a request, it MUST verify the message's validity. A valid message must pass the following checks:

1. Reasonable Syntax

2. Max-Forwards

3. Loop Detection

4. Proxy-Require

5. Proxy-Authorization

If any of these checks fail, the element MUST behave as a user agent server (see Section 8.2) and respond with an error code.

1. Reasonable Syntax check

   The request MUST be well-formed enough to be handled with a server transaction. Any components involved in the remainder of these Request Validation steps or the Request Processing section MUST be well-formed. Any other components, well-formed or not, SHOULD be ignored. For instance, an element SHOULD NOT reject a request because of a malformed Date header field.

   This protocol is designed to be extended. Future extensions may define new methods and header fields at any time. An element MUST NOT refuse to proxy a request because it contains a method or header field it does not know about.

2. Max-Forwards check

   The Max-Forwards header (Section 22.22) is used to limit the number of elements a SIP request can traverse.

   If the request does not contain a Max-Forwards header field, this check is passed.

   If the request contains a Max-Forwards header field with a field value greater than zero, the check is passed.

   If the request contains a Max-Forwards header field with a field value of zero (0), the element MUST NOT forward the request. If the request was for OPTIONS, the element MAY act as the final recipient and respond per Section 11. Otherwise, the element MUST return a 483 (Too many hops) response.

3. Loop Detection check

   An element MUST check for forwarding loops before forwarding a request. If the request contains a Via header field value with A sent-by value that equals a value placed into previous requests by the proxy, the request has been forwarded by this element before. The request has either looped or is legitimately spiraling through the element. To determine if the request has looped, the element MUST perform the branch parameter calculation described in Section 3 on this message and compare it to the parameter received in that Via field value. If the parameters match, the request has looped. If they differ, the request is spiraling, and processing continues. If a loop is detected, the element MUST return a 482 (Loop Detected) response.

An element MUST NOT forward a request to a multicast group which already appears in any of the Via headers.

4. Proxy-Require check

   Future extensions to this protocol may introduce features that require special handling by proxies. Endpoints will include a Proxy-Require header in requests that use these features, telling the proxy it should not process the request unless the feature is understood.

   If the request contains a Proxy-Require header (Section 22.28) with one or more option-tags this element does not understand, the element MUST return a 420 (Bad Extension) response. The response MUST include an Unsupported (Section 22.38) header field listing those option-tags the element did not understand.

5. Proxy-Authorization check

   If an element requires credentials before forwarding a request, the request MUST be inspected as described in Section 20.2.3. That section also defines what the element must do if the inspection fails.

16.4 Making a Routing Decision

At this point, the proxy must decide where to forward the request. This can be modeled as computing a set of destinations for the request. This set will either be predetermined by the contents of the request or will be obtained from an abstract location service. Each destination is represented as a URI and an optional IP address, port and transport. This combination is referred to as a “next-hop location”.

First, the proxy core checks the received request for Route headers. If any Route header fields are present in the request, the element MUST use the URL (including all of its parameters) from the topmost Route header field as only next hop URI in the destination set, with no IP address, port and transport set for that next hop. The destination set is complete, containing **only** this URL, and the proxy MUST proceed to the Request Processing of Section 16.5.

The Route mechanism is used to control the path a request takes through SIP elements, much like strict IP source routing. The UAC will insert Route header fields (see Section 12), usually based on information provided by proxies through Record-Route header fields (see Section 6).

Assuming there were no Route headers in the received request, the proxy checks the Request-URI of the received request. If it has an maddr parameter, and that parameter does not indicate an interface the proxy is listening on, the Request-URI MUST be placed into the destination set as the only next hop URI, with no IP address, port and transport set for that next hop, and the proxy MUST proceed to Section 16.5. If the maddr parameter was present, but did indicate an interface the proxy is listening on, the proxy MUST strip the maddr and continue processing as if no maddr were present.

OPEN ISSUE #213: Do we strip just the maddr, or the port and transport as well?

OPEN ISSUE #218: Are we really sure this ordering of precedence of Route, maddr, and domain is correct?? It is not yet clear. This needs resolution asap finally, since it affects things like loose source routing, outbound proxy processing at a UA, and so on.

If the domain of the Request-URI indicates a domain this element is not responsible for, it SHOULD set the next hop URI to the Request-URI, and leave the IP address, port and transport of the next hop empty. That next hops MUST be placed into the destination set as the only next hop, and the element MUST proceed to the task of Request Processing (Section 16.5.

There are many circumstances in which a proxy might receive a request for a domain it is not responsible for. A firewall proxy handling outgoing calls (the way HTTP proxies handle outgoing requests) is an example of where this is likely to occur.

If the destination set for the request has not been predetermined as described above, this implies that the element is responsible for the domain in the Request-URI, and the element MAY use whatever mechanism it desires to determine where to send the request. Any of these mechanisms can be modeled as accessing an abstract Location Service. This may consist of obtaining information from a location service created by a SIP Registrar, reading a database, consulting a presence server, utilizing other protocols, or simply performing an algorithmic substitution on the Request-URI. The output of these mechanisms is used to construct the destination set.

Any information in or about the request or the current environment of the element MAY be used in the construction of the destination set. For instance, different sets may be constructed depending contents or presence of header fields and bodies, the time of day of the request's arrival, the interface on which the request arrived, failure of previous requests, or even the element's current level of utilization.

As potential destinations are located through these services, their next hops are added to the destination set. Next-hop locations may only be placed in the destination set once. If a next-hop location is already present in the set (based on the definition of equality for the URI type and equality of the optional parameters), it MUST NOT be added again.

A proxy MAY continue to add destinations to the set after beginning Request Processing. It MAY use any information obtained during that processing to determine new locations. For instance, a proxy may choose to incorporate contacts obtained in a redirect response (3xx class) into the destination set. If a proxy uses a dynamic source of information while building the destination set (for instance, if it consults a SIP Registrar), it SHOULD monitor that source for the duration of processing the request. New locations SHOULD be added to the destination set as they become available. As above, any given URI MUST NOT be added to the set more than once.

Allowing a URI to be added to the set only once reduces unnecessary network traffic, and in the case of incorporating contacts from redirect requests prevents infinite recursion.

An example trivial location service is achieved by configuring an element with a default outbound destination. All requests are forwarded to this location. The Request-URI of the request is placed in the destination set with the optional next-hop IP address, port and transport parameters set to the default outbound destination. The destination set is complete, containing **only** this URI, and the element proceeds to the task of Request Processing.

If the Request-URI indicates a resource at this proxy that does not exist, the proxy MUST return a 404 (Not Found) response.

If the destination set remains empty after applying all of the above, the proxy MUST return an error response, which SHOULD be the 480 (Temporarily Unavailable) response.

16.5 Request Processing

As soon as the destination set is non-empty, a proxy MAY begin forwarding the request. A stateful proxy MAY process the set in any order. It MAY process multiple destinations serially, allowing each client transaction to complete before starting the next. It MAY start client transactions with every destination in parallel. It also MAY arbitrarily divide the set into groups, processing the groups serially and processing the destinations in each group in parallel.

A common ordering mechanism is to use the qvalue parameter of destinations obtained from Contact header fields (see Section 22.10). Destinations are processed from highest qvalue to lowest. Destinations with equal qvalues may be processed in parallel.

A stateful proxy must have a mechanism to maintain the destination set as responses are received and associate the responses to each forwarded request with the original request. For the purposes of this model, this mechanism is a "response context" created by the proxy layer before forwarding the first request.

For each destination, the proxy forwards the request following these steps:

1. Make a copy of the received request
2. Update the Request-URI
3. Add a Via header field value
4. Update the Max-Forwards field if present
5. Update the Route header field if present
6. Optionally add a Record-route header field value
7. Optionally add additional headers
8. send the new request

Each of these steps is detailed below:

1. Copy request

   The proxy starts with a copy of the received request. The copy MUST initially contain all of the header fields from the received request. Only those fields detailed in the processing described below may be removed. The copy SHOULD maintain the ordering of the header fields as in the received request. The proxy MUST NOT reorder field values with a common field name (See Section 7.3.1).

   > An actual implementation need not perform a copy; the primary requirement is that the processing of each next hop begin with the same request.

2. Request-URI

   The Request-URI in the copy's start line MUST be replaced with the URI for this destination. If the URI contains any parameters not allowed in a Request-URI, they MUST be removed.

   This is the essence of a proxy's role. This is the mechanism through which a proxy routes a request toward its destination.

3. Via

The proxy MUST insert a Via header field into the copy before the existing Via header fields. The Via header maddr, ttl, and sent-by components will be set when the request is processed by the transport layer (Section 19). The Via headers ensure that responses will follow the same set of elements that the request traversed.

The proxy MUST include a “branch” parameter (Section 22.40) in the Via header. When the path of a request through one or more forking proxies is graphed, the result is a tree. The branch parameter identifies the “branch” each request was forwarded on. The branch parameter value MUST be unique for each client transaction to which the request is forwarded. The precise format of the branch. token is implementation-defined. In order to be able to both detect loops and associate responses with the corresponding request, the parameter SHOULD consist of two parts separable by the implementation. The first part is used to detect loops and distinguish loops from spirals. The second is used to match responses to requests.

Loop detection is performed by verifying that those fields having an impact on the routing decision have not changed. The value placed in the this part of the branch parameter SHOULD reflect all of those fields (which include any Proxy-Require and Proxy-Authorization headers). 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 (Section 3). A common way to create this value is to compute 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, in addition to any Proxy-Require and Proxy-Authorization fields that may be present. The algorithm used to compute the hash is implementation-dependent, but MD5 [23], expressed in hexadecimal, is a reasonable choice. (Note that base64 is not permissible for a token.)

In order to correctly match responses to requests (Section 17.1.3), the value SHOULD also contain a part that is a globally unique function of of the branch on which this request will be forwarded. One example is a hash of a sequence number, local IP address and request-URI of the request

For example: `7a83e5750418bce23d5106b4c06cc632.1`

> The “branch” parameter MUST depend on all information used for routing decisions, including the incoming request-URI and any header values affecting the routing choices. This is necessary to distinguish looped requests from requests whose routing parameters have changed before returning to this server.

Note that the request method MUST NOT be included in the calculation of the branch parameter. In particular, CANCEL and ACK requests MUST have the same branch value as the corresponding request they cancel or acknowledge. The branch parameter is used in correlating those requests at server handling them (see Section 17.2.3 and 9.2).

4. Max-Forwards

If the copy contains a Max-Forwards header field, the proxy must decrement its value by one (1).

5. Route

If the copy contains a Route header field, the proxy must remove the first (topmost) value. Note that this value was placed in the destination set and then into the Request-URI of this copy in previous steps.

6. Record-Route

If this proxy wishes to request to remain on the path of future requests in this dialog, it MUST insert a Record-Route header value (Section refsec:record-route) into the copy before any existing Record-Route header values. See Section 12 for details on whether this request will be honored. Each proxy in the path of a request makes this request independently the presence of a Record-Route header does not obligate this proxy to add a value.

If the request is honored, the information the proxy places in the Record-Route header value will be used at the endpoints to construct Route headers. As shown in the processing steps above, Route headers determine forwarding destinations much like strict IP source routing.

The URL placed in the Record-Route header value MUST be a SIP URL. This URL MAY be different for each destination the request is forwarded to. The URL SHOULD NOT contain the transport parameter unless the proxy has knowledge (such as in a private network) that the next downstream element that will be in the path of subsequent requests supports that transport.

> The URL this proxy provides will be used by some other element to make a routing decision. This proxy, in general, has no way to know what the capabilities of that element are, so it must restrict itself to the mandatory elements of a SIP implementation: SIP URLs and UDP transports.

The URL placed in the Record-Route header value MUST resolve to this element when the server location procedures of Section 24are applied to it. This ensures subsequent requests are routed back to this element.

The URL placed in the Record-Route header value SHOULD be such that if a subsequent 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 elements, including the originating client, traversed by the original request. This improves robustness, ensuring that the Request-URI contains enough information to forward subsequent requests to a reasonable destination even in the absence of Route headers.

The URL placed in the Record-Route header value MUST vary with the Request-URI in the received request. A request may legitimately pass through this proxy more than once on the way to its final destination (this is called a spiraling request). The Request-URI will be different each time the request passes through. If this proxy places the same URL in the Record-Route header field each time, subsequent requests will be rejected as looped requests. It is insufficient to simply copy the Request-URI from each request into the Record-Route header. Some modification, such as adding an maddr parameter, is necessary.

URLs satisfying the above paragraphs can be constructed in many ways. One way is to use a URL that 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 URL in the Record-Route be different for each distinct Request-URI). A call stateful proxy could use a URL of the form sip:proxy.example.com and use information from the stored call state to meet the requirements.

The proxy MAY include Record-Route header parameters in the value it provides. These will be returned in some responses to the request (200 responses to INVITE for example) and may be useful for pushing state into the message.

The Record-Route process is designed to work for any SIP request that initiates a dialog. The only such request in this specification is INVITE. Extensions to the protocol MAY define others, and the mechanisms described here will apply. The request that initiates a dialog and all refreshes (re-INVITE for example) MUST have Record-Route header values added to them if the proxy wishes to remain in the request path. This means a proxy will often need to record-route requests that contain Route headers. Section 12 describes how this will affect a dialog.

Including Record-Route even when Route headers already exist in a request improves robustness in the presence of a preloaded Route header field and recovery from endpoint failure.

If a proxy needs to be in the path of any type of dialog (such as one straddling a firewall), it SHOULD add a Record-Route header value to every request with a method it doesn't understand.

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 in a dialog. It is RECOMMENDED that proxies do not automatically record route. They should do so only if specifically required.

7. Adding Additional Headers

   The proxy MAY add any other appropriate headers to the copy at this point.

8. Forward Request

   A stateful proxy creates a new client transaction for this request as described in Section 17.1. If the next-hop location used in building this request contains the optional addressing parameters, the transaction is instructed to send the request based on those parameters. Otherwise, the proxy uses the procedures of Section 24 to compute an ordered set of addresses from the Request-URI, and as described there, attempts to contact the first one by instructing the client transaction to send the request there. If this fails, the stateful proxy continues down the list. Each attempt is a new client transaction, and therefore represents a new branch, so that the processing described above for each branch would need to be repeated. This results in a requirement to use a different branch ID parameter for each attempt.

16.6 Response Processing

When a response is received by an element, it first tries to locate a client transaction (Section 17.1.3) matching the response. If none is found, the element MUST process the response (even if it is an informational response) as a stateless proxy (described below). If a match is found, the response is handed to the client transaction.

Forwarding responses for which a client transaction (or more generally any knowledge of having sent an associated request) is not found improves robustness. In particular, it ensures that "late" 2xx class responses to INVITE requests are forwarded properly.

As client transactions pass responses to the proxy layer, the following processing MUST take place:

1. Find the appropriate response context

2. Remove the topmost Via

3. Add the response to the response context

4. Check to see if this response should be forwarded

The following processing MUST be performed on each response that is forwarded. Note that more than one response to each request will likely be forwarded - each provisional and one final at the least.

1. Aggregate authorization header fields if necessary

2. Forward the response

3. Generate any necessary CANCEL requests

If no final response has been forwarded after every client transaction associated with the response context has been terminated, the proxy must choose and forward the "best" response from those it has seen so far.

Each of the above steps are detailed below:

1. Find Context

   The proxy locates the "response context" it created before forwarding the original request using the key described in Section 16.5. The remaining processing steps take place in this context.

2. Via

   The proxy removes the topmost Via field value from the response. The address in this value necessarily matches the proxy since the response matched a client transaction above. The branch parameter from this value can be used to determine which branch the response corresponds to.

   If no Via field values remain in the response, the response was meant for this element and MUST NOT be forwarded. The remainder of the processing described in this section is not performed on this message. This will happen, for instance, when the element generates CANCEL requests as described in Section sec:proxy-response-processing-cancel.

3. Add response to context

   Final responses received are stored in the response context until a final response is generated on the server transaction associated with this context. The response may a candidate for the best final response to be returned on that server transaction. Information from this response may be needed in forming the best response even if this response is not chosen.

   If the proxy chooses to recurse on a 3xx class response, it MUST NOT add the response to the response context

4. Check response for forwarding

   Until a final response has been sent on the server transaction, the following responses MUST be forwarded immediately:

   - Any provisional response other than 100 Trying
   - Any 2xx response

   If a 6xx response is received, it is not immediately forwarded, but the stateful proxy SHOULD cancel all pending transactions as described in Section 9.

This is a change from RFC2543, which mandated that the 6xx be forwarded immediately. The problem with this is that it is possible for a 2xx to arrive on another branch, in which case the proxy would have to forward that in the case of an INVITE transaction. The result is that the UAC could receive a 6xx followed by a 2xx, which should never be allowed to happen. So, instead, upon receiving a 6xx, a proxy will CANCEL, which will generally result in 487s to all outstanding client transactions, and then at that point the 6xx is forwarded upstream.

After a final response has been sent on the server transaction, the following responses MUST be forwarded immediately:

- Any 2xx class response to an INVITE request

A stateful proxy MUST NOT immediately forward any other responses. In particular, a stateful proxy MUST NOT forward any 100 Trying response. Those responses that are candidates for forwarding later as the "best" response have been gathered as described in step "Add Response to Context".

Any response chosen for immediate forwarding MUST be processed as described in steps "Aggregate authorization headers" through "Record-Route".

5. Choosing the best response

A stateful proxy MUST send a final response to a response context's server transaction if no final responses have been immediately forwarded by the above rules and all client transactions in this response context have been terminated.

The stateful proxy MUST choose the "best" final response among those received and stored in the response context.

If there are no final responses in the context, the proxy MUST send a 408 (Request Timeout) response to the server transaction.

Otherwise, the proxy MUST forward one of the responses from the lowest response class stored in the response context. The proxy MAY select any response within that lowest class. The proxy SHOULD give preference to responses that provide information affecting resubmission of this request, such as 401, 407, 415, 420, and 484.

A proxy which receives a 503 response 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.

The forwarded response MUST be processed as described in steps "Aggregate authorization headers" through "Record-Route".

For example, if a proxy forwarded a request to 4 locations, and received 503, 407, 501, and 404 responses, it may choose to forward the 407 response.

The tag in the To header field serves to distinguish responses at the UAC. If the forwarded response did not have one, it MUST NOT be inserted into the response by the proxy.

6. Aggregate authorization headers

If the selected response is a 401 or 407, the proxy MUST collect any WWW-Authenticate and Proxy-Authenticate header fields from all other 401 and 407 responses received so for in this response context and add them to this response before forwarding.

This is necessary because any or all of the destinations the request was forwarded to may have requested credentials. The client must receive all of those challenges and supply credentials for each of them when it retries the request. Motivation for this behavior is provided in Section 20.

7. Record-Route

If the selected response contains a Record-Route header field value originally provided by this proxy, the proxy MAY chose to rewrite the value before forwarding the response. This allows the proxy to provide different URLs for itself to the next upstream and downstream elements. A proxy may choose to use this mechanism for any reason. For instance, it is useful for multi-homed hosts.

The new URL provided by the proxy MUST satisfy the same constraints on URLs placed in Record-Route header fields in requests (see Section 6) with the following modifications:

The URL SHOULD NOT contain the transport parameter unless the proxy has knowledge that the next upstream (as opposed to downstream) element that will be in the path of subsequent requests supports that transport.

The URL placed in the Record-Route header value SHOULD be such that if a subsequent 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 element (as opposed to some previous element) as the originally forwarded request.

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.

8. Forward response

After performing the processing described in steps "Aggregate authorization headers" through "Record-Route", the proxy may perform any feature specific manipulations on the selected response. Unless otherwise specified, the proxy MUST NOT remove the message body or any header values other than the Via header value discussed in Section refsec:proxy-response-processing-via. The proxy MUST pass the response to the server transaction associated with the response context. This will result in the response being sent to the location now indicated in the topmost Via field value. If the server transaction is no longer available to handle the transmission, the element MUST forward the response statelessly by sending it to the server transport.

Even after forwarding a final response, the proxy MUST maintain the response context until all of its associated transactions have been terminated.

9. Generate CANCELs

OPEN ISSUE #7: If CANCEL is restricted to INVITE only, this behavior must restrict itself to INVITE requests.

OPEN ISSUE #122: The MUST below reflects list discussion, but the question of how strong this requirement should be was not formally closed.

If the forwarded response was a final response, the proxy MUST generate a CANCEL request for all pending client transactions associated with this response context. A proxy SHOULD also generate a CANCEL request for all pending client transactions associated with this response context when it receives a 6xx response. A pending client transaction is one that has received a provisional response, but no final response and has not had an associated CANCEL generated for it. Generating CANCEL requests is described in Section 9.1.

16.7 Handling transport errors

If the transport layer notifies a proxy of an error when it tries to forward a request (see Section 19.4), the proxy MUST behave as if the forwarded request received a 400 response.

If the proxy is notified of an error when forwarding a response, it drops the response. The proxy SHOULD NOT cancel any outstanding client transactions associated with this response context due to this notification.

If a proxy cancels its outstanding client transactions, a single malicious or misbehaving client can cause all transactions to fail through its Via header field.

16.8 CANCEL Processing

A stateful proxy may generate a CANCEL to any other request it has generated at any time. For instance, it may choose to generate CANCELs based on having a transaction exceed the time specified in the Expire header of certain requests, or as a result of any logic it applies while forwarding requests. A proxy MUST cancel any pending client transactions associated with a response context when it receives a matching CANCEL request.

OPEN ISSUE #185: Should generating CANCEL at a proxy based on Expires in INVITE be deprecated?

While a CANCEL request is handled in a stateful proxy by its own server transaction, a new response context is not created for it. Instead, the proxy layer searches its existing response contexts for the server transaction handling the request associated with this CANCEL. If a matching response context is found, the element MUST immediately return a 200 OK response to the CANCEL request. In this case, the element is acting as a user agent server as defined in Section 8.2. Furthermore, the element MUST generate CANCEL requests for all pending client transactions in the context as described in Section 9.

If a response context is not found, the element does not have any knowledge of the request to apply the CANCEL to. It MUST forward the CANCEL request statelessly (it may have statelessly forwarded the associated request previously).

16.9 Stateless proxy

When acting statelessly, a proxy is a simple message forwarder. Much of the processing performed when acting statelessly is the same as when behaving statefully. The differences are detailed here.

A stateless proxy does not have any notion of a transaction, or of the response context used to describe stateful proxy behavior. Instead, the stateless proxy takes messages, both requests and responses, directly from the transport layer (See section 19). As a result, stateless proxies do not retransmit messages on their own. They do, however, forward all retransmission they receive (they do not have the ability to distinguish a retransmission from the original message). Furthermore, when handling a request statelessly, an element MUST NOT generate its own 100 Trying (or any other provisional) response.

A stateless proxy must validate a request as described in Section 16.3

A stateless proxy must make a routing decision as described in Section 16.4 with the following exception:

- A stateless proxy MUST choose one and only one destination from the destination set. This choice MUST only rely on fields in the message and time-invariant properties of the server. In particular, a retransmitted request MUST be forwarded to the same destination each time it is processed. Furthermore, CANCEL and non-Routed ACK requests MUST generate the same choice as their associated INVITE.

A stateless proxy must process the request before forwarding as described in Section 16.5 with the following exceptions:

- The branch parameter on the inserted Via header field MUST be the same each time a retransmitted request is forwarded. Thus for a stateless proxy, the branch parameter calculation MUST **only** depend on message parameters affecting the routing of the request which are invariant on retransmission.

- The request is sent directly to the transport layer instead of through a client transaction. If the next-hop destination parameters don't provide an explicit destination, the element applies the procedures of Section 24 to the Request-URI to determine where to send the request.

Stateless proxies MUST NOT perform special processing for CANCEL requests. They are processed by the above rules as any other requests.

Response processing as described in Section 16.6 does not apply to a proxy behaving statelessly. When a response arrives at a stateless proxy, the proxy inspects the address in the first (topmost) Via header value. If that address matches the proxy, the proxy MUST remove that value from the response and forward the result to the location indicated in the next Via header value. Unless specified otherwise, the proxy MUST NOT remove any other header values or the message body. If the address does not match the proxy, the message MUST be silently discarded.

17 Transactions

SIP is fundamentally a transactional protocol. This means that interactions between components take place in a series of independent message exchanges. Specifically, a SIP transaction consists of a single request, and any responses to that request (which include zero or more provisional responses and one or more final responses). In the case of a transaction where the request was an INVITE (known as an INVITE transaction), the transaction also includes the ACK only if the final response was not a 2xx response. If the response was a 2xx, the ACK is not considered part of the transaction.

> The reason for this separation is rooted in the importance of delivering all 200 OK responses to an INVITE to the UAC. To deliver them all to the UAC, the UAS alone takes responsibility for retransmitting them, and the UAC alone takes responsibility for acknowledging them with ACK. Since this ACK is retransmitted only by the UAC, it is effectively considered its own transaction.

Transactions have a client side and a server side. The client side is known as a client transaction, and the server side, as a server transaction. The client transaction sends the request, and the server transaction sends the response. The client and server transactions are logical functions that are embedded in any number of elements. Specifically, they exist within user agents and stateful proxy servers. Consider the example of Section 4. In this example, the UAC executes the client transaction, and its outbound proxy executes the server transaction. The outbound proxy also executes a client transaction, which sends the request to a server transaction in the inbound proxy. That proxy also executes a client transaction, which in turn, sends the request to a server transaction in the UAS. This is shown pictorially in Figure 4.

```
+---------+        +---------+        +---------+        +---------+
|      +-+|Request |+-+   +-+|Request |+-+   +-+|Request |+-+      |
|      |C||------->||S|   |C||------->||S|   |C||------->||S|      |
|      |l||        ||e|   |l||        ||e|   |l||        ||e|      |
|      |i||        ||r|   |i||        ||r|   |i||        ||r|      |
|      |e||        ||v|   |e||        ||v|   |e||        ||v|      |
|      |n||        ||e|   |n||        ||e|   |n||        ||e|      |
|      |t||        ||r|   |t||        ||r|   |t||        ||r|      |
|      | ||        || |   | ||        || |   | ||        || |      |
|      |T||        ||T|   |T||        ||T|   |T||        ||T|      |
|      |r||        ||r|   |r||        ||r|   |r||        ||r|      |
|      |a||        ||a|   |a||        ||a|   |a||        ||a|      |
|      |n||        ||n|   |n||        ||n|   |n||        ||n|      |
|      |s||Response||s|   |s||Response||s|   |s||Response||s|      |
|      +-+|<-------|+-+   +-+|<-------|+-+   +-+|<-------|+-+      |
+---------+        +---------+        +---------+        +---------+
   UAC               Outbound           Inbound             UAS
                     Proxy               Proxy
```

Figure 4: Transaction relationships

A stateless proxy does not contain a client or server transaction. The transaction exists between the UA or stateful proxy on one side of the stateless proxy, and the UA or stateful proxy on the other side. As far as SIP transactions are concerned, stateless proxies are effectively transparent. The purpose of the client transaction is to receive a request from the element the client is embedded in (call this element the "Transaction User" or TU; it can be a UA or a stateful proxy), and reliably deliver the request to that server transaction. The client transaction is also responsible for receiving responses, and delivering them to the TU, filtering out any retransmissions or disallowed responses (such as a response to ACK). In the case of an INVITE transaction, that includes generation of the ACK request for any final response excepting a 2xx response.

Similarly, the purpose of the server transaction is to receive requests from the transport layer, and deliver them to the TU. The server transaction filters any request retransmissions from the network. The server transaction accepts responses from the TU, and delivers them to the transport layer for transmission over the network. In the case of an INVITE transaction, it absorbs the ACK request for any final response excepting a 2xx response.

The 2xx response, and the ACK for it, have special treatment. This response is retransmitted only by a

UAS, and its ACK generated only by the UAC. This end-to-end treatment is needed so that a caller knows the entire set of users that have accepted the call. Because of this special handling, retransmissions of the 2xx response are handled by the UA core, not the transaction layer. Similarly, generation of the ACK for the 2xx is handled by the UA core. Each proxy along the path merely forwards each 2xx response to INVITE, and its corresponding ACK.

17.1 Client transaction

The client transaction provides its functionality through the maintenance of a state machine.

The TU communicates with the client transaction through a simple interface. When the TU wishes to initiate a new transaction, it creates a client transaction, and passes it the SIP request to send, a value for timer C (described below), and an IP address, port, and transport to send it to. The client transaction begins execution of its state machine. Valid responses are past up to the TU from the client transaction.

There are two types of client transaction state machines, depending on the method the request passed by the TU. One handles client transactions for INVITE request. This type of machine is referred to as an INVITE client transaction. Another type handles client transactions for all requests except INVITE and ACK. This is referred to as a non-INVITE client transaction. There is no client transaction for ACK. If the TU wishes to send an ACK, it passes one directly to the transport layer for transmission.

The INVITE transaction is different from those of other methods because of its extended duration. Normally, human input is required in order to respond to an INVITE. The long delays expected for sending a response argue for a three way handshake. Requests of other methods, on the other hand, are expected to completely rapidly. In fact, because of its reliance on just a two way handshake, TUs SHOULD respond immediately to non-INVITE requests. Protocol extensions which require longer durations for generation of a response (such as a new method that does require human interaction) SHOULD instead use two transactions - one to send the request, and another in the reverse direction to convey the result of the request.

17.1.1 INVITE Client Transaction

17.1.1.1 Overview of INVITE Transaction The INVITE transaction consists of a three-way handshake. The client transaction sends an INVITE, the server transaction sends responses, and the client transaction sends an ACK. For unreliable transports (such as UDP), the client transaction will retransmit requests at an interval that starts at T1 seconds and doubles after every retransmission. The request is not retransmitted over reliable transports. After receiving a 1xx response, any retransmissions cease altogether, and the client waits for further responses. The server transaction can send additional 1xx responses, which are not transmitted reliably. Eventually, the server transaction decides to send a final response. For unreliable transports, that response is retransmitted periodically, and for reliable transports, its sent once. For each final response that is received at the client transaction, the client transaction sends an ACK, the purpose of which is to quench retransmissions of the response.

17.1.1.2 Formal Description The state machine for the INVITE client transaction is shown in Figure 5. The initial state, “calling”, MUST be entered when the TU initiates a new client transaction with an INVITE request. The client transaction MUST pass the request to the transport layer for transmission (see Section 19). If an unreliable transport is being used, the client transaction SHOULD start timer A with a value of T1, and SHOULD NOT start timer A when a reliable transport is being used (Timer A controls request retransmissions). For any transport, the client transaction MUST start timer B with a value of 64*T1 seconds

```
                   |INVITE from TU
 Timer A fires     |INVITE sent
 Reset A,          V             Timer B fires
 INVITE sent +-----------+       t.o. to TU
   +---------|           |---------------+
   |         |  Calling  |               |
   +-------->|           |-------------->|
             +-----------+ 2xx           |
300-699         |  |       2xx to TU     |
ACK sent        |  |1xx                  |
+---------------+  |1xx to TU            |
|                  |                     |
|  1xx             V       Timer C fires |
|  1xx to TU  -----------+ t.o. to TU    |
|  +---------|           |-------------->|
|  |         |Proceeding |               |
|  +-------->|           |-------------->|
|            +-----------+ 2xx           |
|       300-699    |       2xx to TU     |
|       ACK sent,  |                     |
|       resp. to TU|                     |
|                  |                     |      NOTE:
|  300-699         V                     |
|  ACK sent  +-----------+               |  transitions
|  +---------|           |               |  labeled with
|  |         | Completed |               |  the event
|  +-------->|           |               |  over the action
|            +-----------+               |  to take
|              ^   |                     |
|              |   | Timer D fires       |
+--------------+   | -                   |
                   |                     |
                   V                     |
             +-----------+               |
             |           |               |
             | Terminated|<--------------+
             |           |
```

Figure 5: INVITE client transaction

(Timer B controls transaction timeouts).

When timer A fires, the client transaction SHOULD retransmit the request by passing it to the transport layer, and SHOULD reset the timer with a value of 2*T1. When the timer fires 2*T1 seconds later, the request SHOULDbe retransmitted again (assuming the client transaction is still in this state). This process SHOULDcontinue, so that the request is retransmitted with intervals that double after each transmission. These retransmissions SHOULDonly be done while the client transaction is in the “calling” state.

The default value for T1 is 500ms. T1 is an estimate of the RTT between the client and server transactions. The optional RTT estimation procedure of Section 17.3 MAY be followed, in which case the resulting estimate MAY be used instead of 500ms. If no RTT estimation is used, other values MAYbe used in private networks where it is known that RTT has a different value. On the public Internet, T1 MAY be chosen larger, but SHOULD NOT be smaller.

If the client transaction is still in the “calling” when timer B fires, the client transaction SHOULD inform the TU that a timeout has occurred. The client transaction MUST NOT generate an ACK. The value of 64*T1 is equal to the amount of time required to send seven requests in the case of an unreliable transport.

If the client transaction receives a provisional response while in the ”calling” state, it transitions to the “proceeding” state. Upon entering this state, the client transaction MUST start timer C with the value provided by the TU when the client transaction was created. This timeout dictates how long the client transaction waits for a final response before giving up (i.e., roughly how long does it “let the phone ring”). In the “proceeding” state, the client transaction SHOULD NOT retransmit the request any longer. Furthermore, the provisional response MUST be passed to the TU. Any further provisional responses MUST be passed up to the TU while in the “proceeding” state. When timer C fires, the client transaction MUST transition to the terminated state, and it MUST inform the TU of the timeout.

When in either the ”calling” or “proceeding” states, reception of a response with status code from 300-699 MUST cause the client transaction to transition to “completed”. The client transaction MUST pass the received response up to the TU, and it MUST generate an ACK request, even if the transport is reliable (guidelines for constructing the ACK from the response are given in Section 17.1.1.3) and then pass the ACK to the transport layer for transmission. The ACK MUST be sent to the same address, port and transport that the original request was sent to. The client transaction SHOULD start timer D when it enters the “completed” state, with a value of T3 seconds for unreliable transports, and zero seconds for reliable transports. T3 is the total amount of time that the server transaction can remain in the “completed” state when unreliable transports are used. For the default values of the timers below, this is 16 seconds.

> OPEN ISSUE #210: Timer D should be based on the values of the timers selected at the server, but these values aren’t known by the client. We could alternatively specify an absolute minimum.

Any retransmissions of the final response that are received while in the “completed” state SHOULD cause the ACK to be re-passed to the transport layer for retransmission, but the newly received response MUST NOT be passed up to the TU. A retransmission of the response is defined as any response which would match the same client transaction, based on the rules of Section 17.1.3.

If timer D fires while the client transaction is in the “completed” state, the client transaction MUST move to the terminated state, and it MUST inform the TU of the timeout.

When in either the “calling” or “proceeding” states, reception of a 2xx response MUST cause the client transaction to enter the terminated state, and the response MUST be passed up to the TU. The handling of this response depends on whether the TU is a proxy core or a UAC core. A UAC core will handle generation of the ACK for this response, while a proxy core will always forward the 200 OK upstream. The differing treatment of 200 OK between proxy and UAC is the reason that handling of it does not take place in the transaction layer.

The client transaction MUST be destroyed the instant it enters the terminated state. This is actually necessary to guarantee correct operation. The reason is that 2xx responses to an INVITE are treated differently; each one is forwarded by proxies, and the ACK handling in a UAC is different. Thus, each 2xx needs to be passed to a proxy core (so that it can be forwarded) and to a UAC core (so it can be acknowledged). No transaction layer processing takes place. Whenever a response is received by the transport, if the transport layer finds no matching client transaction (using the rules of Section 17.1.3, the response is passed directly to the core. Since the matching client transaction is destroyed by the first 2xx, subsequent 2xx will find no match and therefore be passed to the core.

17.1.1.3 Construction of the ACK Request The ACK request constructed by the client transaction MUST contain values for the Call-ID, From, and Request-URI which are equal to the values of those headers in the request that created the client transaction (call this the "original request"). The To field in the ACK MUST equal the To field in the response being acknowledged, and will therefore usually differ from the To field in the original request by the addition of the tag parameter. The ACK MUST contain a single Via header, and this MUST be equal to the top Via header of the original request. The ACK request MUST NOT contain any Route headers. The CSeq header in the ACK MUST contain the same value for the sequence number as was present in the original request, but the method parameter MUST be equal to "ACK".

These rules for construction of ACK only apply to the client transaction. A UAC core which generates an ACK for 2xx MUST instead follow the rules described in Section 13.

For example, consider the following request:

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP 10.1.3.3
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=88sja8x
Call-ID: 987asjd97y7atg@10.1.3.3
CSeq: 986759 INVITE
```

The ACK request for a non-2xx final response to this request would look like:

```
ACK sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP 10.1.3.3
To: Bob <sip:bob@biloxi.com>;tag=99sa0xk
From: Alice <sip:alice@atlanta.com>;tag=88sja8x
Call-ID: 987asjd97y7atg@10.1.3.3
CSeq: 986759 ACK
```

17.1.2 non-INVITE Client Transaction

17.1.2.1 Overview of the non-INVITE Transaction non-INVITE transactions do not make use of ACK. They are a simple request-response interaction. For unreliable transports, requests are retransmitted at an interval which starts at T1, and doubles until it hits T2. If a provisional response is received, retransmissions continue for unreliable transports, but at an interval of T2. The server transaction retransmits the last response it sent (which can be a provisional or final response) only when a retransmission of the request is received. This is why request retransmissions need to continue even after a provisional response, they are what ensure reliable delivery of the final response.

Unlike an INVITE transaction, a non-INVITE transaction has no special handling for the 2xx response. The result is that only a single 2xx response to a non-INVITE is ever delivered to a UAC.

17.1.2.2 Formal Description The state machine for the non-INVITE client transaction is shown in Figure 6. It is very similar to the state machine for INVITE.

```
                               |Request from app
                               |send request
           Timer E             V              Timer F
           send request  +-----------+        t.o. to TU
               +---------|           |-------------------+
               |         |  Trying   |                   |
               +-------->|           |                   |
                         +-----------+                   |
            200-699         |   |                        |
            resp. to TU     |   |1xx                     |
            +---------------+   |resp. to TU             |
            |                   |                        |
            |   Timer E         V       Timer F          |
            |   send req +-----------+ t.o.to TU         |
            |  +---------|           |------------------>|
            |  |         |Proceeding |                   |
            |  +-------->|           |-----+             |
            |            +-----------+     |1xx          |
            |              |      ^        |resp to TU   |
            | 200-699      |      +--------+             |
            | resp. to TU  |                             |
            |              |                             |
            |              V                             |
            |            +-----------+                   |
            |            |           |                   |
            |            | Completed |                   |
            |            |           |                   |
            |            +-----------+                   |
            |              ^   |                         |
            |              |   | Timer K                 |
            +--------------+   | -                       |
                               |                         |
                               V                         |
         NOTE:           +-----------+                   |
                         |           |                   |
     transitions         | Terminated|<------------------+
     labeled with        |           |
     the event           +-----------+
     over the action
     to take
```

Figure 6: non-INVITE client transaction

The "Trying" state is entered when the TU initiates a new client transaction with a request. When entering this state, the client transaction SHOULD set Timer F to fire in T3 seconds. The request MUST be passed to the transport layer for transmission. If an unreliable transport is in use, the client transaction MUST set timer E to fire in T1 seconds. If timer E fires while still in this state, the timer is reset, but this time with a value of MIN(2*T1, T2). When the timer fires again, it is reset to a MIN(4*T1, T2). This process continues, so that retransmissions occur with an exponentially increasing inverval that caps at T2. The default value of T2 is 4s, and it represents the amount of time a non-INVITE server transaction will take to respond to a request, if it does not respond immediately. For the default values of T1 and T2, this results in intervals of 500 ms, 1 s, 2 s, 4 s, 4 s, 4s, etc.

If Timer F fires while the client transaction is still in the “Trying” state, the client transaction SHOULD inform the TU about the timeout, and then it SHOULDenter the “Terminated” state. If a provisional response is received while in the “Trying” state, the response MUST be passed to the TU, and then the client transaction SHOULD move to the “Proceeding” state. If a final response (status codes 200-699) is received while in the “Trying” state, the response MUST be passed to the TU, and the client transaction MUST transition to the “Completed” state.

If Timer E fires while in the “Proceeding” state, the request MUST be passed to the transport layer for retransmission, and Timer E MUST be reset with a value of T2 seconds. If timer F fires while in the “Proceeding” state, the TU MUST be informed of a timeout, and the client transaction MUST transition to the terminated state. If a final response (status codes 200-699) is received while in the “Proceeding” state, the response MUST be passed to the TU, and the client transaction MUST transition to the “Completed” state.

Once the client transaction enters the “Completed” state, it MUST set Timer K to fire in T4 seconds for unreliable transports, and zero seconds for reliable transports. The “Completed” state exists to buffer any additional response retransmissions that may be received (which is why the client transaction remains there only for unreliable transports). T4 represents the amount of time the network will take to clear messages between client and server transactions. The default value of T4 is 5s. A response is a retransmission when it matches the same transaction, using the rules specified in Section 17.1.3. If Timer K fires while in this state, the client transaction MUST transition to the “Terminated” state.

OPEN ISSUE #211: This special treatment for reliable transports, where the state machine transactions directly to terminated, is new.

Once the transaction is in the terminated state, it MUST be destroyed. As with client transactions, this is needed to ensure reliability of the 2xx responses to INVITE.

17.1.3 Matching Responses to Client Transactions

When the transport layer in the client receives a response, it has to figure out which client transaction will handle the response, so that the processing of Sections 17.1.1 and 17.1.2 can take place.

A response matches a client transaction through a comparison process with fields in the request that created the transaction. Specifically, the From, Call-ID, CSeq, and the topmost Via header MUST match the same fields in the request, using the matching operations for those headers defined in Section 22. If the To field in the request had a tag, the To field in the response MUST match the To field in the request, as described in Section 22.37. However, if the To field in the request did not contain a tag, the To field in the response MUST match that in the request, except that the tag MUST NOT be considered as part of the matching process. This is needed since a UAS will add a tag to the To field of the response.

17.1.4 Handling Transport Errors

When the client transaction sends a request to the transport layer to be sent, the following procedures are followed if the transport layer indicates a failure.

The client transaction SHOULD inform the TU that a transport failure has occurred, and the client transaction SHOULD transition directly to the terminated state.

17.2 Server Transaction

The server transaction is responsible for the delivery of requests to the TU, and the reliable transmission of responses. It accomplishes this through a state machine. Server transactions are created by the core when a request is received, and transaction handling is desired for that request (this won't always be the case).

As with the client transactions, the state machine depends on whether the received request is an INVITE request or not.

17.2.1 INVITE Server Transaction

The state diagram for the INVITE server transaction is shown in Figure 7.

When a server transaction is constructed with a request, it enters the "Proceeding" state. The server transaction MUST generate a 100 response (not any status code - the specific value of 100) unless it knows that the TU will generate a provisional or final response within 200 ms, in which case it MAY generate a 100 response. This provisional response is needed to rapidly quench request retransmissions in order to avoid network congestion. The request MUST be passed to the TU.

The TU passes any number of provisional responses to the server transaction. So long as the server transaction is in the "Proceeding" state, each of these MUST be passed to the transport layer for transmission. They are not sent reliably (they are not retransmitted), and do not cause a change in the state of the server transaction. If a request retransmission is received while in the "Proceeding" state, the most recent provisional response that was received from the TU MUST be passed to the transport layer for retransmission. A request is a retransmission if it matches the same server transaction based on the rules of Section 17.2.3.

If, while in the "proceeding" state, the TU passes a 2xx Response to the server transaction, the server transaction MUST pass this response to the transport layer for transmission. It is not retransmitted by the server transaction; retransmissions of 2xx responses are handled by the TU. The server transaction MUST then transition to the "terminated" state.

While in the "Proceeding" state, if the TU passes a response with status code from 300 to 699 to the server transaction, the response MUST be passed to the transport layer for transmission, and the state machine MUST enter the "Completed" state. For unreliable transports, timer G is set to fire in T1 seconds, and is not set to fire for reliable transports.

> This is a change from RFC2543, where responses were always retransmitted, even over reliable transports.

When the "Completed" state is entered, timer H MUST be set to fire in 64*T1 seconds, for all transports. Timer H determines when the server transaction gives up retransmitting the response. Its value is chosen to equal Timer B, the amount of time a client transaction will continue to retry sending a request. If timer G fires, the response is passed to the transport layer once more for retransmission, and timer G is set to fire in MIN(2*T1, T2) seconds. From then on, when timer G fires, the response is passed to the transport again for transmission, and timer G is reset with a value that doubles, unless that value exceeds T2, in which case it is reset with the value of T2. This is identical to the retransmit behavior for requests in the "Trying" state of the non- INVITE client transaction. Furthermore, while in the "completed" state, if a request retransmission is received, the server SHOULD pass the response to the transport for retransmission.

```
                   |INVITE
                   |pass to TU, send 100
INVITE             V
send response+-----------+
    +--------|           |--------+101-199 from TU
    |        | Proceeding|        |send response
    +------->|           |<-------+
             +-----------+
300-699 from TU |     |2xx from TU
send response   |     |send response
                |     +-------------------+
                |                         |
INVITE          V          Timer G fires  |
send response+-----------+ send response  |
    +--------|           |--------+       |
    |        | Completed |        |       |
    +------->|           |<-------+       |
             +-----------+                |
                |     |                   |
            ACK |     |                   |
            -   |     +------------------>+
                |          Timer H fires  |
                V          fail to TU     |
             +-----------+                |
             |           |                |
             | Confirmed |                |
             |           |                |
             +-----------+                |
                   |                      |
                   |Timer I fires         |
                   |-                     |
                   |                      |
                   V                      |
             +-----------+                |
             |           |                |
             | Terminated|<---------------+
             |           |
             +-----------+
```

Figure 7: INVITE server transaction

If an ACK is received while the server transaction is in the “Completed” state, the server transaction MUST transition to the “confirmed” state. As Timer G is ignored in this state, any retransmissions of the response will cease.

If timer H fires while in the “Completed” state, it implies that the ACK was never received. In this case, the server transaction MUST transition to the terminated state, and MUST indicate to the TU that a transaction failure has occurred.

The purpose of the "confirmed" state is to absorb any additional ACK messages that arrive, triggered from retransmissions of the final response. When this state is entered, timer I is set to fire in T4 seconds for unreliable transports, and zero seconds for reliable transports. Once timer I fires, the server MUST transition to the "Terminated" state.

Once the transaction is in the terminated state, it MUST be destroyed. As with client transactions, this is needed to ensure reliability of the 2xx responses to INVITE.

17.2.2 non-INVITE Server Transaction

The state machine for the non-INVITE server transaction is shown in Figure 8.

The state machine is initialized in the "Trying" state, and is passed a request other than INVITE or ACK when initialized. This request is passed up to the TU. Once in the "Trying" state, any further request retransmissions are discarded. A request is a retransmission if it matches the same server transaction, using the rules specified in Section 17.2.3.

While in the "Trying" state, if the TU passes a provisional response to the server transaction, the server transaction MUST enter the "Proceeding" state. The response MUST be passed to the transport layer for transmission. Any further provisional responses that are received from the TU while in the "Proceeding" state MUST be passed to the transport layer for transmission. If a retransmission of the request is received while in the "Proceeding" state, the most recently sent provisional response MUST be passed to the transport layer for retransmission. If the TU passes a final response (status codes 200-699) to the server while in the "Proceeding" state, the transaction MUST enter the "Completed" state, and the response MUST be passed to the transport layer for transmission.

When the server transaction enters the "Completed" state, it MUST set Timer J to fire in T3 seconds for unreliable transports, and zero seconds for reliable transports. While in the "Completed" state, the server transaction MUST pass the final response to the transport layer for retransmission whenever a retransmission of the request is received. Any other final responses passed by the TU to the server transaction MUST be discarded while in the "Completed" state. The server transaction remains in this state until Timer J fires, at which point it MUST transition to the "Terminated" state.

The server transaction MUST be destroyed the instant it enters the "Terminated" state.

17.2.3 Matching Requests to Server Transactions

When an INVITE or ACK request is received from the network by the server, it has to be matched to an existing INVITE transaction. The INVITE request matches a transaction if the Request-URI, To, From, Call-ID, CSeq, and top Via header match those of the INVITE request which created the transaction. The ACK request matches a transaction if the Request-URI, From, Call-ID, CSeq method (not the number), and top Via header match those of the INVITE request which created the transaction, and the To field of the ACK matches the To field of the response sent by the server transaction (which then includes the tag). Matching is done based on the matching rules defined for each of those headers. The usage of the tag in the To field helps disambiguate ACK for 2xx from ACK for other responses at a proxy which may have forwarded both responses (which can occur in unusual conditions).

For all other request methods, a request is matched to a transaction if the Request-URI, To, From, Call-ID and Cseq (including the method) and top Via header match those of the request which created the transaction. Matching is done based on the matching rules defined for each of those headers.

Because the matching rules include the Request-URI, the server cannot match a response to a transaction. When the TU passes a response to the server, it must inform the TU which transaction the response is

```
                   |Request received
                   |pass to TU
                   V
             +-----------+
             |           |
             | Trying    |-------------+
             |           |             |
             +-----------+             |200-699 from TU
                   |                   |send response
                   |1xx from TU        |
                   |send response      |
                   |                   |
Request            V      1xx from TU  |
send response+-----------+send response|
    +--------|           |--------+    |
    |        | Proceeding|        |    |
    +------->|           |<-------+    |
             +-----------+             |
                   |                   |
                   |                   |
                   |200-699 from TU    |
                   |send response      |
Request            V                   |
send response+-----------+             |
    +--------|           |             |
    |        | Completed |-------------+
    +------->|           |
             +-----------+
                   |
                   |Timer J fires
                   |-
                   |
                   V
             +-----------+
             |           |
             | Terminated|
             |           |
             +-----------+
```

Figure 8: non-INVITE server transaction

for.

17.3 RTT Estimation

Most of the timeouts used in the transaction state machines derive from T1, which is an estimate of the RTT between the client and server transactions. This subsection defines optional procedures that a client can use to build up estimates of the RTT to a particular IP address. To perform this procedure, the client MUST maintain a table of variables for each destination IP address to which an RTT estimate is being made.

OPEN ISSUE #212: Is destination IP address the right index for an RTT estimate? How about Request-URI?

If a client wishes to measure RTT for a particular IP address, it MUST include a Timestamp header into a request containing the time when the request is initially created and passed to a new client transaction, which transmits the request. If a 100 response (not any 1xx, only the 100 response) is received before the client transaction generates a retransmission, an RTT estimate is made. This is consistent with the RFC 2988 requirements on TCP for using Karn's algorithm in RTT estimation.

The estimate, called R, is made by computing the difference between the current time and the value of Timestamp header in the 100 response. The value of R is applied to the estimation of RTO as described in Section 2 of RFC 2988 [24], with the following differences. First, the initial value of RTO is 500 ms for SIP, not 3 s as is used for TCP. Second, there is no minimum value for the RTO, as there is for TCP, if SIP is being run on a private network. When run on the public Internet, the minimum is 500 ms, as opposed to 1 s for TCP. This difference is because of the expected usage of SIP in private networks where rapid call setup times are service critical. Once RTO is computed, the timer T1 is set to the value of RTO, and all other timers scale proportionally as described above.

18 Reliability of Provisional Responses

Placeholder.

Reliability of provisional responses will be incorporated into bis. This is a heads up on that.

19 Transport

The transport layer is responsible for the actual transmission of requests and responses over network transports. This includes determination of the connection to use for a request or response, in the case of connection oriented transports.

The transport layer is responsible for managing any persistent connections (for transports like TCP, TLS and SCTP) including ones it opened, as well as ones opened to it. This includes connections opened by the client or server transports, so that connections are shared between client and server transport functions. It is RECOMMENDED that connections be kept open for some implementation defined time after the last message was sent or received over that connection. This time SHOULD be at least 16 seconds in order to ensure with high probability that responses can be sent over the same connection a request was sent.

All SIP elements MUST support UDP at a minimum.

19.1 Clients

19.1.1 Sending Requests

The client side of the transport layer is responsible for sending the request and receiving responses. The user of the transport layer passes the client transport the request, an IP address, port, transport, and possibly TTL for multicast destinations.

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.)

Requests sent to multicast groups SHOULD be scoped to ensure that they are not forwarded beyond the administrative domain to which they were targeted. This scooping MAY be done with either TTL or administrative scopes [19], depending on what is implemented in the network.

It is important to note that the layers above the transport layer do not operate differently for multicast as opposed to unicast requests. This means that SIP treats multicast more like anycast, assuming that there is a single recipient generating responses to requests. If this is not the case, the first response will end up "winning", based on the client transaction rules. Any other responses from different UA will appear as retransmissions and be discarded. This limits the utility of multicast to cases where an anycast type of function is desired, such as registrations.

OPEN ISSUE #7: This is a proposed resolution to whether or not multicast should be removed entirely.

Before a request is sent, the client transport MUST insert a value of the sent-by field into the Via header. This field contains an IP address or host name, and port. In certain cases discussed in Section 19.2.2, this IP address and port are used to construct a SIP URL for sending the response. The transport layer MUST be prepared to receive incoming connections (and receive responses sent over such connections) on any IP addresses and ports that this SIP URL might resolve to using the procedures defined in Section 24. The transport layer MUST also be prepared to receive an incoming connection on the source IP address that the request was sent from, and port number in the sent-by field. The client transport MUST also be prepared to receive the response on the same connection used to send the request.

For unreliable unicast transports, the client transport MUST be prepared to receive responses on the source IP address that the request is sent from (as responses are sent back to the source address), but the port number in the sent-by field. Furthermore, as with reliable transports, in certain cases the IP address and port are used to construct a URL for sending the response. The client transport MUST be prepared to receive responses on any IP address/port combinations that this SIP URL might resolve to using the procedures of Section 24.

For multicast, the client transport MUST be prepared to receive responses on the same multicast group and port that the request is sent to.

If a request is destined to an IP address, port, and transport to which an existing connection is open, it is RECOMMENDED that this connection be used to send the request, but another connection MAY be opened and used.

If a request is sent using multicast, it is sent to the group address, port, and TTL provided by the transport user. If a request is sent using unicast unreliable transports, it is sent to the IP address and port provided by the transport user.

19.1.2 Receiving Responses

When a response is received, the client transport examines the top Via header. If the value of the sent-by parameter in that header does not correspond to a value that the client transport is configured to insert into requests, the response MUST be rejected.

If there are any client transactions in existence, the client transport uses the matching procedures of Section 17.1.3 to attempt to match the response to an existing transaction. If there is a match, the response MUST be passed to that transaction. Otherwise, the response MUST be passed to the core (whether it be stateless proxy, stateful proxy, or UA) for further processing. Handling of these "stray" responses is dependent on the core (a stateless proxy will forward all responses, for example).

19.2 Servers

19.2.1 Receiving Requests

When the server transport receives a request over any transport, it MUST examine the value of the sent-by parameter in the top Via header field. If the host portion of the sent-by parameter contains a domain name, or if it contains an IP address that differs from the packet source address, the server MUST add a “received” attribute to that Via header field. This attribute MUST contain the source address that the packet was received from. This is to assist the server transport layer in sending the response, since it must be sent to the source IP address that the request came from.

Consider a request received by the server transport which looks like, in part:

```
INVITE sip:bob@Biloxi.com SIP/2.0
Via: SIP/2.0/UDP bobspc.biloxi.com:5060
```

The request is received with a source IP address of 1.2.3.4. Before passing the request up, the transport would add a received parameter, so that the request would look like, in part:

```
INVITE sip:bob@Biloxi.com SIP/2.0
Via: SIP/2.0/UDP bobspc.biloxi.com:5060
```

Next, the client transport attempts to match the request to the client transaction. It does so using the matching rules described in Section 17.2.3. If a matching server transaction is found, the request is passed to that transaction for processing. If no match is found, the request is passed to the core, which may decide to construct a new server transaction for that request.

19.2.2 Sending Responses

The server transport uses the value of the top Via header in order to determine where to send a response. It MUST follow the following process:

- If the “sent-protocol” is a reliable transport protocol such as TCP, TLS or SCTP, the response MUST be sent using the existing connection to the source of the original request that created the transaction, if that connection is still open. This does require the server transport to maintain an association between server transactions and transport connections. If that connection is no longer open, the server MAY open a connection to the IP address in the received parameter, if present, using the port in the sent-by value, or the default port for that transport, if no port is specified (5060 for UDP and TCP, 5061 for TLS and SSL). If that connection attempt fails, the server SHOULD construct a SIP URL of the form “sip:¡sent-by host¿;transport=¡sent-protocol¿” and then use the procedures defined in Section 24 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 (for unreliable unicast transports), if the top Via has a received parameter, 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 Section 24 to send the response.

- Otherwise, if it is not receiver-tagged, send the response to the address indicated by the "sent-by" value.

19.3 Framing

In the case of message oriented transports (such as UDP), if the message has a Content-Length header, the message body is assumed to contain that many bytes. If there are additional bytes in the transport packet below the end of the body, they MUST be discarded. If the transport packet ends before the end of the message body, this is considered an error. If the message is a response, it MUST be discarded. If its a request, the element SHOULD generate a 400 class response. If the message has no Content-Length header, the message body is assumed to end at the end of the transport packet.

In the case of stream oriented transports (such as TCP), the Content-Length header indicates the size of the body. The Content-Length header MUST be used with stream oriented transports.

19.4 Error Handling

Error handling is independent of whether the message was a request or response.

If the transport user asks for a message to be sent over an unreliable transport, and the result is an ICMP error, the behavior depends on the type of ICMP error. A host, network, port or protocol unreachable errors, or parameter problem errors SHOULD cause the transport layer to inform the transport user of a failure in sending. Source quench and TTL exceeded ICMP errors SHOULD be ignored.

If the transport user asks for a request to be sent over a reliable transport, and the result is a connection failure, the transport layer SHOULD inform the transport user of a failure in sending.

20 Security Considerations

The fundamental security issues confronting SIP are: preserving the confidentiality and integrity of messaging, preventing replay attacks or message spoofing, ensuring the privacy of the participants in a session, and preventing denial of service attacks.

SIP messages frequently contain sensitive information about their senders not just what they have to say, but with whom they communicate, when they communicate and for how long, and from where they participate in sessions. Many applications and their users require that this sort of private information be hidden from any parties that do not need to know it.

Encryption provides the best means to preserve the confidentiality of signaling it can also guarantee that messages are not modified by any malicious intermediaries. However, SIP requests and responses cannot be encrypted end-to-end (that is, between a pair of distinct user agents who share encryption keys) in their entirety because message fields such as the Request-URI, Route and Via need, in most network architectures, to be visible to proxies so that SIP requests are routed correctly. Note that proxy servers need to modify signaling as well (adding Via headers) in order for SIP to function. Proxy servers must therefore be a part of trust relationships in SIP networks.

Note that there are also less direct ways in which private information can be divulged. If a user or service chooses to be reachable at an address that is guessable from the person's name and organizational affiliation (which describes most addresses of record), the traditional method of ensuring privacy by having an unlisted "phone number" is compromised. A user location service can infringe on the privacy of the recipient of a session invitation by divulging their specific whereabouts to the caller; an implementation consequently 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.

SIP entities also have a need to identify one another in a secure fashion. Ordinarily a SIP UA asserts an identity for the initiator of a request in the From header field, but in many systems this information is controlled directly by the end user, and thus spoofing the contents of the From is trivial. When a SIP endpoint asserts the identity of its user to a peer user agent or to a proxy server, that identity should in some way be verifiable. A cryptographic authentication mechanism is provided in SIP to address this requirement.

The most comprehensive mechanisms for securing SIP messages (providing confidentiality and integrity guarantees for signaling as well as authentication) make use of transport or network layer encryption. encryption encrypts the entire SIP request or response on the wire so that packet sniffers or other eavesdroppers cannot see who is calling whom.

Note that the security of SIP signaling itself has no bearing on the security of protocols used in concert with SIP such as RTP, or with any MIME types carried as SIP bodies, such as SDP. Any media associated with a session can be encrypted end-to-end without any of the problems associated with encrypting SIP signaling. Media encryption is outside the scope of this document.

20.1 Transport and Network Layer Security

SIP requests and responses MAY be protected by security mechanisms at the transport or network layer. No particular mechanism is recommended by this document, but two popular alternatives are briefly examined: protection at the transport layer can be afforded by TLS [25], and network layer security is provided by IPSec [26].

Transport or network layer security encrypts signaling traffic, guaranteeing message confidentiality and integrity (note however that the originator and recipient of a session may be deducible by observers performing a network traffic analysis). The keys used to establish encrypt traffic can also be used to verify an asserted identity in many architectures, and therefore provide a means of authentication.

IPSec is a network layer protocol essentially, a secure replacement for traditional IP (Internet Protocol). IPSec is most suited to VPN (virtual private network) architectures in which a set of SIP hosts (mingled user agents and proxy servers) or bridged administrative domains have a trust relationship with one another.

TLS is a transport protocol and hence, like TCP and UDP, TLS can be specified as the desired transport protocol within a Via header field or a SIP-URI. TLS is most suited to architectures in which a chain of trust joins together a set of hosts (e.g. Alice trusts her local proxy server, which in turn trust Bob's local proxy server, which Bob trusts, hence Bob and Alice can communicate securely).

TLS must be tightly coupled with a SIP application. Note that transport mechanisms are specified on a hop-by-hop basis in SIP, and that in some networks TLS might be used for only certain portions of the signaling path.

It is RECOMMENDED that SIP endpoints support TLS as a secure transport for SIP.

20.2 SIP Authentication

SIP provides a stateless challenged-based mechanism for authentication. Any time that a proxy server or user agent receives a request, they MAY challenge the initiator of the request to provide assurance of their identity. Once the originator has been identified, the recipient of the request SHOULD ascertain whether or not this user is authorized to make the request in question. No authorization systems are recommended or discussed in this document.

The “basic” and “digest” authentication mechanisms described in this section provide message authentication only, without message integrity or confidentiality. Protective measures above and beyond authentication need to be taken to prevent active attackers from modifying and/or replaying SIP requests and responses.

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

20.2.1 Framework

The framework for SIP authentication closely parallels that of HTTP (RFC 2617 [27]). 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 identity of a user agent client. Additionally, registrars and redirect servers MAY make use of 401 (Unauthorized) responses for authentication, but proxies MUST NOT, and instead MAY use the 407 (Proxy Authentication Required) response. The requirements for inclusion of the Proxy-Authenticate, Proxy- Authorization, WWW-Authenticate, and Authorization in the various messages are identical to those described in RFC 2617 [27].

Since SIP does not have the concept of a canonical root URL, the notion of protection spaces is interpreted differently in 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:bob@biloxi.com SIP/2.0
WWW-Authenticate:  Basic realm="business"
```

and

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

Generally, SIP authentication is for a specific request 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 (e.g. an INVITE to ’sip:10.3.6.6’). 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.

20.2.2 User to User Authentication

When a UAS receives a request from a UAC, the UAS MAY authenticate the originator before the request is processed. If no credentials (in the Authorization header field are provided in the request, the UAS can challenge the originator to provide credentials by rejecting the request with a 401 (Unauthorized) status code.

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.

An example of the WWW-Authenticate in a 401 challenge is:

```
WWW-Authenticate:  Basic realm="business"
```

When the originating UAC receives the 401 it SHOULD, if it is able, re-originate the request with the proper credentials. The UAC may require input from the originating user before proceeding. The content of the "realm" parameter of the WWW-Authenticate header SHOULD be displayed to the user. Once authentication credentials have been supplied (either directly by the user, or discovered in a keyring), user agents SHOULD cache the credentials for a given value of the Request-URI and "realm" and attempt to re-use these values on the next request for that destination.

Any 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.

An example of the Authorization header is:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

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.

20.2.3 Proxy to User Authentication

Similarly, when a UAC sends a request to a proxy server, the proxy server MAY authenticate the originator before the request is processed. If no credentials (in the Proxy-Authorization header field) are provided in the request, the UAS can challenge the originator to provide credentials by rejecting the request with a 407 (Proxy Authentication Required) status code. The proxy MUST populate the 407 (Proxy Authentication Required) message with a Proxy- Authenticate header applicable to the proxy for the requested resource.

The use of the Proxy-Authentication and Proxy-Authorization parallel that described in [27, 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 UAC following the procedures for any other response. It is the client's responsibility to add the Proxy-Authorization header containing credentials for the realm of 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.

When the originating UAC receives the 407 it SHOULD, if it is able, re-originate the request with the proper credentials. It should follow the same procedures for the display of the "realm" parameter that are given above for responding to 401.

Any user agent that wishes to authenticate itself to a proxy server – usually, but not necessarily, after receiving a 407 response – MAY do so by including an Proxy-Authorization header field with the request. 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.

A Proxy-Authorization header field applies only to the proxy whose realm is identifier in the "realm" parameter (this proxy may previously have demanded authentication using the Proxy-Authenticate field). When multiple proxies are used in a chain, the Proxy-Authorization header field MUST NOT be consumed by any proxy whose realm does not match the "realm" parameter specified in the Proxy-Authorization header.

Note that if an authentication scheme is used in the Proxy- Authorization that does not support realms, a proxy server MUST attempt to parse all Proxy-Authorization headers to determine whether or not one of them has what it considers to be valid credentials. Because this is potentially very time consuming in large networks, proxy servers SHOULD use an authentication scheme that supports realms in the Proxy-Authorization header.

It is also possible that a 401 or 407 response will contain several challenges, from a mixture of proxies and user agent servers, if the request was forked. If at least one user agent responds to a request with a challenge, than a 401 should be used; otherwise a 407 should be used. When resubmitting its request in response to the challenge, the UAC needs to include an Authorization for each WWW-Authenticate and Proxy- Authorization for each Proxy-Authenticate.

See [H14.34] for a definition of the syntax of Proxy- Authentication and Proxy-Authorization.

20.2.4 Authentication Schemes

SIP implementations MAY use HTTP's basic and digest authentication mechanisms ([27]) to provide a rudimentary form of security. This section overviews usage of these mechanisms in SIP. The scheme usage is almost completely identical to that for HTTP [27]. This section outlines this operation, pointing to RFC 2617 ([27]) for details and noting the differences that arise when using SIP. Since RFC 2543 is based on HTTP basic and digest as defined in RFC 2069 [28], SIP servers supporting RFC 2617 MUST ensure they are backwards compatible with RFC 2069. Procedures for this backwards compatibility are specified in RFC 2617.

20.2.4.1 HTTP Basic The rules for basic authentication follow those defined in [27, 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 [27, 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.

20.2.4.2 HTTP Digest The rules for digest authentication follow those defined in [27, 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 26

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

5. The text in RFC 2617 [27] regarding cache operation does not apply to SIP.

6. RFC 2617 [27] 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 for whom that the server is willing to accept forwarded or direct calls.

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 [27] 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.

20.3 SIP Encryption

No mechanism is currently specified for encrypting entire SIP messages end-to-end for the purpose of confidentiality. This is a hard problem because network intermediaries (like proxy servers) need to view certain headers in order to route messages correctly, and if these intermediaries are excluded from security associations then SIP messages will essentially be unroutable.

That much said, SIP messages carry MIME bodies and the MIME standard includes mechanisms for securing MIME contents to ensure both integrity and confidentiality (including the 'multipart/encrypted' MIME type, see [29]), but detailed description of the use of secure MIME types are outside the scope of this document. Implementors should note, however, that there may be rare network intermediaries (not typical proxy servers) that rely on viewing or modifying the bodies of SIP messages (especially SDP), and that secure MIME may prevent these sorts of intermediaries from functioning.

This applies particularly to certain types of firewalls.

End-to-end encryption relies on keys shared by the two user agents involved in the request. Typically, the message is sent encrypted with the public key of the recipient, so that only that recipient can read the message. SIP does not define any mechanism for end-to-end key exchange.

Note that the PGP mechanism for encrypting the headers and bodies of SIP messages described in RFC2543 has been deprecated.

20.4 Denial of Service

Denial of service attacks focus on rendering a particular network element unavailable, usually by directing an excessive amount of network traffic at its interfaces. A distributed denial of service attack allows one network user to cause multiple network hosts to flood a target host with a large amount of network traffic.

In many architectures SIP proxy servers face the public Internet in order to accept requests from worldwide IP endpoints. When the host on which a SIP proxy server is operating is routable from the public Internet, it should be deployed in an administrative domain with secure routing policies (blocking source-routed traffic, preferably filtering ping traffic).

SIP creates a number of potential opportunities for distributed denial of service attacks that must be recognized and addressed by the implementors and operators of SIP systems.

Floods of messages directed at proxy servers can lock up proxy server resources and prevent desirable traffic from reaching its destination. There is a computational expense associated with processing a SIP transaction at a proxy server, and that expense is greater for stateful proxy servers that it is for stateless proxy servers. Therefore stateful proxies are more susceptible to flooding than stateless proxy servers.

Attackers can create bogus requests that contain a falsified Via header field which identifies a targeted host as the originator of the message and then send this message to a large number of SIP network elements, thereby using hapless SIP UAs or proxies to generate denial of service traffic aimed at the target.

Similarly, attackers might use falsified Route headers in a request that identify the target host and then send such messages to forking proxies that will amplify messaging sent to the target. Record-Route could be used to similar effect when the attacker is certain that the SIP dialog initiated by the request will result in numerous transactions originating in the backwards direction.

One could prevent one's host from being commandeered for such an attack by disallowing requests that do not make use of a persistent security association established through a transport or network layer security instrument such as TLS or IPsec. This could be an appropriate security solution for two proxy servers that trust one another and exchange significant amounts of signaling traffic with one another, or between a user agent and its outbound proxy.

Both TLS and IPSec can also make use of bastion hosts at the edges of administrative domains that participate in the security associations to aggregate secure tunnels and sockets. These bastion hosts can also take the brunt of denial of service attacks, ensuring that SIP hosts within the administrative domain are not encumbered with superfluous messaging.

If such a persistent security association is not feasible, user agents and proxy servers SHOULD challenge questionable requests with only a *single* 401 (Unauthorized) or 407 (Proxy Authentication Required) forgoing the normal response retransmission algorithm.

> Retransmitting the 401 or 407 status response amplifies the problem of an attacker using a falsified header (such as Via) to direct traffic to a third party.

A number of denial of service attacks open up if REGISTER requests are not properly authenticated and authorized by registrars. Attackers could de-register some or all users in an administrative domain, thereby preventing these users from being invited to new sessions. An attacker could also register a large number of contacts designating the same host for a given address of record in order to use the registrar and any associated proxy servers as amplifiers in a denial of service attack. Attackers might also attempt to deplete available memory and disk resources of a registrar by registering huge numbers of bindings.

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.

The use of multicast to transmit SIP requests can greatly increase the potential for denial of service attacks.

21 Common Message Components

There are certain components of SIP messages that appear in various places within SIP messages (and sometimes, outside of them), which merit separate discussion.

21.1 SIP Uniform Resource Locators

A SIP URL identifies a communications resource. Like all URLs, SIP URLs may be placed in web pages, email messages or printed literature. They contain sufficient information to initiate and maintain a communication session with the resource.

Examples of communications resources include

- a user of an online service
- an appearance on a multiline phone
- a mailbox on a messaging system
- a PSTN phone number at a gateway service
- a group (such as "sales" or "helpdesk") in an organization

21.1.1 SIP URL components

The "sip:" scheme follows the guidelines in RFC 2396 [9]. It uses a form similar to the mailto URL, allowing the specification of SIP request-header fields and the SIP message- body. This makes it possible to specify the subject, media type, or urgency of sessions initiated by using a URL on a web page or in an email message. The formal syntax for a SIP URL is presented in Section 26. Its general form is

sip:user:password@host:port;url-parameters?headers

These tokens, and some of the tokens in their expansion, have the following meanings.

user: The identifier of a particular resource at the host being addressed. Note that "host" as used here may, and frequently does, refer to a domain.

The "userpart" of a URL consists of this user field, the password field and the @ sign following them. The userpart of a URL is optional and MAY be absent when the destination host does not have a notion of users or when the host itself is the resource being identified. If the @ sign is present in a SIP URL, the user field MUST NOT be empty.

If the host being addressed is capable of processing telephone numbers, an Internet telephony gateway for instance, a telephone- subscriber field defined in RFC 2806 [13] MAY be used to populate the user field. There are special escaping rules for encoding telephone-subscriber fields in SIP URLs described in Section 21.1.2.

password: A password associated with the user

While the SIP URL syntax allows this field to be present, its use 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. For instance, transporting a PIN number in this field exposes the PIN.

host: The entity hosting the SIP resource

The host part contains either a fully-qualified domain name or numeric IPv4 or IPv6 address. Using the fully-qualified domain name form is RECOMMENDED whenever possible.

port: The port number where the request is to be sent.

URL parameters: Parameters affecting a request constructed from the URL.

URL parameters are added after the hostport component and are separated by semi-colons. This extensible mechanism includes the transport, maddr, ttl, user, and method parameters.

The transport parameter determines the transport mechanism to be used for sending SIP messages. SIP can use any network transport protocol. Parameter names are defined for UDP [30], TCP [31], TLS [25], and SCTP [32].

The maddr parameter indicates the server address to be contacted for this user, overriding any address derived from the host field. Section 24 describes the proper interpretation of the transport, maddr and hostport in order to obtain the destination address, port and transport for sending a request.

> The maddr field can be used as a simple form of loose source routing. It allows a URL to specify a specific proxy that must be traversed en-route to the destination. This capability is useful for a roaming user that is forced to use an outbound proxy, but wishes to force requests through their home proxy.

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 `alice@atlanta.com` using multicast to 239.255.255.1 with a ttl of 15, the following URL would be used:

```
sip:alice@atlanta.com;maddr=239.255.255.1;ttl=15
```

The set of valid telephone-subscriber strings is a subset of valid user strings. The user URL parameter exists to distinguish telephone numbers from user names that happen to look like telephone numbers. If the user string contains a telephone number formatted as a telephone-subscriber, the user parameter value "phone" SHOULD be present. 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.

The method of the SIP request constructed from the URL can be specified with the method parameter.

Since the url-parameter mechanism is extensible, SIP elements MUST silently ignore any url-parameters that they do not understand.

Headers: Headers to be included in a request constructed from the URL.

Headers fields in the SIP request can be specified with the "?" mechanism within a SIP URL. The header names and values are encoded in ampersand separated hname = hvalue pairs. The special hname "body" indicates that the associated hvalue is the message-body of the SIP request.

Table 1 summarizes the use of SIP URL components based on the context in which the URL appears. The external column describes URLs appearing anywhere outside of a SIP message, for instance on a web page or business card. Entries marked "m" are mandatory, those marked "o" are optional, and those marked "-" are not allowed. Elements processing URLs SHOULD ignore any disallowed components if they are present. The second column indicates the default value of an optional element if it is not present. "–" indicates that the element is either not optional, or has no default value.

SIP URLs in Contact header fields have different restrictions depending on the context in which the header field appears. One set applies to messages that establish and maintain dialogs (INVITE and its 200 OK response). The other applies to registration and redirection messages (REGISTER, its 200 OK response, and 3xx class responses to any method).

OPEN ISSUE #203: maddr is disallowed in To/From, but not port. Should port be disallowed?

OPEN ISSUE #204: Password is disallowed in From, but not To. Why?

OPEN ISSUE #205: Should we allow method and header URL components in registration/redirect Contacts. What do they mean?

	default	Req.-URI	To	From	reg./redir. Contact	dialog Contact/ R-R/Route	external
user	–	o	o	o	o	o	o
password	–	o	o	-	o	o	o
host	–	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	o	o
ttl-param	1	o	-	-	o	-	o
transp.-param	udp	o	-	-	o	o	o
other-param	–	o	o	o	o	o	o
headers	–	-	-	-	o	-	o

Table 1: Use and default values of URL components for SIP headers, Request-URI and references

21.1.2 Character escaping requirements

SIP follows the requirements and guidelines of RFC 2396 when defining the set of characters that must be escaped in a SIP URL, and uses its ""%" HEX HEX" mechanism for escaping. From RFC 2396:

> The set of characters actually 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. [9].

Excluded US-ASCII characters [9, 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.

For each component, the set of valid BNF expansions defines exactly which characters may appear unescaped. All other characters MUST be escaped.

For example, “@” is not in the set of characters in the user component, so the user “j@s0n” must have at least the @ sign encoded, as in “j%40s0n”.

Expanding the hname and hvalue tokens in Section 26 show that all URL reserved characters in header names and values MUST be escaped.

The telephone-subscriber subset of the user component has special escaping considerations. The set of characters not reserved in the RFC 2806 [13] description of telephone-subscriber contains a number of characters in various syntax elements that need to be escaped when used in SIP URLs. Any characters occurring in a telephone-subscriber that do not appear in an expansion of the BNF for the user rule MUST be escaped.

21.1.3 Example SIP URLs

```
sip:alice@atlanta.com
sip:alice:secretword@atlanta.com;transport=tcp
sip:alice@atlanta.com?subject=project%20x&priority=urgent
sip:+1-212-555-1212:1234@gateway.com;user=phone
sip:1212@gateway.com
sip:alice@10.1.1.1
sip:atlanta.com;method=REGISTER?to=alice%40atlanta.com
sip:alice;day=tuesday@atlanta.com
```

The last example URL above has a user field value of “alice;day=tuesday”. The escaping rules defined above allow a semicolon to appear unescaped in this field. Note, however, that for the purposes of this protocol, the field is opaque. The apparent structure in that value is only useful to the entity responsible for the resource.

21.1.4 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. (OPEN ISSUE #100 : There is a proposal to make only quoted string comparisons case-sensitive.)
- The ordering of parameters and headers is not significant in comparing SIP URLs.
- Characters other than those in the “reserved” and “unsafe” sets (see RFC 2396 [9]) 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.
- For two URLs to be equal, the user, password, host, and port components must match. A URL omitting the optional port component will match a URL explicitly declaring port 5060. A URL

omitting the user component will **not** match a URL that includes one. A URL omitting the password component will **not** match a URL that includes one.

- URL url-parameter components are compared as follows
  - Any url-parameter appearing in both URLs must match.
  - A user, transport, ttl, or method url-parameter appearing in only one URL must contain its default value or the URLs do not match.
  - All other url-parameters appearing in only one URL are ignored when comparing the URLs.
- URL header components are never ignored. Any present header component MUST be present in both URLs and match for the URLs to match. The matching rules are defined for each header in Section sec:header-fields.

The URLs within each of the following sets are equivalent:

```
sip:alice@%61tlanta.com:5060
sip:alice@AtLanTa.CoM;Transport=udp

sip:carol@chicago.com
sip:carol@chicago.com;newparam=5
sip:carol@chicago.com;security=on

sip:biloxi.com;transport=tcp;method=REGISTER?to=sip:bob%40biloxi.com
sip:biloxi.com;method=REGISTER;transport=tcp?to=sip:bob%40biloxi.com

sip:alice@atlanta.com?subject=project%20x&priority=urgent
sip:alice@atlanta.com?priority=urgent&subject=project%20x
```

The URLs within each of the following sets are **not** equivalent:

```
SIP:ALICE@AtLanTa.CoM;Transport=udp               (different usernames)
sip:alice@AtLanTa.CoM;Transport=UDP

sip:bob@biloxi.com                       (different port and transport)
sip:bob@biloxi.com:6000;transport=tcp

sip:carol@chicago.com                      (different header component)
sip:carol@chicago.com?Subject=next%20meeting

sip:bob@phone21.boxesbybob.com     (even though that's what
sip:bob@10.4.1.4                    phone21.boxesbybob.com resolves to)
```

Note that equality is not transitive:

sip:carol@chicago.com and sip:carol@chicago.com;security=on are equivalent and sip:carol@chicago.com and sip:carol@chicago.com;security=off are equivalent

But sip:carol@chicago.com;security=on and sip:carol@chicago.com;security=off are **not** equivalent

Comparing URLs is a major part of comparing several SIP headers (see Section 22).

21.2 Option Tags

Option tags are unique identifiers used to designate new options (extensions) in SIP. These tags are used in Require (Section 22.30), Proxy-Require (Section 22.28, Supported (Section 22.35) and Unsupported (Section 22.38) header fields. Note that these options appear as parameters in those headers in an option-tag = token form (see Section 26 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) (See Section 27).

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.

21.3 Tags

The “tag” parameter is used in the To and From fields of SIP messages. It serves as a general mechanism to identify a particular instance of a user agent for a particular SIP URI.

As proxies can fork requests, the same request can reach multiple instances of a user (mobile and home phones, for example). Since each can respond, there needs to be a means for the originator of a session to distinguish the responses. Tag fields in the To and From disambiguate these multiple instances of the same user.

This situation also arises with multicast requests.

When a tag is generated by a UA for insertion into a request or response, it MUST be globally unique and cryptographically random with at least 32 bits of randomness. A property of this selection requirement is that a UA will place a different tag into the From header of an INVITE as it would place into the To header of the response to the same INVITE. This is needed in order for a UA to invite itself to a session, a common case for “hairpinning” of calls in PSTN gateways.

Besides the requirement for global uniqueness, the algorithm for generating a tag is implementation specific. Tags are helpful in fault tolerant systems, where a dialog is to be recovered on an alternate server after a failure. A UAS can select the tag in such a way that a backup can recognize a request as part of a dialog on the failed server, and therefore determine that it should attempt to recover the dialog and any other state associated with it.

22 Header Fields

The general syntax for header fields is covered in Section 7.3. This section lists the full set of header fields along with notes on syntax, meaning, and usage. Throughout this section, we use [HX.Y] to refer to Section X.Y of the current HTTP/1.1 specification RFC 2617 [27]. Examples of each header field are given.

Information about header fields in relation to methods and proxy processing is summarized in Tables 2 and 3.

The “where” column describes the request and response types in which the header field can be used. Values in this column are:

R: refers to header fields that can be used in requests.

r: designates a 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.

c: indicates a header field is copied from the request to the response.

The “proxy” column describes the operations a proxy may perform on a header.

c: indicates that a proxy can add (concatenate) comma-separated elements to the header

m: indicates that a proxy can modify the header

a: indicates that a proxy can add the header if not present

r: indicates that a proxy must be be able to read the header. Headers that need to be read cannot be encrypted.

The next six columns relate to the presence of a header field in a method, with the contents indicating:

o: for optional

m: for mandatory

m*: indicates a header that SHOULD be sent, but servers need to be prepared to receive messages without that header field.

***:** indicates that the header fields are required if the message body is not empty. See sections 22.14, 22.15 and 7.4 for details.

-: for not applicable.

“Optional” means thata 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 (The exception to this rule is the Require header field discussed in 22.30). A “mandatory” 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 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.

A compact form of some common header fields is also defined for use when overall message size is an issue.

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.

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG
Accept	R		-	o	-	m*	o	o
Accept	2xx		-	-	-	m*	o	o
Accept	415		-	o	-	o	o	o
Accept-Encoding	R		-	o	-	m*	o	o
Accept-Encoding	2xx		-	-	-	m*	o	o
Accept-Encoding	415		-	o	-	o	o	o
Accept-Language	R		-	o	-	m*	o	o
Accept-Language	2xx		-	-	-	m*	o	o
Accept-Language	415		-	o	-	o	o	o
Alert-Info	R	am	-	-	-	o	-	-
Alert-Info	180	am	-	-	-	o	-	-
Allow	R		o	o	o	o	o	o
Allow	2xx		-	o	o	m*	m*	o
Allow	r		-	o	o	o	o	o
Allow	405		-	m	m	m	m	m
Authentication-Info	2xx		-	o	-	o	o	o
Authorization	R		o	o	o	o	o	o
Call-ID	c	r	m	m	m	m	m	m
Call-Info		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			o	o	-	o	o	o
Content-Encoding			o	o	-	o	o	o
Content-Language			o	o	-	o	o	o
Content-Length		r	m*	m*	m*	m*	m*	m*
Content-Type			*	*	-	*	*	*
CSeq	c	r	m	m	m	m	m	m
Date		a	o	o	o	o	o	o
Error-Info	300-699		-	o	o	o	o	o
Expires			-	-	-	o	-	o
From	c	r	m	m	m	m	m	m
In-Reply-To	R		-	-	-	o	-	-
Max-Forwards	R	rm	o	o	o	o	o	o
MIME-Version			o	o	o	o	o	o
Organization		am	-	-	-	o	o	o

Table 2: Summary of header fields, A–O

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

Table 3: Summary of header fields, P–Z; (1): copied with possible addition of tag

22.1 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`.

Example:

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

22.2 Accept-Encoding

The Accept-Encoding 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].

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 this header is not present, the default value is identity. This differs slightly from the HTTP definition, which indicates that when not present, any encoding can be used, but the identity encoding is preferred.

Example:

```
Accept-Encoding: gzip
```

22.3 Accept-Language

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

The Accept-Language header is used in requests to indicate the preferred languages for reason phrases, session descriptions or status responses carried as message bodies in the response. If no Accept-Language header field is present in a request, the server assumes all languages are acceptable to the client.

Example:

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

22.4 Alert-Info

When present in an INVITE request, the Alert-Info header field specifies an alternative ring tone to the UAS. When present in a 180 (Ringing) response, the Alert-Info header field specifies an alternative ringback tone to the UAC. A typical usage is for a proxy to insert this header to provide a distinctive ring feature.

The Alert-Info header can introduce security risks. These risks, and the ways to handle them, are discussed in Section 22.9 which discusses the Call-Info header, as the risks are identical.

In addition, a user SHOULD be able to disable this feature selectively.

> This helps prevent disruptions that could result from the use of this header by untrusted elements.

Example:

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

22.5 Allow

The Allow header field lists the set of methods supported by the user agent generating the message.

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.

Example:

```
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE
```

22.6 Authentication-Info

The Authentication-Info header provides for mutual authentication with HTTP Digest. A UAS MAY include this header in a 2xx response to a request that was successfully authenticated using digest based on the Authorization header.

Syntax and semantics follow those specified in RFC2617 [27].

Example:

```
Authentication-Info: nextnonce="47364c23432d2e131a5fb210812c"
```

22.7 Authorization

The Authorization header field contains authentication credentials of a UA. Section 20.2.2 overviews the use of the Authorization header field, and Section 20.2.4 describes the syntax and semantics when used with HTTP Basic and Digest authentication.

Note that this header field, along with Proxy-Authorization breaks the general rules about multiple header fields. Although not a comma-separated list, this header field may be present multiple times, and MUST NOT be combined into a single header using the usual rules described in Section 7.3.

Example:

```
Authorization: Digest username="Alice", realm="Bob's Friends",
 nonce="84a4cc6f3082121f32b42a2187831a9e",
 response="7587245234b3434cc3412213e5f113a5432"
```

22.8 Call-ID

The Call-ID 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.Call-IDs are case- sensitive and are simply compared byte-by-byte.

The compact form of the Call-IDheader field is i.

Examples:

```
Call-ID: f81d4fae-7dec-11d0-a765-00a0c91e6bf6@biloxi.com
i:f81d4fae-7dec-11d0-a765-00a0c91e6bf6@10.4.1.4
```

22.9 Call-Info

The Call-Info 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 [33] or LDIF [34] formats). Additonal tokens can be registered using IANA and the procedures in Section 27.

Usage of the Call-Info header can pose a security risk. If a callee fetches the URLs provided by an malicious caller, the callee may be at risk for displaying inappropriate or offensive content, dangerous or illegal content, and so on. Therefore, it is RECOMMENDED that a UA only render the information in the Call-Info header if it can verify the authenticity of the element which originated the header, and trusts that element. This need not be the peer UA; a proxy can insert this header into requests.

The use of this header is important in converged applications.

Example:

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

22.10 Contact

The Contact header field provides a URL whose meaning depends on the the type of request or response it is in.

Parameters defined for Contact include “q” and “expires”. Additional parameters may be defined in other specifications.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.

The compact form of the Contact header field is m (for ”moved”).

Examples:

```
Contact: "Mr. Watson" <sip:watson@worcester.bell-telephone.com>
   ;q=0.7; expires=3600,
   "Mr. Watson" <mailto:watson@bell-telephone.com> ;q=0.1
m: <sip:bob@10.5.1.5>
```

22.11 Content-Disposition

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 [35]).

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. The parameter has defined values of “optional” and “required”. 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.

Example:

```
Content-Disposition: session
```

22.12 Content-Encoding

The Content-Encoding 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. 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.

The compact form of the Content-Encoding header field is e.

Examples:

```
Content-Encoding: gzip
e: tar
```

22.13 Content-Language

See [H14.12].

Example:

```
Content-Language: fr
```

22.14 Content-Length

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

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.

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

The short form of the header is l.

Examples:

```
Content-Length: 349
l: 173
```

22.15 Content-Type

The Content-Type 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).

The short form of the header is c.

Examples:

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

22.16 CSeq

A CSeq header field in a request contains a single decimal sequence number and the request method. The sequence number MUST be expressible as a 32-bit unsigned integer. The CSeq header serves to order transactions within a dialog, and to provide a means to uniquely identify transactions, and to differentiate between new requests and request retransmissions.

Example:

```
CSeq: 4711 INVITE
```

22.17 Date

The Date header field contains an RFC 1123 date (see [H14.18]). Note that unlike HTTP/1.1, SIP only supports the most recent RFC 1123 [36] 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.

> 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.

Example:

```
Date: Sat, 13 Nov 2001 23:29:00 GMT
```

22.18 Error-Info

The Error-Info header field provides a pointer to additional information about the error status response.

> SIP UACs have user interface capabilities ranging from pop up windows and audio on PC softclients to audio-only on "black" phones or endpoints connected via gateways. Rather than forcing a server generating an error to choose between sending an error status code with a detailed reason phrase and playing an audio recording, the Error-Info header field allows both to be sent. The UAC then has the choice of which error indicator to render to the caller.

A UAC MAY treat a SIP URL in an Error-Info header field as if it were a Contact in a redirect and generate a new INVITE, resulting an a recorded announcement session being established. A non-SIP URL MAY be rendered to the user.

Examples:

```
SIP/2.0 404 The number you have dialed is not in service
Error-Info: <sip:not-in-service-recording@atlanta.com>
```

22.19 Expires

The Expires header field gives the date and time after which the message (or content) expires. The precise meaning of this is method dependent.

Note that the expiration time in an INVITE 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 date (see the Date header field) 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.

Examples:

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

22.20 From

The From header field indicates the initiator of the request. (Note that this may be different from the initiator of the dialog. Requests sent by the callee to the caller use the callee's address in the From header field.)

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.

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 7.3.1.

The short form of the header is f.

Examples:

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

22.21 In-Reply-To

The In-Reply-To header field enumerates the Call-IDs that this call references or returns. These Call-IDs may have been cached by the client then included in this header in a return call.

> 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.

Example:

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

22.22 Max-Forwards

The Max-Forwards 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.

The Max-Forwards value is a decimal integer indicating the remaining number of times this request message is allowed to be forwarded. This count is decremented by each server that forwards the request.

Example:

```
Max-Forwards: 6
```

22.23 MIME-Version

See [H19.4.1].

Example:

```
MIME-Version: 1.0
```

22.24 Organization

The Organization header field conveys the name of the organization to which the entity issuing the request or response belongs.

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

Example:

```
Organization: Boxes by Bob
```

22.25 Priority

The Priority header field indicates the urgency of the request as perceived by the client. Defined values include “non-urgent”, “normal”, “urgent”, and “emergency”.

It is RECOMMENDED that the value of “emergency” only be used when life, limb or property are in imminent danger. Otherwise, there are no semantics defined for this header field.

> These are the values of RFC 2076 [37], with the addition of “emergency”.

Examples:

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

or

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

22.26 Proxy-Authenticate

The Proxy-Authenticate header field consists of a challenge that indicates the authentication scheme and parameters applicable to the proxy for this Request-URI.

The syntax for this header and use is defined in [H14.33]. See 20.2.3 for further details on its usage. Example:

```
   Proxy-Authenticate: Digest realm="Carrier SIP",
    domain="sip:ss1.carrier.com",
    nonce="f84f1cec41e6cbe5aea9c8e88d359",
    opaque="", stale=FALSE, algorithm=MD5
```

22.27 Proxy-Authorization

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

See [H14.34] for a definition of the syntax, and section 20.2.3 for a discussion of its usage.

Note that this header field, along with Authorization breaks the general rules about multiple header fields. Although not a comma-separated list, this header field may be present multiple times, and MUST NOT be combined into a single header using the usual rules described in Section 7.3.1.

Example:

```
Proxy-Authorization: Digest username="Alice", realm="Atlanta ISP",
   nonce="c60f3082ee1212b402a21831ae",
   response="245f23415f11432b3434341c022"
```

22.28 Proxy-Require

The Proxy-Require header field is used to indicate proxy-sensitive features that must be supported by the proxy. See Section 22.30 for more details on the mechanics of this message and a usage example.

Example:

```
  Proxy-Require: foo
```

22.29 Record-Route

The Record-Route is inserted by proxies in a request to force future requests in the session to route through the proxy.

Details of its use with the Route header field are described in Section 16.4.

Example:

```
  Record-Route: <sip:bob@biloxi.com;maddr=10.1.1.1>,
   <sip:bob@biloxi.com;maddr=10.2.1.1>
```

22.30 Require

The Require 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. Although an optional header, the Require MUST NOT be ignored if it is present.

> 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.

Example:

```
Require: com.example.billing
```

22.31 Retry-After

The Retry-After 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.

Examples:

```
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.

22.32 Route

The Route is used to force routing for a request through the listed set of proxies. Details of its use with the Record-Route header field are described in Section 13.

Example:

```
Route: <sip:bob@biloxi.com;maddr=10.1.1.1>, <sip:bob@10.4.1.4>
```

22.33 Server

The Server 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].

Example:

```
Server: HomeProxy v2
```

22.34 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.)

The short form of the header is s.

Example:

```
Subject: Need more boxes
s: Tech Support
```

22.35 Supported

The Supported header field enumerates all the extensions upported by the client or server. If empty, it means that no extensions are supported.

Example:

```
Supported: foo, bar
```

22.36 Timestamp

The Timestamp header field describes when the client sent the request to the server. The use of the Timestamp is covered in Section 13.

Example:

```
Timestamp: 54
```

22.37 To

The To header field specifies the logical recipient of the request.

The optional “display-name” is meant to be rendered by a human-user interface. The “tag” parameter serves as a general mechanism to distinguish multiple instances of a user identified by a single SIP URL.

See Section 13 for details of the “tag” parameter.

Section 22.20 describes how To and From header fields are compared for the purpose of matching requests to dialogs. 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 short form of the header is t.

The following are examples of valid To headers:

```
To: The Operator <sip:operator@cs.columbia.edu>;tag=287447
t: sip:+12125551212@server.phone2net.com
```

22.38 Unsupported

The Unsupported header field lists the features not supported by the server. See Section 22.30 for a usage example and motivation.

Example:

```
Unsupported: foo
```

22.39 User-Agent

The User-Agent header field contains information about the client user agent originating the request. The syntax and semantics are defined in [H14.43].

Example:

```
User-Agent: Softphone Beta1.5
```

22.40 Via

The Via field indicates the path taken by the request so far and indicate the path that should be followed in routing responses.

The Via header field contains 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. The Via header field can also contains parameters such as "maddr", "ttl", "received", and "branch"whose meaning and use are described in other sections.

The short form of the header is v.

Example:

```
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.

Another example:

```
Via: SIP/2.0/UDP first.example.com:4000;ttl=16
  ;maddr=224.2.0.1 ;branch=a7c6a8dlze.1
```

22.41 Warning

The Warning header field is used to carry additional information about the status of a response. Warning headers are sent with responses and contain a three digit warning code, host name, and warning text.

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].

The first digit of warning codes beginning with "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'"
```

22.42 WWW-Authenticate

The WWW-Authenticate header field consists of a challenge that indicates the authentication scheme and parameters applicable for this Request-URI.

The syntax for this header and use is defined in [H14.47]. See 20.2.2 for further details on its usage.

Example:

```
WWW-Authenticate: Digest realm="Bob's Friends",
  domain="sip:boxesbybob.com",
  nonce="f84f1cec41e6cbe5aea9c8e88d359",
  opaque="", stale=FALSE, algorithm=MD5
```

23 Response Codes

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.

23.1 Provisional 1xx

Provisional responses indicate that the server or proxy contacted is performing some further action and does not yet have a definitive response. A server typically sends a 1xx response if it expects to takemore than 200 ms to obtain a final response. Note that 1xx responses are not transmitted reliably, that is, they do not cause the client to send an ACK.

Provisional (1xx) responses MAY contain message bodies, including session descriptions.

Provisional responses are also known as informational responses.

23.1.1 100 Trying

This response indicates that the request has been received by the next hop server and that some unspecified action is being taken on behalf of this call (e.g., a database is being consulted). This response stops retransmissions of an INVITE by a UAC.

23.1.2 180 Ringing

The user agent receiving the INVITE is trying to alert the user. This response MAY be used to initiate local ringback.

23.1.3 181 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.

23.1.4 182 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.

23.1.5 183 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, header fields, or message body MAY be used to convey more details about the call progress.

23.2 Successful 2xx

The request was successful.

23.2.1 200 OK

The request has succeeded. The information returned with the response depends on the method used in the request.

23.3 Redirection 3xx

3xx responses give information about the user's new location, or about alternative services that might be able to satisfy the call.

23.3.1 300 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 MAY include a message body 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 choices SHOULD also be listed as Contact fields (Section 22.10). 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.

23.3.2 301 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 22.10). 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.

23.3.3 302 Moved Temporarily

The requesting client SHOULD retry the request at the new address(es) given by the Contact header field (Section 22.10). 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. In the second approach, UAs MAY also use the Contact information for the To header field, as well as a new Call-ID value.

The duration of the redirection can be indicated through an Expires (Section 22.19) header. If there is no explicit expiration time, the address is only valid for this call and MUST NOT be cached for future calls.

23.3.4 305 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.

23.3.5 380 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.

23.4 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.

23.4.1 400 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”.

23.4.2 401 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.

23.4.3 402 Payment Required

Reserved for future use.

23.4.4 403 Forbidden

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

23.4.5 404 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.

23.4.6 405 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.

23.4.7 406 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.

23.4.8 407 Proxy Authentication Required

This code is similar to 401 (Unauthorized), but indicates that the client MUST first authenticate itself with the proxy. SIP access authentication is explained in section 20 and 20.2.3.

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

23.4.9 408 Request Timeout

The server could not produce a response within a suitable amount of time, for example, if it could not determine the location of the user in time. The client MAY repeat the request without modifications at any later time.

23.4.10 410 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.

23.4.11 413 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.

23.4.12 414 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.

23.4.13 415 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. UAC processing of this response is described in Section 8.1.3.4.

23.4.14 420 Bad Extension

The server did not understand the protocol extension specified in a Proxy-Require (Section 22.28) or Require (Section 22.30) header field. The server SHOULD include a list of the unsupported extensions in an Unsupported header in the response. UAC processing of this response is described in Section 8.1.3.4.

23.4.15 421 Extension Required

The UAS needs a particular extension to process the request, but this extension is not listed in a Supported header in the request. Responses with this status code MUST contain a Require header listing the required extensions.

In general, a UAS SHOULD NOT use this response when it wishes to apply an extension to a request. The end result will often be no service at all, and a break in interoperability. Rather, servers SHOULD process the request using baseline SIP capabilities and any extensions supported by the client.

23.4.16 480 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). 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.

23.4.17 481 Call/Transaction Does Not Exist

This status indicates that the UAS received a request that does not match any existing dialog or transaction.

23.4.18 482 Loop Detected

The server has detected a loop (Section 3).

23.4.19 483 Too Many Hops

The server received a request that contains a Max-Forwards (Section 22.22) header with the value zero.

23.4.20 484 Address Incomplete

The server received a request with a 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.

23.4.21 485 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.

23.4.22 486 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. Status 600 (Busy Everywhere) SHOULD be used if the client knows that no other end system will be able to accept this call.

23.4.23 487 Request Terminated

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

23.4.24 488 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.

23.5 Server Failure 5xx

5xx responses are failure responses given when a server itself has erred.

23.5.1 500 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.

23.5.2 501 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.)

23.5.3 502 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.

23.5.4 503 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.

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.

23.5.5 504 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.

23.5.6 505 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.

23.5.7 513 Message Too Large

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

23.6 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.

23.6.1 600 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.

23.6.2 603 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.

23.6.3 604 Does Not Exist Anywhere

The server has authoritative information that the user indicated in the Request-URI does not exist anywhere.

23.6.4 606 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 22.41. It is hoped that negotiation will not frequently be needed, and when a new user is being invited to join an already existing conference, negotiation may not be possible. It is up to the invitation initiator to decide whether or not to act on a 606 (Not Acceptable) response.

24 Locating a SIP Server

NOTE: Usage of SRV records is still under discussion with IESG, and therefore this section is likely to change in subsequent versions of bis.

The SIP URI provides a way to identify a communications resource. For this URI to be useful in a SIP element, a mechanism is necessary to take this URI and determine the IP address, port, and transport of one or more servers that message destined for this URI should be sent to. We refer to the combination of an IP address, port, and transport as a *next hop*. There are two ways to determine the next hop. The next hop can be configured to be the same for all URIs. In this case, the next hop is referred to as a *outbound proxy*. This is commonly used in a user agent which is required to send all requests to a specific server for policy processing or firewall traversal, for example. The outbound proxy can be configured by any mechanism, including DHCP [39].

When the next hop is not configured, a mechanism is needed to determine one or more next hops from the URI. Section 24.1 provides an algorithm which can be used to determine an ordered list of next hops. Typically, the URI that is used is from the Request-URI of a request, in order to determine where to send that request. However, in certain circumstances (which are documented in Section 19.2.2), a URI may have been extracted from a response in order to determine where to send the response.

Once the ordered list of next hops is computed, they are used according to the procedures of Section 24.2.

24.1 Computing the List of Next Hops

The algorithm for computing the list of next hops begins by setting three variables. The first variable is called the *target address*. The target address MUST be set to the contents of the maddr parameter of the URI, if present. If not present, it MUST be set to the host element of the URI. The next variable is called the *target port*. The target port MUST be set to the port element of the URI if present, else the target port MUST remain empty. The target transport MUST be set to the headertransport element of the URI if present, else the target transport MUST remain empty.

The algorithm begins by examining the target address. If it contains a numeric IP address, the procedures of Section 24.1.1 MUST be followed. Otherwise, the target transport is examined. If it is empty, and the target port is either empty or contains a value of 5060, the procedures of Section 24.1.2 MUST be followed. If the target transport is not empty, and the target port is empty, the procedures of Section 24.1.2 MUST be followed if the target transport is UDP. If the target transport and target port are not empty, but the target port contains the default port for the target transport (5060 for UDP, TCP, and SCTP, 5061 for TLS), the procedures of Section 24.1.2 MUST also be followed. Otherwise, the procedures of Section 24.1.3 MUST be followed. Effectively, this case occurs when the target port and target transport don't "match", taking into account their defaults if empty.

24.1.1 Numeric Destination Address

The addresses of the next hops are all the same, and MUST be equal to the value of the target address.

If the target transport is specified, and the element supports that transport, there is only a single next hop, using the target transport. If the target transport is not specified, the number of next hops is equal to the number of transports the element supports. The first next hop MUST be UDP, and the ordering of the remaining transports is at the discretion of the element.

For each next hop, the port number is equal to the target port, if specified, otherwise the default port for that transport of that next hop.

For example, consider the SIP URI `sip:joe@1.2.3.4` present in the Request-URI of a request. A UAC wishes to use this URI to determine the set of next hops. The UAC supports UDP and TLS. It applies the algorithm in this section, and ends up with the following ordered list of IP address, port, transport:

```
{1.2.3.4, 5060, UDP}
{1.2.3.4, 5061, TLS}
```

24.1.2 SRV Resolution of Host Name

DNS SRV records are retrieved according to RFC 2782 [40]. The service identifier for DNS SRV records is "_sip". If the target transport is not empty, only records for that transport are retrieved. (If the element does not support the transport specified, the lookup fails.) If the target transport is empty, the element retrieves records for all transport protocols it supports. The results of all queries are merged and then sorted according to priority, independent of the transport protocol. If this list is empty, follow the procedure in Section 24.1.3.

Note that the behavior above differs slightly from that described in RFC 2782. There, A records are consulted if the query for one transport protocol fails; here, we only abandon the SRV lookup if none of the transport protocols supported by the client yield an answer.

Clients MUST NOT cache query results except according to the rules in RFC 1035 [41].

24.1.3 Address Record Resolution of Host Name

When the target address is not a numeric IP, and there is a target port which does not match the default port for the target transport, SRV records are not used. This is because SRV will normally provide ports, so if one is provided that is not a default, this would seem to imply the the URL is trying to explicitly identify the destination, rather than using SRV.

In this case, 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.

The DNS address records are kept sorted in the order returned by the DNS server. For each address, the port is set to the target port. For each address, the transport is set to the target transport if not empty, otherwise, the target transport MUST be UDP for the first address, and is at the discretion of the implementation for the others.

> OPEN ISSUE #221: Selection of transports for the case when multiple A records are returned requires more work.

Clients MUST NOT cache query results except according to the rules in RFC 1035 [41].

24.2 Contacting the Next Hops

The algorithms of the previous section will result in an ordered list of next hops. This section describes how that list is used.

If the ordered list was obtained through SRV, servers are contacted as specified in the "Usage rules" section of RFC 2782 [40], which describes procedures for using the weight field to randomly select servers amongst those of equal priority.

The SIP element takes the ordered list, and it tries to contact each next hop in turn, until a server responds. If contacting a next hop results in a failure, as defined in the next paragraph, the element moves to the next next hop in the list, until the list is exhausted. If the list is exhausted, then the element gives up.

Failures SHOULD be detected through network failure indications or timeouts. If the element sending the message is a client sending a request using a client transaction, the client transaction will report any transport layer failures. If the element sending the message is a client sending a request directly to the transport layer, the transport layer will report any failures (See Section 19.4). In either case, the client SHOULD try the next address. This will involve creating a new client transaction for it in the former case. The new request MUST have a new branch ID in the Via header. Note also that the new destination might be with a different transport, which might require a change in other parts of the Via header.

Response failures are handled by the transport layer itself, which may retry the response to the next next hop. See Section 19.2.2.

Failures can be detected through timeouts only if the element is a client sending a request through the client transaction. In that case, if a timeout is reported by the client transaction, the client SHOULD try the next next hop in the list.

> OPEN ISSUE #219: It might be easier to encapsulate the SRV processing in one place, at the transport layer, rather than the behavior being dependent on client v. server. This can only be done if merging of srv records across transports is deprecated, along with failures based on timeouts.

Once a next hop is successfully contacted, that same next hop address MUST be used for all subsequent messages that share the same Call-ID. More specifically, once a request is delivered successfully to a particular next hop, all subsequent requests with the same Call-ID MUST be delivered to that next hop. Once a response is delivered successfully to a particular next hop, all subsequent responses with the same Call-ID MUST be delivered to that next hop. However, if that next hop fails, the selection algorithms MUST be re-run for the top.

> This is a change from RFC2543, which only used the same address for requests within a transaction. Broadening the scope to Call-ID helps, for example, ensure that requests with credentials after a challenge are delivered to the same server that issued the challenge.

A stateless proxy can accomplish this, for example, by using the modulo N of a hash of the Call-ID value as the uniform random number described in the weighting algorithm of RFC 2782 [40]. Here, N is the sum of weights within the priority class.

> OPEN ISSUE #220: This stateless selection algorithm doesn't work if there are failures.

25 Examples

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

25.1 Registration

Bob registers on start-up. The message flow is shown in Figure 9.

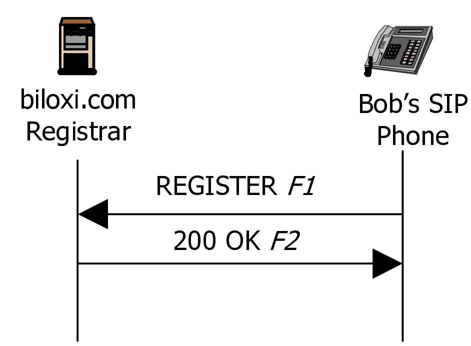

Figure 9: SIP Registration Example

```
F1 REGISTER Bob -> Registrar

  REGISTER sip:registrar.biloxi.com
  Via: SIP/2.0/UDP 10.4.1.4:5060
  To: Bob <sip:bob@biloxi.com>
  From: Bob <sip:bob@biloxi.com>;tag=456248
```

```
  Call-ID: 843817637684230@phone21.boxesbybob.com
  CSeq: 1826 REGISTER
  Contact: <sip:bob@10.4.1.4>
  Expires: 7200
  Contact-Length: 0
```

The registration expires after two hours. The registrar responds with a 200 OK:

```
F2 200 OK Registrar -> Bob

  SIP/2.0 200 OK
  Via: SIP/2.0/UDP 10.4.1.4:5060
  To: Bob <sip:bob@biloxi.com>
  From: Bob <sip:bob@biloxi.com>;tag=456248
  Call-ID: 843817637684230@phone21.boxesbybob.com
  CSeq: 1826 REGISTER
  Contact: <sip:bob@10.4.1.4>
  Expires: 7200
  Contact-Length: 0
```

25.2 Session Setup

This example contains the full details of the example session setup in Section 4. The message flow is shown in Figure 1.

```
F1 INVITE Alice -> atlanta.com proxy

  INVITE sip:bob@biloxi.com SIP/2.0
  Via: SIP/2.0/UDP 10.1.3.3:5060
  To: Bob <sip:bob@biloxi.com>
  From: Alice <sip:alice@atlanta.com>;tag=1928301774
  Call-ID: a84b4c76e66710@10.1.3.3
  CSeq: 314159 INVITE
  Contact: <sip:alice@10.1.3.3>
  Content-Type: application/sdp
  Contact-Length: 142

  (Alice’s SDP not shown)
```

```
F2 100 Trying atlanta.com proxy -> Alice
```

```
SIP/2.0 100 Trying
Via: SIP/2.0/UDP 10.1.3.3:5060
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@10.1.3.3
CSeq: 314159 INVITE
Contact-Length: 0
```

F3 INVITE atlanta.com proxy -> biloxi.com proxy

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP 10.1.1.1:5060;branch=77ef4c2312983.1
Via: SIP/2.0/UDP 10.1.3.3:5060
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@10.1.3.3
CSeq: 314159 INVITE
Contact: <sip:alice@10.1.3.3>
Content-Type: application/sdp
Contact-Length: 142

(Alice’s SDP not shown)
```

F4 100 Trying biloxi.com proxy -> atlanta.com proxy

```
SIP/2.0 100 Trying
Via: SIP/2.0/UDP 10.1.1.1:5060;branch=77ef4c2312983.1
Via: SIP/2.0/UDP 10.1.3.3:5060
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@10.1.3.3
CSeq: 314159 INVITE
Contact-Length: 0
```

F5 INVITE biloxi.com proxy -> Bob

```
INVITE sip:bob@10.4.1.4 SIP/2.0
Via: SIP/2.0/UDP 10.2.1.1:5060;branch=4b43c2ff8.1
Via: SIP/2.0/UDP 10.1.1.1:5060;branch=77ef4c2312983.1
Via: SIP/2.0/UDP 10.1.3.3:5060
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
```

```
  Call-ID: a84b4c76e66710@10.1.3.3
  CSeq: 314159 INVITE
  Contact: <sip:alice@10.1.3.3>
  Content-Type: application/sdp
  Contact-Length: 142

  (Alice’s SDP not shown)
```

```
F6 180 Ringing Bob -> biloxi.com proxy

  SIP/2.0 180 Ringing
  Via: SIP/2.0/UDP 10.2.1.1:5060;branch=4b43c2ff8.1
  Via: SIP/2.0/UDP 10.1.1.1:5060;branch=77ef4c2312983.1
  Via: SIP/2.0/UDP 10.1.3.3:5060
  To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
  From: Alice <sip:alice@atlanta.com>;tag=1928301774
  Call-ID: a84b4c76e66710@10.1.3.3
  CSeq: 314159 INVITE
  Contact-Length: 0
```

```
F7 180 Ringing biloxi.com proxy -> atlanta.com proxy

  SIP/2.0 180 Ringing
  Via: SIP/2.0/UDP 10.1.1.1:5060;branch=77ef4c2312983.1
  Via: SIP/2.0/UDP 10.1.3.3:5060
  To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
  From: Alice <sip:alice@atlanta.com>;tag=1928301774
  Call-ID: a84b4c76e66710@10.1.3.3
  CSeq: 314159 INVITE
  Contact-Length: 0
```

```
F8 180 Ringing atlanta.com proxy -> Alice

  SIP/2.0 180 Ringing
  Via: SIP/2.0/UDP 10.1.3.3:5060
  To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
  From: Alice <sip:alice@atlanta.com>;tag=1928301774
  Call-ID: a84b4c76e66710@10.1.3.3
  CSeq: 314159 INVITE
  Contact-Length: 0
```

F9 200 OK Bob -> biloxi.com proxy

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 10.2.1.1:5060;branch=4b43c2ff8.1
Via: SIP/2.0/UDP 10.1.1.1:5060;branch=77ef4c2312983.1
Via: SIP/2.0/UDP 10.1.3.3:5060
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@10.1.3.3
CSeq: 314159 INVITE
Contact: <sip:bob@10.4.1.4>
Content-Type: application/sdp
Contact-Length: 131

(Bob’s SDP not shown)
```

F10 200 OK biloxi.com proxy -> atlanta.com proxy

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 10.1.1.1:5060;branch=77ef4c2312983.1
Via: SIP/2.0/UDP 10.1.3.3:5060
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@10.1.3.3
CSeq: 314159 INVITE
Contact: <sip:bob@10.4.1.4>
Content-Type: application/sdp
Contact-Length: 131

(Bob’s SDP not shown)
```

F11 200 OK atlanta.com proxy -> Alice

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 10.1.3.3:5060
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@10.1.3.3
CSeq: 314159 INVITE
Contact: <sip:bob@10.4.1.4>
Content-Type: application/sdp
Contact-Length: 131
```

```
  (Bob's SDP not shown)
```

```
F12 ACK Alice -> Bob

  ACK sip:bob@10.4.1.4 SIP/2.0
  Via: SIP/2.0/UDP 10.1.3.3:5060
  To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
  From: Alice <sip:alice@atlanta.com>;tag=1928301774
  Call-ID: a84b4c76e66710@10.1.3.3
  CSeq: 314159 ACK
  Contact-Length: 0
```

The media session between Alice and Bob is now established.

Bob hangs up first. Note that Bob's SIP phone maintains its own CSeq numbering space, which, in this example, begins with 231. Also not that since Bob is making the request, the To and From URLs and tags have been swapped.

```
F13 BYE Bob -> Alice

  BYE sip:alice@10.1.3.3 SIP/2.0
  Via: SIP/2.0/UDP 10.4.1.4:5060
  From: Bob <sip:bob@biloxi.com>;tag=a6c85cf
  To: Alice <sip:alice@atlanta.com>;tag=1928301774
  Call-ID: a84b4c76e66710@10.1.3.3
  CSeq: 231 BYE
  Contact-Length: 0
```

```
F14 200 OK Alice -> Bob

  SIP/2.0 200 OK
  Via: SIP/2.0/UDP 10.4.1.4:5060
  From: Bob <sip:bob@biloxi.com>;tag=a6c85cf
  To: Alice <sip:alice@atlanta.com>;tag=1928301774
  Call-ID: a84b4c76e66710@10.1.3.3
  CSeq: 231 BYE
  Contact-Length: 0
```

The SIP Call Flows document [42] contains further examples of SIP messages.

;; This buffer is for notes you don't want to save, and for Lisp evaluation. ;; If you want to create a file, first visit that file with C-x C-f, ;; then enter the text in that file's own buffer.

26 Augmented BNF for the SIP Protocol

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 [12] and RFC 2234 [43]. 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.

26.1 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 [9] 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. This is intended to behave exactly as HTTP 1.1 as described in RFC2615 [8].

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

To separate the header name from the rest of value, a colon is used, which, by the above rule allows whitespace before, but no line break, and whitespace after, including a linebreak. The HCOLON defines this construct.

```
HCOLON  =  *( SP | HT ) ":" LWS
```

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 [11]). 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. Some elements (authentication) force hex alphas to be lower case.

```
LHEX  =  digit | "a" | "b" | "c" | "d" | "e" | "f"
```

Others allow mixed upped and lower case

```
hex  =  LHEX | "A" | "B" | "C" | "D" | "E" | "F"
```

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   =  ”(” | ”)” | ”<” | ”>” | ”@” |
                ”,” | ”;” | ”:” | ”\” | <”> |
                ”/” | ”[” | ”]” | ”?” | ”=” |
4487            ”{” | ”}” | SP | HT
```

4488 When tokens are used or separators are used between elements, whitespace is often allowed before or
4489 after these characters:

```
MINUS     =  LWS ”-” LWS ; minus
DOT       =  LWS ”.” LWS ; period
PERCENT   =  LWS ”%” LWS ; percent
BANG      =  LWS ”!” LWS ; exclamation
PLUS      =  LWS ”+” LWS ; plus
STAR      =  LWS ”*” LWS ; askerisk
TILDE     =  LWS ”˜” LWS ; tilde
EQUAL     =  LWS ”=” LWS ; equal
LPAREN    =  LWS ”(” LWS ; left parenthesis
RPAREN    =  LWS ”)” LWS ; right parenthesis
LANGLE    =  LWS ”<” LWS ; left angle bracket
RAQUOT    =  ”>” LWS ; right angle quote
LAQUOT    =  LWS ”<”; left angle quote
RANGLE    =  LWS ”>” LWS ; right angle bracket
BAR       =  LWS ”—” LWS ; vertical bar
ATSIGN    =  LWS ”@” LWS ; atsign
COMMA     =  LWS ”,” LWS ; comma
SEMI      =  LWS ”;” LWS ; semicolon
COLON     =  LWS ”:” LWS ; colon
DQUOT     =  LWS <”> LWS ; double quotation mark
LDQUOT    =  LWS <”>; open double quotation mark
RDQUOT    =  <”> LWS ; close double quotation mark
LBRACK    =  LWS ”{” LWS ; left square bracket
4490 RBRACK    =  LWS ”}” LWS ; right square bracket
```

4491 Comments can be included in some SIP header fields by surrounding the comment text with parentheses.
4492 Comments are only allowed in fields containing “comment” as part of their field value definition. In all other
4493 fields, parentheses are considered part of the field value.

```
comment   =  LPAREN *(ctext | quoted-pair | comment) RPAREN
4494 ctext     =  <any TEXT-UTF8 excluding “(“ and “)”>
```

4495 A string of text is parsed as a single word if it is quoted using double-quote marks. In quoted strings,
4496 quotation marks (”) and backslashes (\) need to be escaped.

```
quoted-string  =  ( LWS <”> *(qdtext | quoted-pair ) <”> )
qdtext         =  LWS | %x21 | %x23-5b | %x5d-7e
4497                 |                                    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)

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 )
4505 hnv-unreserved   =  "[" | "]" | "/" | "?" | ":" | "+" | "$"

     SIP-message    =  Request | Response
     Request        =  Request-Line
                       *( message-header )
                       CRLF
                       [ message-body ]
     Request-Line   =  Method SP Request-URI SP SIP-Version CRLF
     Request-URI    =  SIP-URL | absoluteURI
4506 SIP-Version    =  "SIP/2.0"
```

```
message-header
        =  Accept
        |  Accept-Encoding
        |  Accept-Language
        |  Alert-Info
        |  Allow
        |  Authentication-Info
        |  Authorization
        |  Call-ID
        |  Call-Info
        |  Contact
        |  Content-Disposition
        |  Content-Encoding
        |  Content-Language
        |  Content-Length
        |  Content-Type
        |  CSeq
        |  Date
        |  Error-Info
        |  Expires
        |  From
        |  In-Reply-To
        |  Max-Forwards
        |  MIME-Version
        |  Organization
        |  Priority
        |  Proxy-Authenticate
        |  Proxy-Authorization
        |  Proxy-Require
        |  Record-Route
        |  Require
        |  Retry-After
        |  Route
        |  Server
        |  Subject
        |  Supported
        |  Timestamp
        |  To
        |  Unsupported
        |  User-Agent
        |  Via
        |  Warning
4507    |  WWW-Authenticate
```

```
Method            =  "INVITE" | "ACK" | "OPTIONS" | "BYE"
                     | "CANCEL" | "REGISTER" | extension-method
extension-method  =  token
option-tag        =  token
Response
                  =  Status-Line
                     *( message-header )
                     CRLF
4508                 [ message-body ]

Status-Line    =  SIP-version SP Status-Code SP Reason-Phrase CRLF
Status-Code
               =  Informational
               |  Redirection
               |  Success
               |  Client-Error
               |  Server-Error
               |  Global-Failure
               |  extension-code
4509 extension-code =  3DIGIT

Reason-Phrase
               =  *<TEXT-UTF8, excluding CR, LF>
Informational
               =  "100"                        ; Trying
               |  "180"                        ; Ringing
               |  "181"                        ; Call Is Being Forwarded
               |  "182"                        ; Queued
4510           |  "183"                        ; Session Progress

4511 Success  =  "200"  ; OK

Redirection  =  "300"  ; Multiple Choices
             |  "301"  ; Moved Permanently
             |  "302"  ; Moved Temporarily
             |  "305"  ; Use Proxy
4512         |  "380"  ; Alternative Service
```

```
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
              |  "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
4513          |  "488"  ; Not Acceptable Here

Server-Error  =  "500"  ; Internal Server Error
              |  "501"  ; Not Implemented
              |  "502"  ; Bad Gateway
              |  "503"  ; Service Unavailable
              |  "504"  ; Server Time-out
4514          |  "505"  ; SIP Version not supported

Global-Failure  =  "600"  ; Busy Everywhere
                |  "603"  ; Decline
                |  "604"  ; Does not exist anywhere
4515            |  "606"  ; Not Acceptable

Accept            =  "Accept" HCOLON
                     #( media-range [ accept-params ] )
media-range       =  ( "*/*"
                     | ( type LWS "/" "*" LWS )
                     | ( type SLASH subtype )
                     ) *( SEMI parameter )
accept-params     =  SEMI "q" EQUAL qvalue *( accept-extension )
4516 accept-extension  =  SEMI token [ EQUAL ( token | quoted-string ) ]
```

```
     Accept-Encoding   =  "Accept-Encoding" HCOLON
                          1#( codings [ SEMI "q" EQUAL qvalue ] LWS )
     codings           =  ( content-coding | "*" )
     content-coding    =  token
     qvalue            =  ( "0" [ "." 0*3DIGIT ] )
4517                      — ( "1" [ "." 0*3("0") ] )

     Accept-Language   =  "Accept-Language" HCOLON
                          1#( language-range [ SEMI "q" EQUAL qvalue ] )
4518 language-range    =  ( ( 1*8ALPHA *( MINUS 1*8ALPHA ) ) — "*" )

     Alert-Info      =  "Alert-Info" HCOLON #
                        ( LAQUOT URI RAQUOT *( COLON generic-param ))
     generic-param   =  token [ EQUAL ( token | host |
4519                    quoted-string ) ]

4520 Allow   =   "Allow" HCOLON 1#Method

     Authorization     =  "Authorization" HCOLON credentials
     credentials       =  LWS "Digest" digest-response
     digest-response   =  1#( username | realm | nonce | digest-uri
                          | dresponse | [ algorithm ] | [cnonce]
                          | [opaque] | [message-qop]
                          | [nonce-count] | [auth-param] )
     username          =  "username" EQUAL username-value
     username-value    =  quoted-string
     digest-uri        =  "uri" EQUAL digest-uri-value
     digest-uri-value  =  request-uri ; As specified by HTTP/1.1
     message-qop       =  "qop" EQUAL qop-value
     cnonce            =  "cnonce" EQUAL cnonce-value
     cnonce-value      =  nonce-value
     nonce-count       =  "nc" EQUAL nc-value
     dresponse         =  "response" EQUAL request-digest
4521 request-digest    =  LDQUOT 32LHEX RDQUOT

     AuthenticationInfo  =  "Authentication-info" HCOLON 1#( digest — nextnonce )
     nextnonce           =  "nextnonce" EQUAL nonce-value

4522 callid              =  token [ ATSIGN token ]

4523 Call-ID   =   ( "Call-ID" | "i" ) HCOLON callid

     Call-Info    =  "Call-Info" HCOLON # ( LAQUOT URI RAQUOT
                     *( SEMI info-param) )
     info-param   =  "purpose" EQUAL ( "icon" | "info"
4524                 | "card" | token ) | generic-param
```

```
     Contact        =  ( "Contact" | "m" ) HCOLON
                       (STAR | (1# (( name-addr | addr-spec )
                       *( SEMI contact-params ))))
     name-addr      =  [ display-name ] LAQUOT addr-spec RAQUOT
     addr-spec      =  SIP-URL | URI
4525 display-name   =  LWS (*token | quoted-string)
       contact-params     =  "q"                   EQUAL  qvalue
                          |  "action"              EQUAL  "proxy" | "redirect"
                          |  "expires"             EQUAL  delta-seconds |
                             LDQUOT SIP-date RDQUOT
                          |  contact-extension
       contact-extension  =  generic-param
       qvalue             =  ( "0" [ "." 0*3DIGIT ] )
4526                         | ( "1" [ "." 0*3("0") ] )

4527        delta-seconds   =   1*DIGIT

            Content-Disposition   =  "Content-Disposition" HCOLON
                                     disposition-type *( SEMI disposition-param )
            disposition-type      =  "render" | "session" | "icon" | "alert"
                                     | disp-extension-token
            disposition-param     =  "handling" EQUAL
                                     ( "optional" | "required" |
                                     other-handling ) | generic-param
            other-handling        =  token
4528        disp-extension-token  =  token

            Content-Encoding   =  ( "Content-Encoding" | "e" ) HCOLON
4529                              1#content-coding

            Content-Language   =  "Content-Language" HCOLON 1#language-tag
            language-tag       =  primary-tag *( MINUS subtag )
            primary-tag        =  1*8ALPHA
4530        subtag             =  1*8ALPHA

4531        Content-Length   =   ( "Content-Length" | "l" ) HCOLON 1*DIGIT

4532        Content-Type   =   ( "Content-Type" | "c" ) HCOLON media-type

4533        CSeq   =   "CSeq" HCOLON 1*DIGIT Method
```

```
Date          =  "Date" HCOLON SIP-date
SIP-date      =  rfc1123-date
rfc1123-date  =  wkday COMMA SP date1 SP time SP "GMT"
date1         =  2DIGIT SP month SP 4DIGIT
                 ; day month year (e.g., 02 Jun 1982)
time          =  2DIGIT ":" 2DIGIT ":" 2DIGIT
                 ; 00:00:00 - 23:59:59
wkday         =  "Mon" | "Tue" | "Wed"
                 | "Thu" | "Fri" | "Sat" | "Sun"
month         =  "Jan" | "Feb" | "Mar" | "Apr"
                 | "May" | "Jun" | "Jul" | "Aug"
4534             | "Sep" | "Oct" | "Nov" | "Dec"

Error-Info  =  "Error-Info" HCOLON #
               ( LAQUOT URI RAQUOT
4535           *( SEMI generic-param ))

Expires     =  "Expires" HCOLON ( SIP-date | delta-seconds )
From        =  ( "From" | "f" ) HCOLON
               ( name-addr | addr-spec )
               *( SEMI from-param )
from-param  =  tag-param | generic-param
4536 tag-param   =  "tag" EQUAL token

4537 In-Reply-To  =  "In-Reply-To" HCOLON 1# callid

4538 Max-Forwards  =  "Max-Forwards" HCOLON 1*DIGIT

4539 MIME-Version  =  "MIME-Version" HCOLON 1*DIGIT "." 1*DIGIT

4540 Organization  =  "Organization" HCOLON TEXT-UTF8-TRIM

Priority        =  "Priority" HCOLON priority-value
priority-value  =  "emergency" | "urgent" | "normal"
                   |                        "non-urgent" | other-priority
4541 other-priority  =  token
```

```
     Proxy-Authenticate  =  ”Proxy-Authenticate” HCOLON 1#challenge
     challenge           =  LWS ”Digest” digest-challenge
     digest-challenge    =  1#( realm | [ domain ] | nonce |
                            [ opaque ] | [ stale ] | [ algorithm ] |
                            [ qop-options ] | [auth-param] )
     realm               =  ”realm” EQUALS realm-value
     realm-value         =  quoted-string
     domain              =  ”domain” EQUAL LDQUOT URI
                            ( 1*SP URI ) RDQUOT
     URI                 =  absoluteURI | abs_path
     nonce               =  ”nonce” EQUAL nonce-value
     nonce-value         =  quoted-string
     opaque              =  ”opaque” EQUAL quoted-string
     stale               =  ”stale” EQUAL ( ”true” | ”false” )
     algorithm           =  ”algorithm” EQUAL ( ”MD5” | ”MD5-sess” |
                            token )
     qop-options         =  ”qop” EQUAL LDQUOT 1#qop-value RDQUOT
4542 qop-value           =  ”auth” | ”auth-int” | token

4543 Proxy-Authorization  =  ”Proxy-Authorization” HCOLON credentials

4544 Proxy-Require  =  ”Proxy-Require” HCOLON 1#option-tag

     Record-Route  =  ”Record-Route” HCOLON 1#
                      ( name-addr *( SEMI rr-param ))
     rr-param      =  generic-param
4545 Require       =  ”Require” HCOLON 1#option-tag

     Retry-After  =  ”Retry-After” HCOLON
                     ( SIP-date | delta-seconds )
                     [ comment ] *( SEMI retry-param )
     retry-param  =  ”duration” EQUAL delta-seconds |
4546                 generic-param

     Route  =  ”Route” HCOLON 1# ( name-addr
4547           *( SEMI rr-param ))

     Server           =  ”Server” HCOLON 1*( product — comment )
     product          =  token [SLASH product-version]
4548 product-version  =  token

4549 Subject  =  ( ”Subject” | ”s” ) HCOLON TEXT-UTF8-TRIM

4550 Supported  =  ( ”Supported” | ”k” ) HCOLON 0#option-tag
```

```
     Timestamp  =  "Timestamp" HCOLON *(DIGIT)
                   [ "." *(DIGIT) ] [ delay ]
4551 delay      =  *(DIGIT) [ "." *(DIGIT) ]

     To        =  ( "To" | "t" ) HCOLON ( name-addr |
                  addr-spec ) *( SEMI to-param )
4552 to-param  =  tag-param | generic-param

4553 Unsupported  =  "Unsupported" HCOLON 1#option-tag

4554 User-Agent  =  "User-Agent" HCOLON 1*( product — comment )

     Via               =  ( "Via" | "v" ) HCOLON
                          1#( sent-protocol sent-by
                          *( SEMI via-params ) [ comment ] )
     via-params        =  via-hidden | via-ttl | via-maddr
                          | via-received | via-branch
                          | via-extension
     via-hidden        =  "hidden"
     via-ttl           =  "ttl" EQUAL ttl
     via-maddr         =  "maddr" EQUAL host
     via-received      =  "received" EQUAL host
     via-branch        =  "branch" EQUAL token
     via-extension     =  generic-param
     sent-protocol     =  protocol-name SLASH protocol-version
                          SLASH transport
     protocol-name     =  "SIP" | token
     protocol-version  =  token
     transport         =  "UDP" | "TCP" | "TLS" | "SCTP"
                          | other-transport
     sent-by           =  host [ COLON port ]
4555 ttl               =  1*3DIGIT                      ; 0 to 255

     Warning        =  "Warning" HCOLON 1#warning-value
     warning-value  =  warn-code SP warn-agent SP warn-text
     warn-code      =  3DIGIT
     warn-agent     =  ( host [ COLON port ] ) | pseudonym
                       ; the name or pseudonym of the server adding
                       ; the Warning header, for use in debugging
     warn-text      =  quoted-string
4556 pseudonym      =  token

4557 WWW-Authenticate  =  "WWW-Authenticate" HCOLON challenge
```

27 IANA Considerations

All new or experimental method names, header field names, and status codes used in SIP applications SHOULD be registered with IANA in order to prevent potential naming conflicts. It is RECOMMENDED that new "option- tag"s and "warn-code"s also be registered. Before IANA registration, new protcol elements SHOULD be characterized in an Internet- Draft or, preferably, an RFC.

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.

When a registration for either a new header field, new method or new status code is created based on an Internet-Draft, and that Internet-Draft 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.

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

27.1 Option Tags

Option tags are used in headers such as Require, Supported, Proxy-Require and Unsupported in support of SIP compatibility mechanisms for extensions. For more on the use of option tags in these headers see Section 21.2. The option tag itself is a string that is associated with a particular SIP option (e.g. an extension) in order to identify the option in signaling between SIP endpoints.

When registering a new SIP option with IANA, 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 Section 26) 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)

This procedure has been borrowed from RTSP [4] and the RTP AVP [44].

27.2 Warn-Codes

Warning codes provide information supplemental to the status code in SIP response messages when the failure of the transaction results from a Session Description Protocol (SDP, [6]). New "warn-code" values can be registered with IANA as they arise.

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

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.

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

27.3 Header Field Names

Header field names do not require working group or working group chair review prior to IANA registration, but SHOULD be documented in an RFC or Internet- Draft before IANA is consulted.

The following information needs to be provided to IANA in order to register a new header field 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 field is defined.
- A copy of the draft or RFC where the header field is defined.

Header fields SHOULD NOT use the X- prefix notation and MUST NOT duplicate the names of header fields 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 7.3.3). Compact forms can only be assigned after SIP working group review. In the absence of this working group, a designated expert reviews the request.

27.4 Method and Response Codes

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. The same procedures apply to new method names.

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

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

28 Changes Made in Version 00

- 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 22.40.
- 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.
- Clarified that CANCEL and INVITE are separate transactions (Fig. 7). 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 6)
- In Section 22.40, noted that the maddr and received parameters also need to be encrypted when doing Via hiding.
- Simplified Fig. 7 to only show INVITE transaction.
- Added definition of the use of Contact (Section 22.10) for OPTIONS.
- Added HTTP/RFC822 headers Content-Language and MIME-Version.
- Added note in minimal section indicating that UAs need to support UDP.
- Added explanation explaining what a UA should do when receiving an initial INVITE with a tag.
- Clarified UA and proxy behavior for 302 responses.

- 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 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 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 22.19, noted that registration updates can shorten the validity period.
- Added note to enclose the URI for digest 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 22.10 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 SDP section, 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 22.16).
- Added IPv6 literal addresses to the SIP URL definition, according to RFC 2732 [45]. Modified the IPv4 address to limit segments to at most three digits.
- 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.
- Changed references regarding DNS SRV records from RFC 2052 to RFC 2782, which is now a Proposed Standard. Integrated SRV into the search procedure 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 1, 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. 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. 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 2. It is mandatory.
- Added: "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: "If an INVITE request for an existing session fails, the session description agreed upon in the last successful INVITE transaction remains in force."
- Clarified 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.
- 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 2 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.
- 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. 22.10.

- 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 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.

29 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 SDP section, 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 2 and 3 to show proxy interaction with headers rather than "end-to-end" or "hop-by-hop".

30 Changes Made in Version 02

- Added "or UAS" in description of received headers in Section 22.40. 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 22.14).
- Added note that Allow MAY be included in requests, to indicate requestor capabilities for the same call ID.
- Added note to Section 22.17 indicating that registrars MUST include the Date header to accomodate UAs that do not have a notion of absolute time.
- Added note emphasizing that non-SIP URIs are permissible in REGISTER.
- 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.

31 Changes Made in Version 03

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

- 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 22.16) 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.

32 Changes Made in Version 04

- Removed Unsupported as a request header from Table 3.
- 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.

33 Changes Made in Version 05

- Updated Table 2 to reflect that Error-Info is a response header in 3xx-6xx responses (it was previously listed as a request header).
- Removed WWW-Authenticate as a request header from Table 3. Authentication of responses is now done according to RFC2617.
- Updated the Accept, Accept-Encoding and Accept-Language sections. More details on precise semantics for the various requests and responses is now provided. Presence of these headers is now a SHOULD for INVITE and 2xx to INVITE when a non-default value is present. Extra emphasis is placed on including the Accept-Language in INVITE and 2xx in order to support internationalization. Usage of these three headers in CANCEL has been removed since it makes no sense.
- Generalized local outbound processing rules in Section 16.4.1 to cover the case where the UAS is using a local outbound proxy which was not in the initial call setup path.
- Updated record-routing section, so that a proxy can insert a transport param if it knows that the proxy on one side supports the specific transport (the previous text required the proxy to know whether the proxies on both sides supported the specific transport).
- Added Authentication-Info to Section 10.
- Clarified the meaning of Table 2 for responses.
- Updated Table 1 to reflect that maddr is no longer mandatory in Record-Route.
- Updated Table 3 so that header fields in responses to ACK are never listed as optional, mandatory, etc. - only not applicable. This is because responses to ACK are not allowed. Also improved wording in Section 5.1.1 to clarify that there MUST NOT be responses to ACK.
- Updated SRV procedures. Old text said to treat a failure to contact a server as a 4xx, which would stop the SRV processing. But, this is not so. Sentence was stricken.
- Updated 12.1 to clarify that 2xx INVITE responses MUST contain session descriptions.
- Changed User-Agent to a request header in Table 3.
- Updated SDP section, so that a UA cannot change the SDP when it gets a re-INVITE with no SDP.
- Clarified Appendix B that a unicast offer MUST have a unicast response.
- Clarified that any request can be record-routed, but it may not be used by the UA, depending on the method.
- non-2xx responses to INVITE no longer retransmitted over TCP.
- Removed lower bound on T1 and T2 in private networks, which can use lower values. Furthermore, T1 can be smaller on the public Internet if proper RTT estimation is used.
- UAS Cannot send a BYE for a call leg until it receives ACK, in order to eliminate a race condition between BYE and 200 OK.

- Support of CR or LF alone as line terminators, as opposed to CRLF, is no longer required.
- Client behavior on receipt of a 3xx to re-INVITE is now specified, and it is no longer forbidden to generate a 3xx. This is needed to maintain the idempotency of INVITE, as a proxy might redirect without knowing its a 3xx.
- CANCEL cannot be sent before a 1xx is received, in order to eliminate race condition between request and CANCEL.
- Termination of the client and server transactions is now based entirely on timeouts, rather than retransmission counters, in order to unify TCP and UDP behavior. Timeout values scale as a function of the RTT estimate, defined as T1. For reliable transports, many of these timers are now set to zero. Many timeouts differ than in bis-04.
- Added a working RTT estimation algorithm using the Timestamp header, and specified it to be compliant to RFC 2988.
- UAS accepting requests with unknown schemes in the URI in the To field is now a RECOMMENDED instead of SHOULD. This reflects the fact that processing a request when the To field doesn't match is a matter of policy.
- Bodies are now allowed in any request and response, including CANCEL, although there may not be any semantics associated with that.
- Supporting of INVITE without SDP is now a MUST (no strength was previously specified).
- Registration procedures for visiting, which had a few sentences in bis-04, have been removed. Roaming is a complex issue, and should be treated elsewhere.
- Bis-04 mandated that a 2xx response to REGISTER contain expires Contact parameters indicating the expiration time of a contact. This behavior has now been made consistent with requests, so that the expiration time of a contact is the same in either case: the expires param is used first if present, then the Expires header if present, else one hour for SIP URLs.
- Action parameter in contact registrations is deprecated.
- 2xx to REGISTER MUST contain current contacts. This was just a SHOULD in bis-04.
- Multicast operation radically changed. Now, the treatment is no different than unicast. That is, only the first non-1xx response to a multicast request will be used. This is a natural consequence of the layering now applied to the protocol. This still enables anycast types of functions, mirroring the real usage of registrar discovery.
- To completely separate transport rules from transaction rules, the rule in bis-04 that said a UAC SHOULD keep a connection opened until a response is received, has been turned into a timer recommendation. Specifically, the spec now says that it is RECOMMENDED that connections be kept opened for a minimum interval of sufficient duration to guarantee, with high probability, that responses are sent over the same connections as a request.

- Re-use of existing connections for new requests to the same address and port is now RECOMMENDED, it was only a MAY in bis-04.
- Modification of headers below the Authorization header by proxies is no longer disallowed, since the only mechanism that used Authorization in that way, PGP, has been deprecated previously.
- Authentication of registrations now RECOMMENDED; no strength was defined previously.
- Registering of new headers with IANA is now SHOULD; no strength was defined previously.
- Proxy aggregation of challenges now a SHOULD; no strength was defined previously.
- Server support of basic authentication downgraded from SHOULD to MAY.
- UAC resubmitting requests with credentials after a challenge upgraded from MAY to SHOULD.
- TLS is now RECOMMENDED as the transport layer security for SIP signaling.
- UA recursion on a redirect is now SHOULD; no strength was assigned previously.
- UA reuse of headers in a recursed request is now SHOULD; no strength was assigned previously.
- Security considerations added for Call-Info and Alert-Info.
- Proxies no longer forward a 6xx immediately on receiving it. Instead, they CANCEL pending branches immediately. This avoids a potential race condition that would result in a UAC getting a 6xx followed by a 2xx. In all cases except this race condition, the result will be the same - the 6xx is forwarded upstream.
- The term call-leg has been eliminated from the spec; a more generic term, dialog, is used in its place.
- For SRV processing, subsequent requests with the same Call-ID (as opposed to the same transaction in bis-04) are sent to the same server.
- SRV processing generalized to deal with the fact that the default port is transport dependent.
- Per IESG request, draft-ietf-sip-serverfeatures has been integrated into bis.
- Per IESG request, draft-ietf-sip-100rel will be integrated into bis. This is marked with a placeholder in this draft.
- The BNF has been converted from implicit LWS to explicit LWS.
- Caching of responses in a proxy to avoid redoing location server lookups used to be a SHOULD. Caching behavior for responses is now fully encapsulated in the transaction processing.
- Proxy usage of SRV in processing Route headers upgraded from SHOULD to MUST.

34 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.

Brian Rosen provided the compiled BNF.

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

35 Authors' Addresses

Authors addresses are listed alphabetically for the editors, the writers, and then the original authors of RFC 2543.

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

Henning Schulzrinne
Dept. of Computer Science
Columbia University
1214 Amsterdam Avenue
New York, NY 10027
USA
electronic mail: schulzrinne@cs.columbia.edu

Gonzalo Camarillo
Ericsson
Advanced Signalling Research Lab.
FIN-02420 Jorvas
Finland
electronic mail: Gonzalo.Camarillo@ericsson.com

Alan Johnston
WorldCom
100 South 4th Street
St. Louis, MO 63102
USA
electronic mail: alan.johnston@wcom.com

Jon Peterson
NeuStar, Inc

1800 Sutter Street, Suite 570
Concord, CA 94520
USA
electronic mail: jon.peterson@neustar.com

Robert Sparks
dynamicsoft, Inc.
5100 Tennyson Parkway
Suite 1200
Plano, Texas 75024
USA
electronic mail: rsparks@dynamicsoft.com

Mark Handley
ACIRI
electronic mail: mjh@aciri.org

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

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. 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.

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

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

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

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

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

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

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

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

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

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

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

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

[21] S. Donovan, “The SIP INFO method,” Request for Comments 2976, Internet Engineering Task Force, Oct. 2000.

[22] J. Rosenberg and H. Schulzrinne, “An offer/answer model with sdp,” Internet Draft, Internet Engineering Task Force, Oct. 2001. Work in progress.

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

[24] V. Paxson and M. Allman, “Computing TCP’s retransmission timer,” Request for Comments 2988, Internet Engineering Task Force, Nov. 2000.

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

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

[27] 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.

[28] 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.

[29] J. Galvin, S. Murphy, S. Crocker, and N. Freed, “Security multiparts for MIME: multipart/signed and multipart/encrypted,” Request for Comments 1847, Internet Engineering Task Force, Oct. 1995.

[30] J. Postel, “User datagram protocol,” Request for Comments 768, Internet Engineering Task Force, Aug. 1980.

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

[32] 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.

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

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

[35] 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.

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

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

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

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

[40] 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.

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

[42] A. Johnston, S. Donovan, R. Sparks, C. Cunningham, D. Willis, J. Rosenberg, K. Summers, and H. Schulzrinne, “SIP telephony call flow examples,” Internet Draft, Internet Engineering Task Force, Apr. 2001. Work in progress.

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

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

[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