

SIP: Session Initiation Protocol

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as “work in progress.”

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/1id-abstracts.txt>

To view the list Internet-Draft Shadow Directories, see <http://www.ietf.org/shadow.html>.

Copyright Notice

Copyright (c) The Internet Society (2002). All Rights Reserved.

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 that 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 user’s current location, authenticate and authorize users for services, implement provider call-routing policies, and provide features to users. SIP also provides a registration function that allows users to upload their current locations for use by proxy servers. SIP runs on top of several different transport protocols.

Contents

1	Introduction	8
2	Overview of SIP Functionality	8
3	Terminology	9
4	Overview of Operation	9
5	Structure of the Protocol	14
6	Definitions	16
7	SIP Messages	20
7.1	Requests	20
7.2	Responses	21

34	7.3	Header Fields	22
35	7.3.1	Header Field Format	22
36	7.3.2	Header Field Classification	24
37	7.3.3	Compact Form	24
38	7.4	Bodies	25
39	7.4.1	Message Body Type	25
40	7.4.2	Message Body Length	25
41	7.5	Framing SIP messages	25
42	8	General User Agent Behavior	25
43	8.1	UAC Behavior	26
44	8.1.1	Generating the Request	26
45	8.1.2	Sending the Request	30
46	8.1.3	Processing Responses	30
47	8.2	UAS Behavior	33
48	8.2.1	Method Inspection	33
49	8.2.2	Header Inspection	33
50	8.2.3	Content Processing	34
51	8.2.4	Applying Extensions	35
52	8.2.5	Processing the Request	35
53	8.2.6	Generating the Response	35
54	8.2.7	Stateless UAS Behavior	36
55	8.3	Redirect Servers	36
56	9	Canceling a Request	37
57	9.1	Client Behavior	38
58	9.2	Server Behavior	39
59	10	Registrations	39
60	10.1	Overview	39
61	10.2	Constructing the REGISTER Request	41
62	10.2.1	Adding Bindings	42
63	10.2.2	Removing Bindings	43
64	10.2.3	Fetching Bindings	43
65	10.2.4	Refreshing Bindings	43
66	10.2.5	Setting the Internal Clock	43
67	10.2.6	Discovering a Registrar	43
68	10.2.7	Transmitting a Request	44
69	10.2.8	Error Responses	44
70	10.3	Processing REGISTER Requests	44
71	11	Querying for Capabilities	46
72	11.1	Construction of OPTIONS Request	47
73	11.2	Processing of OPTIONS Request	47
74	12	Dialogs	48

75	12.1	Creation of a Dialog	49
76	12.1.1	UAS behavior	49
77	12.1.2	UAC Behavior	50
78	12.2	Requests within a Dialog	50
79	12.2.1	UAC Behavior	51
80	12.2.2	UAS Behavior	52
81	12.3	Termination of a Dialog	53
82	13	Initiating a Session	53
83	13.1	Overview	53
84	13.2	UAC Processing	54
85	13.2.1	Creating the Initial INVITE	54
86	13.2.2	Processing INVITE Responses	55
87	13.3	UAS Processing	57
88	13.3.1	Processing of the INVITE	57
89	14	Modifying an Existing Session	59
90	14.1	UAC Behavior	59
91	14.2	UAS Behavior	60
92	15	Terminating a Session	61
93	15.1	Terminating a Session with a BYE Request	61
94	15.1.1	UAC Behavior	61
95	15.1.2	UAS Behavior	62
96	16	Proxy Behavior	62
97	16.1	Overview	62
98	16.2	Stateful Proxy	63
99	16.3	Request Validation	63
100	16.4	Route Information Preprocessing	66
101	16.5	Determining request targets	66
102	16.6	Request Forwarding	68
103	16.7	Response Processing	73
104	16.8	Processing Timer C	78
105	16.9	Handling Transport Errors	78
106	16.10	CANCEL Processing	78
107	16.11	Stateless Proxy	79
108	16.12	Summary of Proxy Route Processing	80
109	16.12.1	Examples	81
110	17	Transactions	84
111	17.1	Client Transaction	85
112	17.1.1	INVITE Client Transaction	86
113	17.1.2	Non-INVITE Client Transaction	89
114	17.1.3	Matching Responses to Client Transactions	91
115	17.1.4	Handling Transport Errors	92

116	17.2 Server Transaction	92
117	17.2.1 INVITE Server Transaction	92
118	17.2.2 Non-INVITE Server Transaction	94
119	17.2.3 Matching Requests to Server Transactions	94
120	17.2.4 Handling Transport Errors	96
121	18 Transport	97
122	18.1 Clients	97
123	18.1.1 Sending Requests	97
124	18.1.2 Receiving Responses	98
125	18.2 Servers	99
126	18.2.1 Receiving Requests	99
127	18.2.2 Sending Responses	100
128	18.3 Framing	100
129	18.4 Error Handling	100
130	19 Common Message Components	101
131	19.1 SIP and SIPS Uniform Resource Indicators	101
132	19.1.1 SIP and SIPS URI Components	101
133	19.1.2 Character Escaping Requirements	103
134	19.1.3 Example SIP and SIPS URIs	104
135	19.1.4 URI Comparison	105
136	19.1.5 Forming Requests from a URI	107
137	19.1.6 Relating SIP URIs and tel URLs	107
138	19.2 Option Tags	109
139	19.3 Tags	109
140	20 Header Fields	109
141	20.1 Accept	111
142	20.2 Accept-Encoding	111
143	20.3 Accept-Language	111
144	20.4 Alert-Info	113
145	20.5 Allow	114
146	20.6 Authentication-Info	114
147	20.7 Authorization	114
148	20.8 Call-ID	115
149	20.9 Call-Info	115
150	20.10 Contact	115
151	20.11 Content-Disposition	116
152	20.12 Content-Encoding	117
153	20.13 Content-Language	117
154	20.14 Content-Length	117
155	20.15 Content-Type	117
156	20.16 CSeq	118
157	20.17 Date	118

158	20.18 Error-Info	118
159	20.19 Expires	119
160	20.20 From	119
161	20.21 In-Reply-To	119
162	20.22 Max-Forwards	119
163	20.23 Min-Expires	120
164	20.24 MIME-Version	120
165	20.25 Organization	120
166	20.26 Priority	120
167	20.27 Proxy-Authenticate	121
168	20.28 Proxy-Authorization	121
169	20.29 Proxy-Require	121
170	20.30 Record-Route	122
171	20.31 Reply-To	122
172	20.32 Require	122
173	20.33 Retry-After	122
174	20.34 Route	123
175	20.35 Server	123
176	20.36 Subject	123
177	20.37 Supported	123
178	20.38 Timestamp	123
179	20.39 To	124
180	20.40 Unsupported	124
181	20.41 User-Agent	124
182	20.42 Via	124
183	20.43 Warning	125
184	20.44 WWW-Authenticate	126
185	21 Response Codes	127
186	21.1 Provisional 1xx	127
187	21.1.1 100 Trying	127
188	21.1.2 180 Ringing	127
189	21.1.3 181 Call Is Being Forwarded	127
190	21.1.4 182 Queued	127
191	21.1.5 183 Session Progress	127
192	21.2 Successful 2xx	127
193	21.2.1 200 OK	128
194	21.3 Redirection 3xx	128
195	21.3.1 300 Multiple Choices	128
196	21.3.2 301 Moved Permanently	128
197	21.3.3 302 Moved Temporarily	128
198	21.3.4 305 Use Proxy	129
199	21.3.5 380 Alternative Service	129
200	21.4 Request Failure 4xx	129
201	21.4.1 400 Bad Request	129

202	21.4.2	401 Unauthorized	129
203	21.4.3	402 Payment Required	129
204	21.4.4	403 Forbidden	129
205	21.4.5	404 Not Found	129
206	21.4.6	405 Method Not Allowed	129
207	21.4.7	406 Not Acceptable	130
208	21.4.8	407 Proxy Authentication Required	130
209	21.4.9	408 Request Timeout	130
210	21.4.10	410 Gone	130
211	21.4.11	413 Request Entity Too Large	130
212	21.4.12	414 Request-URI Too Long	130
213	21.4.13	415 Unsupported Media Type	130
214	21.4.14	416 Unsupported URI Scheme	131
215	21.4.15	420 Bad Extension	131
216	21.4.16	421 Extension Required	131
217	21.4.17	423 Interval Too Brief	131
218	21.4.18	480 Temporarily Unavailable	131
219	21.4.19	481 Call/Transaction Does Not Exist	131
220	21.4.20	482 Loop Detected	131
221	21.4.21	483 Too Many Hops	132
222	21.4.22	484 Address Incomplete	132
223	21.4.23	485 Ambiguous	132
224	21.4.24	486 Busy Here	132
225	21.4.25	487 Request Terminated	132
226	21.4.26	488 Not Acceptable Here	132
227	21.4.27	491 Request Pending	133
228	21.4.28	493 Undecipherable	133
229	21.5	Server Failure 5xx	133
230	21.5.1	500 Server Internal Error	133
231	21.5.2	501 Not Implemented	133
232	21.5.3	502 Bad Gateway	133
233	21.5.4	503 Service Unavailable	133
234	21.5.5	504 Server Time-out	134
235	21.5.6	505 Version Not Supported	134
236	21.5.7	513 Message Too Large	134
237	21.6	Global Failures 6xx	134
238	21.6.1	600 Busy Everywhere	134
239	21.6.2	603 Decline	134
240	21.6.3	604 Does Not Exist Anywhere	134
241	21.6.4	606 Not Acceptable	135
242	22	Usage of HTTP Authentication	135
243	22.1	Framework	135
244	22.2	User-to-User Authentication	137
245	22.3	Proxy-to-User Authentication	138

246	22.4 The Digest Authentication Scheme	139
247	23 S/MIME	140
248	23.1 S/MIME Certificates	141
249	23.2 S/MIME Key Exchange	141
250	23.3 Securing MIME bodies	143
251	23.4 SIP Header Privacy and Integrity using S/MIME: Tunneling SIP	144
252	23.4.1 Integrity and Confidentiality Properties of SIP Headers	145
253	23.4.2 Tunneling Integrity and Authentication	146
254	23.4.3 Tunneling Encryption	147
255	24 Examples	149
256	24.1 Registration	149
257	24.2 Session Setup	150
258	25 Augmented BNF for the SIP Protocol	155
259	25.1 Basic Rules	156
260	26 Security Considerations: Threat Model and Security Usage Recommendations	170
261	26.1 Attacks and Threat Models	170
262	26.1.1 Registration Hijacking	171
263	26.1.2 Impersonating a Server	171
264	26.1.3 Tampering with Message Bodies	172
265	26.1.4 Tearing Down Sessions	172
266	26.1.5 Denial of Service and Amplification	173
267	26.2 Security Mechanisms	173
268	26.2.1 Transport and Network Layer Security	174
269	26.2.2 SIPS URI Scheme	174
270	26.2.3 HTTP Authentication	175
271	26.2.4 S/MIME	175
272	26.3 Implementing Security Mechanisms	176
273	26.3.1 Requirements for Implementers of SIP	176
274	26.3.2 Security Solutions	176
275	26.4 Limitations	180
276	26.4.1 HTTP Digest	180
277	26.4.2 S/MIME	180
278	26.4.3 TLS	181
279	26.4.4 SIPS URIs	181
280	26.5 Privacy	182
281	27 IANA Considerations	182
282	27.1 Option Tags	183
283	27.2 Warn-Codes	183
284	27.3 Header Field Names	183
285	27.4 Method and Response Codes	184
286	27.5 The “message/sip” MIME type.	184

287	27.6 New Content-Disposition Parameter Registrations	185
288	28 Changes From RFC 2543	185
289	28.1 Major Functional Changes	185
290	28.2 Minor Functional Changes	188
291	29 Acknowledgments	189
292	30 Authors' Addresses	189
293	A Table of Timer Values	193

294 1 Introduction

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

307 2 Overview of SIP Functionality

308 SIP is an application-layer control protocol that can establish, modify, and terminate multimedia sessions
 309 (conferences) such as Internet telephony calls. SIP can also invite participants to already existing sessions,
 310 such as multicast conferences. Media can be added to (and removed from) an existing session. SIP trans-
 311 parently supports name mapping and redirection services, which supports *personal mobility* [27] - users can
 312 maintain a single externally visible identifier regardless of their network location.

313 SIP supports five facets of establishing and terminating multimedia communications:

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

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

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

317 **Session setup:** “ringing”, establishment of session parameters at both called and calling party;

318 **Session management:** including transfer and termination of sessions, modifying session parameters, and
 319 invoking services.

320 SIP is not a vertically integrated communications system. SIP is rather a component that can be used with
321 other IETF protocols to build a complete multimedia architecture. Typically, these architectures will include
322 protocols such as the real-time transport protocol (RTP) (RFC 1889 [28]) for transporting real-time data and
323 providing QoS feedback, the real-time streaming protocol (RTSP) (RFC 2326 [29]) for controlling delivery
324 of streaming media, the Media Gateway Control Protocol (MEGACO) (RFC 3015 [30]) for controlling
325 gateways to the Public Switched Telephone Network (PSTN), and the session description protocol (SDP)
326 (RFC 2327 [1]) for describing multimedia sessions. Therefore, SIP should be used in conjunction with other
327 protocols in order to provide complete services to the users. However, the basic functionality and operation
328 of SIP does not depend on any of these protocols.

329 SIP does not provide services. SIP rather provides primitives that can be used to implement different
330 services. For example, SIP can locate a user and deliver an opaque object to his current location. If this
331 primitive is used to deliver a session description written in SDP, for instance, the endpoints can agree on the
332 parameters of a session. If the same primitive is used to deliver a photo of the caller as well as the session
333 description, a "caller ID" service can be easily implemented. As this example shows, a single primitive is
334 typically used to provide several different services.

335 SIP does not offer conference control services such as floor control or voting and does not prescribe how
336 a conference is to be managed. SIP can be used to initiate a session that uses some other conference control
337 protocol. Since SIP messages and the sessions they establish can pass through entirely different networks,
338 SIP cannot, and does not, provide any kind of network resource reservation capabilities.

339 The nature of the services provided make security particularly important. To that end, SIP provides a
340 suite of security services, which include denial-of-service prevention, authentication (both user to user and
341 proxy to user), integrity protection, and encryption and privacy services.

342 SIP works with both IPv4 and IPv6.

343 **3 Terminology**

344 In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD",
345 "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" are to be inter-
346 preted as described in RFC 2119 [2] and indicate requirement levels for compliant SIP implementations.

347 **4 Overview of Operation**

348 This section introduces the basic operations of SIP using simple examples. This section is tutorial in nature
349 and does not contain any normative statements.

350 The first example shows the basic functions of SIP: location of an end point, signal of a desire to com-
351 municate, negotiation of session parameters to establish the session, and teardown of the session once es-
352 tablished.

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

359 Alice "calls" Bob using his SIP identity, a type of Uniform Resource Identifier (URI) called a *SIP*

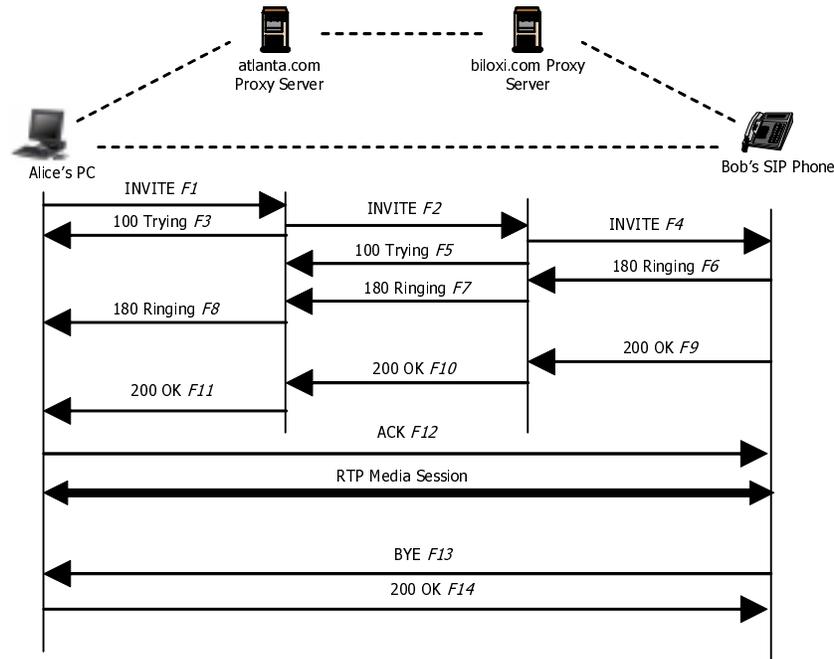


Figure 1: SIP session setup example with SIP trapezoid

360 *URI* and which is defined in Section 19.1. It has a similar form to an email address, typically containing
 361 a username and a host name. In this case, it is sip:bob@biloxi.com, where biloxi.com is the domain of
 362 Bob's SIP service provider (which can be an enterprise, retail provider, etc). Alice also has a SIP URI
 363 of sip:alice@atlanta.com. Alice might have typed in Bob's URI or perhaps clicked on a hyperlink or
 364 an entry in an address book. SIP also provides a secure URI, called a SIPS URI. An example would be
 365 sips:bob@biloxi.com. A call made to a SIPS URI guarantees that secure, encrypted transport (namely TLS)
 366 is used to carry all SIP messages from the caller to the domain of the callee. From there, the request is sent
 367 securely to the callee, but with security mechanisms that depend on the policy of the domain of the callee.

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

```

376 INVITE sip:bob@biloxi.com SIP/2.0
377 Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds
378 Max-Forwards: 70
379 To: Bob <sip:bob@biloxi.com>
380 From: Alice <sip:alice@atlanta.com>;tag=1928301774
381 Call-ID: a84b4c76e66710@pc33.atlanta.com

```

```
382   CSeq: 314159 INVITE
383   Contact: <sip:alice@pc33.atlanta.com>
384   Content-Type: application/sdp
385   Content-Length: 142
386
387   (Alice's SDP not shown)
```

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

391 **Via** contains the address (pc33.atlanta.com) at which Alice is expecting to receive responses to this
392 request. It also contains a branch parameter that contains an identifier for this transaction.

393 **To** contains a display name (Bob) and a SIP or SIPS URI (sip:bob@biloxi.com) towards which the
394 request was originally directed. Display names are described in RFC 2822 [3].

395 **From** also contains a display name (Alice) and a SIP or SIPS URI (sip:alice@atlanta.com) that indicate
396 the originator of the request. This header field also has a **tag** parameter containing a pseudorandom string
397 (1928301774) that was added to the URI by the softphone. It is used for identification purposes.

398 **Call-ID** contains a globally unique identifier for this call, generated by the combination of a pseudoran-
399 dom string and the softphone's IP address. The combination of the **To** tag, **From** tag, and **Call-ID** completely
400 define a peer-to-peer SIP relationship between Alice and Bob and is referred to as a *dialog*.

401 **CSeq** or Command Sequence contains an integer and a method name. The **CSeq** number is incremented
402 for each new request within a dialog and is a traditional sequence number.

403 **Contact** contains a SIP or SIPS URI that represents a direct route to contact Alice, usually composed
404 of a username at a fully qualified domain name (FQDN). While an FQDN is preferred, many end systems
405 do not have registered domain names, so IP addresses are permitted. While the **Via** header field tells other
406 elements where to send the response, the **Contact** header field tells other elements where to send future
407 requests.

408 **Max-Forwards** serves to limit the number of hops a request can make on the way to its destination. It
409 consists of an integer that is decremented by one at each hop.

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

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

412 The complete set of SIP header fields is defined in Section 20.

413 The details of the session, type of media, codec, sampling rate, etc. are not described using SIP. Rather,
414 the body of a SIP message contains a description of the session, encoded in some other protocol format.
415 One such format is the Session Description Protocol (SDP) (RFC 2327 [1]). This SDP message (not shown
416 in the example) is carried by the SIP message in a way that is analogous to a document attachment being
417 carried by an email message, or a web page being carried in an HTTP message.

418 Since the softphone does not know the location of Bob or the SIP server in the biloxi.com domain, the
419 softphone sends the INVITE to the SIP server that serves Alice's domain, atlanta.com. The address of the
420 atlanta.com SIP server could have been configured in Alice's softphone, or it could have been discovered by
421 DHCP, for example.

422 The atlanta.com SIP server is a type of SIP server known as a proxy server. A proxy server receives
423 SIP requests and forwards them on behalf of the requestor. In this example, the proxy server receives the
424 INVITE request and sends a 100 (Trying) response back to Alice's softphone. The 100 (Trying) response
425 indicates that the INVITE has been received and that the proxy is working on her behalf to route the INVITE

426 to the destination. Responses in SIP use a three-digit code followed by a descriptive phrase. This response
427 contains the same To, From, Call-ID, CSeq and branch parameter in the Via as the INVITE, which allows
428 Alice's softphone to correlate this response to the sent INVITE. The atlanta.com proxy server locates the
429 proxy server at biloxi.com, possibly by performing a particular type of DNS (Domain Name Service) lookup
430 to find the SIP server that serves the biloxi.com domain. This is described in [4]. As a result, it obtains the
431 IP address of the biloxi.com proxy server and forwards, or proxies, the INVITE request there. Before
432 forwarding the request, the atlanta.com proxy server adds an additional Via header field value that contains
433 its own address (the INVITE already contains Alice's address in the first Via). The biloxi.com proxy server
434 receives the INVITE and responds with a 100 (Trying) response back to the atlanta.com proxy server to
435 indicate that it has received the INVITE and is processing the request. The proxy server consults a database,
436 generically called a location service, that contains the current IP address of Bob. (We shall see in the next
437 section how this database can be populated.) The biloxi.com proxy server adds another Via header field
438 value with its own address to the INVITE and proxies it to Bob's SIP phone.

439 Bob's SIP phone receives the INVITE and alerts Bob to the incoming call from Alice so that Bob can
440 decide whether to answer the call, that is, Bob's phone rings. Bob's SIP phone indicates this in a 180
441 (Ringing) response, which is routed back through the two proxies in the reverse direction. Each proxy uses
442 the Via header field to determine where to send the response and removes its own address from the top.
443 As a result, although DNS and location service lookups were required to route the initial INVITE, the 180
444 (Ringing) response can be returned to the caller without lookups or without state being maintained in the
445 proxies. This also has the desirable property that each proxy that sees the INVITE will also see all responses
446 to the INVITE.

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

449 In this example, Bob decides to answer the call. When he picks up the handset, his SIP phone sends a
450 200 (OK) response to indicate that the call has been answered. The 200 (OK) contains a message body with
451 the SDP media description of the type of session that Bob is willing to establish with Alice. As a result, there
452 is a two-phase exchange of SDP messages: Alice sent one to Bob, and Bob sent one back to Alice. This
453 two-phase exchange provides basic negotiation capabilities and is based on a simple offer/answer model of
454 SDP exchange. If Bob did not wish to answer the call or was busy on another call, an error response would
455 have been sent instead of the 200 (OK), which would have resulted in no media session being established.
456 The complete list of SIP response codes is in Section 21. The 200 (OK) (message F9 in Figure 1) might
457 look like this as Bob sends it out:

```
458 SIP/2.0 200 OK
459 Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bKnashds8
460     ;received=192.0.2.3
461 Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
462     ;received=192.0.2.2
463 Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds
464     ;received=192.0.2.1
465 To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
466 From: Alice <sip:alice@atlanta.com>;tag=1928301774
467 Call-ID: a84b4c76e66710
468 CSeq: 314159 INVITE
469 Contact: <sip:bob@192.0.2.4>
```

470 Content-Type: application/sdp

471 Content-Length: 131

472

473 (Bob's SDP not shown)

474 The first line of the response contains the response code (200) and the reason phrase (OK). The remain-
475 ing lines contain header fields. The Via, To, From, Call-ID, and CSeq header fields are copied from the
476 INVITE request. (There are three Via header field values - one added by Alice's SIP phone, one added by
477 the atlanta.com proxy, and one added by the biloxi.com proxy.) Bob's SIP phone has added a tag parameter
478 to the To header field. This tag will be incorporated by both endpoints into the dialog and will be included
479 in all future requests and responses in this call. The Contact header field contains a URI at which Bob can
480 be directly reached at his SIP phone. The Content-Type and Content-Length refer to the message body
481 (not shown) that contains Bob's SDP media information.

482 In addition to DNS and location service lookups shown in this example, proxy servers can make flexible
483 "routing decisions" to decide where to send a request. For example, if Bob's SIP phone returned a 486 (Busy
484 Here) response, the biloxi.com proxy server could proxy the INVITE to Bob's voicemail server. A proxy
485 server can also send an INVITE to a number of locations at the same time. This type of parallel search is
486 known as *forking*.

487 In this case, the 200 (OK) is routed back through the two proxies and is received by Alice's softphone,
488 which then stops the ringback tone and indicates that the call has been answered. Finally, Alice's softphone
489 sends an acknowledgement message, ACK to Bob's SIP phone to confirm the reception of the final response
490 (200 (OK)). In this example, the ACK is sent directly from Alice's softphone to Bob's SIP phone, bypassing
491 the two proxies. This occurs because the endpoints have learned each other's address from the Contact
492 header fields through the INVITE/200 (OK) exchange, which was not known when the initial INVITE was
493 sent. The lookups performed by the two proxies are no longer needed, so the proxies drop out of the call
494 flow. This completes the INVITE/200/ACK three-way handshake used to establish SIP sessions. Full details
495 on session setup are in Section 13.

496 Alice and Bob's media session has now begun, and they send media packets using the format to which
497 they agreed in the exchange of SDP. In general, the end-to-end media packets take a different path from the
498 SIP signaling messages.

499 During the session, either Alice or Bob may decide to change the characteristics of the media session.
500 This is accomplished by sending a re-INVITE containing a new media description. This re-INVITE refer-
501 ences the existing dialog so that the other party knows that it is to modify an existing session instead of
502 establishing a new session. The other party sends a 200 (OK) to accept the change. The requestor responds
503 to the 200 (OK) with an ACK. If the other party does not accept the change, he sends an error response such
504 as 406 (Not Acceptable), which also receives an ACK. However, the failure of the re-INVITE does not cause
505 the existing call to fail - the session continues using the previously negotiated characteristics. Full details on
506 session modification are in Section 14.

507 At the end of the call, Bob disconnects (hangs up) first and generates a BYE message. This BYE is
508 routed directly to Alice's softphone, again bypassing the proxies. Alice confirms receipt of the BYE with a
509 200 (OK) response, which terminates the session and the BYE transaction. No ACK is sent - an ACK is only
510 sent in response to a response to an INVITE request. The reasons for this special handling for INVITE will
511 be discussed later, but relate to the reliability mechanisms in SIP, the length of time it can take for a ringing
512 phone to be answered, and forking. For this reason, request handling in SIP is often classified as either
513 INVITE or non-INVITE, referring to all other methods besides INVITE. Full details on session termination

514 are in Section 15.

515 Full details of all the messages shown in the example of Figure 1 are shown in Section 24.2.

516 In some cases, it may be useful for proxies in the SIP signaling path to see all the messaging between the
517 endpoints for the duration of the session. For example, if the biloxi.com proxy server wished to remain in the
518 SIP messaging path beyond the initial INVITE, it would add to the INVITE a required routing header field
519 known as **Record-Route** that contained a URI resolving to the hostname or IP address of the proxy. This
520 information would be received by both Bob's SIP phone and (due to the **Record-Route** header field being
521 passed back in the 200 (OK)) Alice's softphone and stored for the duration of the dialog. The biloxi.com
522 proxy server would then receive and proxy the ACK, BYE, and 200 (OK) to the BYE. Each proxy can
523 independently decide to receive subsequent messaging, and that messaging will go through all proxies that
524 elect to receive it. This capability is frequently used for proxies that are providing mid-call features.

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

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

537 The location service is just an abstract concept. It generally contains information that allows a proxy to
538 input a URI and receive a set of zero or more URIs that tell the proxy where to send the request. Registrations
539 are one way to create this information, but not the only way. Arbitrary mapping functions can be configured
540 at the discretion of the administrator.

541 Finally, it is important to note that in SIP, registration is used for routing incoming SIP requests and
542 has no role in authorizing outgoing requests. Authorization and authentication are handled in SIP either
543 on a request-by-request basis with a challenge/response mechanism, or by using a lower layer scheme as
544 discussed in Section 26.

545 The complete set of SIP message details for this registration example is in Section 24.1.

546 Additional operations in SIP, such as querying for the capabilities of a SIP server or client using **OP-**
547 **TIONS**, or canceling a pending request using **CANCEL**, will be introduced in later sections.

548 5 Structure of the Protocol

549 SIP is structured as a layered protocol, which means that its behavior is described in terms of a set of fairly
550 independent processing stages with only a loose coupling between each stage. The protocol behavior is
551 described as layers for the purpose of presentation, allowing the description of functions common across
552 elements in a single section. It does not dictate an implementation in any way. When we say that an element
553 "contains" a layer, we mean it is compliant to the set of rules defined by that layer.

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

557 The lowest layer of SIP is its syntax and encoding. Its encoding is specified using an augmented Backus-
558 Naur Form grammar (BNF). The complete BNF is specified in Section 25; an overview of a SIP message's
559 structure can be found in Section 7.

560 The second layer is the transport layer. It defines how a client sends requests and receives responses and
561 how a server receives requests and sends responses over the network. All SIP elements contain a transport
562 layer. The transport layer is described in Section 18.

563 The third layer is the transaction layer. Transactions are a fundamental component of SIP. A transaction
564 is a request sent by a client transaction (using the transport layer) to a server transaction, along with all
565 responses to that request sent from the server transaction back to the client. The transaction layer handles
566 application-layer retransmissions, matching of responses to requests, and application-layer timeouts. Any
567 task that a user agent client (UAC) accomplishes takes place using a series of transactions. Discussion of
568 transactions can be found in Section 17. User agents contain a transaction layer, as do stateful proxies.
569 Stateless proxies do not contain a transaction layer. The transaction layer has a client component (referred
570 to as a client transaction) and a server component (referred to as a server transaction), each of which are
571 represented by a finite state machine that is constructed to process a particular request.

572 The layer above the transaction layer is called the transaction user (TU). Each of the SIP entities, except
573 the stateless proxy, is a transaction user. When a TU wishes to send a request, it creates a client transaction
574 instance and passes it the request along with the destination IP address, port, and transport to which to send
575 the request. A TU that creates a client transaction can also cancel it. When a client cancels a transaction,
576 it requests that the server stop further processing, revert to the state that existed before the transaction was
577 initiated, and generate a specific error response to that transaction. This is done with a CANCEL request,
578 which constitutes its own transaction, but references the transaction to be cancelled (Section 9).

579 The SIP elements, that is, user agent clients and servers, stateless and stateful proxies and registrars,
580 contain a *core* that distinguishes them from each other. Cores, except for the stateless proxy, are transaction
581 users. While the behavior of the UAC and UAS cores depends on the method, there are some common rules
582 for all methods (Section 8). For a UAC, these rules govern the construction of a request; for a UAS, they
583 govern the processing of a request and generating a response. Since registrations play an important role in
584 SIP, a UAS that handles a REGISTER is given the special name registrar. Section 10 describes UAC and
585 UAS core behavior for the REGISTER method. Section 11 describes UAC and UAS core behavior for the
586 OPTIONS method, used for determining the capabilities of a UA.

587 Certain other requests are sent within a dialog. A dialog is a peer-to-peer SIP relationship between two
588 user agents that persists for some time. The dialog facilitates sequencing of messages and proper routing
589 of requests between the user agents. The INVITE method is the only way defined in this specification to
590 establish a dialog. When a UAC sends a request that is within the context of a dialog, it follows the common
591 UAC rules as discussed in Section 8 but also the rules for mid-dialog requests. Section 12 discusses dialogs
592 and presents the procedures for their construction and maintenance, in addition to construction of requests
593 within a dialog.

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

599 The procedures of Sections 8, 10, 11, 12, 13, 14, and 15 deal entirely with the UA core (Section 9
600 describes cancellation, which applies to both UA core and proxy core). Section 16 discusses the proxy
601 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 terms and generic syntax of URI and URL are defined in RFC 2396 [5]. The following terms have special significance for SIP.

Address-of-Record: An address-of-record (AOR) is a SIP or SIPS URI that points to a domain with a location service that can map the URI to another URI where the user might be available. Typically, the location service is populated through registrations. An AOR is frequently thought of as the “public address” of the user.

Back-to-Back User Agent: A back-to-back user agent (B2BUA) is a logical entity that receives a request and processes it as an user agent server (UAS). In order to determine how the request should be answered, it acts as an user agent client (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 some communication between peers generally set up for the purposes of a multimedia conversation.

Call Leg: Another name for a dialog [31]; no longer used in this specification.

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 transaction stateful, but the converse is not necessarily 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.

Core: Core designates the functions specific to a particular type of SIP entity, i.e., specific to either a stateful or stateless proxy, a user agent or registrar. All cores except those for the stateless proxy are transaction users.

Dialog: A dialog is a peer-to-peer SIP relationship between two 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 tag, and a remote tag. A dialog was formerly known as a call leg in RFC 2543.

Downstream: A direction of message forwarding within a transaction that 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.

Header: A header is a component of a SIP message that conveys information about the message. It is structured as a sequence of header fields.

Header field: A header field is a component of the SIP message header. It consists of one or more header field values separated by comma or having the same header field name.

638 **Header field value:** A header field value is a singular value, which can be one of many in a header field.

639 **Home Domain:** The domain providing service to a SIP user. Typically, this is the domain present in the
640 URI in the address-of-record of a registration.

641 **Informational Response:** Same as a provisional response.

642 **Initiator, Calling Party, Caller:** The party initiating a session (and dialog) with an INVITE request. A
643 caller retains this role from the time it sends the initial INVITE that established a dialog until the
644 termination of that dialog.

645 **Invitation:** An INVITE request.

646 **Invitee, Invited User, Called Party, Callee:** The party that receives an INVITE request for the purposes of
647 establishing a new session. A callee retains this role from the time it receives the INVITE until the
648 termination of the dialog established by that INVITE.

649 **Location Service:** A location service is used by a SIP redirect or proxy server to obtain information about
650 a callee's possible location(s). It contains a list of bindings of address-of-record keys to zero or more
651 contact addresses. The bindings can be created and removed in many ways; this specification defines
652 a REGISTER method that updates the bindings.

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

658 **Loose Routing:** A proxy is said to be loose routing if it follows the procedures defined in this specification
659 for processing of the Route header field. These procedures separate the destination of the request
660 (present in the Request-URI) from the set of proxies that need to be visited along the way (present
661 in the Route header field). A proxy compliant to these mechanisms is also known as a loose router.

662 **Message:** Data sent between SIP elements as part of the protocol. SIP messages are either requests or
663 responses.

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

666 **Outbound Proxy:** A *proxy* that receives requests from a client, even though it may not be the server re-
667 solved by the Request-URI. Typically, a UA is manually configured with an outbound proxy, or can
668 learn about one through auto-configuration protocols.

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

- 673 **Provisional Response:** A response used by the server to indicate progress, but that does not terminate a SIP
674 transaction. 1xx responses are provisional, other responses are considered *final*. Provisional responses
675 are not sent reliably.
- 676 **Proxy, Proxy Server:** An intermediary entity that acts as both a server and a client for the purpose of
677 making requests on behalf of other clients. A proxy server primarily plays the role of routing, which
678 means its job is to ensure that a request is sent to another entity “closer” to the targeted user. Proxies
679 are also useful for enforcing policy (for example, making sure a user is allowed to make a call). A
680 proxy interprets, and, if necessary, rewrites specific parts of a request message before forwarding it.
- 681 **Recursion:** A client recurses on a 3xx response when it generates a new request to one or more of the URIs
682 in the Contact header field in the response.
- 683 **Redirect Server:** A redirect server is a user agent server that generates 3xx responses to requests it receives,
684 directing the client to contact an alternate set of URIs.
- 685 **Registrar:** A registrar is a server that accepts REGISTER requests and places the information it receives
686 in those requests into the location service for the domain it handles.
- 687 **Regular Transaction:** A regular transaction is any transaction with a method other than INVITE, ACK, or
688 CANCEL.
- 689 **Request:** A SIP message sent from a client to a server, for the purpose of invoking a particular operation.
- 690 **Response:** A SIP message sent from a server to a client, for indicating the status of a request sent from the
691 client to the server.
- 692 **Ringback:** Ringback is the signaling tone produced by the calling party’s application indicating that a
693 called party is being alerted (ringing).
- 694 **Route Set:** A route set is a collection of ordered SIP or SIPS URI which represent a list of proxies that
695 must be traversed when sending a particular request. A route set can be learned, through headers like
696 Record-Route, or it can be configured.
- 697 **Server:** A server is a network element that receives requests in order to service them and sends back re-
698 sponses to those requests. Examples of servers are proxies, user agent servers, redirect servers, and
699 registrars.
- 700 **Sequential Search:** In a sequential search, a proxy server attempts each contact address in sequence, pro-
701 ceeding to the next one only after the previous has generated a final response. A 2xx or 6xx class final
702 response always terminates a sequential search.
- 703 **Session:** From the SDP specification: “A multimedia session is a set of multimedia senders and receivers
704 and the data streams flowing from senders to receivers. A multimedia conference is an example of a
705 multimedia session.” (RFC 2327 [1]) (A session as defined for SDP can comprise one or more RTP
706 sessions.) As defined, a callee can be invited several times, by different calls, to the same session. If
707 SDP is used, a session is defined by the concatenation of the *SDP user name*, *session id*, *network type*,
708 *address type*, and *address* elements in the origin field.

- 709 **SIP Transaction:** A SIP transaction occurs between a client and a server and comprises all messages from
710 the first request sent from the client to the server up to a final (non-1xx) response sent from the server
711 to the client. If the request is INVITE and the final response is a non-2xx, the transaction also includes
712 an ACK to the response. The ACK for a 2xx response to an INVITE request is a separate transaction.
- 713 **Spiral:** A spiral is a SIP request that is routed to a proxy, forwarded onwards, and arrives once again at that
714 proxy, but this time differs in a way that will result in a different processing decision than the original
715 request. Typically, this means that the request's Request-URI differs from its previous arrival. A
716 spiral is not an error condition, unlike a loop. A typical cause for this is call forwarding. A user calls
717 joe@example.com. The example.com proxy forwards it to Joe's PC, which in turn, forwards it to
718 bob@example.com. This request is proxied back to the example.com proxy. However, this is not a
719 loop. Since the request is targeted at a different user, it is considered a spiral, and is a valid condition.
- 720 **Stateful Proxy:** A logical entity that maintains the client and server transaction state machines defined by
721 this specification during the processing of a request. Also known as a transaction stateful proxy. The
722 behavior of a stateful proxy is further defined in Section 16. A (transaction) stateful proxy is not the
723 same as a call stateful proxy.
- 724 **Stateless Proxy:** A logical entity that does not maintain the client or server transaction state machines
725 defined in this specification when it processes requests. A stateless proxy forwards every request it
726 receives downstream and every response it receives upstream.
- 727 **Strict Routing:** A proxy is said to be strict routing if it follows the Route processing rules of RFC 2543
728 and many prior Internet Draft versions of this RFC. That rule caused proxies to destroy the contents of
729 the Request-URI when a Route header field was present. Strict routing behavior is not used in this
730 specification, in favor of a loose routing behavior. Proxies that perform strict routing are also known
731 as strict routers.
- 732 **Target Refresh Request:** A target refresh request sent within a dialog is defined as a request that can
733 modify the remote target of the dialog.
- 734 **Transaction User (TU):** The layer of protocol processing that resides above the transaction layer. Trans-
735 action users include the UAC core, UAS core, and proxy core.
- 736 **Upstream:** A direction of message forwarding within a transaction that refers to the direction that responses
737 flow from the user agent server back to the user agent client.
- 738 **URL-encoded:** A character string encoded according to RFC 1738, Section 2.2 [6].
- 739 **User Agent Client (UAC):** A user agent client is a logical entity that creates a new request, and then uses
740 the client transaction state machinery to send it. The role of UAC lasts only for the duration of that
741 transaction. In other words, if a piece of software initiates a request, it acts as a UAC for the duration
742 of that transaction. If it receives a request later, it assumes the role of a user agent server for the
743 processing of that transaction.
- 744 **UAC Core:** The set of processing functions required of a UAC that reside above the transaction and trans-
745 port layers.

746 **User Agent Server (UAS):** A user agent server is a logical entity that generates a response to a SIP request.
 747 The response accepts, rejects, or redirects the request. This role lasts only for the duration of that
 748 transaction. In other words, if a piece of software responds to a request, it acts as a UAS for the
 749 duration of that transaction. If it generates a request later, it assumes the role of a user agent client for
 750 the processing of that transaction.

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

753 **User Agent (UA):** A logical entity that can act as both a user agent client and user agent server.

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

758 Proxy, location, and registrar servers defined above are *logical* entities; implementations MAY combine
 759 them into a single application.

760 7 SIP Messages

761 SIP is a text-based protocol and uses the UTF-8 charset (RFC 2279 [7]).

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

763 Both **Request** (section 7.1) and **Response** (section 7.2) messages use the basic format of RFC 2822 [3],
 764 even though the syntax differs in character set and syntax specifics. (SIP allows header fields that would not
 765 be valid RFC 2822 header fields, for example.) Both types of messages consist of a **start-line**, one or more
 766 header fields, an empty line indicating the end of the header fields, and an optional **message-body**.

```

767     generic-message = start-line
                       *message-header
                       CRLF
                       [ message-body ]
     start-line      = Request-Line / Status-Line
```

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

770 Except for the above difference in character sets, much of SIP's message and header field syntax is
 771 identical to HTTP/1.1. Rather than repeating the syntax and semantics here, we use [HX.Y] to refer to
 772 Section X.Y of the current HTTP/1.1 specification (RFC 2616 [8]).

773 However, SIP is not an extension of HTTP.

774 7.1 Requests

775 SIP requests are distinguished by having a **Request-Line** for a **start-line**. A **Request-Line** contains a
 776 method name, a **Request-URI**, and the protocol version separated by a single space (SP) character.

777 The **Request-Line** ends with CRLF. No CR or LF are allowed except in the end-of-line CRLF se-
 778 quence. No linear whitespace (LWS) is allowed in any of the elements.

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

780 **Method:** This specification defines six methods: REGISTER for registering contact information, INVITE,
781 ACK, and CANCEL for setting up sessions, BYE for terminating sessions, and OPTIONS for query-
782 ing servers about their capabilities. SIP extensions, documented in standards track RFCs, may define
783 additional methods.

784 **Request-URI:** The Request-URI is a SIP or SIPS URI as described in Section 19.1 or a general URI
785 (RFC 2396 [5]). It indicates the user or service to which this request is being addressed. The Request-
786 URI MUST NOT contain unescaped spaces or control characters and MUST NOT be enclosed in "<>".
787 SIP elements MAY support Request-URIs with schemes other than "sip" and "sips", for example the
788 "tel" URI scheme of RFC 2806 [9]. SIP elements MAY translate non-SIP URIs using any mechanism
789 at their disposal, resulting in either SIP URI, SIPS URI, or some other scheme.

790 **SIP-Version:** Both request and response messages include the version of SIP in use, and follow [H3.1] (with
791 HTTP replaced by SIP, and HTTP/1.1 replaced by SIP/2.0) regarding version ordering, compliance
792 requirements, and upgrading of version numbers. To be compliant with this specification, applications
793 sending SIP messages MUST include a SIP-Version of "SIP/2.0". The SIP-Version string is case-
794 insensitive, but implementations MUST send upper-case.

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

797 7.2 Responses

798 SIP responses are distinguished from requests by having a Status-Line as their start-line. A Status-Line
799 consists of the protocol version followed by a numeric Status-Code and its associated textual phrase, with
800 each element separated by a single SP character.

801 No CR or LF is allowed except in the final CRLF sequence.

802 Status-Line = SIP-Version SP Status-Code SP Reason-Phrase CRLF

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

807 While this specification suggests specific wording for the reason phrase, implementations MAY choose
808 other text, for example, in the language indicated in the Accept-Language header field of the request.

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

813 **1xx:** Provisional – request received, continuing to process the request;

814 **2xx:** Success – the action was successfully received, understood, and accepted;

815 **3xx:** Redirection – further action needs to be taken in order to complete the request;

816 **4xx:** Client Error – the request contains bad syntax or cannot be fulfilled at this server;

817 **5xx:** Server Error – the server failed to fulfill an apparently valid request;

818 **6xx:** Global Failure – the request cannot be fulfilled at any server.

819 Section 21 defines these classes and describes the individual codes.

820 7.3 Header Fields

821 SIP header fields are similar to HTTP header fields in both syntax and semantics. In particular, SIP header
822 fields follow the [H4.2] definitions of syntax for **message-header** and the rules for extending header fields
823 over multiple lines. However, the latter is specified in HTTP with implicit whitespace and folding. This
824 specification conforms with RFC 2234 [10] and uses only explicit whitespace and folding as an integral part
825 of the grammar.

826 [H4.2] also specifies that multiple header fields of the same field name whose value is a comma-separated
827 list can be combined into one header field. That applies to SIP as well, but the specific rule is different
828 because of the different grammars. Specifically, any SIP header whose grammar is of the form:

829 `header = "header-name" HCOLON header-value *(COMMA header-value)`

830 allows for combining header fields of the same name into a comma-separated list. This is also true for
831 the **Contact** header, as long as none of the header field values are “*”.

832 7.3.1 Header Field Format

833 Header fields follow the same generic header format as that given in Section 2.2 of RFC 2822 [3]. Each
834 header field consists of a field name followed by a colon (":") and the field value.

835 `field-name: field-value`

836 The formal grammar for a **message-header** specified in Section 25 allows for an arbitrary amount of
837 whitespace on either side of the colon; however, implementations should avoid spaces between the field
838 name and the colon and use a single space (SP) between the colon and the **field-value**. Thus,

839 `Subject: lunch`

840 `Subject : lunch`

841 `Subject :lunch`

842 `Subject: lunch`

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

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

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

848 `Subject: I know you're there,`

849 `pick up the phone`

850 `and talk to me!`

851 The relative order of header fields with different field names is not significant. However, it is RECOM-
852 MENDED that header fields which are needed for proxy processing (Via, Route, Record-Route, Proxy-
853 Require, Max-Forwards, and Proxy-Authorization, for example) appear towards the top of the message
854 to facilitate rapid parsing. The relative order of header field rows with the same field name is important.
855 Multiple header field rows with the same field-name MAY be present in a message if and only if the entire
856 field-value for that header field is defined as a comma-separated list (that is, if follows the grammar defined
857 in Section 7.3). It MUST be possible to combine the multiple header field rows into one "field-name: field-
858 value" pair, without changing the semantics of the message, by appending each subsequent field-value to
859 the first, each separated by a comma. The exceptions to this rule are the WWW-Authenticate, Authoriza-
860 tion, Proxy-Authenticate, and Proxy-Authorization header fields. Multiple header field rows with these
861 names MAY be present in a message, but since their grammar does not follow the general form listed in
862 Section 7.3, they MUST NOT be combined into a single header field row.

863 Implementations MUST be able to process multiple header field rows with the same name in any combi-
864 nation of the single-value-per-line or comma-separated value forms.

865 The following groups of header field rows are valid and equivalent:

```
866 Route: <sip:alice@atlanta.com>
867 Subject: Lunch
868 Route: <sip:bob@biloxi.com>
869 Route: <sip:carol@chicago.com>
870
871 Route: <sip:alice@atlanta.com>, <sip:bob@biloxi.com>
872 Route: <sip:carol@chicago.com>
873 Subject: Lunch
874
875 Subject: Lunch
876 Route: <sip:alice@atlanta.com>, <sip:bob@biloxi.com>, <sip:carol@chicago.com>
```

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

```
878 Route: <sip:alice@atlanta.com>
879 Route: <sip:bob@biloxi.com>
880 Route: <sip:carol@chicago.com>
881
882 Route: <sip:bob@biloxi.com>
883 Route: <sip:alice@atlanta.com>
884 Route: <sip:carol@chicago.com>
885
886 Route: <sip:alice@atlanta.com>, <sip:carol@chicago.com>, <sip:bob@biloxi.com>
```

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

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

892 Even though an arbitrary number of parameter pairs may be attached to a header field value, any given
893 **parameter-name** MUST NOT appear more than once.

894 When comparing header fields, field names are always case-insensitive. Unless otherwise stated in
895 the definition of a particular header field, field values, parameter names, and parameter values are case-
896 insensitive. Tokens are always case-insensitive. Unless specified otherwise, values expressed as quoted
897 strings are case-sensitive.

898 For example,

899 Contact: <sip:alice@atlanta.com>;expires=3600

900 is equivalent to

901 CONTACT: <sip:alice@atlanta.com>;EXPIRES=3600

902 and

903 Content-Disposition: session;handling=optional

904 is equivalent to

905 content-disposition: Session;HANDLING=OPTIONAL

906 The following two header fields are not equivalent:

907 Warning: 370 devnull "Choose a bigger pipe"

908 Warning: 370 devnull "CHOOSE A BIGGER PIPE"

909 7.3.2 Header Field Classification

910 Some header fields only make sense in requests or responses. These are called request header fields and
911 response header fields, respectively. If a header field appears in a message not matching its category (such
912 as a request header field in a response), it MUST be ignored. Section 20 defines the classification of each
913 header field.

914 7.3.3 Compact Form

915 SIP provides a mechanism to represent common header field names in an abbreviated form. This may
916 be useful when messages would otherwise become too large to be carried on the transport available to it
917 (exceeding the maximum transmission unit (MTU) when using UDP, for example). These compact forms
918 are defined in Section 20. A compact form MAY be substituted for the longer form of a header field name at
919 any time without changing the semantics of the message. A header field name MAY appear in both long and
920 short forms within the same message. Implementations MUST accept both the long and short forms of each
921 header name.

922 7.4 Bodies

923 Requests, including new requests defined in extensions to this specification, MAY contain message bodies
924 unless otherwise noted. The interpretation of the body depends on the request method.

925 For response messages, the request method and the response status code determine the type and inter-
926 pretation of any message body. All responses MAY include a body.

927 7.4.1 Message Body Type

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

932 The "multipart" MIME type defined in RFC 2046 [11] MAY be used within the body of the message.
933 Implementations that send requests containing multipart message bodies MUST send a session description
934 as a non-multipart message body if the remote implementation requests this through an Accept header field
935 that does not contain multipart.

936 Note that SIP messages MAY contain binary bodies or body parts.

937 7.4.2 Message Body Length

938 The body length in bytes is provided by the Content-Length header field. Section 20.14 describes the
939 necessary contents of this header field in detail.

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

942 7.5 Framing SIP messages

943 Unlike HTTP, SIP implementations can use UDP or other unreliable datagram protocols. Each such data-
944 gram carries one request or response. See Section 18 on constraints on usage of unreliable transports.

945 Implementations processing SIP messages over stream-oriented transports MUST ignore any CRLF ap-
946 pearing before the start-line [H4.1].

947 The Content-Length header field value is used to locate the end of each SIP message in a stream. It will always
948 be present when SIP messages are sent over stream-oriented transports.

949 8 General User Agent Behavior

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

955 When a UAC sends a request, the request passes through some number of proxy servers, which forward
956 the request towards the UAS. When the UAS generates a response, the response is forwarded towards the
957 UAC.

958 UAC and UAS procedures depend strongly on two factors. First, based on whether the request or
959 response is inside or outside of a dialog, and second, based on the method of a request. Dialogs are discussed

960 thoroughly in Section 12; they represent a peer-to-peer relationship between user agents and are established
961 by specific SIP methods, such as INVITE.

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

965 Security procedures for requests and responses outside of a dialog are described in Section 26. Specif-
966 ically, mechanisms exist for the UAS and UAC to mutually authenticate. A limited set of privacy features
967 are also supported through encryption of bodies using S/MIME.

968 8.1 UAC Behavior

969 This section covers UAC behavior outside of a dialog.

970 8.1.1 Generating the Request

971 A valid SIP request formulated by a UAC MUST at a minimum contain the following header fields: To,
972 From, CSeq, Call-ID, Max-Forwards, and Via; all of these header fields are mandatory in all SIP mes-
973 sages. These six header fields are the fundamental building blocks of a SIP message, as they jointly provide
974 for most of the critical message routing services including the addressing of messages, the routing of re-
975 sponses, limiting message propagation, ordering of messages, and the unique identification of transactions.
976 These header fields are in addition to the mandatory request line, which contains the method, Request-URI,
977 and SIP version.

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

980 **8.1.1.1 Request-URI** The initial Request-URI of the message SHOULD be set to the value of the URI
981 in the To field. One notable exception is the REGISTER method; behavior for setting the Request-URI
982 of REGISTER is given in Section 10. It may also be undesirable for privacy reasons or convenience to
983 set these fields to the same value (especially if the originating UA expects that the Request-URI will be
984 changed during transit).

985 In some special circumstances, the presence of a pre-existing route set can affect the Request-URI of
986 the message. A pre-existing route set is an ordered set of URIs that identify a chain of servers, to which a
987 UAC will send outgoing requests that are outside of a dialog. Commonly, they are configured on the UA by
988 a user or service provider manually, or through some other non-SIP mechanism. When a provider wishes
989 to configure a UA with an outbound proxy, it is RECOMMENDED that this be done by providing it with a
990 pre-existing route set with a single URI, that of the outbound proxy.

991 When a pre-existing route set is present, the procedures for populating the Request-URI and Route
992 header field detailed in Section 12.2.1.1 MUST be followed, even though there is no dialog.

993 **8.1.1.2 To** The To header field first and foremost specifies the desired “logical” recipient of the request,
994 or the address-of-record of the user or resource that is the target of this request. This may or may not be
995 the ultimate recipient of the request. The To header field MAY contain a SIP or SIPS URI, but it may also
996 make use of other URI schemes (the tel URL (RFC 2806 [9]), for example) when appropriate. All SIP
997 implementations MUST support the SIP and URI scheme. Any implementation that supports TLS MUST
998 support the SIPS URI scheme. The To header field allows for a display name.

999 A UAC may learn how to populate the **To** header field for a particular request in a number of ways.
1000 Usually the user will suggest the **To** header field through a human interface, perhaps inputting the URI
1001 manually or selecting it from some sort of address book. Frequently, the user will not enter a complete URI,
1002 but rather a string of digits or letters (for example, "bob"). It is at the discretion of the UA to choose how
1003 to interpret this input. Using the string to form the user part of a SIP URI implies that the UA wishes the
1004 name to be resolved in the domain to the right-hand side (RHS) of the at-sign in the SIP URI (for instance,
1005 sip:bob@example.com). Using the string to form the user part of a SIPS URI implies that the UA wishes to
1006 communicate securely, and that the name is to be resolved in the domain to the RHS of the at-sign. The RHS
1007 will frequently be the home domain of the user, which allows for the home domain to process the outgoing
1008 request. This is useful for features like "speed dial" that require interpretation of the user part in the home
1009 domain. The tel URL may be used when the UA does not wish to specify the domain that should interpret a
1010 telephone number that has been inputted by the user. Rather, each domain through which the request passes
1011 would be given that opportunity. As an example, a user in an airport might log in and send requests through
1012 an outbound proxy in the airport. If they enter "411" (this is the phone number for local directory assistance
1013 in the United States), that needs to be interpreted and processed by the outbound proxy in the airport, not
1014 the user's home domain. In this case, tel:411 would be the right choice.

1015 A request outside of a dialog **MUST NOT** contain a tag; the tag in the **To** field of a request identifies the
1016 peer of the dialog. Since no dialog is established, no tag is present.

1017 For further information on the **To** header field, see Section 20.39. The following is an example of valid
1018 **To** header field:

```
1019 To: Carol <sip:carol@chicago.com>
```

1020 **8.1.1.3 From** The **From** header field indicates the logical identity of the initiator of the request, possibly
1021 the user's address-of-record. Like the **To** header field, it contains a URI and optionally a display name. It
1022 is used by SIP elements to determine which processing rules to apply to a request (for example, automatic
1023 call rejection). As such, it is very important that the **From** URI not contain IP addresses or the FQDN of the
1024 host on which the UA is running, since these are not logical names.

1025 The **From** header field allows for a display name. A UAC **SHOULD** use the display name "Anonymous",
1026 along with a syntactically correct, but otherwise meaningless URI (like sip:thisis@anonymous.invalid), if
1027 the identity of the client is to remain hidden.

1028 Usually the value that populates the **From** header field in requests generated by a particular UA is pre-
1029 provisioned by the user or by the administrators of the user's local domain. If a particular UA is used by
1030 multiple users, it might have switchable profiles that include a URI corresponding to the identity of the
1031 profiled user. Recipients of requests can authenticate the originator of a request in order to ascertain that
1032 they are who their **From** header field claims they are (see Section 22 for more on authentication).

1033 The **From** field **MUST** contain a new "tag" parameter, chosen by the UAC. See Section 19.3 for details
1034 on choosing a tag.

1035 For further information on the **From** header field, see Section 20.20. Examples:

```
1036 From: "Bob" <sips:bob@biloxi.com> ;tag=a48s  
1037 From: sip:+12125551212@phone2net.com;tag=887s  
1038 From: Anonymous <sip:c8oqz84zk7z@privacy.org>;tag=hyh8
```

1039 **8.1.1.4 Call-ID** The Call-ID header field acts as a unique identifier to group together a series of mes-
1040 sages. It MUST be the same for all requests and responses sent by either UA in a dialog. It SHOULD be the
1041 same in each registration from a UA.

1042 In a new request created by a UAC outside of any dialog, the Call-ID header field MUST be selected by
1043 the UAC as a globally unique identifier over space and time unless overridden by method-specific behavior.
1044 All SIP UAs must have a means to guarantee that the Call-ID header fields they produce will not be inad-
1045 vertently generated by any other UA. Note that when requests are retried after certain failure responses that
1046 solicit an amendment to a request (for example, a challenge for authentication), these retried requests are
1047 not considered new requests, and therefore do not need new Call-ID header fields; see Section 8.1.3.5.

1048 Use of cryptographically random identifiers (RFC 1750 [12]) in the generation of Call-IDs is RECOM-
1049 MENDED. Implementations MAY use the form "localid@host". Call-IDs are case-sensitive and are simply
1050 compared byte-by-byte.

1051 Using cryptographically random identifiers provides some protection against session hijacking and reduces the
1052 likelihood of unintentional Call-ID collisions.

1053 No provisioning or human interface is required for the selection of the Call-ID header field value for a
1054 request.

1055 For further information on the Call-ID header field, see Section 20.8.

1056 Example:

1057 Call-ID: f81d4fae-7dec-11d0-a765-00a0c91e6bf6@foo.bar.com

1058 **8.1.1.5 CSeq** The CSeq header field serves as a way to identify and order transactions. It consists
1059 of a sequence number and a method. The method MUST match that of the request. For non-REGISTER
1060 requests outside of a dialog, the sequence number value is arbitrary. The sequence number value MUST
1061 be expressible as a 32-bit unsigned integer and MUST be less than 2**31. As long as it follows the above
1062 guidelines, a client may use any mechanism it would like to select CSeq header field values.

1063 Section 12.2.1.1 discusses construction of the CSeq for requests within a dialog.

1064 Example:

1065 CSeq: 4711 INVITE

1066 **8.1.1.6 Max-Forwards** The Max-Forwards header field serves to limit the number of hops a request
1067 can transit on the way to its destination. It consists of an integer that is decremented by one at each hop.
1068 If the Max-Forwards value reaches 0 before the request reaches its destination, it will be rejected with a
1069 483(Too Many Hops) error response.

1070 A UAC MUST insert a Max-Forwards header field into each request it originates with a value which
1071 SHOULD be 70. This number was chosen to be sufficiently large to guarantee that a request would not be
1072 dropped in any SIP network when there were no loops, but not so large as to consume proxy resources when
1073 a loop does occur. Lower values should be used with caution and only in networks where topologies are
1074 known by the UA.

1075 **8.1.1.7 Via** The Via header field indicates the transport used for the transaction and identifies the location
1076 where the response is to be sent. A Via header field value is added only after the transport that will be used
1077 to reach the next hop has been selected (which may involve the usage of the procedures in [4]).

1078 When the UAC creates a request, it MUST insert a **Via** into that request. The protocol name and protocol
1079 version in the header field MUST be SIP and 2.0, respectively. The **Via** header field value MUST contain a
1080 branch parameter. This parameter is used to identify the transaction created by that request. This parameter
1081 is used by both the client and the server.

1082 The **branch** parameter value MUST be unique across space and time for all requests sent by the UA.
1083 The exceptions to this rule are **CANCEL** and **ACK** for non-2xx responses. As discussed below, a **CAN-**
1084 **CEL** request will have the same value of the **branch** parameter as the request it cancels. As discussed in
1085 Section 17.1.1.3, an **ACK** for a non-2xx response will also have the same **branch ID** as the **INVITE** whose
1086 response it acknowledges.

1087 The uniqueness property of the **branch ID** parameter, to facilitate its use as a transaction ID, was not part of
1088 RFC 2543

1089 The **branch ID** inserted by an element compliant with this specification MUST always begin with the
1090 characters "z9hG4bK". These 7 characters are used as a magic cookie (7 is deemed sufficient to ensure that
1091 an older RFC 2543 implementation would not pick such a value), so that servers receiving the request can
1092 determine that the **branch ID** was constructed in the fashion described by this specification (that is, globally
1093 unique). Beyond this requirement, the precise format of the **branch** token is implementation-defined.

1094 The **Via** header **maddr**, **ttl**, and **sent-by** components will be set when the request is processed by the
1095 transport layer (Section 18).

1096 **Via** processing for proxies is described in Section 16.6 Item 8 and Section 16.7 Item 3.

1097 **8.1.1.8 Contact** The **Contact** header field provides a SIP URI that can be used to contact that specific
1098 instance of the UA for subsequent requests. The **Contact** header field MUST be present and contain exactly
1099 one SIP or SIPS URI in any request that can result in the establishment of a dialog. For the methods defined
1100 in this specification, that includes only the **INVITE** request. For these requests, the scope of the **Contact**
1101 is global. That is, the **Contact** header field value contains the URI at which the UA would like to receive
1102 requests, and this URI MUST be valid even if used in subsequent requests outside of any dialogs.

1103 If the **Request-URI** or top **Route** header field value contains a SIPS URI, the **Contact** header field
1104 MUST contain a SIPS URI as well.

1105 For further information on the **Contact** header field, see Section 20.10.

1106 **8.1.1.9 Supported and Require** If the UAC supports extensions to SIP that can be applied by the
1107 server to the response, the UAC SHOULD include a **Supported** header field in the request listing the option
1108 tags (Section 19.2) for those extensions.

1109 The option tags listed MUST only refer to extensions defined in standards-track RFCs. This is to pre-
1110 vent servers from insisting that clients implement non-standard, vendor-defined features in order to receive
1111 service. Extensions defined by experimental and informational RFCs are explicitly excluded from usage
1112 with the **Supported** header field in a request, since they too are often used to document vendor-defined
1113 extensions.

1114 If the UAC wishes to insist that a UAS understand an extension that the UAC will apply to the request
1115 in order to process the request, it MUST insert a **Require** header field into the request listing the option tag
1116 for that extension. If the UAC wishes to apply an extension to the request and insist that any proxies that are
1117 traversed understand that extension, it MUST insert a **Proxy-Require** header field into the request listing the
1118 option tag for that extension.

1119 As with the Supported header field, the option tags in the Require and Proxy-Require header fields
1120 MUST only refer to extensions defined in standards-track RFCs.

1121 **8.1.1.10 Additional Message Components** After a new request has been created, and the header fields
1122 described above have been properly constructed, any additional optional header fields are added, as are any
1123 header fields specific to the method.

1124 SIP requests MAY contain a MIME-encoded message-body. Regardless of the type of body that a request
1125 contains, certain header fields must be formulated to characterize the contents of the body. For further
1126 information on these header fields, see Sections 20.11 through 20.15.

1127 **8.1.2 Sending the Request**

1128 The destination for the request is then computed. Unless there is local policy specifying otherwise, then
1129 the destination MUST be determined by applying the DNS procedures described in [4] as follows. If the
1130 first element in the route set indicated a strict router (resulting in forming the request as described in Sec-
1131 tion 12.2.1.1), the procedures MUST be applied to the Request-URI of the request. Otherwise, the pro-
1132 cedures are applied to the first Route header field value in the request (if one exists), or to the request's
1133 Request-URI if there is no Route header field present. These procedures yield an ordered set of address,
1134 port, and transports to attempt. Independent of which URI is used as input to the procedures of [4], if the
1135 Request-URI specifies a SIPS resource, the UAC MUST follow the procedures of [4] as if the input URI
1136 were a SIPS URI.

1137 Local policy MAY specify an alternate set of destinations to attempt. If the Request-URI contains a
1138 SIPS URI, any alternate destinations MUST be contacted with TLS. Beyond that, there are no restrictions on
1139 the alternate destinations if the request contains no Route header field. This provides a simple alternative
1140 to a pre-existing route set as a way to specify an outbound proxy. However, that approach for configuring
1141 an outbound proxy is NOT RECOMMENDED; a pre-existing route set with a single URI SHOULD be used
1142 instead. If the request contains a Route header field, the request SHOULD be sent to the locations derived
1143 from its topmost value, but MAY be sent to any server that the UA is certain will honor the Route and
1144 Request-URI policies specified in this document (as opposed to those in RFC 2543). In particular, a UAC
1145 configured with an outbound proxy SHOULD attempt to send the request to the location indicated in the first
1146 Route header field value instead of adopting the policy of sending all messages to the outbound proxy.

1147 This ensures that outbound proxies that do not add Record-Route header field values will drop out of the path of
1148 subsequent requests. It allows endpoints that cannot resolve the first Route URI to delegate that task to an outbound
1149 proxy.

1150 The UAC SHOULD follow the procedures defined in [4] for stateful elements, trying each address until
1151 a server is contacted. Each try constitutes a new transaction, and therefore each carries a different topmost
1152 Via header field value with a new branch parameter. Furthermore, the transport value in the Via header field
1153 is set to whatever transport was determined for the target server.

1154 **8.1.3 Processing Responses**

1155 Responses are first processed by the transport layer and then passed up to the transaction layer. The trans-
1156 action layer performs its processing and then passes the response up to the TU. The majority of response
1157 processing in the TU is method specific. However, there are some general behaviors independent of the
1158 method.

1159 **8.1.3.1 Transaction Layer Errors** In some cases, the response returned by the transaction layer will not
1160 be a SIP message, but rather a transaction layer error. When a timeout error is received from the transaction
1161 layer, it **MUST** be treated as if a 408 (Request Timeout) status code has been received. If a fatal transport
1162 error is reported by the transport layer (generally, due to fatal ICMP errors in UDP or connection failures in
1163 TCP), the condition **MUST** be treated as a 503 (Service Unavailable) status code.

1164 **8.1.3.2 Unrecognized Responses** A UAC **MUST** treat any final response it does not recognize as being
1165 equivalent to the x00 response code of that class, and **MUST** be able to process the x00 response code for
1166 all classes. For example, if a UAC receives an unrecognized response code of 431, it can safely assume that
1167 there was something wrong with its request and treat the response as if it had received a 400 (Bad Request)
1168 response code. A UAC **MUST** treat any provisional response different than 100 that it does not recognize as
1169 183 (Session Progress). A UAC **MUST** be able to process 100 and 183 responses.

1170 **8.1.3.3 Vias** If more than one *Via* header field value is present in a response, the UAC **SHOULD** discard
1171 the message.

1172 The presence of additional *Via* header field values that precede the originator of the request suggests that the
1173 message was misrouted or possibly corrupted.

1174 **8.1.3.4 Processing 3xx Responses** Upon receipt of a redirection response (for example, a 301 response
1175 status code), clients **SHOULD** use the URI(s) in the *Contact* header field to formulate one or more new
1176 requests based on the redirected request. This process is similar to that of a proxy recursing on a 3xx class
1177 response as detailed in Sections 16.5 and 16.6. A client starts with an initial target set containing exactly one
1178 URI, the *Request-URI* of the original request. If a client wishes to formulate new requests based on a 3xx
1179 class response to that request, it places the URIs to try into the target set. Subject to the restrictions in this
1180 specification, a client can choose which *Contact* URIs it places into the target set. As with proxy recursion,
1181 a client processing 3xx class responses **MUST NOT** add any given URI to the target set more than once. If
1182 the original request had a SIPS URI in the *Request-URI*, the client **MAY** choose to recurse to a non-SIPS
1183 URI, but **SHOULD** inform the user of the redirection to an insecure URI.

1184 Any new request may receive 3xx responses themselves containing the original URI as a contact. Two locations
1185 can be configured to redirect to each other. Placing any given URI in the target set only once prevents infinite
1186 redirection loops.

1187 As the target set grows, the client **MAY** generate new requests to the URIs in any order. A common
1188 mechanism is to order the set by the "q" parameter value from the *Contact* header field value. Requests to
1189 the URIs **MAY** be generated serially or in parallel. One approach is to process groups of decreasing q-values
1190 serially and process the URIs in each q-value group in parallel. Another is to perform only serial processing
1191 in decreasing q-value order, arbitrarily choosing between contacts of equal q-value.

1192 If contacting an address in the list results in a failure, as defined in the next paragraph, the element moves
1193 to the next address in the list, until the list is exhausted. If the list is exhausted, then the request has failed.

1194 Failures **SHOULD** be detected through failure response codes (codes greater than 399); for network errors
1195 the client transaction will report any transport layer failures to the transaction user. Note that some response
1196 codes (detailed in 8.1.3.5) indicate that the request can be retried; requests that are reattempted should not
1197 be considered failures.

1198 When a failure for a particular contact address is received, the client **SHOULD** try the next contact
1199 address. This will involve creating a new client transaction to deliver a new request.

1200 In order to create a request based on a contact address in a 3xx response, a UAC MUST copy the en-
1201 tire URI from the target set into the Request-URI, except for the “method-param” and “header” URI
1202 parameters (see Section 19.1.1 for a definition of these parameters). It uses the “header” parameters to
1203 create header field values for the new request, overwriting header field values associated with the redirected
1204 request in accordance with the guidelines in Section 19.1.5.

1205 Note that in some instances, header fields that have been communicated in the contact address may
1206 instead append to existing request header fields in the original redirected request. As a general rule, if the
1207 header field can accept a comma-separated list of values, then the new header field value MAY be appended
1208 to any existing values in the original redirected request. If the header field does not accept multiple values,
1209 the value in the original redirected request MAY be overwritten by the header field value communicated in
1210 the contact address. For example, if a contact address is returned with the following value:

```
1211 sip:user@host?Subject=foo&Call-Info=<http://www.foo.com>
```

1212 Then any Subject header field in the original redirected request is overwritten, but the HTTP URL is
1213 merely appended to any existing Call-Info header field values.

1214 It is RECOMMENDED that the UAC reuse the same To, From, and Call-ID used in the original redirected
1215 request, but the UAC MAY also choose to update the Call-ID header field value for new requests, for example.

1216 Finally, once the new request has been constructed, it is sent using a new client transaction, and therefore
1217 MUST have a new branch ID in the top Via field as discussed in Section 8.1.1.7.

1218 In all other respects, requests sent upon receipt of a redirect response SHOULD re-use the header fields
1219 and bodies of the original request.

1220 In some instances, Contact header field values may be cached at UAC temporarily or permanently de-
1221 pending on the status code received and the presence of an expiration interval; see Sections 21.3.2 and 21.3.3.

1222 **8.1.3.5 Processing 4xx Responses** Certain 4xx response codes require specific UA processing, indepen-
1223 dent of the method.

1224 If a 401 (Unauthorized) or 407 (Proxy Authentication Required) response is received, the UAC SHOULD
1225 follow the authorization procedures of Section 22.2 and Section 22.3 to retry the request with credentials.

1226 If a 413 (Request Entity Too Large) response is received (Section 21.4.11), the request contained a body
1227 that was longer than the UAS was willing to accept. If possible, the UAC SHOULD retry the request, either
1228 omitting the body or using one of a smaller length.

1229 If a 415 (Unsupported Media Type) response is received (Section 21.4.13), the request contained media
1230 types not supported by the UAS. The UAC SHOULD retry sending the request, this time only using content
1231 with types listed in the Accept header field in the response, with encodings listed in the Accept-Encoding
1232 header field in the response, and with languages listed in the Accept-Language in the response.

1233 If a 416 (Unsupported URI Scheme) response is received (Section 21.4.14), the Request-URI used a
1234 URI scheme not supported by the server. The client SHOULD retry the request, this time, using a SIP URI.

1235 If a 420 (Bad Extension) response is received (Section 21.4.15), the request contained a Require or
1236 Proxy-Require header field listing an option-tag for a feature not supported by a proxy or UAS. The UAC
1237 SHOULD retry the request, this time omitting any extensions listed in the Unsupported header field in the
1238 response.

1239 In all of the above cases, the request is retried by creating a new request with the appropriate modifica-
1240 tions. This new request SHOULD have the same value of the Call-ID, To, and From of the previous request,
1241 but the CSeq should contain a new sequence number that is one higher than the previous.

1242 With other 4xx responses, including those yet to be defined, a retry may or may not be possible depend-
1243 ing on the method and the use case.

1244 8.2 UAS Behavior

1245 When a request outside of a dialog is processed by a UAS, there is a set of processing rules that are followed,
1246 independent of the method. Section 12 gives guidance on how a UAS can tell whether a request is inside or
1247 outside of a dialog.

1248 Note that request processing is atomic. If a request is accepted, all state changes associated with it **MUST**
1249 be performed. If it is rejected, all state changes **MUST NOT** be performed.

1250 UASs **SHOULD** process the requests in the order of the steps that follow in this section (that is, starting
1251 with authentication, then inspecting the method, the header fields, and so on throughout the remainder of
1252 this section).

1253 8.2.1 Method Inspection

1254 Once a request is authenticated (or authentication is skipped), the UAS **MUST** inspect the method of the
1255 request. If the UAS recognizes but does not support the method of a request, it **MUST** generate a 405
1256 (Method Not Allowed) response. Procedures for generating responses are described in Section 8.2.6. The
1257 UAS **MUST** also add an **Allow** header field to the 405 (Method Not Allowed) response. The **Allow** header
1258 field **MUST** list the set of methods supported by the UAS generating the message. The **Allow** header field is
1259 presented in Section 20.5.

1260 If the method is one supported by the server, processing continues.

1261 8.2.2 Header Inspection

1262 If a UAS does not understand a header field in a request (that is, the header field is not defined in this spec-
1263 ification or in any supported extension), the server **MUST** ignore that header field and continue processing
1264 the message. A UAS **SHOULD** ignore any malformed header fields that are not necessary for processing
1265 requests.

1266 **8.2.2.1 To and Request-URI** The **To** header field identifies the original recipient of the request desig-
1267 nated by the user identified in the **From** field. The original recipient may or may not be the UAS processing
1268 the request, due to call forwarding or other proxy operations. A UAS **MAY** apply any policy it wishes to
1269 determine whether to accept requests when the **To** header field is not the identity of the UAS. However, it is
1270 **RECOMMENDED** that a UAS accept requests even if they do not recognize the URI scheme (for example, a
1271 `tel:URI`) in the **To** header field, or if the **To** header field does not address a known or current user of this
1272 UAS. If, on the other hand, the UAS decides to reject the request, it **SHOULD** generate a response with a 403
1273 (Forbidden) status code and pass it to the server transaction for transmission.

1274 However, the **Request-URI** identifies the UAS that is to process the request. If the **Request-URI** uses
1275 a scheme not supported by the UAS, it **SHOULD** reject the request with a 416 (Unsupported URI Scheme)
1276 response. If the **Request-URI** does not identify an address that the UAS is willing to accept requests for,
1277 it **SHOULD** reject the request with a 404 (Not Found) response. Typically, a UA that uses the **REGISTER**
1278 method to bind its address-of-record to a specific contact address will see requests whose **Request-URI**
1279 equals that contact address. Other potential sources of received **Request-URIs** include the **Contact** header
1280 fields of requests and responses sent by the UA that establish or refresh dialogs.

1281 **8.2.2.2 Merged Requests** If the request has no tag in the **To** header field, the UAS core **MUST** check the
1282 request against ongoing transactions. If the **To** tag, **From** tag, **Call-ID**, **CSeq** exactly match those associated
1283 with an ongoing transaction, but the request does not match that transaction (based on the matching rules in
1284 Section 17.2.3), the UAS core **SHOULD** generate a 482 (Loop Detected) response and pass it to the server
1285 transaction.

1286 The same request has arrived at the UAS more than once, following different paths, most likely due to forking.
1287 The UAS processes the first such request received and responds with a 482 (Loop Detected) to the rest of them.

1288 **8.2.2.3 Require** Assuming the UAS decides that it is the proper element to process the request, it ex-
1289 amines the **Require** header field, if present.

1290 The **Require** header field is used by a UAC to tell a UAS about SIP extensions that the UAC expects
1291 the UAS to support in order to process the request properly. Its format is described in Section 20.32. If a
1292 UAS does not understand an option-tag listed in a **Require** header field, it **MUST** respond by generating a
1293 response with status code 420 (Bad Extension). The UAS **MUST** add an **Unsupported** header field, and list
1294 in it those options it does not understand amongst those in the **Require** header field of the request.

1295 Note that **Require** and **Proxy-Require** **MUST NOT** be used in a SIP **CANCEL** request, or in an **ACK**
1296 request sent for a non-2xx response. These header fields **MUST** be ignored if they are present in these
1297 requests.

1298 An **ACK** request for a 2xx response **MUST** contain only those **Require** and **Proxy-Require** values that
1299 were present in the initial request.

1300 Example:

```
1301 UAC->UAS:   INVITE sip:watson@bell-telephone.com SIP/2.0
1302             Require: 100rel
```

1303
1304

```
1305 UAS->UAC:   SIP/2.0 420 Bad Extension
1306             Unsupported: 100rel
```

1307 This behavior ensures that the client-server interaction will proceed without delay when all options are under-
1308 stood by both sides, and only slow down if options are not understood (as in the example above). For a well-matched
1309 client-server pair, the interaction proceeds quickly, saving a round-trip often required by negotiation mechanisms.
1310 In addition, it also removes ambiguity when the client requires features that the server does not understand. Some
1311 features, such as call handling fields, are only of interest to end systems.

1312 **8.2.3 Content Processing**

1313 Assuming the UAS understands any extensions required by the client, the UAS examines the body of the
1314 message, and the header fields that describe it. If there are any bodies whose type (indicated by the **Content-**
1315 **Type**), language (indicated by the **Content-Language**) or encoding (indicated by the **Content-Encoding**)
1316 are not understood, and that body part is not optional (as indicated by the **Content-Disposition** header
1317 field), the UAS **MUST** reject the request with a 415 (Unsupported Media Type) response. The response **MUST**
1318 contain an **Accept** header field listing the types of all bodies it understands, in the event the request contained
1319 bodies of types not supported by the UAS. If the request contained content encodings not understood by the
1320 UAS, the response **MUST** contain an **Accept-Encoding** header field listing the encodings understood by
1321 the UAS. If the request contained content with languages not understood by the UAS, the response **MUST**
1322 contain an **Accept-Language** header field indicating the languages understood by the UAS. Beyond these

1323 checks, body handling depends on the method and type. For further information on the processing of
1324 content-specific header fields, see Section 7.4 as well as Section 20.11 through 20.15.

1325 **8.2.4 Applying Extensions**

1326 A UAS that wishes to apply some extension when generating the response **MUST NOT** do so unless support
1327 for that extension is indicated in the **Supported** header field in the request. If the desired extension is not
1328 supported, the server **SHOULD** rely only on baseline SIP and any other extensions supported by the client. In
1329 rare circumstances, where the server cannot process the request without the extension, the server **MAY** send
1330 a 421 (Extension Required) response. This response indicates that the proper response cannot be generated
1331 without support of a specific extension. The needed extension(s) **MUST** be included in a **Require** header
1332 field in the response. This behavior is **NOT RECOMMENDED**, as it will generally break interoperability.

1333 Any extensions applied to a non-421 response **MUST** be listed in a **Require** header field included in the
1334 response. Of course, the server **MUST NOT** apply extensions not listed in the **Supported** header field in the
1335 request. As a result of this, the **Require** header field in a response will only ever contain option tags defined
1336 in standards-track RFCs.

1337 **8.2.5 Processing the Request**

1338 Assuming all of the checks in the previous subsections are passed, the UAS processing becomes method-
1339 specific. Section 10 covers the **REGISTER** request, section 11 covers the **OPTIONS** request, section 13
1340 covers the **INVITE** request, and section 15 covers the **BYE** request.

1341 **8.2.6 Generating the Response**

1342 When a UAS wishes to construct a response to a request, it follows the general procedures detailed in the
1343 following subsections. Additional behaviors specific to the response code in question, which are not detailed
1344 in this section, may also be required.

1345 Once all procedures associated with the creation of a response have been completed, the UAS hands the
1346 response back to the server transaction from which it received the request.

1347 **8.2.6.1 Sending a Provisional Response** One largely non-method-specific guideline for the generation
1348 of responses is that UASs **SHOULD NOT** issue a provisional response for a non-**INVITE** request. Rather,
1349 UASs **SHOULD** generate a final response to a non-**INVITE** request as soon as possible.

1350 When a 100 (Trying) response is generated, any **Timestamp** header field present in the request **MUST** be
1351 copied into this 100 (Trying) response. If there is a delay in generating the response, the UAS **SHOULD** add
1352 a delay value into the **Timestamp** value in the response. This value **MUST** contain the difference between
1353 time of sending of the response and receipt of the request, measured in seconds.

1354 **8.2.6.2 Headers and Tags** The **From** field of the response **MUST** equal the **From** header field of the
1355 request. The **Call-ID** header field of the response **MUST** equal the **Call-ID** header field of the request. The
1356 **CSeq** header field of the response **MUST** equal the **CSeq** field of the request. The **Via** header field values in
1357 the response **MUST** equal the **Via** header field values in the request and **MUST** maintain the same ordering.

1358 If a request contained a **To** tag in the request, the **To** header field in the response **MUST** equal that of
1359 the request. However, if the **To** header field in the request did not contain a tag, the URI in the **To** header

1360 field in the response MUST equal the URI in the To header field; additionally, the UAS MUST add a tag to
1361 the To header field in the response (with the exception of the 100 (Trying) response, in which a tag MAY be
1362 present). This serves to identify the UAS that is responding, possibly resulting in a component of a dialog
1363 ID. The same tag MUST be used for all responses to that request, both final and provisional (again excepting
1364 the 100 (Trying)). Procedures for generation of tags are defined in Section 19.3.

1365 8.2.7 Stateless UAS Behavior

1366 A stateless UAS is a UAS that does not maintain transaction state. It replies to requests normally, but
1367 discards any state that would ordinarily be retained by a UAS after a response has been sent. If a stateless
1368 UAS receives a retransmission of a request, it regenerates the response and resends it, just as if it were
1369 replying to the first instance of the request. Stateless UASs do not use a transaction layer; they receive
1370 requests directly from the transport layer and send responses directly to the transport layer.

1371 The stateless UAS role is needed primarily to handle unauthenticated requests for which a challenge
1372 response is issued. If unauthenticated requests were handled statefully, then malicious floods of unau-
1373 thenticated requests could create massive amounts of transaction state that might slow or completely halt
1374 call processing in a UAS, effectively creating a denial of service condition; for more information see Sec-
1375 tion 26.1.5.

1376 The most important behaviors of a stateless UAS are the following:

- 1377 • A stateless UAS MUST NOT send provisional (1xx) responses.
- 1378 • A stateless UAS MUST NOT retransmit responses.
- 1379 • A stateless UAS MUST ignore ACK requests.
- 1380 • A stateless UAS MUST ignore CANCEL requests.
- 1381 • To header tags MUST be generated for responses in a stateless manner - in a manner that will generate
1382 the same tag for the same request consistently. For information on tag construction see Section 19.3.

1383 In all other respects, a stateless UAS behaves in the same manner as a stateful UAS. A UAS can operate
1384 in either a stateful or stateless mode for each new request.

1385 8.3 Redirect Servers

1386 In some architectures it may be desirable to reduce the processing load on proxy servers that are responsible
1387 for routing requests, and improve signaling path robustness, by relying on redirection. Redirection allows
1388 servers to push routing information for a request back in a response to the client, thereby taking themselves
1389 out of the loop of further messaging for this transaction while still aiding in locating the target of the request.
1390 When the originator of the request receives the redirection, it will send a new request based on the URI(s)
1391 it has received. By propagating URIs from the core of the network to its edges, redirection allows for
1392 considerable network scalability.

1393 A redirect server is logically constituted of a server transaction layer and a transaction user that has
1394 access to a location service of some kind (see Section 10 for more on registrars and location services). This
1395 location service is effectively a database containing mappings between a single URI and a set of one or more
1396 alternative locations at which the target of that URI can be found.

1397 A redirect server does not issue any SIP requests of its own. After receiving a request other than CAN-
1398 CEL, the server either refuses the request or gathers the list of alternative locations from the location service
1399 and returns a final response of class 3xx. For well-formed CANCEL requests, it SHOULD return a 2xx re-
1400 sponse. This response ends the SIP transaction. The redirect server maintains transaction state for an entire
1401 SIP transaction. It is the responsibility of clients to detect forwarding loops between redirect servers.

1402 When a redirect server returns a 3xx response to a request, it populates the list of (one or more) alter-
1403 native locations into the Contact header field. An “expires” parameter to the Contact header field values
1404 may also be supplied to indicate the lifetime of the Contact data.

1405 The Contact header field contains URIs giving the new locations or user names to try, or may simply
1406 specify additional transport parameters. A 301 (Moved Permanently) or 302 (Moved Temporarily) response
1407 may also give the same location and username that was targeted by the initial request but specify additional
1408 transport parameters such as a different server or multicast address to try, or a change of SIP transport from
1409 UDP to TCP or vice versa.

1410 However, redirect servers MUST NOT redirect a request to a URI equal to the one in the Request-URI;
1411 instead, provided that the URI does not point to itself, the server MAY proxy the request to the destination
1412 URI, or MAY reject it with a 404.

1413 If a client is using an outbound proxy, and that proxy actually redirects requests, a potential arises for infinite
1414 redirection loops.

1415 Note that a Contact header field value MAY also refer to a different resource than the one originally
1416 called. For example, a SIP call connected to PSTN gateway may need to deliver a special informational
1417 announcement such as “The number you have dialed has been changed.”

1418 A Contact response header field can contain any suitable URI indicating where the called party can
1419 be reached, not limited to SIP URIs. For example, it could contain URIs for phones, fax, or irc (if they
1420 were defined) or a mailto: (RFC 2368 [32]) URL. Section 26.4.4 discusses implications and limitations of
1421 redirecting a SIPS URI to a non-SIPS URI.

1422 The “expires” parameter of a Contact header field value indicates how long the URI is valid. The value
1423 of the parameter is a number indicating seconds. If this parameter is not provided, the value of the Expires
1424 header field determines how long the URI is valid. Malformed values SHOULD be treated as equivalent to
1425 3600.

1426 This provides a modest level of backwards compatibility with RFC 2543, which allowed absolute times in this
1427 header field. If an absolute time is received, it will be treated as malformed, and then default to 3600.

1428 Redirect servers MUST ignore features that are not understood (including unrecognized header fields, any
1429 unknown option tags in Require, or even method names) and proceed with the redirection of the request in
1430 question.

1431 9 Canceling a Request

1432 The previous section has discussed general UA behavior for generating requests and processing responses
1433 for requests of all methods. In this section, we discuss a general purpose method, called CANCEL.

1434 The CANCEL request, as the name implies, is used to cancel a previous request sent by a client. Specif-
1435 ically, it asks the UAS to cease processing the request and to generate an error response to that request.
1436 CANCEL has no effect on a request to which a UAS has already given a final response. Because of this,
1437 it is most useful to CANCEL requests to which it can take a server long time to respond. For this reason,
1438 CANCEL is best for INVITE requests, which can take a long time to generate a response. In that usage,

1439 a UAS that receives a CANCEL request for an INVITE, but has not yet sent a final response, would “stop
1440 ringing”, and then respond to the INVITE with a specific error response (a 487).

1441 CANCEL requests can be constructed and sent by both proxies and user agent clients. Section 15
1442 discusses under what conditions a UAC would CANCEL an INVITE request, and Section 16.10 discusses
1443 proxy usage of CANCEL.

1444 A stateful proxy responds to a CANCEL, rather than simply forwarding a response it would receive
1445 from a downstream element. For that reason, CANCEL is referred to as a “hop-by-hop” request, since it is
1446 responded to at each stateful proxy hop.

1447 9.1 Client Behavior

1448 A CANCEL request SHOULD NOT be sent to cancel a request other than INVITE.

1449 Since requests other than INVITE are responded to immediately, sending a CANCEL for a non-INVITE request
1450 would always create a race condition.

1451 The following procedures are used to construct a CANCEL request. The Request-URI, Call-ID, To,
1452 the numeric part of CSeq, and From header fields in the CANCEL request MUST be identical to those in
1453 the request being cancelled, including tags. A CANCEL constructed by a client MUST have only a single
1454 Via header field value matching the top Via value in the request being cancelled. Using the same values
1455 for these header fields allows the CANCEL to be matched with the request it cancels (Section 9.2 indicates
1456 how such matching occurs). However, the method part of the CSeq header field MUST have a value of
1457 CANCEL. This allows it to be identified and processed as a transaction in its own right (See Section 17).

1458 If the request being cancelled contains a Route header field, the CANCEL request MUST include that
1459 Route header field's values.

1460 This is needed so that stateless proxies are able to route CANCEL requests properly.

1461 The CANCEL request MUST NOT contain any Require or Proxy-Require header fields.

1462 Once the CANCEL is constructed, the client SHOULD check whether it has received any response (pro-
1463 visional or final) for the request being cancelled (herein referred to as the “original request”).

1464 If no provisional response has been received, the CANCEL request MUST NOT be sent; rather, the client
1465 MUST wait for the arrival of a provisional response before sending the request. If the original request has
1466 generated a final response, the CANCEL SHOULD NOT be sent, as it is an effective no-op, since CANCEL
1467 has no effect on requests that have already generated a final response. When the client decides to send the
1468 CANCEL, it creates a client transaction for the CANCEL and passes it the CANCEL request along with
1469 the destination address, port, and transport. The destination address, port, and transport for the CANCEL
1470 MUST be identical to those used to send the original request.

1471 If it was allowed to send the CANCEL before receiving a response for the previous request, the server could
1472 receive the CANCEL before the original request.

1473 Note that both the transaction corresponding to the original request and the CANCEL transaction will
1474 complete independently. However, a UAC canceling a request cannot rely on receiving a 487 (Request
1475 Terminated) response for the original request, as an RFC 2543-compliant UAS will not generate such a
1476 response. If there is no final response for the original request in $64 * T1$ seconds ($T1$ is defined in Sec-
1477 tion 17.1.1.1), the client SHOULD then consider the original transaction cancelled and SHOULD destroy the
1478 client transaction handling the original request.

1479 9.2 Server Behavior

1480 The CANCEL method requests that the TU at the server side cancel a pending transaction. The TU deter-
1481 mines the transaction to be cancelled by taking the CANCEL request, and then assuming that the request
1482 method is anything but CANCEL and applying the transaction matching procedures of Section 17.2.3. The
1483 matching transaction is the one to be cancelled.

1484 The processing of a CANCEL request at a server depends on the type of server. A stateless proxy will
1485 forward it, a stateful proxy might respond to it and generate some CANCEL requests of its own, and a UAS
1486 will respond to it. See Section 16.10 for proxy treatment of CANCEL.

1487 A UAS first processes the CANCEL request according to the general UAS processing described in
1488 Section 8.2. However, since CANCEL requests are hop-by-hop and cannot be resubmitted, they cannot be
1489 challenged by the server in order to get proper credentials in an Authorization header field. Note also that
1490 CANCEL requests do not contain a Require header field.

1491 If the UAS did not find a matching transaction for the CANCEL according to the procedure above, it
1492 SHOULD respond to the CANCEL with a 481 (Call Leg/Transaction Does Not Exist). If the transaction
1493 for the original request still exists, the behavior of the UAS on receiving a CANCEL request depends on
1494 whether it has already sent a final response for the original request. If it has, the CANCEL request has no
1495 effect on the processing of the original request, no effect on any session state, and no effect on the responses
1496 generated for the original request. If the UAS has not issued a final response for the original request, its
1497 behavior depends on the method of the original request. If the original request was an INVITE, the UAS
1498 SHOULD immediately respond to the INVITE with a 487 (Request Terminated). The behavior upon reception
1499 of a CANCEL request for any other method defined in this specification is effectively no-op.

1500 Regardless of the method of the original request, as long as the CANCEL matched an existing transac-
1501 tion, the UAS answers the CANCEL request itself with a 200 (OK) response. This response is constructed
1502 following the procedures described in Section 8.2.6 noting that the To tag of the response to the CANCEL
1503 and the To tag in the response to the original request SHOULD be the same. The response to CANCEL is
1504 passed to the server transaction for transmission.

1505 10 Registrations

1506 10.1 Overview

1507 SIP offers a discovery capability. If a user wants to initiate a session with another user, SIP must discover the
1508 current host(s) at which the destination user is reachable. This discovery process is frequently accomplished
1509 by SIP network elements such as proxy servers and redirect servers which are responsible for receiving a
1510 request, determining where to send it based on knowledge of the location of the user, and then sending
1511 it there. To do this, SIP network elements consult an abstract service known as a *location service*, which
1512 provides address bindings for a particular domain. These address bindings map an incoming SIP or SIPS
1513 URI, sip:bob@biloxi.com, for example, to one or more URIs that are somehow “closer” to the desired
1514 user, sip:bob@engineering.biloxi.com, for example. Ultimately, a proxy will consult a location
1515 service that maps a received URI to the user agent(s) at which the desired recipient is currently residing.

1516 Registration creates bindings in a location service for a particular domain that associate an address-of-
1517 record URI with one or more contact addresses. Thus, when a proxy for that domain receives a request whose
1518 Request-URI matches the address-of-record, the proxy will forward the request to the contact addresses
1519 registered to that address-of-record. Generally, it only makes sense to register an address-of-record at a

1520 domain's location service when requests for that address-of-record would be routed to that domain. In
 1521 most cases, this means that the domain of the registration will need to match the domain in the URI of the
 1522 address-of-record.

1523 There are many ways by which the contents of the location service can be established. One way is
 1524 administratively. In the above example, Bob is known to be a member of the engineering department through
 1525 access to a corporate database. However, SIP provides a mechanism for a UA to create a binding explicitly.
 1526 This mechanism is known as registration.

1527 Registration entails sending a REGISTER request to a special type of UAS known as a registrar. A
 1528 registrar acts as the front end to the location service for a domain, reading and writing mappings based on
 1529 the contents of REGISTER requests. This location service is then typically consulted by a proxy server that
 1530 is responsible for routing requests for that domain.

1531 An illustration of the overall registration process is given in 2. Note that the registrar and proxy server
 1532 are logical roles that can be played by a single device in a network; for purposes of clarity the two are
 1533 separated in this illustration. Also note that UAs may send requests through a proxy server in order to reach
 1534 a registrar if the two are separate elements.

1535 SIP does not mandate a particular mechanism for implementing the location service. The only require-
 1536 ment is that a registrar for some domain MUST be able to read and write data to the location service, and
 1537 a proxy or redirect server for that domain MUST be capable of reading that same data. A registrar MAY be
 1538 co-located with a particular SIP proxy server for the same domain.

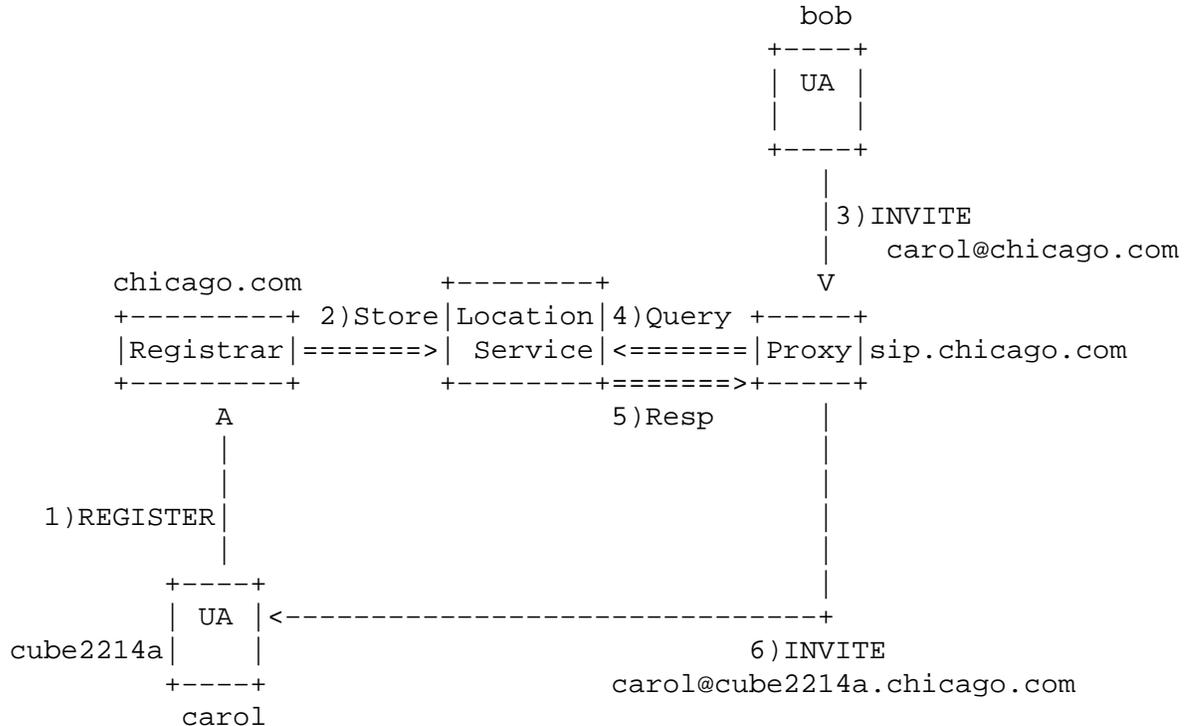


Figure 2: REGISTER example

1539 10.2 Constructing the REGISTER Request

1540 REGISTER requests add, remove, and query bindings. A REGISTER request can add a new binding
1541 between an address-of-record and one or more contact addresses. Registration on behalf of a particular
1542 address-of-record can be performed by a suitably authorized third party. A client can also remove previous
1543 bindings or query to determine which bindings are currently in place for an address-of-record.

1544 Except as noted, the construction of the REGISTER request and the behavior of clients sending a
1545 REGISTER request is identical to the general UAC behavior described in Section 8.1 and Section 17.1.

1546 A REGISTER request does *not* establish a dialog. A UAC MAY include a Route header field in a
1547 REGISTER request based on a pre-existing route set as described in Section 8.1. The Record-Route
1548 header field has no meaning in REGISTER requests or responses, and MUST be ignored if present. In
1549 particular, the UAC MUST NOT create a new route set based on the presence or absence of a Record-Route
1550 header field in any response to a REGISTER request.

1551 The following header fields, except Contact, MUST be included in a REGISTER request. A Contact
1552 header field MAY be included:

1553 **Request-URI:** The Request-URI names the domain of the location service for which the registration is
1554 meant (for example, "sip:chicago.com"). The "userinfo" and "@" components of the SIP URI MUST
1555 NOT be present.

1556 **To:** The To header field contains the address of record whose registration is to be created, queried, or
1557 modified. The To header field and the Request-URI field typically differ, as the former contains a
1558 user name. This address-of-record MUST be a SIP URI or SIPS URI.

1559 **From:** The From header field contains the address-of-record of the person responsible for the registration.
1560 The value is the same as the To header field unless the request is a third-party registration.

1561 **Call-ID:** All registrations from a UAC SHOULD use the same Call-ID header field value for registrations
1562 sent to a particular registrar.

1563 If the same client were to use different Call-ID values, a registrar could not detect whether a delayed
1564 REGISTER request might have arrived out of order.

1565 **CSeq:** The CSeq value guarantees proper ordering of REGISTER requests. A UA MUST increment the
1566 CSeq value by one for each REGISTER request with the same Call-ID.

1567 **Contact:** REGISTER requests MAY contain a Contact header field with zero or more values containing
1568 address bindings.

1569 UAs MUST NOT send a new registration (that is, containing new Contact header field values, as opposed
1570 to a retransmission) until they have received a final response from the registrar for the previous one or the
1571 previous REGISTER request has timed out.

1572 The following Contact header parameters have a special meaning in REGISTER requests:

1573 **action:** The "action" parameter from RFC 2543 has been deprecated. UACs SHOULD NOT use the
1574 "action" parameter.

1575 **expires:** The “expires” parameter indicates how long the UA would like the binding to be valid. The value
1576 is a number indicating seconds. If this parameter is not provided, the value of the Expires header field
1577 is used instead. Implementations MAY treat values larger than 2**32-1 (4294967295 seconds or 136
1578 years) as equivalent to 2**32-1. Malformed values SHOULD be treated as equivalent to 3600.

1579 10.2.1 Adding Bindings

1580 The REGISTER request sent to a registrar includes the contact address(es) to which SIP requests for the
1581 address-of-record should be forwarded. The address-of-record is included in the To header field of the
1582 REGISTER request.

1583 The Contact header field values of the request typically consist of SIP or SIPS URIs that identify
1584 particular SIP endpoints (for example, “sip:carol@cube2214a.chicago.com”), but they MAY use any URI
1585 scheme. A SIP UA can choose to register telephone numbers (with the tel URL, RFC 2806 [9]) or email
1586 addresses (with a mailto URL, RFC 2368[32]) as Contacts for an address-of-record, for example.

1587 For example, Carol, with address-of-record “sip:carol@chicago.com”, would register with the SIP reg-
1588 istrar of the domain chicago.com. Her registrations would then be used by a proxy server in the chicago.com
1589 domain to route requests for Carol’s address-of-record to her SIP endpoint.

1590 Once a client has established bindings at a registrar, it MAY send subsequent registrations containing
1591 new bindings or modifications to existing bindings as necessary. The 2xx response to the REGISTER
1592 request will contain, in a Contact header field, a complete list of bindings that have been registered for this
1593 address-of-record at this registrar.

1594 If the address-of-record in the To header field of a REGISTER request is a SIPS URI, then any Contact
1595 header field values in the request SHOULD also be SIPS URIs. Clients should only register non-SIPS URIs
1596 under a SIPS address-of-record when the security of the resource represented by the contact address is
1597 guaranteed by other means. This may be applicable to URIs that invoke protocols other than SIP, or SIP
1598 devices secured by protocols other than TLS.

1599 Registrations do not need to update all bindings. Typically, a UA only updates its own contact addresses.

1600 **10.2.1.1 Setting the Expiration Interval of Contact Addresses** When a client sends a REGISTER
1601 request, it MAY suggest an expiration interval that indicates how long the client would like the registration
1602 to be valid. (As described in Section 10.3, the registrar selects the actual time interval based on its local
1603 policy.)

1604 There are two ways in which a client can suggest an expiration interval for a binding: through an
1605 Expires header field or an “expires” Contact header parameter. The latter allows expiration intervals to
1606 be suggested on a per-binding basis when more than one binding is given in a single REGISTER request,
1607 whereas the former suggests an expiration interval for all Contact header field values that do not contain
1608 the “expires” parameter.

1609 If neither mechanism for expressing a suggested expiration time is present in a REGISTER, a default
1610 suggestion of one hour SHOULD be assumed.

1611 **10.2.1.2 Preferences among Contact Addresses** If more than one Contact is sent in a REGISTER
1612 request, the registering UA intends to associate all of the URIs in these Contact header field values with the
1613 address-of-record present in the To field. This list can be prioritized with the “q” parameter in the Contact
1614 header field. The “q” parameter indicates a relative preference for the particular Contact header field value

1615 compared to other bindings present in this REGISTER message or existing within the location service of
1616 the registrar. Section 16.6 describes how a proxy server uses this preference indication.

1617 **10.2.2 Removing Bindings**

1618 Registrations are soft state and expire unless refreshed, but can also be explicitly removed. A client can
1619 attempt to influence the expiration interval selected by the registrar as described in Section 10.2.1. A UA
1620 requests the immediate removal of a binding by specifying an expiration interval of "0" for that contact
1621 address in a REGISTER request. UAs SHOULD support this mechanism so that bindings can be removed
1622 before their expiration interval has passed.

1623 The REGISTER-specific Contact header field value of "*" applies to all registrations, but it MUST NOT
1624 be used unless the Expires header field is present with a value of "0".

1625 Use of the "*" Contact header field value allows a registering UA to remove all of bindings associated with an
1626 address-of-record without knowing their precise values.

1627 **10.2.3 Fetching Bindings**

1628 A success response to any REGISTER request contains the complete list of existing bindings, regardless of
1629 whether the request contained a Contact header field. If no Contact header field is present in a REGISTER
1630 request, the list of bindings is left unchanged.

1631 **10.2.4 Refreshing Bindings**

1632 Each UA is responsible for refreshing the bindings that it has previously established. A UA SHOULD NOT
1633 refresh bindings set up by other UAs.

1634 The 200 (OK) response from the registrar contains a list of Contact fields enumerating all current
1635 bindings. The UA compares each contact address to see if it created the contact address, using comparison
1636 rules in Section 19.1.4. If so, it updates the expiration time interval according to the expires parameter or,
1637 if absent, the Expires field value. The UA then issues a REGISTER request for each of its bindings before
1638 the expiration interval has elapsed. It MAY combine several updates into one REGISTER request.

1639 A UA SHOULD use the same Call-ID for all registrations during a single boot cycle. Registration re-
1640 freshes SHOULD be sent to the same network address as the original registration, unless redirected.

1641 **10.2.5 Setting the Internal Clock**

1642 If the response for a REGISTER request contains a Date header field, the client MAY use this header field
1643 to learn the current time in order to set any internal clocks.

1644 **10.2.6 Discovering a Registrar**

1645 UAs can use three ways to determine the address to which to send registrations: by configuration, using the
1646 address-of-record, and multicast. A UA can be configured, in ways beyond the scope of this specification,
1647 with a registrar address. If there is no configured registrar address, the UA SHOULD use the host part of the
1648 address-of-record as the Request-URI and address the request there, using the normal SIP server location
1649 mechanisms [4]. For example, the UA for the user "sip:carol@chicago.com" addresses the REGISTER
1650 request to "sip:chicago.com".

1651 Finally, a UA can be configured to use multicast. Multicast registrations are addressed to the well-known
1652 “all SIP servers” multicast address “sip.mcast.net” (224.0.1.75 for IPv4). No well-known IPv6 multicast
1653 address has been allocated; such an allocation will be documented separately when needed. SIP UAs MAY
1654 listen to that address and use it to become aware of the location of other local users (see [33]); however, they
1655 do not respond to the request.

1656 Multicast registration may be inappropriate in some environments, for example, if multiple businesses share the
1657 same local area network.

1658 **10.2.7 Transmitting a Request**

1659 Once the REGISTER method has been constructed, and the destination of the message identified, UACs
1660 follow the procedures described in Section 8.1.2 to hand off the REGISTER to the transaction layer.

1661 If the transaction layer returns a timeout error because the REGISTER yielded no response, the UAC
1662 SHOULD NOT immediately re-attempt a registration to the same registrar.

1663 An immediate re-attempt is likely to also timeout. Waiting some reasonable time interval for the conditions
1664 causing the timeout to be corrected reduces unnecessary load on the network. No specific interval is mandated.

1665 **10.2.8 Error Responses**

1666 If a UA receives a 423 (Interval Too Brief) response, it MAY retry the registration after making the expiration
1667 interval of all contact addresses in the REGISTER request equal to or greater than the expiration interval
1668 within the Min-Expires header field of the 423 (Interval Too Brief) response.

1669 **10.3 Processing REGISTER Requests**

1670 A registrar is a UAS that responds to REGISTER requests and maintains a list of bindings that are accessible
1671 to proxy servers and redirect servers within its administrative domain. A registrar handles requests according
1672 to Section 8.2 and Section 17.2, but it accepts only REGISTER requests. A registrar MUST not generate
1673 6xx responses.

1674 A registrar MAY redirect REGISTER requests as appropriate. One common usage would be for a
1675 registrar listening on a multicast interface to redirect multicast REGISTER requests to its own unicast
1676 interface with a 302 (Moved Temporarily) response.

1677 Registrars MUST ignore the Record-Route header field if it is included in a REGISTER request. Reg-
1678 istrars MUST NOT include a Record-Route header field in any response to a REGISTER request.

1679 A registrar might receive a request that traversed a proxy which treats REGISTER as an unknown request and
1680 which added a Record-Route header field value.

1681 A registrar has to know (for example, through configuration) the set of domain(s) for which it maintains
1682 bindings. REGISTER requests MUST be processed by a registrar in the order that they are received. REG-
1683 ISTER requests MUST also be processed atomically, meaning that a particular REGISTER request is either
1684 processed completely or not at all. Each REGISTER message MUST be processed independently of any
1685 other registration or binding changes.

1686 When receiving a REGISTER request, a registrar follows these steps:

- 1687 1. The registrar inspects the Request-URI to determine whether it has access to bindings for the domain
1688 identified in the Request-URI. If not, and if the server also acts as a proxy server, the server SHOULD
1689 forward the request to the addressed domain, following the general behavior for proxying messages
1690 described in Section 16.

- 1691 2. To guarantee that the registrar supports any necessary extensions, the registrar MUST process the
1692 **Require** header field values as described for UASs in Section 8.2.2.
- 1693 3. A registrar SHOULD authenticate the UAC. Mechanisms for the authentication of SIP user agents
1694 are described in Section 22. Registration behavior in no way overrides the generic authentication
1695 framework for SIP. If no authentication mechanism is available, the registrar MAY take the **From**
1696 address as the asserted identity of the originator of the request.
- 1697 4. The registrar SHOULD determine if the authenticated user is authorized to modify registrations for
1698 this address-of-record. For example, a registrar might consult a authorization database that maps user
1699 names to a list of addresses-of-record for which that user has authorization to modify bindings. If the
1700 authenticated user is not authorized to modify bindings, the registrar MUST return a 403 (Forbidden)
1701 and skip the remaining steps.

1702 In architectures that support third-party registration, one entity may be responsible for updating the regis-
1703 trations associated with multiple addresses-of-record.

- 1704 5. The registrar extracts the address-of-record from the **To** header field of the request. If the address-of-
1705 record is not valid for the domain in the **Request-URI**, the registrar MUST send a 404 (Not Found)
1706 response and skip the remaining steps. The URI MUST then be converted to a canonical form. To do
1707 that, all URI parameters MUST be removed (including the **user-param**), and any escaped characters
1708 MUST be converted to their unescaped form. The result serves as an index into the list of bindings.
- 1709 6. The registrar checks whether the request contains the **Contact** header field. If not, it skips to the last
1710 step. If the **Contact** header field is present, the registrar checks if there is one **Contact** field value
1711 that contains the special value "*" and an **Expires** field. If the request has additional **Contact** fields
1712 or an expiration time other than zero, the request is invalid, and the server MUST return a 400 Invalid
1713 Request and skip the remaining steps. If not, the registrar checks whether the **Call-ID** agrees with the
1714 value stored for each binding. If not, it MUST remove the binding. If it does agree, it MUST remove
1715 the binding only if the **CSeq** in the request is higher than the value stored for that binding. Otherwise
1716 the registrar MUST leave the binding as is. It then skips to the last step.
- 1717 7. The registrar now processes each contact address in the **Contact** header field in turn. For each address,
1718 it determines the expiration interval as follows:
- 1719 • If the field value has an "expires" parameter, that value MUST be used.
 - 1720 • If there is no such parameter, but the request has an **Expires** header field, that value MUST be
1721 used.
 - 1722 • If there is neither, a locally-configured default value MUST be used.

1723 The registrar MAY shorten the expiration interval. If and only if the expiration interval is greater than
1724 zero AND smaller than one hour AND less than a registrar-configured minimum, the registrar MAY
1725 reject the registration with a response of 423 (Registration Too Brief). This response MUST contain a
1726 **Min-Expires** header field that states the minimum expiration interval the registrar is willing to honor.
1727 It then skips the remaining steps.

1728 Allowing the registrar to set the registration interval protects it against excessively frequent registration
1729 refreshes while limiting the state that it needs to maintain and decreasing the likelihood of registrations going

1730 stale. The expiration interval of a registration is frequently used in the creation of services. An example is a
1731 follow-me service, where the user may only be available at a terminal for a brief period. Therefore, registrars
1732 should accept brief registrations; a request should only be rejected if the interval is so short that the refreshes
1733 would degrade registrar performance.

1734 For each address, the registrar then searches the list of current bindings using the URI comparison
1735 rules. If the binding does not exist, it is tentatively added. If the binding does exist, the registrar
1736 checks the Call-ID value. If the Call-ID value in the existing binding differs from the Call-ID value in
1737 the request, the binding **MUST** be removed if the expiration time is zero and updated otherwise. If they
1738 are the same, the registrar compares the CSeq value. If the value is higher than that of the existing
1739 binding, it **MUST** update or remove the binding as above. If not, the update **MUST** be aborted and the
1740 request fails.

1741 This algorithm ensures that out-of-order requests from the same UA are ignored.

1742 Each binding record records the Call-ID and CSeq values from the request.

1743 The binding updates **MUST** be committed (that is, made visible to the proxy or redirect server) if and
1744 only if all binding updates and additions succeed. If any one of them fails (for example, because the
1745 back-end database commit failed), the request **MUST** fail with a 500 (Server Error) response and all
1746 tentative binding updates **MUST** be removed.

1747 8. The registrar returns a 200 (OK) response. The response **MUST** contain Contact header field values
1748 enumerating all current bindings. Each Contact value **MUST** feature an “expires” parameter indi-
1749 cating its expiration interval chosen by the registrar. The response **SHOULD** include a Date header
1750 field.

1751 11 Querying for Capabilities

1752 The SIP method OPTIONS allows a UA to query another UA or a proxy server as to its capabilities. This
1753 allows a client to discover information about the supported methods, content types, extensions, codecs, etc.
1754 without “ringing” the other party. For example, before a client inserts a Require header field into an INVITE
1755 listing an option that it is not certain the destination UAS supports, the client can query the destination UAS
1756 with an OPTIONS to see if this option is returned in a Supported header field. All UAs **MUST** support the
1757 OPTIONS method.

1758 The target of the OPTIONS request is identified by the Request-URI, which could identify another
1759 UA or a SIP server. If the OPTIONS is addressed to a proxy server, the Request-URI is set without a user
1760 part, similar to the way a Request-URI is set for a REGISTER request.

1761 Alternatively, a server receiving an OPTIONS request with a Max-Forwards header field value of 0
1762 **MAY** respond to the request regardless of the Request-URI.

1763 This behavior is common with HTTP/1.1. This behavior can be used as a “traceroute” functionality to check the
1764 capabilities of individual hop servers by sending a series of OPTIONS requests with incremented Max-Forwards
1765 values.

1766 As is the case for general UA behavior, the transaction layer can return a timeout error if the OPTIONS
1767 yields no response. This may indicate that the target is unreachable and hence unavailable.

1768 An OPTIONS request **MAY** be sent as part of an established dialog to query the peer on capabilities that
1769 may be utilized later in the dialog.

1770 11.1 Construction of OPTIONS Request

1771 An OPTIONS request is constructed using the standard rules for a SIP request as discussed Section 8.1.1.

1772 A Contact header field MAY be present in an OPTIONS.

1773 An Accept header field SHOULD be included to indicate the type of message body the UAC wishes to
1774 receive in the response. Typically, this is set to a format that is used to describe the media capabilities of a
1775 UA, such as SDP (application/sdp).

1776 The response to an OPTIONS request is assumed to be scoped to the Request-URI in the original
1777 request. However, only when an OPTIONS is sent as part of an established dialog is it guaranteed that
1778 future requests will be received by the server that generated the OPTIONS response.

1779 Example OPTIONS request:

```
1780 OPTIONS sip:carol@chicago.com SIP/2.0
1781 Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKhjhs8ass877
1782 Max-Forwards: 70
1783 To: <sip:carol@chicago.com>
1784 From: Alice <sip:alice@atlanta.com>;tag=1928301774
1785 Call-ID: a84b4c76e66710
1786 CSeq: 63104 OPTIONS
1787 Contact: <sip:alice@pc33.atlanta.com>
1788 Accept: application/sdp
1789 Content-Length: 0
```

1790 11.2 Processing of OPTIONS Request

1791 The response to an OPTIONS is constructed using the standard rules for a SIP response as discussed in
1792 Section 8.2.6. The response code chosen MUST be the same that would have been chosen had the request
1793 been an INVITE. That is, a 200 (OK) would be returned if the UAS is ready to accept a call, a 486 (Busy
1794 Here) would be returned if the UAS is busy, etc. This allows an OPTIONS request to be used to determine
1795 the basic state of a UAS, which can be an indication of whether the UAS will accept an INVITE request.

1796 An OPTIONS request received within a dialog generates a 200 (OK) response that is identical to one
1797 constructed outside a dialog and does not have any impact on the dialog.

1798 This use of OPTIONS has limitations due the differences in proxy handling of OPTIONS and INVITE
1799 requests. While a forked INVITE can result in multiple 200 (OK) responses being returned, a forked OP-
1800 TIONS will only result in a single 200 (OK) response, since it is treated by proxies using the non-INVITE
1801 handling. See Section 16.7 for the normative details.

1802 If the response to an OPTIONS is generated by a proxy server, the proxy returns a 200 (OK) listing the
1803 capabilities of the server. The response does not contain a message body.

1804 Allow, Accept, Accept-Encoding, Accept-Language, and Supported header fields SHOULD be
1805 present in a 200 (OK) response to an OPTIONS request. If the response is generated by a proxy, the
1806 Allow header field SHOULD be omitted as it is ambiguous since a proxy is method agnostic. Contact header
1807 fields MAY be present in a 200 (OK) response and have the same semantics as in a 3xx response. That is,
1808 they may list a set of alternative names and methods of reaching the user. A Warning header field MAY be
1809 present.

1810 A message body MAY be sent, the type of which is determined by the Accept header field in the OP-
1811 TIONS request (application/sdp is the default if the Accept header field is not present). If the types include

1812 one that can describe media capabilities, the UAS SHOULD include a body in the response for that purpose.
1813 Details on construction of such a body in the case of application/sdp are described in [13].

1814 Example **OPTIONS** response generated by a UAS (corresponding to the request in Section 11.1):

```
1815 SIP/2.0 200 OK
1816 Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKhjhs8ass877
1817 ;received=192.0.2.4
1818 To: <sip:carol@chicago.com>;tag=93810874
1819 From: Alice <sip:alice@atlanta.com>;tag=1928301774
1820 Call-ID: a84b4c76e66710
1821 CSeq: 63104 OPTIONS
1822 Contact: <sip:carol@chicago.com>
1823 Contact: <mailto:carol@chicago.com>
1824 Allow: INVITE, ACK, CANCEL, OPTIONS, BYE
1825 Accept: application/sdp
1826 Accept-Encoding: gzip
1827 Accept-Language: en
1828 Supported: foo
1829 Content-Type: application/sdp
1830 Content-Length: 274
1831
1832 (SDP not shown)
```

1833 12 Dialogs

1834 A key concept for a user agent is that of a dialog. A dialog represents a peer-to-peer SIP relationship between
1835 two user agents that persists for some time. The dialog facilitates sequencing of messages between the user
1836 agents and proper routing of requests between both of them. The dialog represents a context in which to
1837 interpret SIP messages. Section 8 discussed method independent UA processing for requests and responses
1838 outside of a dialog. This section discusses how those requests and responses are used to construct a dialog,
1839 and then how subsequent requests and responses are sent within a dialog.

1840 A dialog is identified at each UA with a dialog ID, which consists of a **Call-ID** value, a local tag and a
1841 remote tag. The dialog ID at each UA involved in the dialog is not the same. Specifically, the local tag at one
1842 UA is identical to the remote tag at the peer UA. The tags are opaque tokens that facilitate the generation of
1843 unique dialog IDs.

1844 A dialog ID is also associated with all responses and with any request that contains a tag in the **To** field.
1845 The rules for computing the dialog ID of a message depend on whether the SIP element is a UAC or UAS.
1846 For a UAC, the **Call-ID** value of the dialog ID is set to the **Call-ID** of the message, the remote tag is set to
1847 the tag in the **To** field of the message, and the local tag is set to the tag in the **From** field of the message
1848 (these rules apply to both requests and responses). As one would expect, for a UAS, the **Call-ID** value of
1849 the dialog ID is set to the **Call-ID** of the message, the remote tag is set to the tag in the **From** field of the
1850 message, and the local tag is set to the tag in the **To** field of the message.

1851 A dialog contains certain pieces of state needed for further message transmissions within the dialog.
1852 This state consists of the dialog ID, a local sequence number (used to order requests from the UA to its

1853 peer), a remote sequence number (used to order requests from its peer to the UA), a local URI, a remote
1854 URI, the Contact URI of the peer, a boolean flag called “secure”, and a route set, which is an ordered list of
1855 URIs. The route set is the list of servers that need to be traversed to send a request to the peer. A dialog can
1856 also be in the “early” state, which occurs when it is created with a provisional response, and then transition
1857 to the “confirmed” state when a 2xx final response arrives. For other responses, or if no response arrives at
1858 all on that dialog, the early dialog terminates.

1859 **12.1 Creation of a Dialog**

1860 Dialogs are created through the generation of non-failure responses to requests with specific methods.
1861 Within this specification, only 2xx and 101-199 responses with a To tag to INVITE establish a dialog.
1862 A dialog established by a non-final response to a request is in the “early” state and it is called an early dia-
1863 log. Extensions MAY define other means for creating dialogs. Section 13 gives more details that are specific
1864 to the INVITE method. Here, we describe the process for creation of dialog state that is not dependent on
1865 the method.

1866 UAs MUST assign values to the dialog ID components as described below.

1867 **12.1.1 UAS behavior**

1868 When a UAS responds to a request with a response that establishes a dialog (such as a 2xx to INVITE),
1869 the UAS MUST copy all Record-Route header field values from the request into the response (including
1870 the URIs, URI parameters, and any Record-Route header field parameters, whether they are known or
1871 unknown to the UAS) and MUST maintain the order of those values. The UAS MUST add a Contact header
1872 field to the response. The Contact header field contains an address where the UAS would like to be con-
1873 tacted for subsequent requests in the dialog (which includes the ACK for a 2xx response in the case of an
1874 INVITE). Generally, the host portion of this URI is the IP address or FQDN of the host. The URI provided
1875 in the Contact header field MUST be a SIP or SIPS URI. If the request that initiated the dialog contained
1876 a SIPS URI in the Request-URI or in the top Record-Route header field value, if there was any, or the
1877 Contact header field if there was no Record-Route header field, the Contact header field in the response
1878 MUST be a SIPS URI. The URI SHOULD have global scope (that is, the same URI can be used in messages
1879 outside this dialog). The same way, the scope of the URI in the Contact header field of the INVITE is not
1880 limited to this dialog either. It can therefore be used in messages to the UAC even outside this dialog.

1881 The UAS then constructs the state of the dialog. This state MUST be maintained for the duration of the
1882 dialog.

1883 If the request arrived over TLS, and the Request-URI contained a SIPS URI, the “secure” flag is set to
1884 TRUE.

1885 The route set MUST be set to the list of URIs in the Record-Route header field from the request, taken
1886 in order and preserving all URI parameters. If no Record-Route header field is present in the request, the
1887 route set MUST be set to the empty set. This route set, even if empty, overrides any pre-existing route set for
1888 future requests in this dialog. The remote target MUST be set to the URI from the Contact header field of
1889 the request.

1890 The remote sequence number MUST be set to the value of the sequence number in the CSeq header field
1891 of the request. The local sequence number MUST be empty. The call identifier component of the dialog ID
1892 MUST be set to the value of the Call-ID in the request. The local tag component of the dialog ID MUST be
1893 set to the tag in the To field in the response to the request (which always includes a tag), and the remote tag
1894 component of the dialog ID MUST be set to the tag from the From field in the request. A UAS MUST be

1895 prepared to receive a request without a tag in the **From** field, in which case the tag is considered to have a
1896 value of null.

1897 This is to maintain backwards compatibility with RFC 2543, which did not mandate **From** tags.

1898 The remote URI **MUST** be set to the URI in the **From** field, and the local URI **MUST** be set to the URI in
1899 the **To** field.

1900 12.1.2 UAC Behavior

1901 When a UAC sends a request that can establish a dialog (such as an **INVITE**) it **MUST** provide a SIP or SIPS
1902 URI with global scope (i.e., the same SIP URI can be used in messages outside this dialog) in the **Contact**
1903 header field of the request. If the request has a **Request-URI** or a topmost **Route** header field value with a
1904 SIPS URI, the **Contact** header field **MUST** contain a SIPS URI.

1905 When a UAC receives a response that establishes a dialog, it constructs the state of the dialog. This state
1906 **MUST** be maintained for the duration of the dialog.

1907 If the request was sent over TLS, and the **Request-URI** contained a SIPS URI, the “secure” flag is set
1908 to **TRUE**.

1909 The route set **MUST** be set to the list of URIs in the **Record-Route** header field from the response,
1910 taken in reverse order and preserving all URI parameters. If no **Record-Route** header field is present in
1911 the response, the route set **MUST** be set to the empty set. This route set, even if empty, overrides any pre-
1912 existing route set for future requests in this dialog. The remote target **MUST** be set to the URI from the
1913 **Contact** header field of the response.

1914 The local sequence number **MUST** be set to the value of the sequence number in the **CSeq** header field
1915 of the request. The remote sequence number **MUST** be empty (it is established when the remote UA sends
1916 a request within the dialog). The call identifier component of the dialog ID **MUST** be set to the value of the
1917 **Call-ID** in the request. The local tag component of the dialog ID **MUST** be set to the tag in the **From** field
1918 in the request, and the remote tag component of the dialog ID **MUST** be set to the tag in the **To** field of the
1919 response. A UAC **MUST** be prepared to receive a response without a tag in the **To** field, in which case the
1920 tag is considered to have a value of null.

1921 This is to maintain backwards compatibility with RFC 2543, which did not mandate **To** tags.

1922 The remote URI **MUST** be set to the URI in the **To** field, and the local URI **MUST** be set to the URI in
1923 the **From** field.

1924 12.2 Requests within a Dialog

1925 Once a dialog has been established between two UAs, either of them **MAY** initiate new transactions as needed
1926 within the dialog. The UA sending the request will take the UAC role for the transaction. The UA receiving
1927 the request will take the UAS role. Note that these may be different roles than the UAs held during the
1928 transaction that established the dialog.

1929 Requests within a dialog **MAY** contain **Record-Route** and **Contact** header fields. However, these re-
1930 quests do not cause the dialog’s route set to be modified, although they may modify the remote target URI.
1931 Specifically, requests that are not target refresh requests do not modify the dialog’s remote target URI, and
1932 requests that are target refresh requests do. For dialogs that have been established with an **INVITE**, the only
1933 target refresh request defined is re-**INVITE** (see Section 14). Other extensions may define different target
1934 refresh requests for dialogs established in other ways.

1935 Note that an **ACK** is *NOT* a target refresh request.

1936 Target refresh requests only update the dialog’s remote target URI, and not the route set formed from **Record-**
1937 **Route**. Updating the latter would introduce severe backwards compatibility problems with RFC 2543-compliant
1938 systems.

1939 **12.2.1 UAC Behavior**

1940 **12.2.1.1 Generating the Request** A request within a dialog is constructed by using many of the com-
1941 ponents of the state stored as part of the dialog.

1942 The URI in the **To** field of the request **MUST** be set to the remote URI from the dialog state. The tag
1943 in the **To** header field of the request **MUST** be set to the remote tag of the dialog ID. The **From** URI of the
1944 request **MUST** be set to the local URI from the dialog state. The tag in the **From** header field of the request
1945 **MUST** be set to the local tag of the dialog ID. If the value of the remote or local tags is null, the tag parameter
1946 **MUST** be omitted from the **To** or **From** header fields, respectively.

1947 Usage of the URI from the **To** and **From** fields in the original request within subsequent requests is done for
1948 backwards compatibility with RFC 2543, which used the URI for dialog identification. In this specification, only
1949 the tags are used for dialog identification. It is expected that mandatory reflection of the original **To** and **From** URI
1950 in mid-dialog requests will be deprecated in a subsequent revision of this specification.

1951 The **Call-ID** of the request **MUST** be set to the **Call-ID** of the dialog. Requests within a dialog **MUST**
1952 contain strictly monotonically increasing and contiguous **CSeq** sequence numbers (increasing-by-one) in
1953 each direction (excepting **ACK** and **CANCEL** of course, whose numbers equal the requests being acknowl-
1954 edged or cancelled). Therefore, if the local sequence number is not empty, the value of the local sequence
1955 number **MUST** be incremented by one, and this value **MUST** be placed into the **CSeq** header field. If the
1956 local sequence number is empty, an initial value **MUST** be chosen using the guidelines of Section 8.1.1.5.
1957 The method field in the **CSeq** header field value **MUST** match the method of the request.

1958 With a length of 32 bits, a client could generate, within a single call, one request a second for about 136 years
1959 before needing to wrap around. The initial value of the sequence number is chosen so that subsequent requests within
1960 the same call will not wrap around. A non-zero initial value allows clients to use a time-based initial sequence
1961 number. A client could, for example, choose the 31 most significant bits of a 32-bit second clock as an initial
1962 sequence number.

1963 The UAC uses the remote target and route set to build the **Request-URI** and **Route** header field of the
1964 request.

1965 If the route set is empty, the UAC **MUST** place the remote target URI into the **Request-URI**. The UAC
1966 **MUST NOT** add a **Route** header field to the request.

1967 If the route set is not empty, and the first URI in the route set contains the **lr** parameter (see Sec-
1968 tion 19.1.1), the UAC **MUST** place the remote target URI into the **Request-URI** and **MUST** include a **Route**
1969 header field containing the route set values in order, including all parameters.

1970 If the route set is not empty, and its first URI does not contain the **lr** parameter, the UAC **MUST** place
1971 the first URI from the route set into the **Request-URI**, stripping any parameters that are not allowed in a
1972 **Request-URI**. The UAC **MUST** add a **Route** header field containing the remainder of the route set values
1973 in order, including all parameters. The UAC **MUST** then place the remote target URI into the **Route** header
1974 field as the last value.

1975 For example, if the remote target is sip:user@remoteua and the route set contains

1976 <sip:proxy1>, <sip:proxy2>, <sip:proxy3;lr>, <sip:proxy4>

1977 The request will be formed with the following **Request-URI** and **Route** header field:

1978 METHOD sip:proxy1

1979 Route: <sip:proxy2>, <sip:proxy3;lr>, <sip:proxy4>, <sip:user@remoteua>

1980 If the first URI of the route set does not contain the `lr` parameter, the proxy indicated does not understand the
1981 routing mechanisms described in this document and will act as specified in RFC 2543, replacing the `Request-URI`
1982 with the first `Route` header field value it receives while forwarding the message. Placing the `Request-URI` at the
1983 end of the `Route` header field preserves the information in that `Request-URI` across the strict router (it will be
1984 returned to the `Request-URI` when the request reaches a loose-router).

1985 A UAC SHOULD include a `Contact` header field in any target refresh requests within a dialog, and unless
1986 there is a need to change it, the URI SHOULD be the same as used in previous requests within the dialog. If
1987 the “secure” flag is true, that URI MUST be a SIPS URI. As discussed in Section 12.2.2, a `Contact` header
1988 field in a target refresh request updates the remote target URI. This allows a UA to provide a new contact
1989 address, should its address change during the duration of the dialog.

1990 However, requests that are not target refresh requests do not affect the remote target URI for the dialog.
1991 The rest of the request is formed as described in Section 8.1.1.

1992 Once the request has been constructed, the address of the server is computed and the request is sent,
1993 using the same procedures for requests outside of a dialog (Section 8.1.2).

1994 The procedures in Section 8.1.2 will normally result in the request being sent to the address indicated by the
1995 topmost `Route` header field value or the `Request-URI` if no `Route` header field is present. Subject to certain
1996 restrictions, they allow the request to be sent to an alternate address (such as a default outbound proxy not represented
1997 in the route set).

1998 **12.2.1.2 Processing the Responses** The UAC will receive responses to the request from the transaction
1999 layer. If the client transaction returns a timeout this is treated as a 408 (Request Timeout) response.

2000 The behavior of a UAC that receives a 3xx response for a request sent within a dialog is the same as if
2001 the request had been sent outside a dialog. This behavior is described in Section 8.1.3.4.

2002 Note, however, that when the UAC tries alternative locations, it still uses the route set for the dialog to build the
2003 `Route` header of the request.

2004 When a UAC receives a 2xx response to a target refresh request, it MUST replace the dialog’s remote
2005 target URI with the URI from the `Contact` header field in that response, if present.

2006 If the response for a request within a dialog is a 481 (Call/Transaction Does Not Exist) or a 408 (Request
2007 Timeout), the UAC SHOULD terminate the dialog. A UAC SHOULD also terminate a dialog if no response
2008 at all is received for the request (the client transaction would inform the TU about the timeout.)

2009 For INVITE initiated dialogs, terminating the dialog consists of sending a BYE.

2010 **12.2.2 UAS Behavior**

2011 Requests sent within a dialog, as any other requests, are atomic. If a particular request is accepted by the
2012 UAS, *all* the state changes associated with it are performed. If the request is rejected, *none* of the state
2013 changes is performed.

2014 Note that some requests such as INVITEs affect several pieces of state.

2015 The UAS will receive the request from the transaction layer. If the request has a tag in the `To` header
2016 field, the UAS core computes the dialog identifier corresponding to the request and compares it with existing
2017 dialogs. If there is a match, this is a mid-dialog request. In that case, the UAS first applies the same
2018 processing rules for requests outside of a dialog, discussed in Section 8.2.

2019 If the request has a tag in the `To` header field, but the dialog identifier does not match any existing di-
2020 alogs, the UAS may have crashed and restarted, or it may have received a request for a different (possibly
2021 failed) UAS (the UASs can construct the `To` tags so that a UAS can identify that the tag was for a UAS

2022 for which it is providing recovery). Another possibility is that the incoming request has been simply mis-
2023 routed. Based on the To tag, the UAS MAY either accept or reject the request. Accepting the request for
2024 acceptable To tags provides robustness, so that dialogs can persist even through crashes. UAs wishing to
2025 support this capability must take into consideration some issues such as choosing monotonically increasing
2026 CSeq sequence numbers even across reboots, reconstructing the route set, and accepting out-of-range RTP
2027 timestamps and sequence numbers.

2028 If the UAS wishes to reject the request, because it does not wish to recreate the dialog, it MUST respond
2029 to the request with a 481 (Call/Transaction Does Not Exist) status code and pass that to the server transaction.

2030 Requests that do not change in any way the state of a dialog may be received within a dialog (for
2031 example, an OPTIONS request). They are processed as if they had been received outside the dialog.

2032 If the remote sequence number is empty, it MUST be set to the value of the sequence number in the CSeq
2033 header field value in the request. If the remote sequence number was not empty, but the sequence number of
2034 the request is lower than the remote sequence number, the request is out of order and MUST be rejected with
2035 a 500 (Server Internal Error) response. If the remote sequence number was not empty, and the sequence
2036 number of the request is greater than the remote sequence number, the request is in order. It is possible for
2037 the CSeq sequence number to be higher than the remote sequence number by more than one. This is not
2038 an error condition, and a UAS SHOULD be prepared to receive and process requests with CSeq values more
2039 than one higher than the previous received request. The UAS MUST then set the remote sequence number to
2040 the value of the sequence number in the CSeq header field value in the request.

2041 If a proxy challenges a request generated by the UAC, the UAC has to resubmit the request with credentials. The
2042 resubmitted request will have a new CSeq number. The UAS will never see the first request, and thus, it will notice
2043 a gap in the CSeq number space. Such a gap does not represent any error condition.

2044 When a UAS receives a target refresh request, it MUST replace the dialog's remote target URI with the
2045 URI from the Contact header field in that request, if present.

2046 12.3 Termination of a Dialog

2047 Independent of the method, if a request outside of a dialog generates a non-2xx final response, any early
2048 dialogs created through provisional responses to that request are terminated. The mechanism for terminating
2049 confirmed dialogs is method specific. In this specification, the BYE method terminates a session and the
2050 dialog associated with it. See Section 15 for details.

2051 13 Initiating a Session

2052 13.1 Overview

2053 When a user agent client desires to initiate a session (for example, audio, video, or a game), it formulates an
2054 INVITE request. The INVITE request asks a server to establish a session. This request may be forwarded by
2055 proxies, eventually arriving at one or more UAS that can potentially accept the invitation. These UASs will
2056 frequently need to query the user about whether to accept the invitation. After some time, those UAS can
2057 accept the invitation (meaning the session is to be established) by sending a 2xx response. If the invitation
2058 is not accepted, a 3xx, 4xx, 5xx or 6xx response is sent, depending on the reason for the rejection. Before
2059 sending a final response, the UAS can also send provisional responses (1xx) to advise the UAC of progress
2060 in contacting the called user.

2061 After possibly receiving one or more provisional responses, the UAC will get one or more 2xx responses
2062 or one non-2xx final response. Because of the protracted amount of time it can take to receive final responses
2063 to INVITE, the reliability mechanisms for INVITE transactions differ from those of other requests (like

2064 OPTIONS). Once it receives a final response, the UAC needs to send an ACK for every final response
2065 it receives. The procedure for sending this ACK depends on the type of response. For final responses
2066 between 300 and 699, the ACK processing is done in the transaction layer and follows one set of rules (See
2067 Section 17). For 2xx responses, the ACK is generated by the UAC core.

2068 A 2xx response to an INVITE establishes a session, and it also creates a dialog between the UA that
2069 issued the INVITE and the UA that generated the 2xx response. Therefore, when multiple 2xx responses are
2070 received from different remote UAs (because the INVITE forked), each 2xx establishes a different dialog.
2071 All these dialogs are part of the same call.

2072 This section provides details on the establishment of a session using INVITE. A UA that supports IN-
2073 VITE MUST also support ACK, CANCEL and BYE.

2074 13.2 UAC Processing

2075 13.2.1 Creating the Initial INVITE

2076 Since the initial INVITE represents a request outside of a dialog, its construction follows the procedures of
2077 Section 8.1.1. Additional processing is required for the specific case of INVITE.

2078 An Allow header field (Section 20.5) SHOULD be present in the INVITE. It indicates what methods can
2079 be invoked within a dialog, on the UA sending the INVITE, for the duration of the dialog. For example, a
2080 UA capable of receiving INFO requests within a dialog [34] SHOULD include an Allow header field listing
2081 the INFO method.

2082 A Supported header field (Section 20.37) SHOULD be present in the INVITE. It enumerates all the
2083 extensions understood by the UAC.

2084 An Accept (Section 20.1) header field MAY be present in the INVITE. It indicates which Content-Types
2085 are acceptable to the UA, in both the response received by it, and in any subsequent requests sent to it within
2086 dialogs established by the INVITE. The Accept header field is especially useful for indicating support of
2087 various session description formats.

2088 The UAC MAY add an Expires header field (Section 20.19) to limit the validity of the invitation. If the
2089 time indicated in the Expires header field is reached and no final answer for the INVITE has been received
2090 the UAC core SHOULD generate a CANCEL request for the INVITE, as per Section 9.

2091 A UAC MAY also find it useful to add, among others, Subject (Section 20.36), Organization (Sec-
2092 tion 20.25) and User-Agent (Section 20.41) header fields. They all contain information related to the
2093 INVITE.

2094 The UAC MAY choose to add a message body to the INVITE. Section 8.1.1.10 deals with how to con-
2095 struct the header fields – Content-Type among others – needed to describe the message body.

2096 There are special rules for message bodies that contain a session description - their corresponding
2097 Content-Disposition is "session". SIP uses an offer/answer model where one UA sends a session de-
2098 scription, called the offer, which contains a proposed description of the session. The offer indicates the
2099 desired communications means (audio, video, games), parameters of those means (such as codec types) and
2100 addresses for receiving media from the answerer. The other UA responds with another session description,
2101 called the answer, which indicates which communications means are accepted, the parameters that apply to
2102 those means, and addresses for receiving media from the offerer. The offer/answer model defines restrictions
2103 on when offers and answers can be made. This results in restrictions on where the offers and answers can
2104 appear in SIP messages. In this specification, offers and answers can only appear in INVITE requests and re-
2105 sponses, and ACK. The usage of offers and answers is further restricted. For the initial INVITE transaction,
2106 the rules are:

- 2107 • The initial offer **MUST** be in either an **INVITE** or, if not there, in the first reliable non-failure message
2108 from the UAS back to the UAC. In this specification, that is the final 2xx response.
- 2109 • If the initial offer is in an **INVITE**, the answer **MUST** be in a reliable non-failure message from UAS
2110 back to UAC which is correlated to that **INVITE**. For this specification, that is only the final 2xx
2111 response to that **INVITE**. That same exact answer **MAY** also be placed in any provisional responses
2112 sent prior to the answer. The UAC **MUST** treat the first session description it receives as the answer,
2113 and **MUST** ignore any session descriptions in subsequent responses to the initial **INVITE**.
- 2114 • If the initial offer is in the first reliable non-failure message from the UAS back to UAC, the answer
2115 **MUST** be in the acknowledgement for that message (in this specification, **ACK** for a 2xx response).
- 2116 • After having sent or received an answer to the first offer, the UAC **MAY** generate subsequent offers
2117 in requests, but only if it has received answers to any previous offers, and has not sent any offers to
2118 which it hasn't gotten an answer.
- 2119 • Once the UAS has sent or received an answer to the initial offer, it **MUST NOT** generate subsequent
2120 offers in any responses to the initial **INVITE**. This means that a UAS based on this specification alone
2121 can never generate subsequent offers until completion of the initial transaction.

2122 Concretely, the above rules specify two exchanges - the offer is in the **INVITE**, and the answer in the
2123 2xx (and possibly in a 1xx as well, with the same value), or the offer is in the 2xx, and the answer is in the
2124 **ACK**. All user agents that support **INVITE** **MUST** support these two exchanges.

2125 The Session Description Protocol (SDP) (RFC 2327 [1]) **MUST** be supported by all user agents as a
2126 means to describe sessions, and its usage for constructing offers and answers **MUST** follow the procedures
2127 defined in [13].

2128 The restrictions of the offer-answer model just described only apply to bodies whose **Content-Disposition**
2129 header field value is "session". Therefore, it is possible that both the **INVITE** and the **ACK** contain a body
2130 message (for example, the **INVITE** carries a photo (**Content-Disposition: render**) and the **ACK** a session
2131 description (**Content-Disposition: session**)).

2132 If the **Content-Disposition** header field is missing, bodies of **Content-Type application/sdp** imply the
2133 disposition "session", while other content types imply "render".

2134 Once the **INVITE** has been created, the UAC follows the procedures defined for sending requests outside
2135 of a dialog (Section 8). This results in the construction of a client transaction that will ultimately send the
2136 request and deliver responses to the UAC.

2137 13.2.2 Processing **INVITE** Responses

2138 Once the **INVITE** has been passed to the **INVITE** client transaction, the UAC waits for responses for the
2139 **INVITE**. If the **INVITE** client transaction returns a timeout rather than a response the TU acts as if a 408
2140 (Request Timeout) response had been received, as described in Section 8.1.3.

2141 **13.2.2.1 1xx responses** Zero, one or multiple provisional responses may arrive before one or more
2142 final responses are received. Provisional responses for an **INVITE** request can create "early dialogs". If a
2143 provisional response has a tag in the **To** field, and if the dialog ID of the response does not match an existing
2144 dialog, one is constructed using the procedures defined in Section 12.1.2.

2145 The early dialog will only be needed if the UAC needs to send a request to its peer within the dialog
2146 before the initial INVITE transaction completes. Header fields present in a provisional response are appli-
2147 cable as long as the dialog is in the early state (for example, an Allow header field in a provisional response
2148 contains the methods that can be used in the dialog while this is in the early state).

2149 **13.2.2.2 3xx responses** A 3xx response may contain one or more Contact header field values provid-
2150 ing new addresses where the callee might be reachable. Depending on the status code of the 3xx response
2151 (see Section 21.3) the UAC MAY choose to try those new addresses.

2152 **13.2.2.3 4xx, 5xx and 6xx responses** A single non-2xx final response may be received for the IN-
2153 VITE. 4xx, 5xx and 6xx responses may contain a Contact header field value indicating the location where
2154 additional information about the error can be found. Subsequent final responses (which would only arrive
2155 under error conditions) MUST be ignored.

2156 All early dialogs are considered terminated upon reception of the non-2xx final response.

2157 After having received the non-2xx final response the UAC core considers the INVITE transaction com-
2158 pleted. The INVITE client transaction handles generation of ACKs for the response (see Section 17).

2159 **13.2.2.4 2xx responses** Multiple 2xx responses may arrive at the UAC for a single INVITE request
2160 due to a forking proxy. Each response is distinguished by the tag parameter in the To header field, and each
2161 represents a distinct dialog, with a distinct dialog identifier.

2162 If the dialog identifier in the 2xx response matches the dialog identifier of an existing dialog, the dialog
2163 MUST be transitioned to the “confirmed” state, and the route set for the dialog MUST be recomputed based
2164 on the 2xx response using the procedures of Section 12.2.1.2. Otherwise, a new dialog in the “confirmed”
2165 state MUST be constructed using the procedures of Section 12.1.2.

2166 Note that the only piece of state that is recomputed is the route set. Other pieces of state such as the highest
2167 sequence numbers (remote and local) sent within the dialog are not recomputed. The route set only is recomputed
2168 for backwards compatibility. RFC 2543 did not mandate mirroring of the Record-Route header field in a 1xx, only
2169 2xx. However, we cannot update the entire state of the dialog, since mid-dialog requests may have been sent within
2170 the early dialog, modifying the sequence numbers, for example.

2171 The UAC core MUST generate an ACK request for each 2xx received from the transaction layer. The
2172 header fields of the ACK are constructed in the same way as for any request sent within a dialog (see
2173 Section 12) with the exception of the CSeq and the header fields related to authentication. The sequence
2174 number of the CSeq header field MUST be the same as the INVITE being acknowledged, but the CSeq
2175 method MUST be ACK. The ACK MUST contain the same credentials as the INVITE. If the 2xx contains
2176 an offer (based on the rules above), the ACK MUST carry an answer in its body. If the offer in the 2xx
2177 response is not acceptable, the UAC core MUST generate a valid answer in the ACK and then send a BYE
2178 immediately.

2179 Once the ACK has been constructed, the procedures of [4] are used to determine the destination address,
2180 port and transport. However, the request is passed to the transport layer directly for transmission, rather than
2181 a client transaction. This is because the UAC core handles retransmissions of the ACK, not the transaction
2182 layer. The ACK MUST be passed to the client transport every time a retransmission of the 2xx final response
2183 that triggered the ACK arrives.

2184 The UAC core considers the INVITE transaction completed 64*T1 seconds after the reception of the
2185 first 2xx response. At this point all the early dialogs that have not transitioned to established dialogs are
2186 terminated. Once the INVITE transaction is considered completed by the UAC core, no more new 2xx
2187 responses are expected to arrive.

2188 If, after acknowledging any 2xx response to an INVITE, the UAC does not want to continue with that
2189 dialog, then the UAC MUST terminate the dialog by sending a BYE request as described in Section 15.

2190 **13.3 UAS Processing**

2191 **13.3.1 Processing of the INVITE**

2192 The UAS core will receive INVITE requests from the transaction layer. It first performs the request process-
2193 ing procedures of Section 8.2, which are applied for both requests inside and outside of a dialog.

2194 Assuming these processing states complete without generating a response, the UAS core performs the
2195 additional processing steps:

- 2196 1. If the request is an INVITE that contains an Expires header field the UAS core sets a timer for
2197 the number of seconds indicated in the header field value. When the timer fires, the invitation is
2198 considered to be expired. If the invitation expires before the UAS has generated a final response, a
2199 487 (Request Terminated) response SHOULD be generated.
- 2200 2. If the request is a mid-dialog request, the method-independent processing described in Section 12.2.2
2201 is first applied. It might also modify the session; Section 14 provides details.
- 2202 3. If the request has a tag in the To header field but the dialog identifier does not match any of the
2203 existing dialogs, the UAS may have crashed and restarted, or may have received a request for a
2204 different (possibly failed) UAS. Section 12.2.2 provides guidelines to achieve a robust behavior under
2205 such a situation.

2206 Processing from here forward assumes that the INVITE is outside of a dialog, and is thus for the purposes
2207 of establishing a new session.

2208 The INVITE may contain a session description, in which case the UAS is being presented with an offer
2209 for that session. It is possible that the user is already a participant in that session, even though the INVITE
2210 is outside of a dialog. This can happen when a user is invited to the same multicast conference by multiple
2211 other participants. If desired, the UAS MAY use identifiers within the session description to detect this
2212 duplication. For example, SDP contains a session id and version number in the origin (o) field. If the user
2213 is already a member of the session, and the session parameters contained in the session description have
2214 not changed, the UAS MAY silently accept the INVITE (that is, send a 2xx response without prompting the
2215 user).

2216 If the INVITE does not contain a session description, the UAS is being asked to participate in a session,
2217 and the UAC has asked that the UAS provide the offer of the session. It MUST provide the offer in its first
2218 non-failure reliable message back to the UAC. In this specification, that is a 2xx response to the INVITE.

2219 The UAS can indicate progress, accept, redirect, or reject the invitation. In all of these cases, it formu-
2220 lates a response using the procedures described in Section 8.2.6.

2221 **13.3.1.1 Progress** If the UAS is not able to answer the invitation immediately, it can choose to indicate
2222 some kind of progress to the UAC (for example, an indication that a phone is ringing). This is accomplished
2223 with a provisional response between 101 and 199. These provisional responses establish early dialogs and
2224 therefore follow the procedures of Section 12.1.1 in addition to those of Section 8.2.6. A UAS MAY send
2225 as many provisional responses as it likes. Each of these MUST indicate the same dialog ID. However, these
2226 will not be delivered reliably.

2227 If the UAS desires an extended period of time to answer the INVITE, it will need to ask for an “ex-
2228 tension” in order to prevent proxies from canceling the transaction. A proxy has the option of canceling a
2229 transaction when there is a gap of 3 minutes between messages in a transaction. To prevent cancellation, the
2230 UAS MUST send a non-100 provisional response at every minute, to handle the possibility of lost provisional
2231 responses.

2232 An INVITE transaction can go on for extended durations when the user is placed on hold, or when interworking
2233 with PSTN systems which allow communications to take place without answering the call. The latter is common in
2234 Interactive Voice Response (IVR) systems.

2235 **13.3.1.2 The INVITE is redirected** If the UAS decides to redirect the call, a 3xx response is sent. A
2236 300 (Multiple Choices), 301 (Moved Permanently) or 302 (Moved Temporarily) response SHOULD contain
2237 a Contact header field containing one or more URIs of new addresses to be tried. The response is passed to
2238 the INVITE server transaction, which will deal with its retransmissions.

2239 **13.3.1.3 The INVITE is rejected** A common scenario occurs when the callee is currently not willing
2240 or able to take additional calls at this end system. A 486 (Busy Here) SHOULD be returned in such scenario.
2241 If the UAS knows that no other end system will be able to accept this call a 600 (Busy Everywhere) response
2242 SHOULD be sent instead. However, it is unlikely that a UAS will be able to know this in general, and thus
2243 this response will not usually be used. The response is passed to the INVITE server transaction, which will
2244 deal with its retransmissions.

2245 A UAS rejecting an offer contained in an INVITE SHOULD return a 488 (Not Acceptable Here) response.
2246 Such a response SHOULD include a Warning header field value explaining why the offer was rejected.

2247 **13.3.1.4 The INVITE is accepted** The UAS core generates a 2xx response. This response establishes
2248 a dialog, and therefore follows the procedures of Section 12.1.1 in addition to those of Section 8.2.6.

2249 A 2xx response to an INVITE SHOULD contain the Allow header field and the Supported header field,
2250 and MAY contain the Accept header field. Including these header fields allows the UAC to determine the
2251 features and extensions supported by the UAS for the duration of the call, without probing.

2252 If the INVITE request contained an offer, and the UAS had not yet sent an answer, the 2xx MUST contain
2253 an answer. If the INVITE did not contain an offer, the 2xx MUST contain an offer if the UAS had not yet
2254 sent an offer.

2255 Once the response has been constructed it is passed to the INVITE server transaction. Note, however,
2256 that the INVITE server transaction will be destroyed as soon as it receives this final response and passes it
2257 to the transport. Therefore, it is necessary to pass periodically the response directly to the transport until
2258 the ACK arrives. The 2xx response is passed to the transport with an interval that starts at T1 seconds and
2259 doubles for each retransmission until it reaches T2 seconds (T1 and T2 are defined in Section 17). Response
2260 retransmissions cease when an ACK request for the response is received. This is independent of whatever
2261 transport protocols are used to send the response.

2262 Since 2xx is retransmitted end-to-end, there may be hops between UAS and UAC that are UDP. To ensure reliable
2263 delivery across these hops, the response is retransmitted periodically even if the transport at the UAS is reliable.

2264 If the server retransmits the 2xx response for $64 * T1$ seconds without receiving an ACK, the dialog
2265 is confirmed, but the session SHOULD be terminated. This is accomplished with a BYE as described in
2266 Section 15.

2267 14 Modifying an Existing Session

2268 A successful INVITE request (see Section 13) establishes both a dialog between two user agents and a
2269 session using the offer-answer model. Section 12 explains how to modify an existing dialog using a target
2270 refresh request (for example, changing the remote target URI of the dialog). This section describes how
2271 to modify the actual session. This modification can involve changing addresses or ports, adding a media
2272 stream, deleting a media stream, and so on. This is accomplished by sending a new INVITE request within
2273 the same dialog that established the session. An INVITE request sent within an existing dialog is known as
2274 a re-INVITE.

2275 Note that a single re-INVITE can modify the dialog and the parameters of the session at the same time.

2276 Either the caller or callee can modify an existing session.

2277 The behavior of a UA on detection of media failure is a matter of local policy. However, automated
2278 generation of re-INVITE or BYE is NOT RECOMMENDED to avoid flooding the network with traffic when
2279 there is congestion. In any case, if these messages are sent automatically, they SHOULD be sent after some
2280 randomized interval.

2281 Note that the paragraph above refers to automatically generated BYEs and re-INVITEs. If the user hangs up
2282 upon media failure the UA would send a BYE request as usual.

2283 14.1 UAC Behavior

2284 The same offer-answer model that applies to session descriptions in INVITEs (Section 13.2.1) applies to
2285 re-INVITEs. As a result, a UAC that wants to add a media stream, for example, will create a new offer that
2286 contains this media stream, and send that in an INVITE request to its peer. It is important to note that the full
2287 description of the session, not just the change, is sent. This supports stateless session processing in various
2288 elements, and supports failover and recovery capabilities. Of course, a UAC MAY send a re-INVITE with no
2289 session description, in which case the first reliable non-failure response to the re-INVITE will contain the
2290 offer (in this specification, that is a 2xx response).

2291 If the session description format has the capability for version numbers, the offerer SHOULD indicate
2292 that the version of the session description has changed.

2293 The To, From, Call-ID, CSeq, and Request-URI of a re-INVITE are set following the same rules as
2294 for regular requests within an existing dialog, described in Section 12.

2295 A UAC MAY choose not to add an Alert-Info header field or a body with Content-Disposition "alert"
2296 to re-INVITEs because UASs do not typically alert the user upon reception of a re-INVITE.

2297 Unlike an INVITE, which can fork, a re-INVITE will never fork, and therefore, only ever generate a
2298 single final response. The reason a re-INVITE will never fork is that the Request-URI identifies the target
2299 as the UA instance it established the dialog with, rather than identifying an address-of-record for the user.

2300 Note that a UAC MUST NOT initiate a new INVITE transaction within a dialog while another INVITE
2301 transaction is in progress in either direction.

- 2302 1. If there is an ongoing INVITE client transaction, the TU MUST wait until the transaction reaches the
2303 *completed* or *terminated* state before initiating the new INVITE.
- 2304 2. If there is an ongoing INVITE server transaction, the TU MUST wait until the transaction reaches the
2305 *confirmed* or *terminated* state before initiating the new INVITE.

2306 However, a UA MAY initiate a regular transaction while an INVITE transaction is in progress. A UA
2307 MAY also initiate an INVITE transaction while a regular transaction is in progress.

2308 If a UA receives a non-2xx final response to a re-INVITE, the session parameters MUST remain un-
2309 changed, as if no re-INVITE had been issued. Note that, as stated in Section 12.2.1.2, if the non-2xx final
2310 response is a 481 (Call/Transaction Does Not Exist), or a 408 (Request Timeout), or no response at all is
2311 received for the re-INVITE (that is, a timeout is returned by the INVITE client transaction), the UAC will
2312 terminate the dialog.

2313 If a UAC receives a 491 response to a re-INVITE, it SHOULD start a timer with a value T chosen as
2314 follows:

- 2315 1. If the UAC is the owner of the Call-ID of the dialog ID (meaning it generated the value), T has a
2316 randomly chosen value between 2.1 and 4 seconds in units of 10 ms.
- 2317 2. If the UAC is *not* the owner of the Call-ID of the dialog ID, T has a randomly chosen value of between
2318 0 and 2 seconds in units of 10 ms.

2319 When the timer fires, the UAC SHOULD attempt the re-INVITE once more, if it still desires for that
2320 session modification to take place. For example, if the call was already hung up with a BYE, the re-INVITE
2321 would not take place.

2322 The rules for transmitting a re-INVITE and for generating an ACK for a 2xx response to re-INVITE are
2323 the same as for the initial INVITE (Section 13.2.1).

2324 14.2 UAS Behavior

2325 Section 13.3.1 describes the procedure for distinguishing incoming re-INVITEs from incoming initial IN-
2326 VITEs and handling a re-INVITE for an existing dialog.

2327 A UAS that receives a second INVITE before it sends the final response to a first INVITE with a lower
2328 CSeq sequence number on the same dialog MUST return a 500 (Server Internal Error) response to the second
2329 INVITE and MUST include a Retry-After header field with a randomly chosen value of between 0 and 10
2330 seconds.

2331 A UAS that receives an INVITE on a dialog while an INVITE it had sent on that dialog is in progress
2332 MUST return a 491 (Request Pending) response to the received INVITE.

2333 If a UA receives a re-INVITE for an existing dialog, it MUST check any version identifiers in the session
2334 description or, if there are no version identifiers, the content of the session description to see if it has changed.
2335 If the session description has changed, the UAS MUST adjust the session parameters accordingly, possibly
2336 after asking the user for confirmation.

2337 Versioning of the session description can be used to accommodate the capabilities of new arrivals to a conference,
2338 add or delete media, or change from a unicast to a multicast conference.

2339 If the new session description is not acceptable, the UAS can reject it by returning a 488 (Not Acceptable
2340 Here) response for the re-INVITE. This response SHOULD include a Warning header field.

2341 If a UAS generates a 2xx response and never receives an ACK, it SHOULD generate a BYE to terminate
2342 the dialog.

2343 A UAS MAY choose not to generate 180 (Ringing) responses for a re-INVITE because UACs do not
2344 typically render this information to the user. For the same reason, UASs MAY choose not to use an Alert-
2345 Info header field or a body with Content-Disposition "alert" in responses to a re-INVITE.

2346 A UAS providing an offer in a 2xx (because the INVITE did not contain an offer) SHOULD construct
2347 the offer as if the UAS were making a brand new call, subject to the constraints of sending an offer that
2348 updates an existing session, as described in [13] in the case of SDP. Specifically, this means that it SHOULD
2349 include as many media formats and media types that the UA is willing to support. The UAS MUST ensure

2350 that the session description overlaps with its previous session description in media formats, transports, or
2351 other parameters that require support from the peer. This is to avoid the need for the peer to reject the session
2352 description. If, however, it is unacceptable to the UAC, the UAC SHOULD generate an answer with a valid
2353 session description, and then send a BYE to terminate the session.

2354 15 Terminating a Session

2355 This section describes the procedures for terminating a session established by SIP. The state of the session
2356 and the state of the dialog are very closely related. When a session is initiated with an INVITE, each 1xx or
2357 2xx response from a distinct UAS creates a dialog, and if that response completes the offer/answer exchange,
2358 it also creates a session. As a result, each session is "associated" with a single dialog - the one which resulted
2359 in its creation. If an initial INVITE generates a non-2xx final response, that terminates all sessions (if any)
2360 and all dialogs (if any) that were created through responses to the request. By virtue of completing the
2361 transaction, a non-2xx final response also prevents further sessions from being created as a result of the
2362 INVITE. The BYE request is used to terminate a specific session or attempted session. In this case, the
2363 specific session is the one with the peer UA on the other side of the dialog. When a BYE is received on a
2364 dialog, any session associated with that dialog SHOULD terminate. A UA MUST NOT send a BYE outside of
2365 a dialog. The caller's UA MAY send a BYE for either confirmed or early dialogs, and the callee's UA MAY
2366 send a BYE on confirmed dialogs, but MUST NOT send a BYE on early dialogs. However, the callee's UA
2367 MUST NOT send a BYE on a confirmed dialog until it has received an ACK for its 2xx response or until the
2368 server transaction times out. If no SIP extensions have defined other application layer state associated with
2369 the dialog, the BYE also terminates the dialog.

2370 The impact of a non-2xx final response to INVITE on dialogs and sessions makes the use of CANCEL
2371 attractive. The CANCEL attempts to force a non-2xx response to the INVITE (in particular, a 487). There-
2372 fore, if a UAC wishes to give up on its call attempt entirely, it can send a CANCEL. If the INVITE results in
2373 2xx final response(s) to the INVITE, this means that a UAS accepted the invitation while the CANCEL was
2374 in progress. The UAC MAY continue with the sessions established by any 2xx responses, or MAY terminate
2375 them with BYE.

2376 The notion of "hanging up" is not well defined within SIP. It is specific to a particular, albeit common, user
2377 interface. Typically, when the user hangs up, it indicates a desire to terminate the attempt to establish a session, and
2378 to terminate any sessions already created. For the caller's UA, this would imply a CANCEL request if the initial
2379 INVITE has not generated a final response, and a BYE to all confirmed dialogs after a final response. For the callee's
2380 UA, it would typically imply a BYE; presumably, when the user picked up the phone, a 2xx was generated, and so
2381 hanging up would result in a BYE after the ACK is received. This does not mean a user cannot hang up before
2382 receipt of the ACK, it just means that the software in his phone needs to maintain state for a short while in order to
2383 clean up properly. If the particular UI allows for the user to reject a call before its answered, a 403 (Forbidden) is a
2384 good way to express that. As per the rules above, a BYE can't be sent.

2385 15.1 Terminating a Session with a BYE Request

2386 15.1.1 UAC Behavior

2387 A BYE request is constructed as would any other request within a dialog, as described in Section 12.

2388 Once the BYE is constructed, the UAC core creates a new non-INVITE client transaction, and passes it
2389 the BYE request. The UAC MUST consider the session terminated (and therefore stop sending or listening
2390 for media) as soon as the BYE request is passed to the client transaction. If the response for the BYE is a
2391 481 (Call/Transaction Does Not Exist) or a 408 (Request Timeout) or no response at all is received for the

2392 BYE (that is, a timeout is returned by the client transaction), the UAC MUST consider the session and the
2393 dialog terminated.

2394 15.1.2 UAS Behavior

2395 A UAS first processes the BYE request according to the general UAS processing described in Section 8.2.
2396 A UAS core receiving a BYE request checks if it matches an existing dialog. If the BYE does not match an
2397 existing dialog, the UAS core SHOULD generate a 481 (Call/Transaction Does Not Exist) response and pass
2398 that to the server transaction.

2399 This rule means that a BYE sent without tags by a UAC will be rejected. This is a change from RFC 2543, which
2400 allowed BYE without tags.

2401 A UAS core receiving a BYE request for an existing dialog MUST follow the procedures of Sec-
2402 tion 12.2.2 to process the request. Once done, the UAS SHOULD terminate the session (and therefore stop
2403 sending and listening for media). The only case where it can elect not to are multicast sessions, where par-
2404 ticipation is possible even if the other participant in the dialog has terminated its involvement in the session.
2405 Whether or not it ends its participation on the session, the UAS core MUST generate a 2xx response to the
2406 BYE, and MUST pass that to the server transaction for transmission.

2407 The UAS MUST still respond to any pending requests received for that dialog. It is RECOMMENDED that
2408 a 487 (Request Terminated) response is generated to those pending requests.

2409 16 Proxy Behavior

2410 16.1 Overview

2411 SIP proxies are elements that route SIP requests to user agent servers and SIP responses to user agent clients.
2412 A request may traverse several proxies on its way to a UAS. Each will make routing decisions, modifying
2413 the request before forwarding it to the next element. Responses will route through the same set of proxies
2414 traversed by the request in the reverse order.

2415 Being a proxy is a logical role for a SIP element. When a request arrives, an element that can play the
2416 role of a proxy first decides if it needs to respond to the request on its own. For instance, the request may be
2417 malformed or the element may need credentials from the client before acting as a proxy. The element MAY
2418 respond with any appropriate error code. When responding directly to a request, the element is playing the
2419 role of a UAS and MUST behave as described in Section 8.2.

2420 A proxy can operate in either a stateful or stateless mode for each new request. When stateless, a proxy
2421 acts as a simple forwarding element. It forwards each request downstream to a single element determined by
2422 making a targeting and routing decision based on the request. It simply forwards every response it receives
2423 upstream. A stateless proxy discards information about a message once the message has been forwarded.
2424 A stateful proxy remembers information (specifically, transaction state) about each incoming request and
2425 any requests it sends as a result of processing the incoming request. It uses this information to affect the
2426 processing of future messages associated with that request. A stateful proxy MAY choose to “fork” a request,
2427 routing it to multiple destinations. Any request that is forwarded to more than one location MUST be handled
2428 statefully.

2429 In some circumstances, a proxy MAY forward requests using stateful transports (such as TCP) without
2430 being transaction-stateful. For instance, a proxy MAY forward a request from one TCP connection to another
2431 transaction statelessly as long as it places enough information in the message to be able to forward the

2432 response down the same connection the request arrived on. Requests forwarded between different types of
2433 transports where the proxy's TU must take an active role in ensuring reliable delivery on one of the transports
2434 MUST be forwarded transaction statefully.

2435 A stateful proxy MAY transition to stateless operation at any time during the processing of a request,
2436 so long as it did not do anything that would otherwise prevent it from being stateless initially (forking, for
2437 example, or generation of a 100 response). When performing such a transition, all state is simply discarded.
2438 The proxy SHOULD NOT initiate a CANCEL request.

2439 Much of the processing involved when acting statelessly or statefully for a request is identical. The next
2440 several subsections are written from the point of view of a stateful proxy. The last section calls out those
2441 places where a stateless proxy behaves differently.

2442 16.2 Stateful Proxy

2443 When stateful, a proxy is purely a SIP transaction processing engine. Its behavior is modeled here in terms of
2444 the server and client transactions defined in Section 17. A stateful proxy has a server transaction associated
2445 with one or more client transactions by a higher layer proxy processing component (see figure 3), known as
2446 a proxy core. An incoming request is processed by a server transaction. Requests from the server transaction
2447 are passed to a proxy core. The proxy core determines where to route the request, choosing one or more
2448 next-hop locations. An outgoing request for each next-hop location is processed by its own associated
2449 client transaction. The proxy core collects the responses from the client transactions and uses them to send
2450 responses to the server transaction.

2451 A stateful proxy creates a new server transaction for each new request received. Any retransmissions
2452 of the request will then be handled by that server transaction per Section 17. The proxy core MUST behave
2453 as a UAS with respect to sending an immediate provisional on that server transaction (such as 100 Trying)
2454 as described in Section 8.2.6. Thus, a stateful proxy SHOULD NOT generate 100 Trying responses to non-
2455 INVITE requests.

2456 This is a model of proxy behavior, not of software. An implementation is free to take any approach that
2457 replicates the external behavior this model defines.

2458 For all new requests, including any with unknown methods, an element intending to proxy the request
2459 MUST:

- 2460 1. Validate the request (Section 16.3)
- 2461 2. Preprocess routing information (Section 16.4)
- 2462 3. Determine target(s) for the request (Section 16.5)
- 2463 4. Forward the request to each target (Section 16.6)
- 2464 5. Process all responses (Section 16.7)

2465 16.3 Request Validation

2466 Before an element can proxy a request, it MUST verify the message's validity. A valid message must pass
2467 the following checks:

- 2468 1. Reasonable Syntax

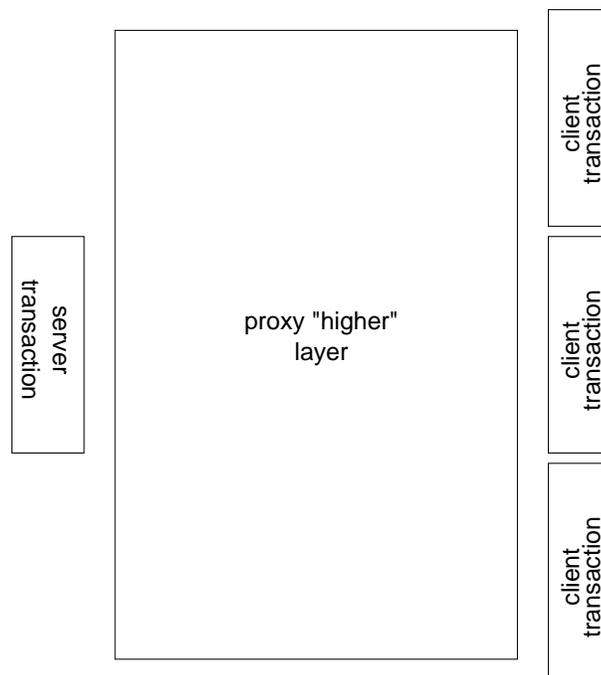


Figure 3: Stateful Proxy Model

- 2469 2. URI scheme
2470 3. Max-Forwards
2471 4. (Optional) Loop Detection
2472 5. Proxy-Require
2473 6. Proxy-Authorization

2474 If any of these checks fail, the element **MUST** behave as a user agent server (see Section 8.2) and respond
2475 with an error code.

2476 Notice that a proxy is not required to detect merged requests and MUST NOT treat merged requests as an
2477 error condition. The endpoints receiving the requests will resolve the merge as described in Section 8.2.2.2.

2478 1. Reasonable syntax check

2479 The request MUST be well-formed enough to be handled with a server transaction. Any components
2480 involved in the remainder of these Request Validation steps or the Request Forwarding section MUST
2481 be well-formed. Any other components, well-formed or not, SHOULD be ignored and remain un-
2482 changed when the message is forwarded. For instance, an element would not reject a request because
2483 of a malformed Date header field. Likewise, a proxy would not remove a malformed Date header
2484 field before forwarding a request.

2485 This protocol is designed to be extended. Future extensions may define new methods and header fields
2486 at any time. An element MUST NOT refuse to proxy a request because it contains a method or header
2487 field it does not know about.

2488 2. URI scheme check

2489 If the Request-URI has a URI whose scheme is not understood by the proxy, the proxy SHOULD
2490 reject the request with a 416 (Unsupported URI Scheme) response.

2491 3. Max-Forwards check

2492 The Max-Forwards header field (Section 20.22) is used to limit the number of elements a SIP request
2493 can traverse.

2494 If the request does not contain a Max-Forwards header field, this check is passed.

2495 If the request contains a Max-Forwards header field with a field value greater than zero, the check is
2496 passed.

2497 If the request contains a Max-Forwards header field with a field value of zero (0), the element MUST
2498 NOT forward the request. If the request was for OPTIONS, the element MAY act as the final recipient
2499 and respond per Section 11. Otherwise, the element MUST return a 483 (Too many hops) response.

2500 4. Optional Loop Detection check

2501 An element MAY check for forwarding loops before forwarding a request. If the request contains a
2502 Via header field with a sent-by value that equals a value placed into previous requests by the proxy,
2503 the request has been forwarded by this element before. The request has either looped or is legitimately
2504 spiraling through the element. To determine if the request has looped, the element MAY perform the
2505 branch parameter calculation described in Step 8 of Section 16.6 on this message and compare it to
2506 the parameter received in that Via header field. If the parameters match, the request has looped. If
2507 they differ, the request is spiraling, and processing continues. If a loop is detected, the element MAY
2508 return a 482 (Loop Detected) response.

2509 5. Proxy-Require check

2510 Future extensions to this protocol may introduce features that require special handling by proxies.
2511 Endpoints will include a Proxy-Require header field in requests that use these features, telling the
2512 proxy not to process the request unless the feature is understood.

2513 If the request contains a Proxy-Require header field (Section 20.29) with one or more option-tags this
2514 element does not understand, the element MUST return a 420 (Bad Extension) response. The response

2515 MUST include an **Unsupported** (Section 20.40) header field listing those option-tags the element did
2516 not understand.

2517 6. Proxy-Authorization check

2518 If an element requires credentials before forwarding a request, the request **MUST** be inspected as
2519 described in Section 22.3. That section also defines what the element must do if the inspection fails.

2520 16.4 Route Information Preprocessing

2521 The proxy **MUST** inspect the **Request-URI** of the request. If the **Request-URI** of the request contains a
2522 value this proxy previously placed into a **Record-Route** header field (see Section 16.6 item 4), the proxy
2523 **MUST** replace the **Request-URI** in the request with the last value from the **Route** header field, and remove
2524 that value from the **Route** header field. The proxy **MUST** then proceed as if it received this modified request.

2525 This will only happen when the element sending the request to the proxy (which may have been an endpoint)
2526 is a strict router. This rewrite on receive is necessary to enable backwards compatibility with those elements. It
2527 also allows elements following this specification to preserve the **Request-URI** through strict-routing proxies (see
2528 Section 12.2.1.1).

2529 This requirement does not obligate a proxy to keep state in order to detect URIs it previously placed in **Record-**
2530 **Route** header fields. Instead, a proxy need only place enough information in those URIs to recognize them as values
2531 it provided when they later appear.

2532 If the **Request-URI** contains an **maddr** parameter, the proxy **MUST** check to see if its value is in the set
2533 of addresses or domains the proxy is configured to be responsible for. If the **Request-URI** has an **maddr**
2534 parameter with a value the proxy is responsible for, and the request was received using the port and transport
2535 indicated (explicitly or by default) in the **Request-URI**, the proxy **MUST** strip the **maddr** and any non-default
2536 port or transport parameter and continue processing as if those values had not been present in the request.

2537 A request may arrive with an **maddr** matching the proxy, but on a port or transport different from that indicated
2538 in the URI. Such a request needs to be forwarded to the proxy using the indicated port and transport.

2539 If the first value in the **Route** header field indicates this proxy, the proxy **MUST** remove that value from
2540 the request.

2541 16.5 Determining request targets

2542 Next, the proxy calculates the target(s) of the request. The set of targets will either be predetermined by
2543 the contents of the request or will be obtained from an abstract location service. Each target in the set is
2544 represented as a URI.

2545 If the **Request-URI** of the request contains an **maddr** parameter, the **Request-URI** **MUST** be placed
2546 into the target set as the only target URI, and the proxy **MUST** proceed to Section 16.6.

2547 If the domain of the **Request-URI** indicates a domain this element is not responsible for, the **Request-**
2548 **URI** **MUST** be placed into the target set as the only target, and the element **MUST** proceed to the task of
2549 Request Forwarding (Section 16.6).

2550 There are many circumstances in which a proxy might receive a request for a domain it is not responsible for.
2551 A firewall proxy handling outgoing calls (the way HTTP proxies handle outgoing requests) is an example of where
2552 this is likely to occur.

2553 If the target set for the request has not been predetermined as described above, this implies that the
2554 element is responsible for the domain in the Request-URI, and the element MAY use whatever mechanism
2555 it desires to determine where to send the request. Any of these mechanisms can be modeled as accessing an
2556 abstract Location Service. This may consist of obtaining information from a location service created by a SIP
2557 Registrar, reading a database, consulting a presence server, utilizing other protocols, or simply performing
2558 an algorithmic substitution on the Request-URI. When accessing the location service constructed by a
2559 registrar, the Request-URI MUST first be canonicalized as described in Section 10.3 before being used as
2560 an index. The output of these mechanisms is used to construct the target set.

2561 If the Request-URI does not provide sufficient information for the proxy to determine the target set,
2562 it SHOULD return a 485 (Ambiguous) response. This response SHOULD contain a Contact header field
2563 containing URIs of new addresses to be tried. For example, an INVITE to sip:John.Smith@company.com
2564 may be ambiguous at a proxy whose location service has multiple John Smiths listed. See Section 21.4.23
2565 for details.

2566 Any information in or about the request or the current environment of the element MAY be used in the
2567 construction of the target set. For instance, different sets may be constructed depending on contents or the
2568 presence of header fields and bodies, the time of day of the request's arrival, the interface on which the
2569 request arrived, failure of previous requests, or even the element's current level of utilization.

2570 As potential targets are located through these services, their URIs are added to the target set. Targets can
2571 only be placed in the target set once. If a target URI is already present in the set (based on the definition of
2572 equality for the URI type), it MUST NOT be added again.

2573 A proxy MUST NOT add additional targets to the target set if the Request-URI of the original request
2574 does not indicate a resource this proxy is responsible for.

2575 A proxy can only change the Request-URI of a request during forwarding if it is responsible for that URI. If
2576 the proxy is not responsible for that URI, it will not recurse on 3xx or 416 responses as described below.

2577 If the Request-URI of the original request indicates a resource this proxy is responsible for, the proxy
2578 MAY continue to add targets to the set after beginning Request Forwarding. It MAY use any information
2579 obtained during that processing to determine new targets. For instance, a proxy may choose to incorporate
2580 contacts obtained in a redirect response (3xx) into the target set. If a proxy uses a dynamic source of
2581 information while building the target set (for instance, if it consults a SIP Registrar), it SHOULD monitor
2582 that source for the duration of processing the request. New locations SHOULD be added to the target set as
2583 they become available. As above, any given URI MUST NOT be added to the set more than once.

2584 Allowing a URI to be added to the set only once reduces unnecessary network traffic, and in the case of incor-
2585 porating contacts from redirect requests prevents infinite recursion.

2586 For example, a trivial location service is a "no-op", where the target URI is equal to the incoming request
2587 URI. The request is sent to a specific next hop proxy for further processing. During request forwarding of
2588 Section 16.6, Item 6, the identity of that next hop, expressed as a SIP or SIPS URI, is inserted as the top-most
2589 Route header field value into the request.

2590 If the Request-URI indicates a resource at this proxy that does not exist, the proxy MUST return a 404
2591 (Not Found) response.

2592 If the target set remains empty after applying all of the above, the proxy MUST return an error response,
2593 which SHOULD be the 480 (Temporarily Unavailable) response.

2594 16.6 Request Forwarding

2595 As soon as the target set is non-empty, a proxy MAY begin forwarding the request. A stateful proxy MAY
2596 process the set in any order. It MAY process multiple targets serially, allowing each client transaction to
2597 complete before starting the next. It MAY start client transactions with every target in parallel. It also MAY
2598 arbitrarily divide the set into groups, processing the groups serially and processing the targets in each group
2599 in parallel.

2600 A common ordering mechanism is to use the qvalue parameter of targets obtained from Contact header
2601 fields (see Section 20.10). Targets are processed from highest qvalue to lowest. Targets with equal qvalues
2602 may be processed in parallel.

2603 A stateful proxy must have a mechanism to maintain the target set as responses are received and associate
2604 the responses to each forwarded request with the original request. For the purposes of this model, this
2605 mechanism is a "response context" created by the proxy layer before forwarding the first request.

2606 For each target, the proxy forwards the request following these steps:

- 2607 1. Make a copy of the received request
- 2608 2. Update the Request-URI
- 2609 3. Update the Max-Forwards header field
- 2610 4. Optionally add a Record-route header field value
- 2611 5. Optionally add additional header fields
- 2612 6. Postprocess routing information
- 2613 7. Determine the next-hop address, port, and transport
- 2614 8. Add a Via header field value
- 2615 9. Add a Content-Length header field if necessary
- 2616 10. Forward the new request
- 2617 11. Set timer C

2618 Each of these steps is detailed below:

2619 1. Copy request

2620 The proxy starts with a copy of the received request. The copy MUST initially contain all of the header
2621 fields from the received request. Fields not detailed in the processing described below MUST NOT be
2622 removed. The copy SHOULD maintain the ordering of the header fields as in the received request.
2623 The proxy MUST NOT reorder field values with a common field name (See Section 7.3.1). The proxy
2624 MUST NOT add to, modify, or remove the message body.

2625 An actual implementation need not perform a copy; the primary requirement is that the processing for each
2626 next hop begin with the same request.

2627 2. Request-URI

2628 The **Request-URI** in the copy's start line **MUST** be replaced with the URI for this target. If the URI
2629 contains any parameters not allowed in a Request-URI, they **MUST** be removed.

2630 This is the essence of a proxy's role. This is the mechanism through which a proxy routes a request
2631 toward its destination.

2632 In some circumstances, the received **Request-URI** is placed into the target set without being modified.
2633 For that target, the replacement above is effectively a no-op.

2634 3. Max-Forwards

2635 If the copy contains a **Max-Forwards** header field, the proxy **MUST** decrement its value by one (1).

2636 If the copy does not contain a **Max-Forwards** header field, the proxy **MUST** add one with a field value
2637 which **SHOULD** be 70.

2638 Some existing UAs will not provide a **Max-Forwards** header field in a request.

2639 4. Record-Route

2640 If this proxy wishes to remain on the path of future requests in a dialog created by this request (as-
2641 suming the request creates a dialog), it **MUST** insert a **Record-Route** header field value into the copy
2642 before any existing **Record-Route** header field values, even if a **Route** header field is already present.

2643 Requests establishing a dialog may contain a preloaded **Route** header field.

2644 If this request is already part of a dialog, the proxy **SHOULD** insert a **Record-Route** header field value
2645 if it wishes to remain on the path of future requests in the dialog. In normal endpoint operation as
2646 described in Section 12 these **Record-Route** header field values will not have any effect on the route
2647 sets used by the endpoints.

2648 The proxy will remain on the path if it chooses to not insert a **Record-Route** header field value into
2649 requests that are already part of a dialog. However, it would be removed from the path when an endpoint that
2650 has failed reconstitutes the dialog.

2651 A proxy **MAY** insert a **Record-Route** header field value into any request. If the request does not
2652 initiate a dialog, the endpoints will ignore the value. See Section 12 for details on how endpoints use
2653 the **Record-Route** header field values to construct **Route** header fields.

2654 Each proxy in the path of a request chooses whether to add a **Record-Route** header field value
2655 independently - the presence of a **Record-Route** header field in a request does not obligate this proxy
2656 to add a value.

2657 The URI placed in the **Record-Route** header field value **MUST** be a SIP URI. This URI **MUST** contain
2658 an **lr** parameter (see Section 19.1.1). This URI **MAY** be different for each destination the request is
2659 forwarded to. The URI **SHOULD NOT** contain the transport parameter unless the proxy has knowledge
2660 (such as in a private network) that the next downstream element that will be in the path of subsequent
2661 requests supports that transport.

2662 The URI this proxy provides will be used by some other element to make a routing decision. This proxy, in
2663 general, has no way to know what the capabilities of that element are, so it must restrict itself to the mandatory
2664 elements of a SIP implementation: SIP URIs and either the TCP or UDP transports.

2665 The URI placed in the **Record-Route** header field **MUST** resolve to the element inserting it (or a
2666 suitable stand-in) when the server location procedures of [4] are applied to it, so that subsequent
2667 requests reach the same SIP element. If the **Request-URI** contains a SIPS URI, or the topmost **Route**
2668 header field value (after the post processing of bullet 6) contains a SIPS URI, the URI placed into the
2669 **Record-Route** header field **MUST** be a SIPS URI. Furthermore, if the request was not received over
2670 TLS, the proxy **MUST** insert a **Record-Route** header field. In a similar fashion, a proxy that receives a
2671 request over TLS, but generates a request without a SIPS URI in the **Request-URI** or topmost **Route**
2672 header field value (after the post processing of bullet 6), **MUST** insert a **Record-Route** header field
2673 that is not a SIPS URI.

2674 A proxy at a security perimeter must remain on the perimeter throughout the dialog.

2675 If the URI placed in the **Record-Route** header field needs to be rewritten when it passes back through
2676 in a response, the URI **MUST** be distinct enough to locate at that time. (The request may spiral through
2677 this proxy, resulting in more than one **Record-Route** header field value being added). Item 8 of
2678 Section 16.7 recommends a mechanism to make the URI sufficiently distinct.

2679 The proxy **MAY** include parameters in the **Record-Route** header field value. These will be echoed in
2680 some responses to the request such as the 200 (OK) responses to INVITE. Such parameters may be
2681 useful for keeping state in the message rather than the proxy.

2682 If a proxy needs to be in the path of any type of dialog (such as one straddling a firewall), it **SHOULD**
2683 add a **Record-Route** header field value to every request with a method it does not understand since
2684 that method may have dialog semantics.

2685 The URI a proxy places into a **Record-Route** header field is only valid for the lifetime of any dialog
2686 created by the transaction in which it occurs. A dialog-stateful proxy, for example, **MAY** refuse to
2687 accept future requests with that value in the **Request-URI** after the dialog has terminated. Non-
2688 dialog-stateful proxies, of course, have no concept of when the dialog has terminated, but they **MAY**
2689 encode enough information in the value to compare it against the dialog identifier of future requests
2690 and **MAY** reject requests not matching that information. Endpoints **MUST NOT** use a URI obtained
2691 from a **Record-Route** header field outside the dialog in which it was provided. See Section 12 for
2692 more information on an endpoint's use of **Record-Route** header fields.

2693 Record-routing may be required by certain services where the proxy needs to observe all messages
2694 in a dialog. However, it slows down processing and impairs scalability and thus proxies should only
2695 record-route if required for a particular service.

2696 The **Record-Route** process is designed to work for any SIP request that initiates a dialog. INVITE is
2697 the only such request in this specification, but extensions to the protocol **MAY** define others.

2698 5. Add Additional Header Fields

2699 The proxy **MAY** add any other appropriate header fields to the copy at this point.

2700 6. Postprocess routing information

2701 A proxy **MAY** have a local policy that mandates that a request visit a specific set of proxies before being
2702 delivered to the destination. A proxy **MUST** ensure that all such proxies are loose routers. Generally,
2703 this can only be known with certainty if the proxies are within the same administrative domain. This
2704 set of proxies is represented by a set of URIs (each of which contains the **lr** parameter). This set **MUST**

2705 be pushed into the **Route** header field of the copy ahead of any existing values, if present. If the
2706 **Route** header field is absent, it **MUST** be added, containing that list of URIs.

2707 If the proxy has a local policy that mandates that the request visit one specific proxy, an alternative to
2708 pushing a **Route** value into the **Route** header field is to bypass the forwarding logic of item 10 below,
2709 and instead just send the request to the address, port, and transport for that specific proxy. If the
2710 request has a **Route** header field, this alternative **MUST NOT** be used unless it is known that next hop
2711 proxy is a loose router. Otherwise, this approach **MAY** be used, but the **Route** insertion mechanism
2712 above is preferred for its robustness, flexibility, generality and consistency of operation. Furthermore,
2713 if the **Request-URI** contains a SIPS URI, TLS **MUST** be used to communicate with that proxy.

2714 If the copy contains a **Route** header field, the proxy **MUST** inspect the URI in its first value. If that
2715 URI does not contain a **lr** parameter, the proxy **MUST** modify the copy as follows:

- 2716 • The proxy **MUST** place the **Request-URI** into the **Route** header field as the last value.
- 2717 • The proxy **MUST** then place the first **Route** header field value into the **Request-URI** and remove
2718 that value from the **Route** header field.

2719 Appending the **Request-URI** to the **Route** header field is part of a mechanism used to pass the information
2720 in that **Request-URI** through strict-routing elements. "Popping" the first **Route** header field value into the
2721 **Request-URI** formats the message the way a strict-routing element expects to receive it (with its own URI in
2722 the **Request-URI** and the next location to visit in the first **Route** header field value).

2723 7. Determine Next-Hop Address, Port, and Transport

2724 The proxy **MAY** have a local policy to send the request to a specific IP address, port, and transport,
2725 independent of the values of the **Route** and **Request-URI**. Such a policy **MUST NOT** be used if the
2726 proxy is not certain that the IP address, port, and transport correspond to a server that is a loose router.
2727 However, this mechanism for sending the request through a specific next hop is **NOT RECOMMENDED**;
2728 instead a **Route** header field should be used for that purpose as described above.

2729 In the absence of such an overriding mechanism, the proxy applies the procedures listed in [4] as
2730 follows to determine where to send the request. If the proxy has reformatted the request to send to
2731 a strict-routing element as described in step 6 above, the proxy **MUST** apply those procedures to the
2732 **Request-URI** of the request. Otherwise, the proxy **MUST** apply the procedures to the first value in the
2733 **Route** header field, if present, else the **Request-URI**. The procedures will produce an ordered set of
2734 (address, port, transport) tuples. Independently of which URI is being used as input to the procedures
2735 of [4], if the **Request-URI** specifies a SIPS resource, the proxy **MUST** follow the procedures of [4] as
2736 if the input URI were a SIPS URI.

2737 As described in [4], the proxy **MUST** attempt to deliver the message to the first tuple in that set, and
2738 proceed through the set in order until the delivery attempt succeeds.

2739 For each tuple attempted, the proxy **MUST** format the message as appropriate for the tuple and send
2740 the request using a new client transaction as detailed in steps 8 through 10. Since each attempt uses a
2741 new client transaction, it represents a new branch. Thus, the branch parameter provided with the **Via**
2742 header field inserted in step 8 **MUST** be different for each attempt.

2743 If the client transaction reports failure to send the request or a timeout from its state machine, the
2744 proxy continues to the next address in that ordered set. If the ordered set is exhausted, the request
2745 cannot be forwarded to this element in the target set. The proxy does not need to place anything in

2746 the response context, but otherwise acts as if this element of the target set returned a 408 (Request
2747 Timeout) final response.

2748 8. Add a Via header field value

2749 The proxy **MUST** insert a **Via** header field value into the copy before the existing **Via** header field
2750 values. The construction of this value follows the same guidelines of Section 8.1.1.7. This implies
2751 that the proxy will compute its own branch parameter, which will be globally unique for that branch,
2752 and contain the requisite magic cookie.

2753 Proxies choosing to detect loops have an additional constraint in the value they use for construction of
2754 the branch parameter. A proxy choosing to detect loops **SHOULD** create a branch parameter separable
2755 into two parts by the implementation. The first part **MUST** satisfy the constraints of Section 8.1.1.7 as
2756 described above. The second is used to perform loop detection and distinguish loops from spirals.

2757 Loop detection is performed by verifying that, when a request returns to a proxy, those fields hav-
2758 ing an impact on the processing of the request have not changed. The value placed in this part of
2759 the branch parameter **SHOULD** reflect all of those fields (including any **Route**, **Proxy-Require** and
2760 **Proxy-Authorization** header fields). This is to ensure that if the request is routed back to the proxy
2761 and one of those fields changes, it is treated as a spiral and not a loop (Section 16.3 A common
2762 way to create this value is to compute a cryptographic hash of the **To** tag, **From** tag, **Call-ID** header
2763 field, the **Request-URI** of the request received (before translation) and the sequence number from
2764 the **CSeq** header field, in addition to any **Proxy-Require** and **Proxy-Authorization** header fields
2765 that may be present. The algorithm used to compute the hash is implementation-dependent, but MD5
2766 (RFC 1321 [35]), expressed in hexadecimal, is a reasonable choice. (Base64 is not permissible for a
2767 token.)

2768 If a proxy wishes to detect loops, the “branch” parameter it supplies **MUST** depend on all information
2769 affecting processing of a request, including the incoming **Request-URI** and any header fields affecting the
2770 request’s admission or routing. This is necessary to distinguish looped requests from requests whose routing
2771 parameters have changed before returning to this server.

2772 The request method **MUST NOT** be included in the calculation of the branch parameter. In particular,
2773 **CANCEL** and **ACK** requests (for non-2xx responses) **MUST** have the same branch value as the cor-
2774 responding request they cancel or acknowledge. The branch parameter is used in correlating those
2775 requests at the server handling them (see Sections 17.2.3 and 9.2).

2776 9. Add a Content-Length header field if necessary

2777 If the request will be sent to the next hop using a stream-based transport and the copy contains no
2778 **Content-Length** header field, the proxy **MUST** insert one with the correct value for the body of the
2779 request (see Section 20.14).

2780 10. Forward Request

2781 A stateful proxy **MUST** create a new client transaction for this request as described in Section 17.1 and
2782 instructs the transaction to send the request using the address, port and transport determined in step 7.

2783 11. Set timer C

2784 In order to handle the case where an **INVITE** request never generates a final response, the TU uses
2785 a timer which is called timer C. Timer C **MUST** be set for each client transaction when an **INVITE**

2786 request is proxied. The timer **MUST** be larger than 3 minutes. Section 16.7 bullet 2 discusses how this
2787 timer is updated with provisional responses, and Section 16.8 discusses processing when it fires.

2788 **16.7 Response Processing**

2789 When a response is received by an element, it first tries to locate a client transaction (Section 17.1.3) match-
2790 ing the response. If none is found, the element **MUST** process the response (even if it is an informational
2791 response) as a stateless proxy (described below). If a match is found, the response is handed to the client
2792 transaction.

2793 Forwarding responses for which a client transaction (or more generally any knowledge of having sent an associ-
2794 ated request) is not found improves robustness. In particular, it ensures that “late” 2xx responses to INVITE requests
2795 are forwarded properly.

2796 As client transactions pass responses to the proxy layer, the following processing **MUST** take place:

- 2797 1. Find the appropriate response context
- 2798 2. Update timer C for provisional responses
- 2799 3. Remove the topmost Via
- 2800 4. Add the response to the response context
- 2801 5. Check to see if this response should be forwarded immediately
- 2802 6. When necessary, choose the best final response from the response context

2803 If no final response has been forwarded after every client transaction associated with the response
2804 context has been terminated, the proxy must choose and forward the “best” response from those it has
2805 seen so far.

2806 The following processing **MUST** be performed on each response that is forwarded. It is likely that
2807 more than one response to each request will be forwarded: at least each provisional and one final
2808 response.

- 2809 7. Aggregate authorization header field values if necessary
- 2810 8. Optionally rewrite Record-Route header field values
- 2811 9. Forward the response
- 2812 10. Generate any necessary **CANCEL** requests

2813 Each of the above steps are detailed below:

- 2814 1. Find Context

2815 The proxy locates the “response context” it created before forwarding the original request using the
2816 key described in Section 16.6. The remaining processing steps take place in this context.

2817 2. Update timer C for provisional responses

2818 For an INVITE transaction, if the response is a provisional response with status codes 101 to 199
2819 inclusive (i.e., anything but 100), the proxy MUST reset timer C for that client transaction. The timer
2820 MAY be reset to a different value, but this value MUST be greater than 3 minutes.

2821 3. Via

2822 The proxy removes the topmost Via header field value from the response.

2823 If no Via header field values remain in the response, the response was meant for this element and
2824 MUST NOT be forwarded. The remainder of the processing described in this section is not performed
2825 on this message, the UAC processing rules described in Section 8.1.3 are followed instead (transport
2826 layer processing has already occurred).

2827 This will happen, for instance, when the element generates CANCEL requests as described in Sec-
2828 tion 10.

2829 4. Add response to context

2830 Final responses received are stored in the response context until a final response is generated on the
2831 server transaction associated with this context. The response may be a candidate for the best final
2832 response to be returned on that server transaction. Information from this response may be needed in
2833 forming the best response even if this response is not chosen.

2834 If the proxy chooses to recurse on any contacts in a 3xx response by adding them to the target set, it
2835 MUST remove them from the response before adding the response to the response context. However,
2836 a proxy SHOULD NOT recurse to a non-SIPS URI if the Request-URI of the original request was a
2837 SIPS URI. If the proxy recurses on all of the contacts in a 3xx response, the proxy SHOULD NOT add
2838 the resulting contactless response to the response context.

2839 Removing the contact before adding the response to the response context prevents the next element up-
2840 stream from retrying a location this proxy has already attempted.

2841 3xx responses may contain a mixture of SIP, SIPS, and non-SIP URIs. A proxy may choose to recurse on
2842 the SIP and SIPS URIs and place the remainder into the response context to be returned potentially in the final
2843 response.

2844 If a proxy receives a 416 (Unsupported URI Scheme) response to a request whose Request-URI
2845 scheme was not SIP, but the scheme in the original received request was SIP or SIPS (that is, the
2846 proxy changed the scheme from SIP or SIPS to something else when it proxied a request), the proxy
2847 SHOULD add a new URI to the target set. This URI SHOULD be a SIP URI version of the non-SIP URI
2848 that was just tried. In the case of the tel URL, this is accomplished by placing the telephone-subscriber
2849 part of the tel URL into the user part of the SIP URI, and setting the hostpart to the domain where the
2850 prior request was sent. See Section 19.1.6 for more detail on forming SIP URIs from tel URLs.

2851 As with a 3xx response, if a proxy “recurses” on the 416 by trying a SIP or SIPS URI instead, the 416
2852 response SHOULD NOT be added to the response context.

2853 5. Check response for forwarding

2854 Until a final response has been sent on the server transaction, the following responses MUST be for-
2855 forwarded immediately:

- 2856 • Any provisional response other than 100 (Trying)
- 2857 • Any 2xx response

2858 If a 6xx response is received, it is not immediately forwarded, but the stateful proxy SHOULD cancel
2859 all client pending transactions as described in Section 10, and it MUST NOT create any new branches
2860 in this context.

2861 This is a change from RFC 2543, which mandated that the proxy was to forward the 6xx response imme-
2862 diately. For an INVITE transaction, this approach had the problem that a 2xx response could arrive on another
2863 branch, in which case the proxy would have to forward the 2xx. The result was that the UAC could receive
2864 a 6xx response followed by a 2xx response, which should never be allowed to happen. Under the new rules,
2865 upon receiving a 6xx, a proxy will issue a CANCEL request, which will generally result in 487 responses from
2866 all outstanding client transactions, and then at that point the 6xx is forwarded upstream.

2867 After a final response has been sent on the server transaction, the following responses MUST be for-
2868 warded immediately:

- 2869 • Any 2xx response to an INVITE request

2870 A stateful proxy MUST NOT immediately forward any other responses. In particular, a stateful proxy
2871 MUST NOT forward any 100 (Trying) response. Those responses that are candidates for forwarding
2872 later as the “best” response have been gathered as described in step “Add Response to Context”.

2873 Any response chosen for immediate forwarding MUST be processed as described in steps “Aggregate
2874 Authorization Header Field Values” through “Record-Route”.

2875 This step, combined with the next, ensures that a stateful proxy will forward exactly one final response
2876 to a non-INVITE request, and either exactly one non-2xx response or one or more 2xx responses to
2877 an INVITE request.

2878 6. Choosing the best response

2879 A stateful proxy MUST send a final response to a response context’s server transaction if no final
2880 responses have been immediately forwarded by the above rules and all client transactions in this
2881 response context have been terminated.

2882 The stateful proxy MUST choose the “best” final response among those received and stored in the
2883 response context.

2884 If there are no final responses in the context, the proxy MUST send a 408 (Request Timeout) response
2885 to the server transaction.

2886 Otherwise, the proxy MUST forward a response from the responses stored in the response context.
2887 It MUST choose from the 6xx class responses if any exist in the context. If no 6xx class responses
2888 are present, the proxy SHOULD choose from the lowest response class stored in the response context.
2889 The proxy MAY select any response within that chosen class. The proxy SHOULD give preference to
2890 responses that provide information affecting resubmission of this request, such as 401, 407, 415, 420,
2891 and 484 if the 4xx class is chosen.

2892 A proxy which receives a 503 (Service Unavailable) response SHOULD NOT forward it upstream
2893 unless it can determine that any subsequent requests it might proxy will also generate a 503. In other
2894 words, forwarding a 503 means that the proxy knows it cannot service any requests, not just the one

2895 for the Request-URI in the request which generated the 503. If the only response that was received
2896 is a 503, the proxy SHOULD generate a 500 response and forward that upstream.

2897 The forwarded response MUST be processed as described in steps "Aggregate Authorization Header
2898 Field Values" through "Record-Route".

2899 For example, if a proxy forwarded a request to 4 locations, and received 503, 407, 501, and 404
2900 responses, it may choose to forward the 407 (Proxy Authentication Required) response.

2901 1xx and 2xx responses may be involved in the establishment of dialogs. When a request does not
2902 contain a To tag, the To tag in the response is used by the UAC to distinguish multiple responses to
2903 a dialog creating request. A proxy MUST NOT insert a tag into the To header field of a 1xx or 2xx
2904 response if the request did not contain one. A proxy MUST NOT modify the tag in the To header field
2905 of a 1xx or 2xx response.

2906 Since a proxy may not insert a tag into the To header field of a 1xx response to a request that did not
2907 contain one, it cannot issue non-100 provisional responses on its own. However, it can branch the
2908 request to a UAS sharing the same element as the proxy. This UAS can return its own provisional
2909 responses, entering into an early dialog with the initiator of the request. The UAS does not have to be
2910 a discreet process from the proxy. It could be a virtual UAS implemented in the same code space as
2911 the proxy.

2912 3-6xx responses are delivered hop-hop. When issuing a 3-6xx response, the element is effectively
2913 acting as a UAS, issuing its own response, usually based on the responses received from downstream
2914 elements. An element SHOULD preserve the To tag when simply forwarding a 3-6xx response to a
2915 request that did not contain a To tag.

2916 A proxy MUST NOT modify the To tag in any forwarded response to a request that contains a To tag.

2917 While it makes no difference to the upstream elements if the proxy replaced the To tag in a forwarded
2918 3-6xx response, preserving the original tag may assist with debugging.

2919 When the proxy is aggregating information from several responses, choosing a To tag from among them
2920 is arbitrary, and generating a new To tag may make debugging easier. This happens, for instance, when
2921 combining 401 (Unauthorized) and 407 (Proxy Authentication Required) challenges, or combining Contact
2922 values from unencrypted and unauthenticated 3xx responses.

2923 7. Aggregate Authorization Header Field Values

2924 If the selected response is a 401 (Unauthorized) or 407 (Proxy Authentication Required), the proxy
2925 MUST collect any WWW-Authenticate and Proxy-Authenticate header field values from all other
2926 401 (Unauthorized) and 407 (Proxy Authentication Required) responses received so far in this re-
2927 sponse context and add them to this response without modification before forwarding. The resulting
2928 401 (Unauthorized) or 407 (Proxy Authentication Required) response could have several WWW-
2929 Authenticate AND Proxy-Authenticate header field values.

2930 This is necessary because any or all of the destinations the request was forwarded to may have re-
2931 quested credentials. The client needs to receive all of those challenges and supply credentials for each
2932 of them when it retries the request. Motivation for this behavior is provided in Section 26.

2933 8. Record-Route

2934 If the selected response contains a Record-Route header field value originally provided by this proxy,
2935 the proxy MAY choose to rewrite the value before forwarding the response. This allows the proxy to

2936 provide different URIs for itself to the next upstream and downstream elements. A proxy may choose
2937 to use this mechanism for any reason. For instance, it is useful for multi-homed hosts.

2938 If the proxy received the request over TLS, and sent it out over a non-TLS connection, the proxy
2939 MUST rewrite the URI in the **Record-Route** header field to be a SIPS URI. If the proxy received the
2940 request over a non-TLS connection, and sent it out over TLS, the proxy MUST rewrite the URI in the
2941 **Record-Route** header field to be a SIP URI.

2942 The new URI provided by the proxy MUST satisfy the same constraints on URIs placed in **Record-**
2943 **Route** header fields in requests (see Step 4 of Section 16.6) with the following modifications:

2944 The URI SHOULD NOT contain the transport parameter unless the proxy has knowledge that the next
2945 upstream (as opposed to downstream) element that will be in the path of subsequent requests supports
2946 that transport.

2947 When a proxy does decide to modify the **Record-Route** header field in the response, one of the
2948 operations it performs is locating the **Record-Route** value that it had inserted. If the request spiraled,
2949 and the proxy inserted a **Record-Route** value in each iteration of the spiral, locating the correct value
2950 in the response (which must be the proper iteration in the reverse direction) is tricky. The rules above
2951 recommend that a proxy wishing to rewrite **Record-Route** header field values insert sufficiently
2952 distinct URIs into the **Record-Route** header field so that the right one may be selected for rewriting.
2953 A RECOMMENDED mechanism to achieve this is for the proxy to append a unique identifier for the
2954 proxy instance to the user portion of the URI.

2955 When the response arrives, the proxy modifies the first **Record-Route** whose identifier matches the
2956 proxy instance. The modification results in a URI without this piece of data appended to the user
2957 portion of the URI. Upon the next iteration, the same algorithm (find the topmost **Record-Route**
2958 header field value with the parameter) will correctly extract the next **Record-Route** header field
2959 value inserted by that proxy.

2960 Not every response to a request to which a proxy adds a **Record-Route** header field value will contain
2961 a **Record-Route** header field. If the response does contain a **Record-Route** header field, it will contain the
2962 value the proxy added.

2963 9. Forward response

2964 After performing the processing described in steps “Aggregate Authorization Header Field Values”
2965 through “Record-Route”, the proxy MAY perform any feature specific manipulations on the selected
2966 response. The proxy MUST NOT add to, modify, or remove the message body. Unless otherwise
2967 specified, the proxy MUST NOT remove any header field values other than the **Via** header field value
2968 discussed in Section 16.7 Item 3. In particular, the proxy MUST NOT remove any “received” parameter
2969 it may have added to the next **Via** header field value while processing the request associated with this
2970 response. The proxy MUST pass the response to the server transaction associated with the response
2971 context. This will result in the response being sent to the location now indicated in the topmost
2972 **Via** header field value. If the server transaction is no longer available to handle the transmission,
2973 the element MUST forward the response statelessly by sending it to the server transport. The server
2974 transaction might indicate failure to send the response or signal a timeout in its state machine. These
2975 errors would be logged for diagnostic purposes as appropriate, but the protocol requires no remedial
2976 action from the proxy.

2977 The proxy MUST maintain the response context until all of its associated transactions have been ter-
2978 minated, even after forwarding a final response.

2979 10. Generate CANCELs

2980 If the forwarded response was a final response, the proxy MUST generate a CANCEL request for all
2981 pending client transactions associated with this response context. A proxy SHOULD also generate a
2982 CANCEL request for all pending client transactions associated with this response context when it
2983 receives a 6xx response. A pending client transaction is one that has received a provisional response,
2984 but no final response (it is in the proceeding state) and has not had an associated CANCEL generated
2985 for it. Generating CANCEL requests is described in Section 9.1.

2986 The requirement to CANCEL pending client transactions upon forwarding a final response does not
2987 guarantee that an endpoint will not receive multiple 200 (OK) responses to an INVITE. 200 (OK)
2988 responses on more than one branch may be generated before the CANCEL requests can be sent and
2989 processed. Further, it is reasonable to expect that a future extension may override this requirement to
2990 issue CANCEL requests.

2991 16.8 Processing Timer C

2992 If timer C should fire, the proxy MUST either reset the timer with any value it chooses, or terminate the
2993 client transaction. If the client transaction has received a provisional response, the proxy MUST generate a
2994 CANCEL request matching that transaction. If the client transaction has not received a provisional response,
2995 the proxy MUST behave as if the transaction received a 408 (Request Timeout) response.

2996 Allowing the proxy to reset the timer allows the proxy to dynamically extend the transaction's lifetime
2997 based on current conditions (such as utilization) when the timer fires.

2998 16.9 Handling Transport Errors

2999 If the transport layer notifies a proxy of an error when it tries to forward a request (see Section 18.4), the
3000 proxy MUST behave as if the forwarded request received a 400 (Bad Request) response.

3001 If the proxy is notified of an error when forwarding a response, it drops the response. The proxy SHOULD
3002 NOT cancel any outstanding client transactions associated with this response context due to this notification.

3003 If a proxy cancels its outstanding client transactions, a single malicious or misbehaving client can cause all
3004 transactions to fail through its Via header field.

3005 16.10 CANCEL Processing

3006 A stateful proxy MAY generate a CANCEL to any other request it has generated at any time (subject to re-
3007 ceiving a provisional response to that request as described in section 9.1). A proxy MUST cancel any pending
3008 client transactions associated with a response context when it receives a matching CANCEL request.

3009 A stateful proxy MAY generate CANCEL requests for pending INVITE client transactions based on the
3010 period specified in the INVITE's Expires header field elapsing. However, this is generally unnecessary
3011 since the endpoints involved will take care of signaling the end of the transaction.

3012 While a CANCEL request is handled in a stateful proxy by its own server transaction, a new response
3013 context is not created for it. Instead, the proxy layer searches its existing response contexts for the server
3014 transaction handling the request associated with this CANCEL. If a matching response context is found, the

3015 element **MUST** immediately return a 200 (OK) response to the **CANCEL** request. In this case, the element is
3016 acting as a user agent server as defined in Section 8.2. Furthermore, the element **MUST** generate **CANCEL**
3017 requests for all pending client transactions in the context as described in Section 16.7 step 10.

3018 If a response context is not found, the element does not have any knowledge of the request to apply
3019 the **CANCEL** to. It **MUST** statelessly forward the **CANCEL** request (it may have statelessly forwarded the
3020 associated request previously).

3021 **16.11 Stateless Proxy**

3022 When acting statelessly, a proxy is a simple message forwarder. Much of the processing performed when
3023 acting statelessly is the same as when behaving statefully. The differences are detailed here.

3024 A stateless proxy does not have any notion of a transaction, or of the response context used to describe
3025 stateful proxy behavior. Instead, the stateless proxy takes messages, both requests and responses, directly
3026 from the transport layer (See section 18). As a result, stateless proxies do not retransmit messages on their
3027 own. They do, however, forward all retransmission they receive (they do not have the ability to distinguish
3028 a retransmission from the original message). Furthermore, when handling a request statelessly, an element
3029 **MUST NOT** generate its own 100 (Trying) or any other provisional response.

3030 A stateless proxy **MUST** validate a request as described in Section 16.3

3031 A stateless proxy **MUST** follow the request processing steps described in Sections 16.4 through 16.5 with
3032 the following exception:

- 3033 • A stateless proxy **MUST** choose one and only one target from the target set. This choice **MUST** only
3034 rely on fields in the message and time-invariant properties of the server. In particular, a retransmitted
3035 request **MUST** be forwarded to the same destination each time it is processed. Furthermore, **CANCEL**
3036 and non-Routed **ACK** requests **MUST** generate the same choice as their associated **INVITE**.

3037 A stateless proxy **MUST** follow the request processing steps described in Section 16.6 with the following
3038 exceptions:

- 3039 • The requirement for unique branch IDs across space and time applies to stateless proxies as well.
3040 However, a stateless proxy cannot simply use a random number generator to compute the first com-
3041 ponent of the branch ID, as described in Section 16.6 bullet 8. This is because retransmissions of
3042 a request need to have the same value, and a stateless proxy cannot tell a retransmission from the
3043 original request. Therefore, the component of the branch parameter that makes it unique **MUST** be
3044 the same each time a retransmitted request is forwarded. Thus for a stateless proxy, the **branch** pa-
3045 rameter **MUST** be computed as a combinatoric function of message parameters which are invariant on
3046 retransmission.

3047 The stateless proxy **MAY** use any technique it likes to guarantee uniqueness of its branch IDs across
3048 transactions. However, the following procedure is **RECOMMENDED**. The proxy examines the branch
3049 ID in the topmost **Via** header field of the received request. If it begins with the magic cookie, the first
3050 component of the branch ID of the outgoing request is computed as a hash of the received branch ID.
3051 Otherwise, the first component of the branch ID is computed as a hash of the topmost **Via**, the tag in
3052 the **To** header field, the tag in the **From** header field, the **Call-ID** header field, the **CSeq** number (but
3053 not method), and the **Request-URI** from the received request. One of these fields will always vary
3054 across two different transactions.

- 3055 • All other message transformations specified in Section 16.6 MUST result in the same transformation
3056 of a retransmitted request. In particular, if the proxy inserts a **Record-Route** value or pushes URIs
3057 into the **Route** header field, it MUST place the same values in retransmissions of the request. As
3058 for the **Via** branch parameter, this implies that the transformations MUST be based on time-invariant
3059 configuration or retransmission-invariant properties of the request.
- 3060 • A stateless proxy determines where to forward the request as described for stateful proxies in Sec-
3061 tion 16.6 Item 10. The request is sent directly to the transport layer instead of through a client trans-
3062 action.

3063 Since a stateless proxy must forward retransmitted requests to the same destination and add identical branch
3064 parameters to each of them, it can only use information from the message itself and time-invariant configuration
3065 data for those calculations. If the configuration state is not time-invariant (for example, if a routing table is updated)
3066 any requests that could be affected by the change may not be forwarded statelessly during an interval equal to the
3067 transaction timeout window before or after the change. The method of processing the affected requests in that
3068 interval is an implementation decision. A common solution is to forward them transaction statefully.

3069 Stateless proxies MUST NOT perform special processing for **CANCEL** requests. They are processed by
3070 the above rules as any other requests. In particular, a stateless proxy applies the same **Route** header field
3071 processing to **CANCEL** requests that it applies to any other request.

3072 Response processing as described in Section 16.7 does not apply to a proxy behaving statelessly. When
3073 a response arrives at a stateless proxy, the proxy MUST inspect the sent-by value in the first (topmost) **Via**
3074 header field value. If that address matches the proxy (it equals a value this proxy has inserted into previous
3075 requests) the proxy MUST remove that header field value from the response and forward the result to the
3076 location indicated in the next **Via** header field value. The proxy MUST NOT add to, modify, or remove the
3077 message body. Unless specified otherwise, the proxy MUST NOT remove any other header field values. If
3078 the address does not match the proxy, the message MUST be silently discarded.

3079 **16.12 Summary of Proxy Route Processing**

3080 In the absence of local policy to the contrary, the processing a proxy performs on a request containing a
3081 **Route** header field can be summarized in the following steps.

- 3082 1. The proxy will inspect the **Request-URI**. If it indicates a resource owned by this proxy, the proxy
3083 will replace it with the results of running a location service. Otherwise, the proxy will not change the
3084 **Request-URI**.
- 3085 2. The proxy will inspect the **URI** in the topmost **Route** header field value. If it indicates this proxy, the
3086 proxy removes it from the **Route** header field (this route node has been reached).
- 3087 3. The proxy will forward the request to the resource indicated by the **URI** in the topmost **Route** header
3088 field value or in the **Request-URI** if no **Route** header field is present. The proxy determines the
3089 address, port and transport to use when forwarding the request by applying the procedures in [4] to
3090 that **URI**.

3091 If no strict-routing elements are encountered on the path of the request, the **Request-URI** will always
3092 indicate the target of the request.

3093 **16.12.1 Examples**

3094 **16.12.1.1 Basic SIP Trapezoid** This scenario is the basic SIP trapezoid, U1 -> P1 -> P2 -> U2, with
3095 both proxies record-routing. Here is the flow.

3096 U1 sends:

```
3097 INVITE sip:callee@domain.com SIP/2.0
3098 Contact: sip:caller@u1.example.com
```

3099 to P1. P1 is an outbound proxy. P1 is not responsible for domain.com, so it looks it up in DNS and
3100 sends it there. It also adds a **Record-Route** header field value:

```
3101 INVITE sip:callee@domain.com SIP/2.0
3102 Contact: sip:caller@u1.example.com
3103 Record-Route: <sip:p1.example.com;lr>
```

3104 P2 gets this. It is responsible for domain.com so it runs a location service and rewrites the **Request-**
3105 **URI**. It also adds a **Record-Route** header field value. There is no **Route** header field, so it resolves the new
3106 **Request-URI** to determine where to send the request:

```
3107 INVITE sip:callee@u2.domain.com SIP/2.0
3108 Contact: sip:caller@u1.example.com
3109 Record-Route: <sip:p2.domain.com;lr>
3110 Record-Route: <sip:p1.example.com;lr>
```

3111 The callee at u2.domain.com gets this and responds with a 200 OK:

```
3112 SIP/2.0 200 OK
3113 Contact: sip:callee@u2.domain.com
3114 Record-Route: <sip:p2.domain.com;lr>
3115 Record-Route: <sip:p1.example.com;lr>
```

3116 The callee at u2 also sets its dialog state's remote target URI to sip:caller@u1.example.com and its route
3117 set to

3118 (<sip:p2.domain.com;lr>, <sip:p1.example.com;lr>)

3119 This is forwarded by P2 to P1 to U1 as normal. Now, U1 sets its dialog state's remote target URI to
3120 sip:callee@u2.domain.com and its route set to

3121 (<sip:p1.example.com;lr>, <sip:p2.domain.com;lr>)

3122 Since all the route set elements contain the lr parameter, U1 constructs the following **BYE** request:

```
3123 BYE sip:callee@u2.domain.com SIP/2.0
```

3124 Route: <sip:p1.example.com;lr>, <sip:p2.domain.com;lr>

3125 As any other element (including proxies) would do, it resolves the URI in the topmost **Route** header
3126 field value using DNS to determine where to send the request. This goes to P1. P1 notices that it is not
3127 responsible for the resource indicated in the **Request-URI** so it doesn't change it. It does see that it is the
3128 first value in the **Route** header field, so it removes that value, and forwards the request to P2:

3129 BYE sip:callee@u2.domain.com SIP/2.0
3130 Route: <sip:p2.domain.com;lr>

3131 P2 also notices it is not responsible for the resource indicated by the **Request-URI** (it is responsible for
3132 domain.com, not u2.domain.com), so it doesn't change it. It does see itself in the first **Route** header field
3133 value, so it removes it and forwards the following to u2.domain.com based on a DNS lookup against the
3134 **Request-URI**:

3135 BYE sip:callee@u2.domain.com SIP/2.0

3136 **16.12.1.2 Traversing a strict-routing proxy** In this scenario, a dialog is established across four prox-
3137 ies, each of which adds **Record-Route** header field values. The third proxy implements the strict-routing
3138 procedures specified in RFC 2543 and the bis drafts up to bis-05.

3139 U1->P1->P2->P3->P4->U2

3140 The INVITE arriving at U2 contains

3141 INVITE sip:callee@u2.domain.com SIP/2.0
3142 Contact: sip:caller@u1.example.com
3143 Record-Route: <sip:p4.domain.com;lr>
3144 Record-Route: <sip:p3.middle.com>
3145 Record-Route: <sip:p2.example.com;lr>
3146 Record-Route: <sip:p1.example.com;lr>

3147 Which U2 responds to with a 200 OK. Later, U2 sends the following **BYE** request to P4 based on the
3148 first **Route** header field value.

3149 BYE sip:caller@u1.example.com SIP/2.0
3150 Route: <sip:p4.domain.com;lr>
3151 Route: <sip:p3.middle.com>
3152 Route: <sip:p2.example.com;lr>
3153 Route: <sip:p1.example.com;lr>

3154 P4 is not responsible for the resource indicated in the **Request-URI** so it will leave it alone. It notices
3155 that it is the element in the first **Route** header field value so it removes it. It then prepares to send the request
3156 based on the now first **Route** header field value of sip:p3.middle.com, but it notices that this URI does not
3157 contain the **lr** parameter, so before sending, it reformats the request to be:

3158 BYE sip:p3.middle.com SIP/2.0
3159 Route: <sip:p2.example.com;lr>
3160 Route: <sip:p1.example.com;lr>
3161 Route: <sip:caller@u1.example.com>

3162 P3 is a strict router, so it forwards the following to P2:

3163 BYE sip:p2.example.com;lr SIP/2.0
3164 Route: <sip:p1.example.com;lr>
3165 Route: <sip:caller@u1.example.com>

3166 P2 sees the request-URI is a value it placed into a Record-Route header field, so before further processing, it rewrites the request to be

3168 BYE sip:caller@u1.example.com SIP/2.0
3169 Route: <sip:p1.example.com;lr>

3170 P2 is not responsible for u1.example.com so it sends the request to P1 based on the resolution of the Route header field value.

3172 P1 notices itself in the topmost Route header field value, so it removes it, resulting in:

3173 BYE sip:caller@u1.example.com SIP/2.0

3174 Since P1 is not responsible for u1.example.com and there is no Route header field, P1 will forward the request to u1.example.com based on the Request-URI.

3176 **16.12.1.3 Rewriting Record-Route header field values** In this scenario, U1 and U2 are in different private namespaces and they enter a dialog through a proxy P1, which acts as a gateway between the namespaces.

3179 U1->P1->U2

3180 U1 sends:

3181 INVITE sip:callee@gateway.leftprivatespace.com SIP/2.0
3182 Contact: <sip:caller@u1.leftprivatespace.com>

3183 P1 uses its location service and sends the following to U2:

3184 INVITE sip:callee@rightprivatespace.com SIP/2.0
3185 Contact: <sip:caller@u1.leftprivatespace.com>
3186 Record-Route: <sip:gateway.rightprivatespace.com;lr>

3187 U2 sends this 200 (OK) back to P1:

3188 SIP/2.0 200 OK
3189 Contact: <sip:callee@u2.rightprivatespace.com>
3190 Record-Route: <sip:gateway.rightprivatespace.com;lr>

3191 P1 rewrites its **Record-Route** header parameter to provide a value that U1 will find useful, and sends
3192 the following to U1:

3193 SIP/2.0 200 OK
3194 Contact: <sip:callee@u2.rightprivatespace.com>
3195 Record-Route: <sip:gateway.leftprivatespace.com;lr>

3196 Later, U1 sends the following **BYE** request to P1:

3197 BYE sip:callee@u2.rightprivatespace.com SIP/2.0
3198 Route: <sip:gateway.leftprivatespace.com;lr>

3199 which P1 forwards to U2 as

3200 BYE sip:callee@u2.rightprivatespace.com SIP/2.0

3201 17 Transactions

3202 SIP is a transactional protocol: interactions between components take place in a series of independent
3203 message exchanges. Specifically, a SIP transaction consists of a single request and any responses to that
3204 request, which include zero or more provisional responses and one or more final responses. In the case
3205 of a transaction where the request was an **INVITE** (known as an **INVITE** transaction), the transaction also
3206 includes the **ACK** only if the final response was not a 2xx response. If the response was a 2xx, the **ACK** is
3207 not considered part of the transaction.

3208 The reason for this separation is rooted in the importance of delivering all 200 (OK) responses to an **INVITE**
3209 to the UAC. To deliver them all to the UAC, the UAS alone takes responsibility for retransmitting them (see Sec-
3210 tion 13.3.1.4), and the UAC alone takes responsibility for acknowledging them with **ACK** (see Section 13.2.2.4).
3211 Since this **ACK** is retransmitted only by the UAC, it is effectively considered its own transaction.

3212 Transactions have a client side and a server side. The client side is known as a client transaction and the
3213 server side as a server transaction. The client transaction sends the request, and the server transaction sends
3214 the response. The client and server transactions are logical functions that are embedded in any number of
3215 elements. Specifically, they exist within user agents and stateful proxy servers. Consider the example in
3216 Section 4. In this example, the UAC executes the client transaction, and its outbound proxy executes the
3217 server transaction. The outbound proxy also executes a client transaction, which sends the request to a
3218 server transaction in the inbound proxy. That proxy also executes a client transaction, which in turn sends
3219 the request to a server transaction in the UAS. This is shown in Figure 4.

3220 A stateless proxy does not contain a client or server transaction. The transaction exists between the UA
3221 or stateful proxy on one side, and the UA or stateful proxy on the other side. As far as SIP transactions are
3222 concerned, stateless proxies are effectively transparent. The purpose of the client transaction is to receive
3223 a request from the element in which the client is embedded (call this element the "Transaction User" or
3224 TU; it can be a UA or a stateful proxy), and reliably deliver the request to a server transaction. The client

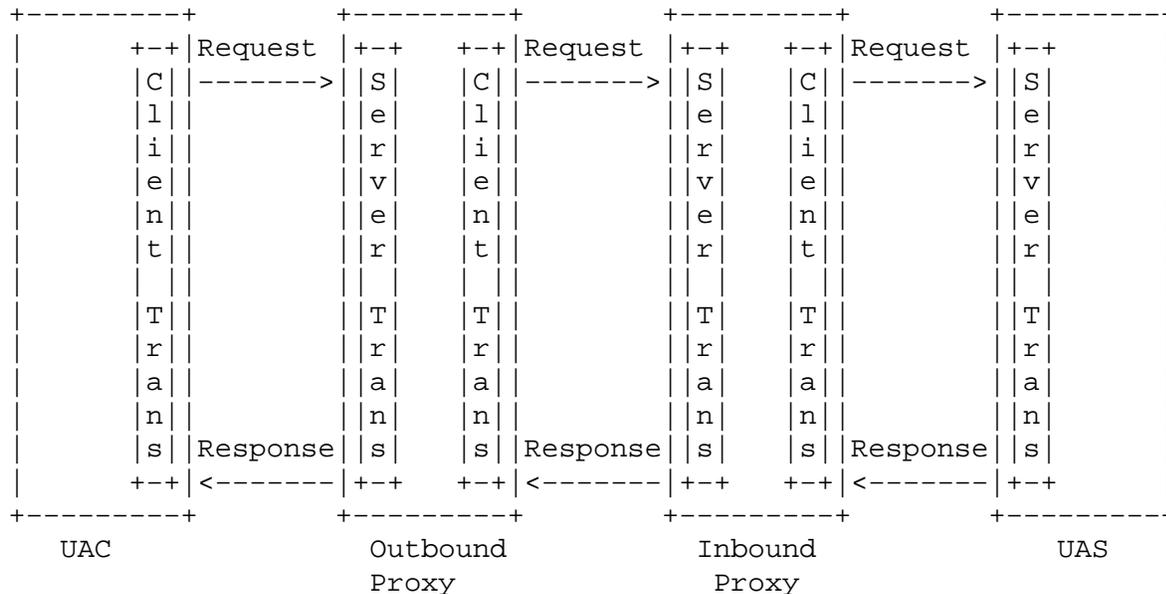


Figure 4: Transaction relationships

3225 transaction is also responsible for receiving responses and delivering them to the TU, filtering out any re-
 3226 sponse retransmissions or disallowed responses (such as a response to ACK). Additionally, in the case of an
 3227 INVITE request, the client transaction is responsible for generating the ACK request for any final response
 3228 excepting a 2xx response.

3229 Similarly, the purpose of the server transaction is to receive requests from the transport layer and deliver
 3230 them to the TU. The server transaction filters any request retransmissions from the network. The server
 3231 transaction accepts responses from the TU and delivers them to the transport layer for transmission over the
 3232 network. In the case of an INVITE transaction, it absorbs the ACK request for any final response excepting
 3233 a 2xx response.

3234 The 2xx response and its ACK receive special treatment. This response is retransmitted only by a UAS,
 3235 and its ACK generated only by the UAC. This end-to-end treatment is needed so that a caller knows the
 3236 entire set of users that have accepted the call. Because of this special handling, retransmissions of the 2xx
 3237 response are handled by the UA core, not the transaction layer. Similarly, generation of the ACK for the 2xx
 3238 is handled by the UA core. Each proxy along the path merely forwards each 2xx response to INVITE and
 3239 its corresponding ACK.

3240 17.1 Client Transaction

3241 The client transaction provides its functionality through the maintenance of a state machine.

3242 The TU communicates with the client transaction through a simple interface. When the TU wishes to
 3243 initiate a new transaction, it creates a client transaction and passes it the SIP request to send and an IP
 3244 address, port, and transport to which to send it. The client transaction begins execution of its state machine.
 3245 Valid responses are passed up to the TU from the client transaction.

3246 There are two types of client transaction state machines, depending on the method of the request passed

3247 by the TU. One handles client transactions for INVITE requests. This type of machine is referred to as
3248 an INVITE client transaction. Another type handles client transactions for all requests except INVITE and
3249 ACK. This is referred to as a non-INVITE client transaction. There is no client transaction for ACK. If the
3250 TU wishes to send an ACK, it passes one directly to the transport layer for transmission.

3251 The INVITE transaction is different from those of other methods because of its extended duration. Nor-
3252 mally, human input is required in order to respond to an INVITE. The long delays expected for sending a
3253 response argue for a three-way handshake. On the other hand, requests of other methods are expected to
3254 complete rapidly. Because of the non-INVITE transaction's reliance on a two-way handshake, TUs SHOULD
3255 respond immediately to non-INVITE requests.

3256 17.1.1 INVITE Client Transaction

3257 **17.1.1.1 Overview of INVITE Transaction** The INVITE transaction consists of a three-way handshake.
3258 The client transaction sends an INVITE, the server transaction sends responses, and the client transaction
3259 sends an ACK. For unreliable transports (such as UDP), the client transaction retransmits requests at an
3260 interval that starts at T1 seconds and doubles after every retransmission. T1 is an estimate of the round-
3261 trip time (RTT), and it defaults to 500 ms. Nearly all of the transaction timers described here scale with
3262 T1, and changing T1 adjusts their values. The request is not retransmitted over reliable transports. After
3263 receiving a 1xx response, any retransmissions cease altogether, and the client waits for further responses.
3264 The server transaction can send additional 1xx responses, which are not transmitted reliably by the server
3265 transaction. Eventually, the server transaction decides to send a final response. For unreliable transports,
3266 that response is retransmitted periodically, and for reliable transports, it is sent once. For each final response
3267 that is received at the client transaction, the client transaction sends an ACK, the purpose of which is to
3268 quench retransmissions of the response.

3269 **17.1.1.2 Formal Description** The state machine for the INVITE client transaction is shown in Figure 5.
3270 The initial state, "calling", MUST be entered when the TU initiates a new client transaction with an INVITE
3271 request. The client transaction MUST pass the request to the transport layer for transmission (see Section 18).
3272 If an unreliable transport is being used, the client transaction MUST start timer A with a value of T1. If a
3273 reliable transport is being used, the client transaction SHOULD NOT start timer A (Timer A controls request
3274 retransmissions). For any transport, the client transaction MUST start timer B with a value of 64*T1 seconds
3275 (Timer B controls transaction timeouts).

3276 When timer A fires, the client transaction MUST retransmit the request by passing it to the transport
3277 layer, and MUST reset the timer with a value of 2*T1. The formal definition of retransmit within the context
3278 of the transaction layer is to take the message previously sent to the transport layer and pass it to the transport
3279 layer once more.

3280 When timer A fires 2*T1 seconds later, the request MUST be retransmitted again (assuming the client
3281 transaction is still in this state). This process MUST continue so that the request is retransmitted with intervals
3282 that double after each transmission. These retransmissions SHOULD only be done while the client transaction
3283 is in the "calling" state.

3284 The default value for T1 is 500 ms. T1 is an estimate of the RTT between the client and server trans-
3285 actions. Elements MAY (though it is NOT RECOMMENDED) use smaller values of T1 within closed, private
3286 networks that do not permit general Internet connection. T1 MAY be chosen larger, and this is RECOM-
3287 MENDED if it is known in advance (such as on high latency access links) that the RTT is larger. Whatever
3288 the value of T1, the exponential backoffs on retransmissions described in this section MUST be used.

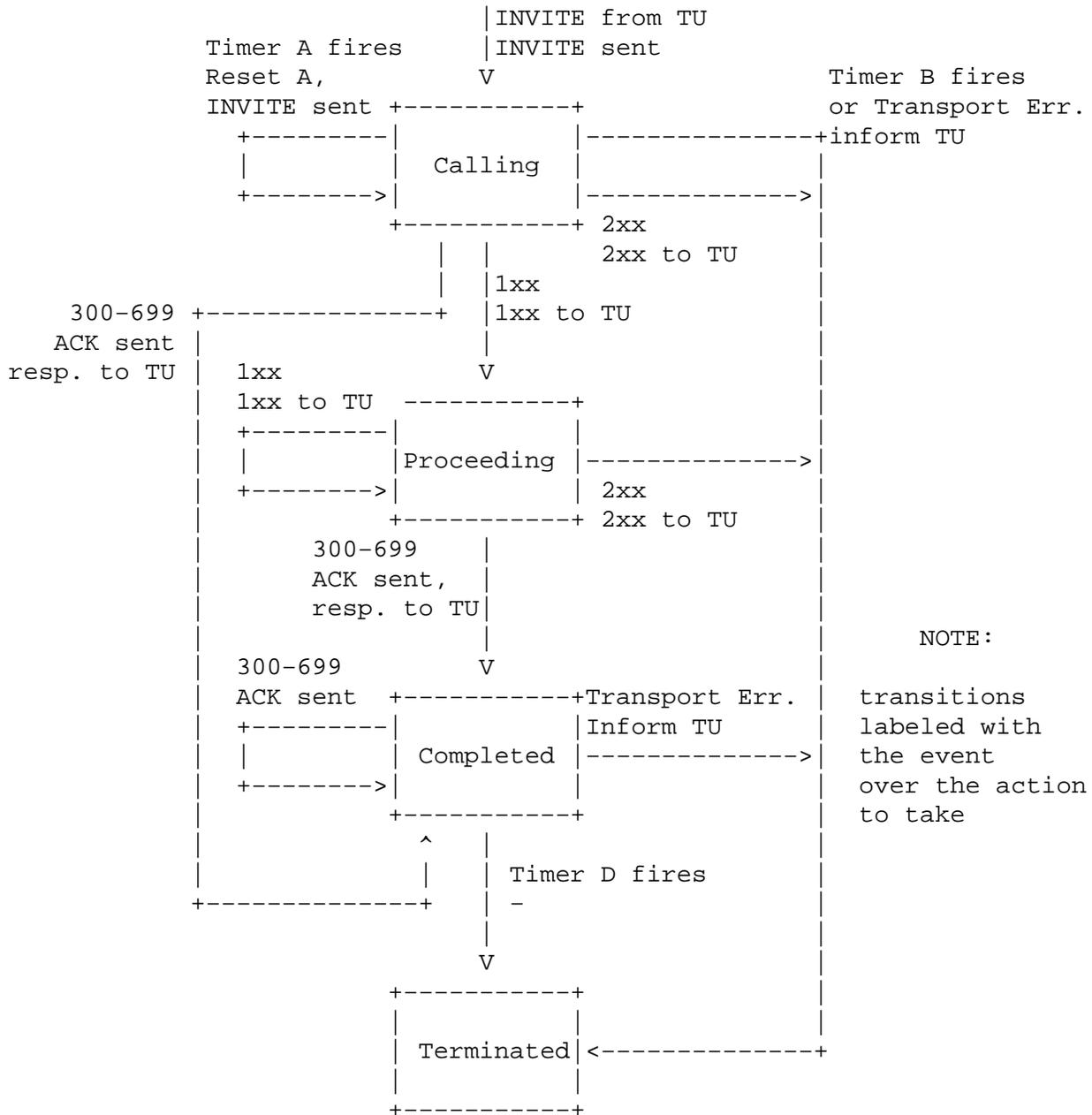


Figure 5: INVITE client transaction

3289 If the client transaction is still in the "calling" state when timer B fires, the client transaction SHOULD
 3290 inform the TU that a timeout has occurred. The client transaction MUST NOT generate an ACK. The value of
 3291 64*T1 is equal to the amount of time required to send seven requests in the case of an unreliable transport.

3292 If the client transaction receives a provisional response while in the "calling" state, it transitions to the
 3293 "proceeding" state. In the "proceeding" state, the client transaction SHOULD NOT retransmit the request any
 3294 longer. Furthermore, the provisional response MUST be passed to the TU. Any further provisional responses

3295 MUST be passed up to the TU while in the "proceeding" state.

3296 When in either the "Calling" or "Proceeding" states, reception of a response with status code from
3297 300-699 MUST cause the client transaction to transition to "Completed". The client transaction MUST pass
3298 the received response up to the TU, and the client transaction MUST generate an ACK request, even if the
3299 transport is reliable (guidelines for constructing the ACK from the response are given in Section 17.1.1.3)
3300 and then pass the ACK to the transport layer for transmission. The ACK MUST be sent to the same address,
3301 port, and transport to which the original request was sent. The client transaction SHOULD start timer D
3302 when it enters the "Completed" state, with a value of at least 32 seconds for unreliable transports, and a
3303 value of zero seconds for reliable transports. Timer D reflects the amount of time that the server transaction
3304 can remain in the "Completed" state when unreliable transports are used. This is equal to Timer H in the
3305 INVITE server transaction, whose default is 64*T1. However, the client transaction does not know the value
3306 of T1 in use by the server transaction, so an absolute minimum of 32s is used instead of basing Timer D on
3307 T1.

3308 Any retransmissions of the final response that are received while in the "Completed" state MUST cause
3309 the ACK to be re-passed to the transport layer for retransmission, but the newly received response MUST
3310 NOT be passed up to the TU. A retransmission of the response is defined as any response which would match
3311 the same client transaction based on the rules of Section 17.1.3.

3312 If timer D fires while the client transaction is in the "Completed" state, the client transaction MUST move
3313 to the terminated state, and it MUST inform the TU of the timeout.

3314 When in either the "Calling" or "Proceeding" states, reception of a 2xx response MUST cause the client
3315 transaction to enter the "Terminated" state, and the response MUST be passed up to the TU. The handling of
3316 this response depends on whether the TU is a proxy core or a UAC core. A UAC core will handle generation
3317 of the ACK for this response, while a proxy core will always forward the 200 (OK) upstream. The differing
3318 treatment of 200 (OK) between proxy and UAC is the reason that handling of it does not take place in the
3319 transaction layer.

3320 The client transaction MUST be destroyed the instant it enters the "Terminated" state. This is actually
3321 necessary to guarantee correct operation. The reason is that 2xx responses to an INVITE are treated differ-
3322 ently; each one is forwarded by proxies, and the ACK handling in a UAC is different. Thus, each 2xx needs
3323 to be passed to a proxy core (so that it can be forwarded) and to a UAC core (so it can be acknowledged). No
3324 transaction layer processing takes place. Whenever a response is received by the transport, if the transport
3325 layer finds no matching client transaction (using the rules of Section 17.1.3), the response is passed directly
3326 to the core. Since the matching client transaction is destroyed by the first 2xx, subsequent 2xx will find no
3327 match and therefore be passed to the core.

3328 **17.1.1.3 Construction of the ACK Request** This section specifies the construction of ACK requests
3329 sent within the client transaction. A UAC core that generates an ACK for 2xx MUST instead follow the rules
3330 described in Section 13.

3331 The ACK request constructed by the client transaction MUST contain values for the Call-ID, From, and
3332 Request-URI that are equal to the values of those header fields in the request passed to the transport by
3333 the client transaction (call this the "original request"). The To header field in the ACK MUST equal the To
3334 header field in the response being acknowledged, and therefore will usually differ from the To header field
3335 in the original request by the addition of the tag parameter. The ACK MUST contain a single Via header
3336 field, and this MUST be equal to the top Via header field of the original request. The CSeq header field in
3337 the ACK MUST contain the same value for the sequence number as was present in the original request, but
3338 the method parameter MUST be equal to "ACK".

3339 If the INVITE request whose response is being acknowledged had Route header fields, those header
3340 fields MUST appear in the ACK. This is to ensure that the ACK can be routed properly through any down-
3341 stream stateless proxies.

3342 Although any request MAY contain a body, a body in an ACK is special since the request cannot be
3343 rejected if the body is not understood. Therefore, placement of bodies in ACK for non-2xx is NOT RECOM-
3344 MENDED, but if done, the body types are restricted to any that appeared in the INVITE, assuming that the
3345 response to the INVITE was not 415. If it was, the body in the ACK MAY be any type listed in the Accept
3346 header field in the 415.

3347 For example, consider the following request:

```
3348 INVITE sip:bob@biloxi.com SIP/2.0
3349 Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKkjshdyff
3350 To: Bob <sip:bob@biloxi.com>
3351 From: Alice <sip:alice@atlanta.com>;tag=88sja8x
3352 Max-Forwards: 70
3353 Call-ID: 987asjd97y7atg
3354 CSeq: 986759 INVITE
```

3355 The ACK request for a non-2xx final response to this request would look like this:

```
3356 ACK sip:bob@biloxi.com SIP/2.0
3357 Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKkjshdyff
3358 To: Bob <sip:bob@biloxi.com>;tag=99sa0xk
3359 From: Alice <sip:alice@atlanta.com>;tag=88sja8x
3360 Max-Forwards: 70
3361 Call-ID: 987asjd97y7atg
3362 CSeq: 986759 ACK
```

3363 17.1.2 Non-INVITE Client Transaction

3364 **17.1.2.1 Overview of the non-INVITE Transaction** Non-INVITE transactions do not make use of ACK.
3365 They are simple request-response interactions. For unreliable transports, requests are retransmitted at an in-
3366 terval which starts at T1 and doubles until it hits T2. If a provisional response is received, retransmissions
3367 continue for unreliable transports, but at an interval of T2. The server transaction retransmits the last re-
3368 sponse it sent, which can be a provisional or final response, only when a retransmission of the request is
3369 received. This is why request retransmissions need to continue even after a provisional response, they are to
3370 ensure reliable delivery of the final response.

3371 Unlike an INVITE transaction, a non-INVITE transaction has no special handling for the 2xx response.
3372 The result is that only a single 2xx response to a non-INVITE is ever delivered to a UAC.

3373 **17.1.2.2 Formal Description** The state machine for the non-INVITE client transaction is shown in Fig-
3374 ure 6. It is very similar to the state machine for INVITE.

3375 The “Trying” state is entered when the TU initiates a new client transaction with a request. When
3376 entering this state, the client transaction SHOULD set timer F to fire in $64 * T1$ seconds. The request MUST be
3377 passed to the transport layer for transmission. If an unreliable transport is in use, the client transaction MUST

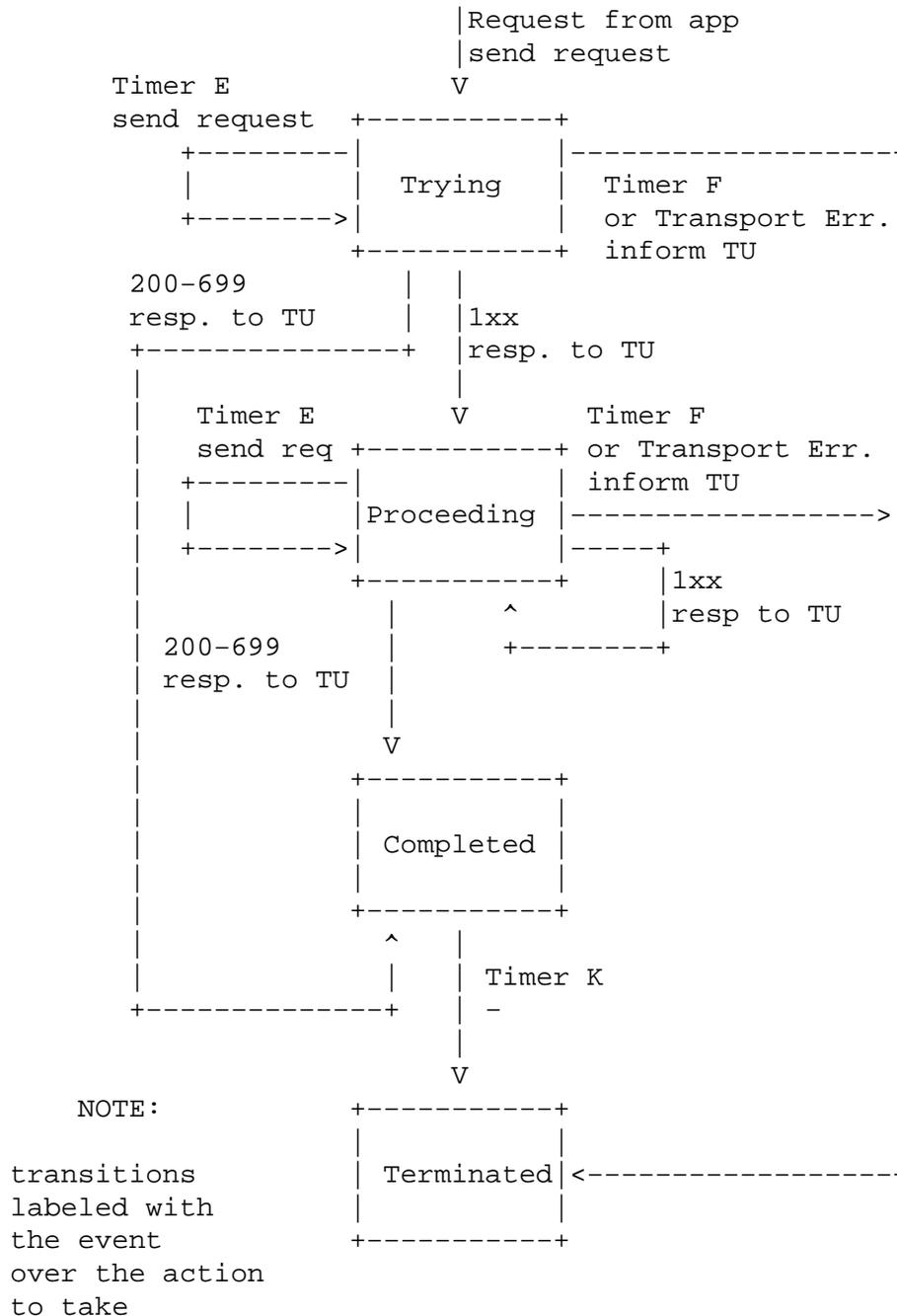


Figure 6: non-INVITE client transaction

3378 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
 3379 value of MIN(2*T1, T2). When the timer fires again, it is reset to a MIN(4*T1, T2). This process continues
 3380 so that retransmissions occur with an exponentially increasing interval that caps at T2. The default value
 3381 of T2 is 4s, and it represents the amount of time a non-INVITE server transaction will take to respond to a

3382 request, if it does not respond immediately. For the default values of T1 and T2, this results in intervals of
3383 500 ms, 1 s, 2 s, 4 s, 4 s, etc.

3384 If Timer F fires while the client transaction is still in the “Trying” state, the client transaction SHOULD
3385 inform the TU about the timeout, and then it SHOULD enter the “Terminated” state. If a provisional response
3386 is received while in the “Trying” state, the response MUST be passed to the TU, and then the client transaction
3387 SHOULD move to the “Proceeding” state. If a final response (status codes 200-699) is received while in the
3388 “Trying” state, the response MUST be passed to the TU, and the client transaction MUST transition to the
3389 “Completed” state.

3390 If Timer E fires while in the “Proceeding” state, the request MUST be passed to the transport layer
3391 for retransmission, and Timer E MUST be reset with a value of T2 seconds. If timer F fires while in the
3392 “Proceeding” state, the TU MUST be informed of a timeout, and the client transaction MUST transition to the
3393 terminated state. If a final response (status codes 200-699) is received while in the “Proceeding” state, the
3394 response MUST be passed to the TU, and the client transaction MUST transition to the “Completed” state.

3395 Once the client transaction enters the “Completed” state, it MUST set Timer K to fire in T4 seconds for
3396 unreliable transports, and zero seconds for reliable transports. The “Completed” state exists to buffer any
3397 additional response retransmissions that may be received (which is why the client transaction remains there
3398 only for unreliable transports). T4 represents the amount of time the network will take to clear messages
3399 between client and server transactions. The default value of T4 is 5s. A response is a retransmission when it
3400 matches the same transaction, using the rules specified in Section 17.1.3. If Timer K fires while in this state,
3401 the client transaction MUST transition to the “Terminated” state.

3402 Once the transaction is in the terminated state, it MUST be destroyed.

3403 **17.1.3 Matching Responses to Client Transactions**

3404 When the transport layer in the client receives a response, it has to determine which client transaction
3405 will handle the response, so that the processing of Sections 17.1.1 and 17.1.2 can take place. The branch
3406 parameter in the top *Via* header field is used for this purpose. A response matches a client transaction under
3407 two conditions:

- 3408 1. If the response has the same value of the branch parameter in the top *Via* header field as the branch
3409 parameter in the top *Via* header field of the request that created the transaction.
- 3410 2. If the method parameter in the *CSeq* header field matches the method of the request that created the
3411 transaction. The method is needed since a *CANCEL* request constitutes a different transaction, but
3412 shares the same value of the branch parameter.

3413 A response that matches a transaction matched by a previous response is considered a retransmission of
3414 that response.

3415 If a request is sent via multicast, it is possible that it will generate multiple responses from different
3416 servers. These responses will all have the same branch parameter in the topmost *Via*, but vary in the *To*
3417 tag. The first response received, based on the rules above, will be used, and others will be viewed as
3418 retransmissions. That is not an error; multicast SIP provides only a rudimentary “single-hop-discovery-
3419 like” service that is limited to processing a single response. See Section 18.1.1 for details.

3420 **17.1.4 Handling Transport Errors**

3421 When the client transaction sends a request to the transport layer to be sent, the following procedures are
3422 followed if the transport layer indicates a failure.

3423 The client transaction SHOULD inform the TU that a transport failure has occurred, and the client trans-
3424 action SHOULD transition directly to the "Terminated" state. The TU will handle the failover mechanisms
3425 described in [4].

3426 **17.2 Server Transaction**

3427 The server transaction is responsible for the delivery of requests to the TU and the reliable transmission of
3428 responses. It accomplishes this through a state machine. Server transactions are created by the core when a
3429 request is received, and transaction handling is desired for that request (this is not always the case).

3430 As with the client transactions, the state machine depends on whether the received request is an INVITE
3431 request.

3432 **17.2.1 INVITE Server Transaction**

3433 The state diagram for the INVITE server transaction is shown in Figure 7.

3434 When a server transaction is constructed with a request, it enters the "Proceeding" state. The server
3435 transaction MUST generate a 100 (Trying) response unless it knows that the TU will generate a provisional
3436 or final response within 200 ms, in which case it MAY generate a 100 (Trying) response. This provisional
3437 response is needed to quench request retransmissions rapidly in order to avoid network congestion. The 100
3438 (Trying) response is constructed according to the procedures in Section 8.2.6, except that the insertion of
3439 tags in the To header field of the response (when none was present in the request) is downgraded from MAY
3440 to SHOULD NOT. The request MUST be passed to the TU.

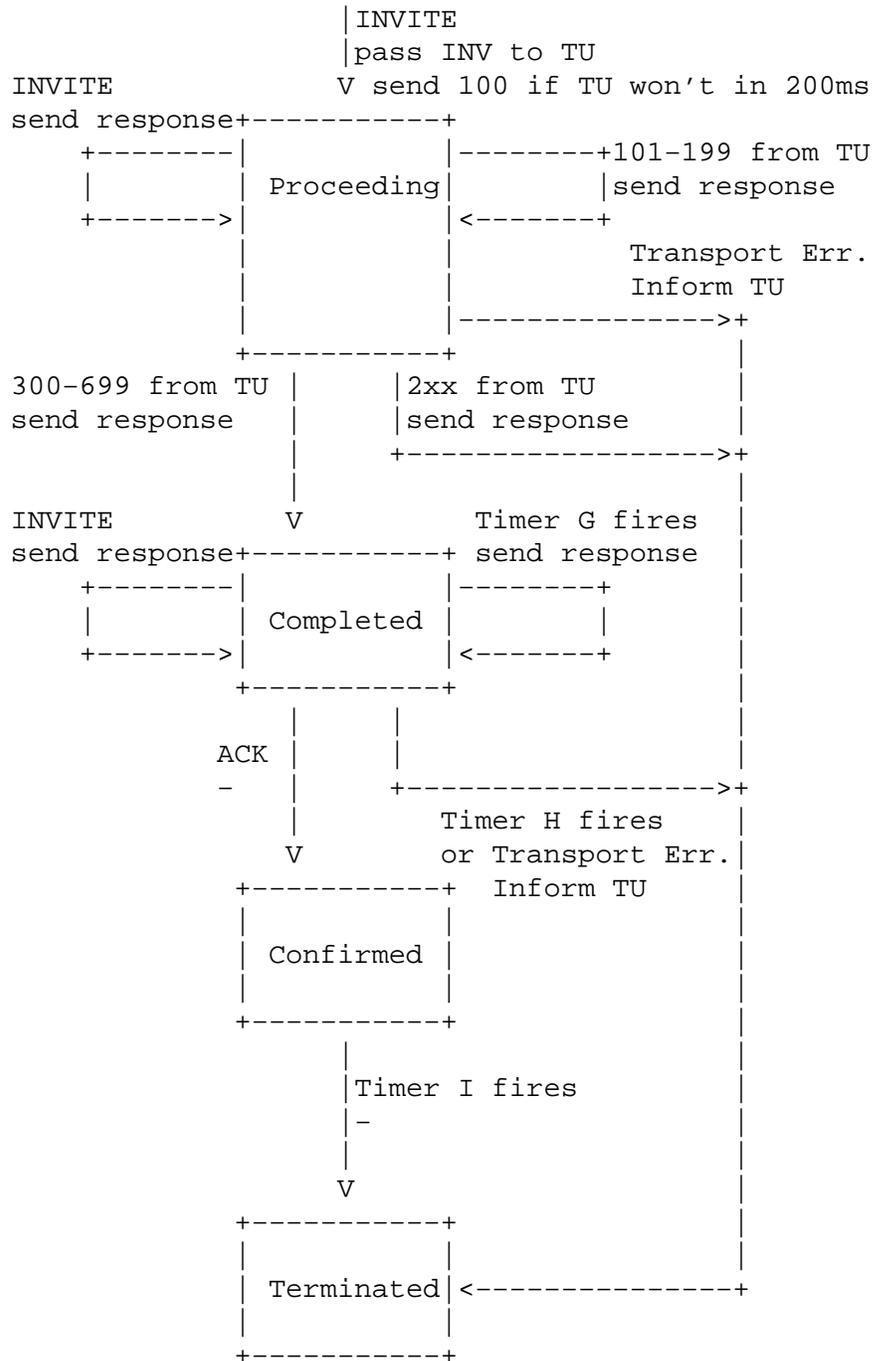
3441 The TU passes any number of provisional responses to the server transaction. So long as the server
3442 transaction is in the "Proceeding" state, each of these MUST be passed to the transport layer for transmission.
3443 They are not sent reliably by the transaction layer (they are not retransmitted by it) and do not cause a change
3444 in the state of the server transaction. If a request retransmission is received while in the "Proceeding" state,
3445 the most recent provisional response that was received from the TU MUST be passed to the transport layer
3446 for retransmission. A request is a retransmission if it matches the same server transaction based on the rules
3447 of Section 17.2.3.

3448 If, while in the "Proceeding" state, the TU passes a 2xx response to the server transaction, the server
3449 transaction MUST pass this response to the transport layer for transmission. It is not retransmitted by the
3450 server transaction; retransmissions of 2xx responses are handled by the TU. The server transaction MUST
3451 then transition to the "Terminated" state.

3452 While in the "Proceeding" state, if the TU passes a response with status code from 300 to 699 to the
3453 server transaction, the response MUST be passed to the transport layer for transmission, and the state machine
3454 MUST enter the "Completed" state. For unreliable transports, timer G is set to fire in T1 seconds, and is not
3455 set to fire for reliable transports.

3456 This is a change from RFC 2543, where responses were always retransmitted, even over reliable transports.

3457 When the "Completed" state is entered, timer H MUST be set to fire in 64*T1 seconds for all transports.
3458 Timer H determines when the server transaction abandons retransmitting the response. Its value is chosen
3459 to equal Timer B, the amount of time a client transaction will continue to retry sending a request. If timer G



3460 fires, the response is passed to the transport layer once more for retransmission, and timer G is set to fire in
3461 $\text{MIN}(2*T1, T2)$ seconds. From then on, when timer G fires, the response is passed to the transport again for
3462 transmission, and timer G is reset with a value that doubles, unless that value exceeds T2, in which case it
3463 is reset with the value of T2. This is identical to the retransmit behavior for requests in the "Trying" state of
3464 the non-INVITE client transaction. Furthermore, while in the "Completed" state, if a request retransmission
3465 is received, the server SHOULD pass the response to the transport for retransmission.

3466 If an ACK is received while the server transaction is in the "Completed" state, the server transaction
3467 MUST transition to the "Confirmed" state. As Timer G is ignored in this state, any retransmissions of the
3468 response will cease.

3469 If timer H fires while in the "Completed" state, it implies that the ACK was never received. In this
3470 case, the server transaction MUST transition to the "Terminated" state, and MUST indicate to the TU that a
3471 transaction failure has occurred.

3472 The purpose of the "Confirmed" state is to absorb any additional ACK messages that arrive, triggered
3473 from retransmissions of the final response. When this state is entered, timer I is set to fire in T4 seconds for
3474 unreliable transports, and zero seconds for reliable transports. Once timer I fires, the server MUST transition
3475 to the "Terminated" state.

3476 Once the transaction is in the "Terminated" state, it MUST be destroyed. As with client transactions, this
3477 is needed to ensure reliability of the 2xx responses to INVITE.

3478 17.2.2 Non-INVITE Server Transaction

3479 The state machine for the non-INVITE server transaction is shown in Figure 8.

3480 The state machine is initialized in the "Trying" state and is passed a request other than INVITE or
3481 ACK when initialized. This request is passed up to the TU. Once in the "Trying" state, any further request
3482 retransmissions are discarded. A request is a retransmission if it matches the same server transaction, using
3483 the rules specified in Section 17.2.3.

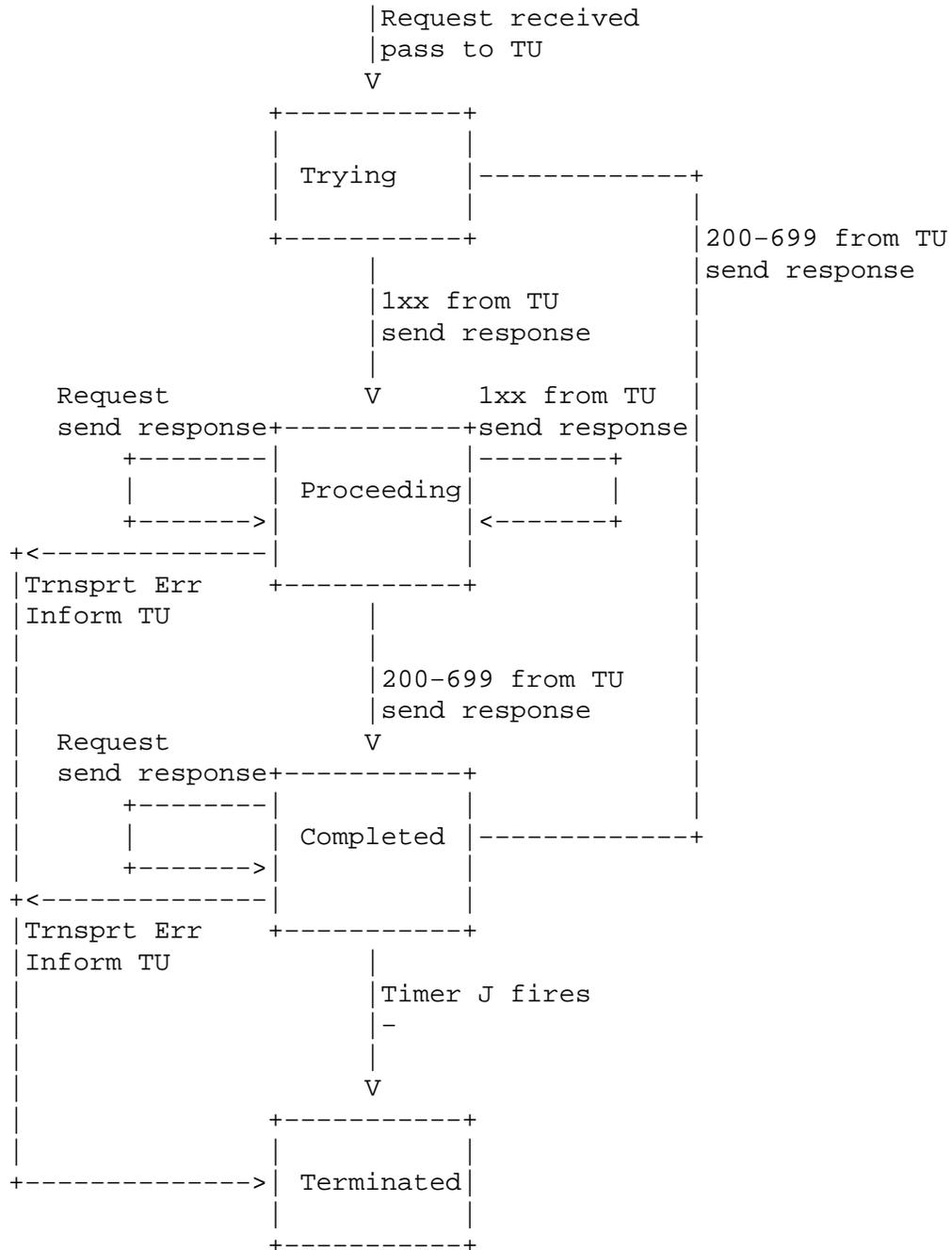
3484 While in the "Trying" state, if the TU passes a provisional response to the server transaction, the server
3485 transaction MUST enter the "Proceeding" state. The response MUST be passed to the transport layer for
3486 transmission. Any further provisional responses that are received from the TU while in the "Proceeding"
3487 state MUST be passed to the transport layer for transmission. If a retransmission of the request is received
3488 while in the "Proceeding" state, the most recently sent provisional response MUST be passed to the transport
3489 layer for retransmission. If the TU passes a final response (status codes 200-699) to the server while in the
3490 "Proceeding" state, the transaction MUST enter the "Completed" state, and the response MUST be passed to
3491 the transport layer for transmission.

3492 When the server transaction enters the "Completed" state, it MUST set Timer J to fire in $64*T1$ seconds
3493 for unreliable transports, and zero seconds for reliable transports. While in the "Completed" state, the server
3494 transaction MUST pass the final response to the transport layer for retransmission whenever a retransmission
3495 of the request is received. Any other final responses passed by the TU to the server transaction MUST be
3496 discarded while in the "Completed" state. The server transaction remains in this state until Timer J fires, at
3497 which point it MUST transition to the "Terminated" state.

3498 The server transaction MUST be destroyed the instant it enters the "Terminated" state.

3499 17.2.3 Matching Requests to Server Transactions

3500 When a request is received from the network by the server, it has to be matched to an existing transaction.
3501 This is accomplished in the following manner.



3502 The branch parameter in the topmost *Via* header field of the request is examined. If it is present and
3503 begins with the magic cookie “z9hG4bK”, the request was generated by a client transaction compliant to this
3504 specification. Therefore, the branch parameter will be unique across all transactions sent by that client. The
3505 request matches a transaction if the branch parameter in the request is equal to the one in the top *Via* header
3506 field of the request that created the transaction, the *sent-by* value in the top *Via* of the request is equal to
3507 the one in the request that created the transaction, and in the case of a *CANCEL* request, the method of
3508 the request that created the transaction was also *CANCEL*. This matching rule applies to both *INVITE* and
3509 non-*INVITE* transactions alike.

3510 The *sent-by* value is used as part of the matching process because there could be accidental or malicious dupli-
3511 cation of branch parameters from different clients.

3512 If the branch parameter in the top *Via* header field is not present, or does not contain the magic cookie,
3513 the following procedures are used. These exist to handle backwards compatibility with RFC 2543 compliant
3514 implementations.

3515 The *INVITE* request matches a transaction if the *Request-URI*, *To* tag, *From* tag, *Call-ID*, *CSeq*, and
3516 top *Via* header field match those of the *INVITE* request which created the transaction. In this case, the
3517 *INVITE* is a retransmission of the original one that created the transaction. The *ACK* request matches a
3518 transaction if the *Request-URI*, *From* tag, *Call-ID*, *CSeq* number (not the method), and top *Via* header
3519 field match those of the *INVITE* request which created the transaction, and the *To* tag of the *ACK* matches the
3520 *To* tag of the response sent by the server transaction. Matching is done based on the matching rules defined
3521 for each of those header fields. Inclusion of the tag in the *To* header field in the *ACK* matching process
3522 helps disambiguate *ACK* for 2xx from *ACK* for other responses at a proxy, which may have forwarded
3523 both responses (This can occur in unusual conditions. Specifically, when a proxy forked a request, and then
3524 crashes, the responses may be delivered to another proxy, which might end up forwarding multiple responses
3525 upstream). An *ACK* request that matches an *INVITE* transaction matched by a previous *ACK* is considered
3526 a retransmission of that previous *ACK*.

3527 For all other request methods, a request is matched to a transaction if the *Request-URI*, *To* tag, *From*
3528 tag, *Call-ID* *Cseq* (including the method), and top *Via* header field match those of the request that created
3529 the transaction. Matching is done based on the matching rules defined for each of those header fields. When
3530 a non-*INVITE* request matches an existing transaction, it is a retransmission of the request that created that
3531 transaction.

3532 Because the matching rules include the *Request-URI*, the server cannot match a response to a transac-
3533 tion. When the TU passes a response to the server transaction, it must pass it to the specific server transaction
3534 for which the response is targeted.

3535 17.2.4 Handling Transport Errors

3536 When the server transaction sends a response to the transport layer to be sent, the following procedures are
3537 followed if the transport layer indicates a failure.

3538 First, the procedures in [4] are followed, which attempt to deliver the response to a backup. If those
3539 should all fail, based on the definition of failure in [4], the server transaction SHOULD inform the TU that a
3540 failure has occurred, and SHOULD transition to the terminated state.

3541 18 Transport

3542 The transport layer is responsible for the actual transmission of requests and responses over network trans-
3543 ports. This includes determination of the connection to use for a request or response in the case of connection-
3544 oriented transports.

3545 The transport layer is responsible for managing persistent connections for transport protocols like TCP
3546 and SCTP, or TLS over those, including ones opened to the transport layer. This includes connections
3547 opened by the client or server transports, so that connections are shared between client and server transport
3548 functions. These connections are indexed by the tuple formed from the address, port, and transport protocol
3549 at the far end of the connection. When a connection is opened by the transport layer, this index is set to the
3550 destination IP, port and transport. When the connection is accepted by the transport layer, this index is set to
3551 the source IP address, port number, and transport. Note that, because the source port is often ephemeral, but
3552 it cannot be known whether it is ephemeral or selected through procedures in [4], connections accepted by
3553 the transport layer will frequently not be reused. The result is that two proxies in a "peering" relationship
3554 using a connection-oriented transport frequently will have two connections in use, one for transactions
3555 initiated in each direction.

3556 It is RECOMMENDED that connections be kept open for some implementation-defined duration after the
3557 last message was sent or received over that connection. This duration SHOULD at least equal the longest
3558 amount of time the element would need in order to bring a transaction from instantiation to the terminated
3559 state. This is to make it likely that transactions complete over the same connection on which they are
3560 initiated (for example, request, response, and in the case of INVITE, ACK for non-2xx responses). This
3561 usually means at least 64*T1 (see Section 17.1.1.1 for a definition of T1). However, it could be larger in an
3562 element that has a TU using a large value for timer C (bullet 11 of Section 16.6), for example.

3563 All SIP elements MUST implement UDP and TCP. SIP elements MAY implement other protocols.

3564 Making TCP mandatory for the UA is a substantial change from RFC 2543. It has arisen out of the need to
3565 handle larger messages, which MUST use TCP, as discussed below. Thus, even if an element never sends large
3566 messages, it may receive one and needs to be able to handle them.

3567 18.1 Clients

3568 18.1.1 Sending Requests

3569 The client side of the transport layer is responsible for sending the request and receiving responses. The
3570 user of the transport layer passes the client transport the request, an IP address, port, transport, and possibly
3571 TTL for multicast destinations.

3572 If a request is within 200 bytes of the path MTU, or if it is larger than 1300 bytes and the path MTU
3573 is unknown, the request MUST be sent using TCP. This prevents fragmentation of messages over UDP
3574 and provides congestion control for larger messages. However, implementations MUST be able to handle
3575 messages up to the maximum datagram packet size. For UDP, this size is 65,535 bytes, including IP and
3576 UDP headers.

3577 The 200 byte "buffer" between the message size and the MTU accommodates the fact that the response in
3578 SIP can be larger than the request. This happens due to the addition of Record-Route header field values to the
3579 responses to INVITE, for example. With the extra buffer, the response can be about 170 bytes larger than the request,
3580 and still not be fragmented on IPv4 (about 30 bytes is consumed by IP/UDP, assuming no IPSec). 1300 is chosen
3581 when path MTU is not known, based on the assumption of a 1500 byte Ethernet MTU.

3582 If an element sends a request over TCP because of these message size constraints, and that request

3583 would have otherwise been sent over UDP, if the attempt to establish the connection generates either an
3584 ICMP Protocol Not Supported, or results in a TCP reset, the element SHOULD retry the request, using UDP.
3585 This is only to provide backwards compatibility with RFC 2543 compliant implementations that do not
3586 support UDP. It is anticipated that this behavior will be deprecated in a future revision of this specification.

3587 A client that sends a request to a multicast address MUST add the "maddr" parameter to its Via header
3588 field value containing the destination multicast address, and for IPv4, SHOULD add the "ttl" parameter with
3589 a value of 1. Usage of IPv6 multicast is not defined in this specification, and will be a subject of future
3590 standardization when the need arises.

3591 These rules result in a purposeful limitation of multicast in SIP. Its primary function is to provide an
3592 "single-hop-discovery-like" service, delivering a request to a group of homogeneous servers, where it is only
3593 required to process the response from any one of them. This functionality is most useful for registrations.
3594 In fact, based on the transaction processing rules in Section 17.1.3, the client transaction will accept the first
3595 response, and view any others as retransmissions because they all contain the same Via branch identifier.

3596 Before a request is sent, the client transport MUST insert a value of the "sent-by" field into the Via header
3597 field. This field contains an IP address or host name, and port. The usage of an FQDN is RECOMMENDED.
3598 This field is used for sending responses under certain conditions, described below. If the port is absent, the
3599 default value depends on the transport. It is 5060 for UDP, TCP and SCTP, 5061 for TLS.

3600 For reliable transports, the response is normally sent on the connection on which the request was re-
3601 ceived. Therefore, the client transport MUST be prepared to receive the response on the same connection
3602 used to send the request. Under error conditions, the server may attempt to open a new connection to send
3603 the response. To handle this case, the transport layer MUST also be prepared to receive an incoming con-
3604 nection on the source IP address from which the request was sent and port number in the "sent-by" field. It
3605 also MUST be prepared to receive incoming connections on any address and port that would be selected by
3606 a server based on the procedures described in Section 5 of [4].

3607 For unreliable unicast transports, the client transport MUST be prepared to receive responses on the
3608 source IP address from which the request is sent (as responses are sent back to the source address) and the
3609 port number in the "sent-by" field. Furthermore, as with reliable transports, in certain cases the response
3610 will be sent elsewhere. The client MUST be prepared to receive responses on any address and port that would
3611 be selected by a server based on the procedures described in Section 5 of [4].

3612 For multicast, the client transport MUST be prepared to receive responses on the same multicast group
3613 and port to which the request is sent (that is, it needs to be a member of the multicast group it sent the request
3614 to.)

3615 If a request is destined to an IP address, port, and transport to which an existing connection is open, it
3616 is RECOMMENDED that this connection be used to send the request, but another connection MAY be opened
3617 and used.

3618 If a request is sent using multicast, it is sent to the group address, port, and TTL provided by the transport
3619 user. If a request is sent using unicast unreliable transports, it is sent to the IP address and port provided by
3620 the transport user.

3621 18.1.2 Receiving Responses

3622 When a response is received, the client transport examines the top Via header field value. If the value of
3623 the "sent-by" parameter in that header field value does not correspond to a value that the client transport is
3624 configured to insert into requests, the response MUST be silently discarded.

3625 If there are any client transactions in existence, the client transport uses the matching procedures of Sec-

3626 tion 17.1.3 to attempt to match the response to an existing transaction. If there is a match, the response MUST
3627 be passed to that transaction. Otherwise, the response MUST be passed to the core (whether it be stateless
3628 proxy, stateful proxy, or UA) for further processing. Handling of these “stray” responses is dependent on
3629 the core (a proxy will forward them, while a UA will discard, for example).

3630 18.2 Servers

3631 18.2.1 Receiving Requests

3632 A server SHOULD be prepared to received requests on any IP address, port and transport combination that can
3633 be the result of a DNS lookup on a SIP or SIPS URI [4] that is handed out for the purposes of communicating
3634 with that server. In this context, “handing out” includes placing a URI in a **Contact** header field in a
3635 **REGISTER** request or a any redirect response, or in a **Record-Route** header field in a request or response.
3636 A URI can also be “handed out” by placing it on a web page or business card. It is also RECOMMENDED
3637 that a server listen for requests on the default SIP ports (5060 for TCP and UDP, 5061 for TLS over TCP)
3638 on all public interfaces. The typical exception would be private networks, or when multiple server instances
3639 are running on the same host. For any port and interface that a server listens on for UDP, it MUST listen on
3640 that same port and interface for TCP. This is because a message may need to be sent using TCP, rather than
3641 UDP, if it is too large. As a result, the converse is not true. A server need not listen for UDP on a particular
3642 address and port just because it is listening on that same address and port for TCP. There may, of course, be
3643 other reasons why a server needs to listen for UDP on a particular address and port.

3644 When the server transport receives a request over any transport, it MUST examine the value of the “sent-
3645 by” parameter in the top **Via** header field value. If the host portion of the “sent-by” parameter contains a
3646 domain name, or if it contains an IP address that differs from the packet source address, the server MUST
3647 add a “received” parameter to that **Via** header field value. This parameter MUST contain the source address
3648 from which the packet was received. This is to assist the server transport layer in sending the response, since
3649 it must be sent to the source IP address from which the request came.

3650 Consider a request received by the server transport which looks like, in part:

```
3651 INVITE sip:bob@Biloxi.com SIP/2.0  
3652 Via: SIP/2.0/UDP bobspc.biloxi.com:5060
```

3653 The request is received with a source IP address of 192.0.2.4. Before passing the request up, the transport
3654 adds a “received” parameter, so that the request would look like, in part:

```
3655 INVITE sip:bob@Biloxi.com SIP/2.0  
3656 Via: SIP/2.0/UDP bobspc.biloxi.com:5060;received=192.0.2.4
```

3657 Next, the server transport attempts to match the request to a server transaction. It does so using the
3658 matching rules described in Section 17.2.3. If a matching server transaction is found, the request is passed
3659 to that transaction for processing. If no match is found, the request is passed to the core, which may
3660 decide to construct a new server transaction for that request. Note that when a UAS core sends a 2xx
3661 response to INVITE, the server transaction is destroyed. This means that when the ACK arrives, there will
3662 be no matching server transaction, and based on this rule, the ACK is passed to the UAS core, where it is
3663 processed.

3664 18.2.2 Sending Responses

3665 The server transport uses the value of the top *Via* header field in order to determine where to send a response.
3666 It MUST follow the following process:

- 3667 • If the “sent-protocol” is a reliable transport protocol such as TCP or SCTP, or TLS over those,
3668 the response MUST be sent using the existing connection to the source of the original request that
3669 created the transaction, if that connection is still open. This requires the server transport to maintain
3670 an association between server transactions and transport connections. If that connection is no longer
3671 open, the server SHOULD open a connection to the IP address in the “received” parameter, if present,
3672 using the port in the “sent-by” value, or the default port for that transport, if no port is specified.
3673 If that connection attempt fails, the server SHOULD use the procedures in [4] for servers in order to
3674 determine the IP address and port to open the connection and send the response to.
- 3675 • Otherwise, if the *Via* header field value contains a “maddr” parameter, the response MUST be for-
3676 forwarded to the address listed there, using the port indicated in “sent-by”, or port 5060 if none is
3677 present. If the address is a multicast address, the response SHOULD be sent using the TTL indicated
3678 in the “ttl” parameter, or with a TTL of 1 if that parameter is not present.
- 3679 • Otherwise (for unreliable unicast transports), if the top *Via* has a “received” parameter, the response
3680 MUST be sent to the address in the “received” parameter, using the port indicated in the “sent-by”
3681 value, or using port 5060 if none is specified explicitly. If this fails, for example, elicits an ICMP
3682 “port unreachable” response, the procedures of Section 5 of [4] SHOULD be used to determine where
3683 to send the response.
- 3684 • Otherwise, if it is not receiver-tagged, the response MUST be sent to the address indicated by the
3685 “sent-by” value, using the procedures in Section 5 of [4].

3686 18.3 Framing

3687 In the case of message-oriented transports (such as UDP), if the message has a *Content-Length* header
3688 field, the message body is assumed to contain that many bytes. If there are additional bytes in the transport
3689 packet beyond the end of the body, they MUST be discarded. If the transport packet ends before the end of
3690 the message body, this is considered an error. If the message is a response, it MUST be discarded. If the
3691 message is a request, the element SHOULD generate a 400 (Bad Request) response. If the message has no
3692 *Content-Length* header field, the message body is assumed to end at the end of the transport packet.

3693 In the case of stream-oriented transports such as TCP, the *Content-Length* header field indicates the
3694 size of the body. The *Content-Length* header field MUST be used with stream oriented transports.

3695 18.4 Error Handling

3696 Error handling is independent of whether the message was a request or response.

3697 If the transport user asks for a message to be sent over an unreliable transport, and the result is an ICMP
3698 error, the behavior depends on the type of ICMP error. Host, network, port or protocol unreachable errors,
3699 or parameter problem errors SHOULD cause the transport layer to inform the transport user of a failure in
3700 sending. Source quench and TTL exceeded ICMP errors SHOULD be ignored.

3701 If the transport user asks for a request to be sent over a reliable transport, and the result is a connection
3702 failure, the transport layer SHOULD inform the transport user of a failure in sending.

3703 19 Common Message Components

3704 There are certain components of SIP messages that appear in various places within SIP messages (and
3705 sometimes, outside of them) that merit separate discussion.

3706 19.1 SIP and SIPS Uniform Resource Indicators

3707 A SIP or SIPS URI identifies a communications resource. Like all URIs, SIP and SIPS URIs may be placed
3708 in web pages, email messages, or printed literature. They contain sufficient information to initiate and
3709 maintain a communication session with the resource.

3710 Examples of communications resources include the following:

- 3711 • a user of an online service
- 3712 • an appearance on a multi-line phone
- 3713 • a mailbox on a messaging system
- 3714 • a PSTN number at a gateway service
- 3715 • a group (such as “sales” or “helpdesk”) in an organization

3716 A SIPS URI specifies that the resource be contacted securely. This means, in particular, that TLS is to
3717 be used between the UAC and the domain that owns the URI. From there, secure communications are used
3718 to reach the user, where the specific security mechanism depends on the policy of the domain. Any resource
3719 described by a SIP URI can be “upgraded” to a SIPS URI by just changing the scheme, if it is desired to
3720 communicate with that resource securely.

3721 19.1.1 SIP and SIPS URI Components

3722 The “sip:” and “sips:” schemes follow the guidelines in RFC 2396 [5]. They use a form similar to the `mailto`
3723 URL, allowing the specification of SIP request-header fields and the SIP message-body. This makes it
3724 possible to specify the subject, media type, or urgency of sessions initiated by using a URI on a web page or
3725 in an email message. The formal syntax for a SIP or SIPS URI is presented in Section 25. Its general form,
3726 in the case of a SIP URI, is

3727 `sip:user:password@host:port;uri-parameters?headers`

3728 The format for a SIPS URI is the same, except that the scheme is “sips” instead of sip. These tokens,
3729 and some of the tokens in their expansions, have the following meanings:

3730 **user:** The identifier of a particular resource at the host being addressed. The term “host” in this context
3731 frequently refers to a domain. The “userinfo” of a URI consists of this user field, the password field,
3732 and the @ sign following them. The `userinfo` part of a URI is optional and MAY be absent when the
3733 destination host does not have a notion of users or when the host itself is the resource being identified.
3734 If the @ sign is present in a SIP or SIPS URI, the user field MUST NOT be empty.

3735 If the host being addressed can process telephone numbers, for instance, an Internet telephony gate-
3736 way, a `telephone-subscriber` field defined in RFC 2806 [9] MAY be used to populate the user field.
3737 There are special escaping rules for encoding `telephone-subscriber` fields in SIP and SIPS URIs
3738 described in Section 19.1.2.

3739 **password:** A password associated with the user. While the SIP and SIPS URI syntax allows this field to
3740 be present, its use is NOT RECOMMENDED, because the passing of authentication information in clear
3741 text (such as URIs) has proven to be a security risk in almost every case where it has been used. For
3742 instance, transporting a PIN number in this field exposes the PIN.

3743 Note that the password field is just an extension of user portion. Implementations not wishing to give
3744 special significance to the password portion of the field MAY simply treat "user:password" as a single
3745 string.

3746 **host:** The host providing the SIP resource. The host part contains either a fully-qualified domain name
3747 or numeric IPv4 or IPv6 address. Using the fully-qualified domain name form is RECOMMENDED
3748 whenever possible.

3749 **port:** The port number where the request is to be sent.

3750 **URI parameters:** Parameters affecting a request constructed from the URI.

3751 URI parameters are added after the hostport component and are separated by semi-colons.

3752 URI parameters take the form:

3753 `parameter-name "=" parameter-value`

3754 Even though an arbitrary number of URI parameters may be included in a URI, any given parameter-
3755 name MUST NOT appear more than once.

3756 This extensible mechanism includes the transport, maddr, ttl, user, method and lr parameters.

3757 The transport parameter determines the transport mechanism to be used for sending SIP messages,
3758 as specified in [4]. SIP can use any network transport protocol. Parameter names are defined for UDP
3759 (RFC 768 [14]), TCP (RFC 761 [15]), and SCTP (RFC 2960 [16]). For a SIPS URI, the transport
3760 parameter MUST indicate a reliable transport.

3761 The maddr parameter indicates the server address to be contacted for this user, overriding any address
3762 derived from the host field. When an maddr parameter is present, the port and transport components
3763 of the URI apply to the address indicated in the maddr parameter value. [4] describes the proper
3764 interpretation of the transport, maddr, and hostport in order to obtain the destination address, port,
3765 and transport for sending a request.

3766 The maddr field has been used as a simple form of loose source routing. It allows a URI to specify a proxy
3767 that must be traversed en-route to the destination. Continuing to use the maddr parameter this way is strongly
3768 discouraged (the mechanisms that enable it are deprecated). Implementations should instead use the Route
3769 mechanism described in this document, establishing a pre-existing route set if necessary (see Section 8.1.1.1).
3770 This provides a full URI to describe the node to be traversed.

3771 The ttl parameter determines the time-to-live value of the UDP multicast packet and MUST only be
3772 used if maddr is a multicast address and the transport protocol is UDP. For example, to specify to call
3773 alice@atlanta.com using multicast to 239.255.255.1 with a ttl of 15, the following URI would
3774 be used:

3775 `sip:alice@atlanta.com;maddr=239.255.255.1;ttl=15`

3776 The set of valid telephone-subscriber strings is a subset of valid user strings. The user URI pa-
3777 rameter exists to distinguish telephone numbers from user names that happen to look like telephone
3778 numbers. If the user string contains a telephone number formatted as a telephone-subscriber, the
3779 user parameter value "phone" SHOULD be present. Even without this parameter, recipients of SIP
3780 and SIPS URIs MAY interpret the pre-@ part as a telephone number if local restrictions on the name
3781 space for user name allow it.

3782 The method of the SIP request constructed from the URI can be specified with the method parameter.
3783 The lr parameter, when present, indicates that the element responsible for this resource implements
3784 the routing mechanisms specified in this document. This parameter will be used in the URIs proxies
3785 place into Record-Route header field values, and may appear in the URIs in a pre-existing route set.

3786 This parameter is used to achieve backwards compatibility with systems implementing the strict-routing
3787 mechanisms of RFC 2543 and the rfc2543bis drafts up to bis-05. An element preparing to send a request
3788 based on a URI not containing this parameter can assume the receiving element implements strict-routing and
3789 reformat the message to preserve the information in the Request-URI.

3790 Since the uri-parameter mechanism is extensible, SIP elements MUST silently ignore any uri-parameters
3791 that they do not understand.

3792 **Headers:** Header fields to be included in a request constructed from the URI.

3793 Headers fields in the SIP request can be specified with the "?" mechanism within a URI. The header
3794 names and values are encoded in ampersand separated hname = hvalue pairs. The special hname
3795 "body" indicates that the associated hvalue is the message-body of the SIP request.

3796 Table 1 summarizes the use of SIP and SIPS URI components based on the context in which the URI
3797 appears. The external column describes URIs appearing anywhere outside of a SIP message, for instance on
3798 a web page or business card. Entries marked "m" are mandatory, those marked "o" are optional, and those
3799 marked "-" are not allowed. Elements processing URIs SHOULD ignore any disallowed components if they
3800 are present. The second column indicates the default value of an optional element if it is not present. "-"
3801 indicates that the element is either not optional, or has no default value.

3802 URIs in Contact header fields have different restrictions depending on the context in which the header
3803 field appears. One set applies to messages that establish and maintain dialogs (INVITE and its 200 (OK)
3804 response). The other applies to registration and redirection messages (REGISTER, its 200 (OK) response,
3805 and 3xx class responses to any method).

3806 19.1.2 Character Escaping Requirements

3807 SIP follows the requirements and guidelines of RFC 2396 [5] when defining the set of characters that must
3808 be escaped in a SIP URI, and uses its "% HEX HEX" mechanism for escaping. From RFC 2396 [5]:

3809 The set of characters actually reserved within any given URI component is defined by that com-
3810 ponent. In general, a character is reserved if the semantics of the URI changes if the character
3811 is replaced with its escaped US-ASCII encoding. [5].

3812 Excluded US-ASCII characters (RFC 2396 [5]), such as space and control characters and characters used as
3813 URI delimiters, also MUST be escaped. URIs MUST NOT contain unescaped space and control characters.

3814 For each component, the set of valid BNF expansions defines exactly which characters may appear
3815 unescaped. All other characters MUST be escaped.

	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	o
host	–	m	m	m	m	m	m
port	(1)	o	-	-	o	o	o
user-param	ip	o	o	o	o	o	o
method	INVITE	-	-	-	-	-	o
maddr-param	–	o	-	-	o	o	o
ttl-param	1	o	-	-	o	-	o
transp.-param	(2)	o	-	-	o	o	o
lr-param	–	o	-	-	-	o	o
other-param	–	o	o	o	o	o	o
headers	–	-	-	-	o	-	o

(1): The default port value is transport and scheme dependent. The default is 5060 for sip: using UDP, TCP, or SCTP. The default is 5061 for sip: using TLS over TCP and sips: over TCP.

(2): The default transport is scheme dependent. For sip:, it is UDP. For sips:, it is TCP.

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

3816 For example, “@” is not in the set of characters in the user component, so the user “j@s0n” must have
3817 at least the @ sign encoded, as in “j%40s0n”.

3818 Expanding the `hname` and `hvalue` tokens in Section 25 show that all URI reserved characters in header
3819 field names and values MUST be escaped.

3820 The `telephone-subscriber` subset of the `user` component has special escaping considerations. The set
3821 of characters not reserved in the RFC 2806 [9] description of `telephone-subscriber` contains a number
3822 of characters in various syntax elements that need to be escaped when used in SIP URIs. Any characters
3823 occurring in a `telephone-subscriber` that do not appear in an expansion of the BNF for the `user` rule MUST
3824 be escaped.

3825 Note that character escaping is not allowed in the host component of a SIP or SIPS URI (the % character
3826 is not valid in its expansion). This is likely to change in the future as requirements for Internationalized
3827 Domain Names are finalized. Current implementations MUST NOT attempt to improve robustness by treating
3828 received escaped characters in the host component as literally equivalent to their unescaped counterpart. The
3829 behavior required to meet the requirements of IDN may be significantly different.

3830 19.1.3 Example SIP and SIPS URIs

```
3831 sip:alice@atlanta.com
3832 sip:alice:secretword@atlanta.com;transport=tcp
3833 sips:alice@atlanta.com?subject=project%20x&priority=urgent
3834 sip:+1-212-555-1212:1234@gateway.com;user=phone
3835 sips:1212@gateway.com
```

3836 sip:alice@192.0.2.4
3837 sip:atlanta.com;method=REGISTER?to=alice%40atlanta.com
3838 sip:alice;day=tuesday@atlanta.com

3839 The last sample URI above has a `user` field value of “alice;day=tuesday”. The escaping rules defined
3840 above allow a semicolon to appear unescaped in this field. For the purposes of this protocol, the field is
3841 opaque. The structure of that value is only useful to the SIP element responsible for the resource.

3842 19.1.4 URI Comparison

3843 Some operations in this specification require determining whether two SIP or SIPS URIs are equivalent.
3844 In this specification, registrars need to compare bindings in `Contact` URIs in `REGISTER` requests (see
3845 Section 10.3.) SIP and SIPS URIs are compared for equality according to the following rules:

- 3846 • A SIP and SIPS URI are never equivalent.
- 3847 • Comparison of the `userinfo` of SIP and SIPS URIs is case-sensitive. This includes `userinfo` containing
3848 passwords or formatted as `telephone-subscribers`. Comparison of all other components of the URI
3849 is case-insensitive unless explicitly defined otherwise.
- 3850 • The ordering of parameters and header fields is not significant in comparing SIP and SIPS URIs.
- 3851 • Characters other than those in the “reserved” and “unsafe” sets (see RFC 2396 [5]) are equivalent to
3852 their “%” HEX HEX” encoding.
- 3853 • An IP address that is the result of a DNS lookup of a host name does **not** match that host name.
- 3854 • For two URIs to be equal, the `user`, `password`, `host`, and `port` components must match.

3855 A URI omitting the `user` component will *not* match a URI that includes one. A URI omitting the
3856 `password` component will **not** match a URI that includes one.

3857 A URI omitting any component with a default value will *not* match a URI explicitly containing that
3858 component with its default value. For instance, a URI omitting the optional `port` component will
3859 *not* match a URI explicitly declaring `port 5060`. The same is true for the `transport-parameter`, `ttl-`
3860 `parameter`, `user-parameter`, and `method` components.

3861 Defining `sip:user@host` to *not* be equivalent to `sip:user@host:5060` is a change from RFC 2543. When de-
3862 riving addresses from URIs, equivalent addresses are expected from equivalent URIs. The URI `sip:user@host:5060`
3863 will always resolve to port 5060. The URI `sip:user@host` may resolve to other ports through the DNS SRV
3864 mechanisms detailed in [4].

- 3865 • URI `uri-parameter` components are compared as follows
 - 3866 – Any `uri-parameter` appearing in both URIs must match.
 - 3867 – A `user`, `ttl`, or `method uri-parameter` appearing in only one URI never matches, even if it
3868 contains the default value.
 - 3869 – A URI that includes an `maddr` parameter will *not* match a URI that contains no `maddr` param-
3870 eter.

3900 and sip:carol@chicago.com and sip:carol@chicago.com;security=off are equivalent

3901 But sip:carol@chicago.com;security=on and sip:carol@chicago.com;security=off are **not** equivalent

3902 **19.1.5 Forming Requests from a URI**

3903 An implementation needs to take care when forming requests directly from a URI. URIs from business cards,
3904 web pages, and even from sources inside the protocol such as registered contacts may contain inappropriate
3905 header fields or body parts.

3906 An implementation **MUST** include any provided transport, maddr, ttl, or user parameter in the Request-
3907 URI of the formed request. If the URI contains a method parameter, its value **MUST** be used as the method
3908 of the request. The method parameter **MUST NOT** be placed in the Request-URI. Unknown URI parameters
3909 **MUST** be placed in the message's Request-URI.

3910 An implementation **SHOULD** treat the presence of any headers or body parts in the URI as a desire to
3911 include them in the message, and choose to honor the request on a per-component basis.

3912 An implementation **SHOULD NOT** honor these obviously dangerous header fields: From, Call-ID, CSeq,
3913 Via, and Record-Route.

3914 An implementation **SHOULD NOT** honor any requested Route header field values in order to not be used
3915 as an unwitting agent in malicious attacks.

3916 An implementation **SHOULD NOT** honor requests to include header fields that may cause it to falsely ad-
3917 vertise its location or capabilities. These include: Accept, Accept-Encoding, Accept-Language, Allow,
3918 Contact (in its dialog usage), Organization, Supported, and User-Agent.

3919 An implementation **SHOULD** verify the accuracy of any requested descriptive header fields, including:
3920 Content-Disposition, Content-Encoding, Content-Language, Content-Length, Content-Type, Date,
3921 Mime-Version, and Timestamp.

3922 If the request formed from constructing a message from a given URI is not a valid SIP request, the URI
3923 is invalid. An implementation **MUST NOT** proceed with transmitting the request. It should instead pursue
3924 the course of action due an invalid URI in the context it occurs.

3925 The constructed request can be invalid in many ways. These include, but are not limited to, syntax error in
3926 header fields, invalid combinations of URI parameters, or an incorrect description of the message body.

3927 Sending a request formed from a given URI may require capabilities unavailable to the implementation.
3928 The URI might indicate use of an unimplemented transport or extension, for example. An implementation
3929 **SHOULD** refuse to send these requests rather than modifying them to match their capabilities. An imple-
3930 mentation **MUST NOT** send a request requiring an extension that it does not support.

3931 For example, such a request can be formed through the presence of a Require header parameter or a method
3932 URI parameter with an unknown or explicitly unsupported value.

3933 **19.1.6 Relating SIP URIs and tel URLs**

3934 When a tel URL (RFC 2806 [9]) is converted to a SIP or SIPS URI, the entire telephone-subscriber portion
3935 of the tel URL, including any parameters, is placed into the userinfo part of the SIP or SIPS URI.

3936 Thus, tel:+358-555-1234567;postd=pp22 becomes

3937 sip:+358-555-1234567;postd=pp22@foo.com;user=phone

3938 or

3939 sips:+358-555-1234567;postd=pp22@foo.com;user=phone

3940 not

3941 sip:+358-555-1234567@foo.com;postd=pp22;user=phone

3942 or

3943 sips:+358-555-1234567@foo.com;postd=pp22;user=phone

3944 In general, equivalent "tel" URLs converted to SIP or SIPS URIs in this fashion may not produce equiv-
3945 alent SIP or SIPS URIs. The userinfo of SIP and SIPS URIs are compared as a case-sensitive string.
3946 Variance in case-insensitive portions of tel URLs and reordering of tel URL parameters does not affect tel
3947 URL equivalence, but does affect the equivalence of SIP URIs formed from them.

3948 For example,

3949 tel:+358-555-1234567;postd=pp22

3950 tel:+358-555-1234567;POSTD=PP22

3951 are equivalent, while

3952 sip:+358-555-1234567;postd=pp22@foo.com;user=phone

3953 sip:+358-555-1234567;POSTD=PP22@foo.com;user=phone

3954 are not.

3955 Likewise,

3956 tel:+358-555-1234567;postd=pp22;isub=1411

3957 tel:+358-555-1234567;isub=1411;postd=pp22

3958 are equivalent, while

3959 sip:+358-555-1234567;postd=pp22;isub=1411@foo.com;user=phone

3960 sip:+358-555-1234567;isub=1411;postd=pp22@foo.com;user=phone

3961 are not.

3962 To mitigate this problem, elements constructing telephone-subscriber fields to place in the userinfo part
3963 of a SIP or SIPS URI SHOULD fold any case-insensitive portion of telephone-subscriber to lower case,
3964 and order the telephone-subscriber parameters lexically by parameter name. (All components of a tel URL
3965 except for future-extension parameters are defined to be compared case-insensitive.)

3966 Following this suggestion, both

3967 tel:+358-555-1234567;postd=pp22

3968 tel:+358-555-1234567;POSTD=PP22

3969 become

3970 sip:+358-555-1234567;postd=pp22@foo.com;user=phone

3971 and both

3972 tel:+358-555-1234567;postd=pp22;isub=1411

3973 tel:+358-555-1234567;isub=1411;postd=pp22

3974 become

3975 sip:+358-555-1234567;isub=1411;postd=pp22;user=phone

3976 19.2 Option Tags

3977 Option tags are unique identifiers used to designate new options (extensions) in SIP. These tags are used in
3978 Require (Section 20.32), Proxy-Require (Section 20.29), Supported (Section 20.37) and Unsupported
3979 (Section 20.40) header fields. Note that these options appear as parameters in those header fields in an
3980 option-tag = token form (see Section 25 for the definition of token).

3981 Option tags are defined in standards track RFCs. This is a change from past practice, and is instituted
3982 to ensure continuing multi-vendor interoperability (see discussion in Section 20.32 and Section 20.37). An
3983 IANA registry of option tags is used to ensure easy reference.

3984 19.3 Tags

3985 The “tag” parameter is used in the To and From header fields of SIP messages. It serves as a general
3986 mechanism to identify a dialog, which is the combination of the Call-ID along with two tags, one from
3987 each participant in the dialog. When a UA sends a request outside of a dialog, it contains a From tag only,
3988 providing “half” of the dialog ID. The dialog is completed from the response(s), each of which contributes
3989 the second half in the To header field. The forking of SIP requests means that multiple dialogs can be
3990 established from a single request. This also explains the need for the two-sided dialog identifier; without a
3991 contribution from the recipients, the originator could not disambiguate the multiple dialogs established from
3992 a single request.

3993 When a tag is generated by a UA for insertion into a request or response, it MUST be globally unique
3994 and cryptographically random with at least 32 bits of randomness. A property of this selection requirement
3995 is that a UA will place a different tag into the From header of an INVITE as it would place into the To
3996 header of the response to the same INVITE. This is needed in order for a UA to invite itself to a session, a
3997 common case for “hairpinning” of calls in PSTN gateways. Similarly, two INVITEs for different calls will
3998 have different From tags, and two responses for different calls will have different To tags.

3999 Besides the requirement for global uniqueness, the algorithm for generating a tag is implementation-
4000 specific. Tags are helpful in fault tolerant systems, where a dialog is to be recovered on an alternate server
4001 after a failure. A UAS can select the tag in such a way that a backup can recognize a request as part of a
4002 dialog on the failed server, and therefore determine that it should attempt to recover the dialog and any other
4003 state associated with it.

4004 20 Header Fields

4005 The general syntax for header fields is covered in Section 7.3. This section lists the full set of header fields
4006 along with notes on syntax, meaning, and usage. Throughout this section, we use [HX.Y] to refer to Section
4007 X.Y of the current HTTP/1.1 specification RFC 2616 [8]. Examples of each header field are given.

4008 Information about header fields in relation to methods and proxy processing is summarized in Tables 2
4009 and 3.

4010 The “where” column describes the request and response types in which the header field can be used.
4011 Values in this column are:

4012 **R:** header field may only appear in requests;

4013 **r:** header field may only appear in responses;

4014 **2xx, 4xx, etc.:** A numerical value or range indicates response codes with which the header field can be
4015 used;

4016 **c:** header field is copied from the request to the response.

4017 An empty entry in the “where” column indicates that the header field may be present in all requests and
4018 responses.

4019 The “proxy” column describes the operations a proxy may perform on a header field:

4020 **a:** A proxy can add or concatenate the header field if not present.

4021 **m:** A proxy can modify an existing header field value.

4022 **d:** A proxy can delete a header field value.

4023 **r:** A proxy must be able to read the header field, and thus this header field cannot be encrypted.

4024 The next six columns relate to the presence of a header field in a method:

4025 **c:** Conditional; requirements on the header field depend on the context of the message.

4026 **m:** The header field is mandatory.

4027 **m*:** The header field SHOULD be sent, but clients/servers need to be prepared to receive messages without
4028 that header field.

4029 **o:** The header field is optional.

4030 **t:** The header field SHOULD be sent, but clients/servers need to be prepared to receive messages without
4031 that header field. If a stream-based protocol (such as TCP) is used as a transport, then the header field
4032 MUST be sent.

4033 ***:** The header field is required if the message body is not empty. See sections 20.14, 20.15 and 7.4 for
4034 details.

4035 **-:** The header field is not applicable.

4036 “Optional” means that a UA MAY include the header field in a request or response, and a UA MAY ignore
4037 the header field if present in the request or response (The exception to this rule is the **Require** header field
4038 discussed in 20.32). A “mandatory” header field MUST be present in a request, and MUST be understood
4039 by the UAS receiving the request. A mandatory response header field MUST be present in the response, and
4040 the header field MUST be understood by the UAC processing the response. “Not applicable” means that the
4041 header field MUST NOT be present in a request. If one is placed in a request by mistake, it MUST be ignored
4042 by the UAS receiving the request. Similarly, a header field labeled “not applicable” for a response means
4043 that the UAS MUST NOT place the header field in the response, and the UAC MUST ignore the header field
4044 in the response.

4045 A UA SHOULD ignore extension header parameters that are not understood.

4046 A compact form of some common header field names is also defined for use when overall message size
4047 is an issue.

4048 The **Contact**, **From**, and **To** header fields contain a URI. If the URI contains a comma, question mark
4049 or semicolon, the URI MUST be enclosed in angle brackets (< and >). Any URI parameters are contained
4050 within these brackets. If the URI is not enclosed in angle brackets, any semicolon-delimited parameters are
4051 header-parameters, not URI parameters.

4052 20.1 Accept

4053 The **Accept** header field follows the syntax defined in [H14.1]. The semantics are also identical, with
4054 the exception that if no **Accept** header field is present, the server SHOULD assume a default value of
4055 `application/sdp`.

4056 An empty **Accept** header field means that no formats are acceptable.

4057 Example:

4058 `Accept: application/sdp;level=1, application/x-private, text/html`

4059 20.2 Accept-Encoding

4060 The **Accept-Encoding** header field is similar to **Accept**, but restricts the content-codings [H3.5] that are
4061 acceptable in the response. See [H14.3]. The syntax of this header field is defined in [H14.3]. The semantics
4062 in SIP are identical to those defined in [H14.3].

4063 An empty **Accept-Encoding** header field is permissible, even though the syntax in [H14.3] does not
4064 provide for it. It is equivalent to **Accept-Encoding: identity**, that is, only the identity encoding, meaning
4065 no encoding, is permissible.

4066 If no **Accept-Encoding** header field is present, the server SHOULD assume a default value of **identity**.

4067 This differs slightly from the HTTP definition, which indicates that when not present, any encoding can
4068 be used, but the identity encoding is preferred.

4069 Example:

4070 `Accept-Encoding: gzip`

4071 20.3 Accept-Language

4072 The **Accept-Language** header field is used in requests to indicate the preferred languages for reason
4073 phrases, session descriptions, or status responses carried as message bodies in the response. If no **Accept-**

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG
Accept	R		-	o	-	o	m*	o
Accept	2xx		-	-	-	o	m*	o
Accept	415		-	c	-	c	c	c
Accept-Encoding	R		-	o	-	o	o	o
Accept-Encoding	2xx		-	-	-	o	m*	o
Accept-Encoding	415		-	c	-	c	c	c
Accept-Language	R		-	o	-	o	o	o
Accept-Language	2xx		-	-	-	o	m*	o
Accept-Language	415		-	c	-	c	c	c
Alert-Info	R	ar	-	-	-	o	-	-
Alert-Info	180	ar	-	-	-	o	-	-
Allow	R		-	o	-	o	o	o
Allow	2xx		-	o	-	m*	m*	o
Allow	r		-	o	-	o	o	o
Allow	405		-	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		ar	-	-	-	o	o	o
Contact	R		o	-	-	m	o	o
Contact	1xx		-	-	-	o	-	-
Contact	2xx		-	-	-	m	o	o
Contact	3xx	d	-	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		ar	t	t	t	t	t	t
Content-Type			*	*	-	*	*	*
CSeq	c	r	m	m	m	m	m	m
Date		a	o	o	o	o	o	o
Error-Info	300-699	a	-	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	amr	m	m	m	m	m	m
Min-Expires	423		-	-	-	-	-	m
MIME-Version			o	o	-	o	o	o
Organization		ar	-	-	-	o	o	o

Table 2: Summary of header fields, A–O

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG
Priority	R	ar	-	-	-	o	-	-
Proxy-Authenticate	407	ar	-	m	-	m	m	m
Proxy-Authenticate	401	ar	-	o	o	o	o	o
Proxy-Authorization	R	dr	o	o	-	o	o	o
Proxy-Require	R	ar	-	o	-	o	o	o
Record-Route	R	ar	o	o	o	o	o	-
Record-Route	2xx,18x	mr	-	o	o	o	o	-
Reply-To			-	-	-	o	-	-
Require		ar	-	c	-	c	c	c
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	adr	c	c	c	c	c	c
Server	r		-	o	o	o	o	o
Subject	R		-	-	-	o	-	-
Supported	R		-	o	o	m*	o	o
Supported	2xx		-	o	o	m*	m*	o
Timestamp			o	o	o	o	o	o
To	c(1)	r	m	m	m	m	m	m
Unsupported	420		-	m	-	m	m	m
User-Agent			o	o	o	o	o	o
Via	R	amr	m	m	m	m	m	m
Via	rc	dr	m	m	m	m	m	m
Warning	r		-	o	o	o	o	o
WWW-Authenticate	401	ar	-	m	-	m	m	m
WWW-Authenticate	407	ar	-	o	-	o	o	o

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

4074 **Language** header field is present, the server SHOULD assume all languages are acceptable to the client.

4075 The **Accept-Language** header field follows the syntax defined in [H14.4]. The rules for ordering the
4076 languages based on the “q” parameter apply to SIP as well.

4077 Example:

4078 `Accept-Language: da, en-gb;q=0.8, en;q=0.7`

4079 20.4 Alert-Info

4080 When present in an INVITE request, the **Alert-Info** header field specifies an alternative ring tone to the UAS.

4081 When present in a 180 (Ringing) response, the **Alert-Info** header field specifies an alternative ringback tone
4082 to the UAC. A typical usage is for a proxy to insert this header field to provide a distinctive ring feature.

4083 The **Alert-Info** header field can introduce security risks. These risks and the ways to handle them are
4084 discussed in Section 20.9, which discusses the **Call-Info** header field since the risks are identical.

4085 In addition, a user SHOULD be able to disable this feature selectively.

4086 This helps prevent disruptions that could result from the use of this header field by untrusted elements.

4087 Example:

4088 Alert-Info: <http://www.example.com/sounds/moo.wav>

4089 20.5 Allow

4090 The Allow header field lists the set of methods supported by the UA generating the message.

4091 All methods, including ACK and CANCEL, understood by the UA MUST be included in the list of
4092 methods in the Allow header field, when present. The absence of an Allow header field MUST NOT be
4093 interpreted to mean that the UA sending the message supports no methods. Rather, it implies that the UA is
4094 not providing any information on what methods it supports.

4095 Supplying an Allow header field in responses to methods other than OPTIONS reduces the number of
4096 messages needed.

4097 Example:

4098 Allow: INVITE, ACK, OPTIONS, CANCEL, BYE

4099 20.6 Authentication-Info

4100 The Authentication-Info header field provides for mutual authentication with HTTP Digest. A UAS MAY
4101 include this header field in a 2xx response to a request that was successfully authenticated using digest based
4102 on the Authorization header field.

4103 Syntax and semantics follow those specified in RFC 2617 [17].

4104 Example:

4105 Authentication-Info: nextnonce="47364c23432d2e131a5fb210812c"

4106 20.7 Authorization

4107 The Authorization header field contains authentication credentials of a UA. Section 22.2 overviews the use
4108 of the Authorization header field, and Section 22.4 describes the syntax and semantics when used with
4109 HTTP authentication.

4110 This header field, along with Proxy-Authorization, breaks the general rules about multiple header field
4111 values. Although not a comma-separated list, this header field name may be present multiple times, and
4112 MUST NOT be combined into a single header line using the usual rules described in Section 7.3.

4113 In the example below, there are no quotes around the Digest parameter:

4114 Authorization: Digest username="Alice", realm="atlanta.com",
4115 nonce="84a4cc6f3082121f32b42a2187831a9e",
4116 response="7587245234b3434cc3412213e5f113a5432"

4117 20.8 Call-ID

4118 The Call-ID header field uniquely identifies a particular invitation or all registrations of a particular client.
4119 A single multimedia conference can give rise to several calls with different Call-IDs, for example, if a user
4120 invites a single individual several times to the same (long-running) conference. Call-IDs are case-sensitive
4121 and are simply compared byte-by-byte.

4122 The compact form of the Call-ID header field is i.

4123 Examples:

4124 Call-ID: f81d4fae-7dec-11d0-a765-00a0c91e6bf6@biloxi.com

4125 i:f81d4fae-7dec-11d0-a765-00a0c91e6bf6@192.0.2.4

4126 20.9 Call-Info

4127 The Call-Info header field provides additional information about the caller or callee, depending on whether
4128 it is found in a request or response. The purpose of the URI is described by the “purpose” parameter.
4129 The “icon” parameter designates an image suitable as an iconic representation of the caller or callee. The
4130 “info” parameter describes the caller or callee in general, for example, through a web page. The “card”
4131 parameter provides a business card, for example, in vCard [36] or LDIF [37] formats. Additional tokens can
4132 be registered using IANA and the procedures in Section 27.

4133 Use of the Call-Info header field can pose a security risk. If a callee fetches the URIs provided by a
4134 malicious caller, the callee may be at risk for displaying inappropriate or offensive content, dangerous or
4135 illegal content, and so on. Therefore, it is RECOMMENDED that a UA only render the information in the
4136 Call-Info header field if it can verify the authenticity of the element that originated the header field and
4137 trusts that element. This need not be the peer UA; a proxy can insert this header field into requests.

4138 Example:

4139 Call-Info: <http://www.example.com/alice/photo.jpg> ;purpose=icon,
4140 <http://www.example.com/alice/> ;purpose=info

4141 20.10 Contact

4142 A Contact header field value provides a URI whose meaning depends on the type of request or response it
4143 is in.

4144 A Contact header field value can contain a display name, a URI with URI parameters, and header
4145 parameters.

4146 This document defines the Contact parameters “q” and “expires”. These parameters are only used
4147 when the Contact is present in a REGISTER request or response, or in a 3xx response. Additional param-
4148 eters may be defined in other specifications.

4149 When the header field value contains a display name, the URI including all URI parameters is enclosed
4150 in “<” and “>”. If no “<” and “>” are present, all parameters after the URI are header parameters, not URI
4151 parameters. The display name can be tokens, or a quoted string, if a larger character set is desired.

4152 Even if the “display-name” is empty, the “name-addr” form MUST be used if the “addr-spec” con-
4153 tains a comma, semicolon, or question mark. There may or may not be LWS between the display-name
4154 and the “<”.

4155 These rules for parsing a display name, URI and URI parameters, and header parameters also apply for

4156 the header fields **To** and **From**.

4157 The **Contact** header field has a role similar to the **Location** header field in HTTP. However, the HTTP header
4158 field only allows one address, unquoted. Since URIs can contain commas and semicolons as reserved characters,
4159 they can be mistaken for header or parameter delimiters, respectively.

4160 The compact form of the **Contact** header field is **m** (for “moved”).

4161 The second example below shows a **Contact** header field value containing both a URI parameter
4162 (**transport**) and a header parameter (**expires**).

```
4163 Contact: "Mr. Watson" <sip:watson@worchester.bell-telephone.com>  
4164         ;q=0.7; expires=3600,  
4165         "Mr. Watson" <mailto:watson@bell-telephone.com> ;q=0.1  
4166 m: <sips:bob@192.0.2.4>;expires=60
```

4167 **20.11 Content-Disposition**

4168 The **Content-Disposition** header field describes how the message body or, for multipart messages, a mes-
4169 sage body part is to be interpreted by the UAC or UAS. This SIP header field extends the MIME **Content-**
4170 **Type** (RFC 2183 [18]).

4171 Several new “**disposition-types**” of the **Content-Disposition** header are defined by SIP. The value
4172 “**session**” indicates that the body part describes a session, for either calls or early (pre-call) media. The
4173 value “**render**” indicates that the body part should be displayed or otherwise rendered to the user. Note that
4174 the value “**render**” is used rather than “**inline**” to avoid the connotation that the MIME body is displayed
4175 as a part of the rendering of the entire message (since the MIME bodies of SIP messages oftentimes are
4176 not displayed to users). For backward-compatibility, if the **Content-Disposition** header field is missing,
4177 the server **SHOULD** assume bodies of **Content-Type** `application/sdp` are the disposition “**session**”,
4178 while other content types are “**render**”.

4179 The disposition type “**icon**” indicates that the body part contains an image suitable as an iconic rep-
4180 resentation of the caller or callee that could be rendered informationally by a user agent when a message
4181 has been received, or persistently while a dialog takes place. The value “**alert**” indicates that the body part
4182 contains information, such as an audio clip, that should be rendered by the user agent in an attempt to alert
4183 the user to the receipt of a request, generally a request that initiates a dialog; this alerting body could for
4184 example be rendered as a ring tone for a phone call after a 180 Ringing provisional response has been sent.

4185 Any MIME body with a “**disposition-type**” that renders content to the user should only be processed
4186 when a message has been properly authenticated.

4187 The handling parameter, **handling-param**, describes how the UAS should react if it receives a message
4188 body whose content type or disposition type it does not understand. The parameter has defined values
4189 of “**optional**” and “**required**”. If the handling parameter is missing, the value “**required**” **SHOULD** be
4190 assumed. The handling parameter is described in RFC 3204 [19].

4191 If this header field is missing, the MIME type determines the default content disposition. If there is
4192 none, “**render**” is assumed.

4193 Example:

```
4194 Content-Disposition: session
```

4195 **20.12 Content-Encoding**

4196 The Content-Encoding header field is used as a modifier to the “media-type”. When present, its value
4197 indicates what additional content codings have been applied to the entity-body, and thus what decoding
4198 mechanisms MUST be applied in order to obtain the media-type referenced by the Content-Type header
4199 field. Content-Encoding is primarily used to allow a body to be compressed without losing the identity of
4200 its underlying media type.

4201 If multiple encodings have been applied to an entity-body, the content codings MUST be listed in the
4202 order in which they were applied.

4203 All content-coding values are case-insensitive. IANA acts as a registry for content-coding value tokens.
4204 See [H3.5] for a definition of the syntax for content-coding.

4205 Clients MAY apply content encodings to the body in requests. A server MAY apply content encodings to
4206 the bodies in responses. The server MUST only use encodings listed in the Accept-Encoding header field
4207 in the request.

4208 The compact form of the Content-Encoding header field is e. Examples:

```
4209 Content-Encoding: gzip  
4210 e: tar
```

4211 **20.13 Content-Language**

4212 See [H14.12]. Example:

```
4213 Content-Language: fr
```

4214 **20.14 Content-Length**

4215 The Content-Length header field indicates the size of the message-body, in decimal number of octets,
4216 sent to the recipient. Applications SHOULD use this field to indicate the size of the message-body to be
4217 transferred, regardless of the media type of the entity. If a stream-based protocol (such as TCP) is used as
4218 transport, the header field MUST be used.

4219 The size of the message-body does *not* include the CRLF separating header fields and body. Any
4220 Content-Length greater than or equal to zero is a valid value. If no body is present in a message, then
4221 the Content-Length header field value MUST be set to zero.

4222 The ability to omit Content-Length simplifies the creation of cgi-like scripts that dynamically generate re-
4223 sponses.

4224 The compact form of the header field is l.

4225 Examples:

```
4226 Content-Length: 349  
4227 l: 173
```

4228 **20.15 Content-Type**

4229 The Content-Type header field indicates the media type of the message-body sent to the recipient. The
4230 “media-type” element is defined in [H3.7]. The Content-Type header field MUST be present if the body is

4231 not empty. If the body is empty, and a **Content-Type** header field is present, it indicates that the body of the
4232 specific type has zero length (for example, an empty audio file).

4233 The compact form of the header field is **c**.

4234 Examples:

```
4235 Content-Type: application/sdp  
4236 c: text/html; charset=ISO-8859-4
```

4237 20.16 CSeq

4238 A **CSeq** header field in a request contains a single decimal sequence number and the request method.
4239 The sequence number **MUST** be expressible as a 32-bit unsigned integer. The method part of **CSeq** is
4240 case-sensitive. The **CSeq** header field serves to order transactions within a dialog, to provide a means
4241 to uniquely identify transactions, and to differentiate between new requests and request retransmissions.
4242 Two **CSeq** header fields are considered equal if the sequence number and the request method are identical.

4243 Example:

```
4244 CSeq: 4711 INVITE
```

4245 20.17 Date

4246 The **Date** header field contains a the date and time. Unlike HTTP/1.1, SIP only supports the most recent
4247 RFC 1123 [20] format for dates. As in [H3.3], SIP restricts the time zone in **SIP-date** to "GMT", while
4248 RFC 1123 allows any time zone. **rfc1123-date** is case-sensitive.

4249 The **Date** header field reflects the time when the request or response is first sent.

4250 The **Date** header field can be used by simple end systems without a battery-backed clock to acquire a notion of
4251 current time. However, in its GMT form, it requires clients to know their offset from GMT.

4252 Example:

```
4253 Date: Sat, 13 Nov 2010 23:29:00 GMT
```

4254 20.18 Error-Info

4255 The **Error-Info** header field provides a pointer to additional information about the error status response.

4256 SIP UACs have user interface capabilities ranging from pop-up windows and audio on PC softclients to audio-
4257 only on "black" phones or endpoints connected via gateways. Rather than forcing a server generating an error to
4258 choose between sending an error status code with a detailed reason phrase and playing an audio recording, the
4259 **Error-Info** header field allows both to be sent. The UAC then has the choice of which error indicator to render to the
4260 caller.

4261 A UAC **MAY** treat a SIP or SIPS URI in an **Error-Info** header field as if it were a **Contact** in a redirect
4262 and generate a new **INVITE**, resulting in a recorded announcement session being established. A non-SIP
4263 URI **MAY** be rendered to the user.

4264 Examples:

```
4265 SIP/2.0 404 The number you have dialed is not in service  
4266 Error-Info: <sip:not-in-service-recording@atlanta.com>
```

4267 20.19 Expires

4268 The Expires header field gives the relative time after which the message (or content) expires.

4269 The precise meaning of this is method dependent.

4270 The expiration time in an INVITE does *not* affect the duration of the actual session that may result
4271 from the invitation. Session description protocols may offer the ability to express time limits on the session
4272 duration, however.

4273 The value of this field is an integral number of seconds (in decimal) between 0 and $(2^{31})-1$, measured
4274 from the receipt of the request.

4275 Example:

4276 Expires: 5

4277 20.20 From

4278 The From header field indicates the initiator of the request. This may be different from the initiator of the
4279 dialog. Requests sent by the callee to the caller use the callee's address in the From header field.

4280 The optional "display-name" is meant to be rendered by a human user interface. A system SHOULD use
4281 the display name "Anonymous" if the identity of the client is to remain hidden. Even if the "display-name"
4282 is empty, the "name-addr" form MUST be used if the "addr-spec" contains a comma, question mark, or
4283 semicolon. Syntax issues are discussed in Section 7.3.1.

4284 Section 12 describes how From header fields are compared for the purpose of matching requests to
4285 dialogs. See Section 20.10 for the rules for parsing a display name, URI and URI parameters, and header
4286 field parameters.

4287 The compact form of the From header field is f.

4288 Examples:

4289 From: "A. G. Bell" <sip:agb@bell-telephone.com> ;tag=a48s

4290 From: sip:+12125551212@server.phone2net.com;tag=887s

4291 f: Anonymous <sip:c8oqz84zk7z@privacy.org>;tag=hyh8

4292 20.21 In-Reply-To

4293 The In-Reply-To header field enumerates the Call-IDs that this call references or returns. These Call-IDs
4294 may have been cached by the client then included in this header field in a return call.

4295 This allows automatic call distribution systems to route return calls to the originator of the first call. This also
4296 allows callees to filter calls, so that only return calls for calls they originated will be accepted. This field is not a
4297 substitute for request authentication.

4298 Example:

4299 In-Reply-To: 70710@saturn.bell-tel.com, 17320@saturn.bell-tel.com

4300 20.22 Max-Forwards

4301 The Max-Forwards header field must be used with any SIP method to limit the number of proxies or
4302 gateways that can forward the request to the next downstream server. This can also be useful when the client
4303 is attempting to trace a request chain that appears to be failing or looping in mid-chain.

4304 The **Max-Forwards** value is an integer in the range 0-255 indicating the remaining number of times this
4305 request message is allowed to be forwarded. This count is decremented by each server that forwards the
4306 request. The recommended value is 70.

4307 This header field should be inserted by elements that can not otherwise guarantee loop detection. For
4308 example, a B2BUA should insert a **Max-Forwards** header field.

4309 Example:

4310 `Max-Forwards: 6`

4311 **20.23 Min-Expires**

4312 The **Min-Expires** header field conveys the minimum refresh interval supported for soft-state elements man-
4313 aged by that server. This includes **Contact** header fields that are stored by a registrar. The header field
4314 contains a decimal integer number of seconds from 0 to $(2^{32})-1$. The use of the header field in a 423
4315 (Registration Too Brief) response is described in Sections 10.2.8, 10.3, and 21.4.17.

4316 Example:

4317 `Min-Expires: 60`

4318 **20.24 MIME-Version**

4319 See [H19.4.1].

4320 Example:

4321 `MIME-Version: 1.0`

4322 **20.25 Organization**

4323 The **Organization** header field conveys the name of the organization to which the SIP element issuing the
4324 request or response belongs.

4325 The field MAY be used by client software to filter calls.

4326 Example:

4327 `Organization: Boxes by Bob`

4328 **20.26 Priority**

4329 The **Priority** header field indicates the urgency of the request as perceived by the client. The **Priority** header
4330 field describes the priority that the SIP request should have to the receiving human or its agent. For example,
4331 it may be factored into decisions about call routing and acceptance. For these decisions, a message contain-
4332 ing no **Priority** header field SHOULD be treated as if it specified a **Priority** of “normal”. The **Priority** header
4333 field does not influence the use of communications resources such as packet forwarding priority in routers or
4334 access to circuits in PSTN gateways. The header field can have the values “non-urgent”, “normal”, “urgent”,
4335 and “emergency”, but additional values can be defined elsewhere. It is RECOMMENDED that the value of
4336 “emergency” only be used when life, limb, or property are in imminent danger. Otherwise, there are no
4337 semantics defined for this header field.

4338 These are the values of RFC 2076 [38], with the addition of "emergency".

4339 Examples:

4340 Subject: A tornado is heading our way!

4341 Priority: emergency

4342 or

4343 Subject: Weekend plans

4344 Priority: non-urgent

4345 **20.27 Proxy-Authenticate**

4346 A Proxy-Authenticate header field value contains an authentication challenge.

4347 The syntax for this header field and its use is defined in [H14.33]. See 22.3 for further details on its
4348 usage.

4349 Example:

4350 Proxy-Authenticate: Digest realm="atlanta.com",

4351 domain="sip:ss1.carrier.com",

4352 nonce="f84f1cec41e6cbe5aea9c8e88d359",

4353 opaque="", stale=FALSE, algorithm=MD5

4354 **20.28 Proxy-Authorization**

4355 The Proxy-Authorization header field allows the client to identify itself (or its user) to a proxy that requires
4356 authentication. A Proxy-Authorization field value consists of credentials containing the authentication
4357 information of the user agent for the proxy and/or realm of the resource being requested.

4358 See [H14.34] for a definition of the syntax, and section 22.3 for a discussion of its usage.

4359 This header field, along with Authorization, breaks the general rules about multiple header field names.
4360 Although not a comma-separated list, this header field name may be present multiple times, and MUST NOT
4361 be combined into a single header line using the usual rules described in Section 7.3.1.

4362 Example:

4363 Proxy-Authorization: Digest username="Alice", realm="atlanta.com",

4364 nonce="c60f3082ee1212b402a21831ae",

4365 response="245f23415f11432b3434341c022"

4366 **20.29 Proxy-Require**

4367 The Proxy-Require header field is used to indicate proxy-sensitive features that must be supported by the
4368 proxy. See Section 20.32 for more details on the mechanics of this message and a usage example.

4369 Example:

4370 Proxy-Require: foo

4371 **20.30 Record-Route**

4372 The Record-Route header field is inserted by proxies in a request to force future requests in the dialog to
4373 be routed through the proxy.

4374 Examples of its use with the Route header field are described in Sections 16.12.1.

4375 Example:

4376 Record-Route: <sip:server10.biloxi.com;lr>, <sip:bigbox3.site3.atlanta.com;lr>

4377 **20.31 Reply-To**

4378 The Reply-To header field contains a logical return URI that may be different from the From header field.
4379 For example, the URI MAY be used to return missed calls or unestablished sessions. If the user wished to
4380 remain anonymous, the header field SHOULD either be omitted from the request or populated in such a way
4381 that does not reveal any private information.

4382 Even if the “display-name” is empty, the “name-addr” form MUST be used if the “addr-spec” con-
4383 tains a comma, question mark, or semicolon. Syntax issues are discussed in Section 7.3.1.

4384 Example:

4385 Reply-To: Bob <sip:bob@biloxi.com>

4386 **20.32 Require**

4387 The Require header field is used by UACs to tell UASs about options that the UAC expects the UAS to
4388 support in order to process the request. Although an optional header field, the Require MUST NOT be
4389 ignored if it is present.

4390 The Require header field contains a list of option tags, described in Section 19.2. Each option tag
4391 defines a SIP extension that MUST be understood to process the request. Frequently, this is used to indicate
4392 that a specific set of extension header fields need to be understood. A UAC compliant to this specification
4393 MUST only include option tags corresponding to standards-track RFCs.

4394 Example:

4395 Require: 100rel

4396 **20.33 Retry-After**

4397 The Retry-After header field can be used with a 503 (Service Unavailable) response to indicate how long
4398 the service is expected to be unavailable to the requesting client and with a 404 (Not Found), 413 (Request
4399 Entity Too Large), 480 (Temporarily Unavailable), 486 (Busy Here), 600 (Busy), or 603 (Decline) response
4400 to indicate when the called party anticipates being available again. The value of this field is a positive integer
4401 number of seconds (in decimal) after the time of the response.

4402 An optional comment can be used to indicate additional information about the time of callback. An
4403 optional “duration” parameter indicates how long the called party will be reachable starting at the initial
4404 time of availability. If no duration parameter is given, the service is assumed to be available indefinitely.

4405 Examples:

4406 Retry-After: 18000;duration=3600

4407 Retry-After: 120 (I'm in a meeting)

4408 **20.34 Route**

4409 The Route header field is used to force routing for a request through the listed set of proxies. Examples of
4410 the use of the Record-Route header field are in Section 16.12.1.

4411 Example:

4412 Route: <sip:bigbox3.site3.atlanta.com;lr>, <sip:server10.biloxi.com;lr>

4413 **20.35 Server**

4414 The Server header field contains information about the software used by the UAS to handle the request.
4415 The syntax for this field is defined in [H14.38].

4416 Revealing the specific software version of the server might allow the server to become more vulnerable
4417 to attacks against software that is known to contain security holes. Implementers SHOULD make the Server
4418 header field a configurable option.

4419 Example:

4420 Server: HomeProxy v2

4421 **20.36 Subject**

4422 The Subject header field provides a summary or indicates the nature of the call, allowing call filtering
4423 without having to parse the session description. The session description does not have to use the same
4424 subject indication as the invitation.

4425 The compact form of the Subject header field is s.

4426 Example:

4427 Subject: Need more boxes

4428 s: Tech Support

4429 **20.37 Supported**

4430 The Supported header field enumerates all the extensions supported by the UAC or UAS.

4431 The Supported header field contains a list of option tags, described in Section 19.2, that are understood
4432 by the UAC or UAS. A UA compliant to this specification MUST only include option tags corresponding to
4433 standards-track RFCs. If empty, it means that no extensions are supported.

4434 Example:

4435 Supported: 100rel

4436 **20.38 Timestamp**

4437 The Timestamp header field describes when the UAC sent the request to the UAS.

4438 See Section 8.2.6 for details on how to generate a response to a request that contains the header field.
4439 Although there is no normative behavior defined here that makes use of the header, it allows for extensions
4440 or SIP applications to obtain RTT estimates.

4441 Example:

4442 `Timestamp: 54`

4443 **20.39 To**

4444 The **To** header field specifies the logical recipient of the request.

4445 The optional “display-name” is meant to be rendered by a human-user interface. The “tag” parameter
4446 serves as a general mechanism for dialog identification.

4447 See Section 19.3 for details of the “tag” parameter.

4448 Section 12 describes how **To** and **From** header fields are compared for the purpose of matching requests
4449 to dialogs. See Section 20.10 for the rules for parsing a display name, URI and URI parameters, and header
4450 field parameters.

4451 The compact form of the **To** header field is **t**.

4452 The following are examples of valid **To** header fields:

4453 `To: The Operator <sip:operator@cs.columbia.edu>;tag=287447`
4454 `t: sip:+12125551212@server.phone2net.com`

4455 **20.40 Unsupported**

4456 The **Unsupported** header field lists the features not supported by the UAS. See Section 20.32 for motivation.

4457 Example:

4458 `Unsupported: foo`

4459 **20.41 User-Agent**

4460 The **User-Agent** header field contains information about the UAC originating the request. The syntax and
4461 semantics are defined in [H14.43].

4462 Revealing the specific software version of the user agent might allow the user agent to become more
4463 vulnerable to attacks against software that is known to contain security holes. Implementers SHOULD make
4464 the **User-Agent** header field a configurable option.

4465 Example:

4466 `User-Agent: Softphone Beta1.5`

4467 **20.42 Via**

4468 The **Via** header field indicates the path taken by the request so far and indicates the path that should be
4469 followed in routing responses. The branch ID parameter in the **Via** header field values serves as a transaction
4470 identifier, and is used by proxies to detect loops.

4471 A **Via** header field value contains the transport protocol used to send the message, the client’s host name
4472 or network address, and possibly the port number at which it wishes to receive responses. A **Via** header

4473 field value can also contain parameters such as “maddr”, “ttl”, “received”, and “branch”, whose meaning
4474 and use are described in other sections. For implementations compliant to this specification, the value of the
4475 **branch** parameter **MUST** start with the magic cookie “z9hG4bK”, as discussed in Section 8.1.1.7.

4476 Transport protocols defined here are “UDP”, “TCP”, “TLS”, and “SCTP”. “TLS” means TLS over
4477 TCP. When a request is sent to a SIPS URI, the protocol still indicates “SIP”, and the transport protocol is
4478 TLS.

```
4479 Via: SIP/2.0/UDP erlang.bell-telephone.com:5060;branch=z9hG4bK87asdk7  
4480 Via: SIP/2.0/UDP 192.0.2.1:5060 ;received=192.0.2.207;branch=z9hG4bK77asjd
```

4481 The compact form of the *Via* header field is *v*.

4482 In this example, the message originated from a multi-homed host with two addresses, 192.0.2.1 and
4483 192.0.2.207. The sender guessed wrong as to which network interface would be used. Erlang.bell-telephone.com
4484 noticed the mismatch and added a parameter to the previous hop’s *Via* header field value, containing the ad-
4485 dress that the packet actually came from.

4486 The host or network address and port number are not required to follow the SIP URI syntax. Specifically,
4487 LWS on either side of the “:” or “/” is allowed, as shown here:

```
4488 Via: SIP / 2.0 / UDP first.example.com: 4000;ttl=16  
4489 ;maddr=224.2.0.1 ;branch=z9hG4bKa7c6a8dlze.1
```

4490 Even though this specification mandates that the *branch* parameter be present in all requests, the BNF
4491 for the header field indicates that it is optional. This allows interoperation with RFC 2543 elements, which
4492 did not have to insert the *branch* parameter.

4493 20.43 Warning

4494 The **Warning** header field is used to carry additional information about the status of a response. **Warning**
4495 header field values are sent with responses and contain a three-digit warning code, host name, and warning
4496 text.

4497 The “**warn-text**” should be in a natural language that is most likely to be intelligible to the human user
4498 receiving the response. This decision can be based on any available knowledge, such as the location of the
4499 user, the **Accept-Language** field in a request, or the **Content-Language** field in a response. The default
4500 language is *i-default* [21].

4501 The currently-defined “**warn-code**”s are listed below, with a recommended **warn-text** in English and a
4502 description of their meaning. These warnings describe failures induced by the session description. The first
4503 digit of warning codes beginning with “3” indicates warnings specific to SIP. Warnings 300 through 329 are
4504 reserved for indicating problems with keywords in the session description, 330 through 339 are warnings
4505 related to basic network services requested in the session description, 370 through 379 are warnings related
4506 to quantitative QoS parameters requested in the session description, and 390 through 399 are miscellaneous
4507 warnings that do not fall into one of the above categories.

4508 **300 Incompatible network protocol:** One or more network protocols contained in the session description
4509 are not available.

4510 **301 Incompatible network address formats:** One or more network address formats contained in the ses-
4511 sion description are not available.

4512 **302 Incompatible transport protocol:** One or more transport protocols described in the session descrip-
4513 tion are not available.

4514 **303 Incompatible bandwidth units:** One or more bandwidth measurement units contained in the session
4515 description were not understood.

4516 **304 Media type not available:** One or more media types contained in the session description are not avail-
4517 able.

4518 **305 Incompatible media format:** One or more media formats contained in the session description are not
4519 available.

4520 **306 Attribute not understood:** One or more of the media attributes in the session description are not sup-
4521 ported.

4522 **307 Session description parameter not understood:** A parameter other than those listed above was not
4523 understood.

4524 **330 Multicast not available:** The site where the user is located does not support multicast.

4525 **331 Unicast not available:** The site where the user is located does not support unicast communication (usu-
4526 ally due to the presence of a firewall).

4527 **370 Insufficient bandwidth:** The bandwidth specified in the session description or defined by the media
4528 exceeds that known to be available.

4529 **399 Miscellaneous warning:** The warning text can include arbitrary information to be presented to a hu-
4530 man user or logged. A system receiving this warning **MUST NOT** take any automated action.

4531 1xx and 2xx have been taken by HTTP/1.1.

4532 Additional "warn-code"s can be defined through IANA, as defined in Section 27.2.

4533 Examples:

4534 Warning: 307 isi.edu "Session parameter 'foo' not understood"

4535 Warning: 301 isi.edu "Incompatible network address type 'E.164'"

4536 20.44 WWW-Authenticate

4537 A WWW-Authenticate header field value contains an authentication challenge. The syntax for this header
4538 field and use is defined in [H14.47]. See 22.2 for further details on its usage.

4539 Example:

```
4540 WWW-Authenticate: Digest realm="atlanta.com",  
4541 domain="sip:boxesbybob.com",  
4542 nonce="f84f1cec41e6cbe5aea9c8e88d359",  
4543 opaque="", stale=FALSE, algorithm=MD5
```

4544 **21 Response Codes**

4545 The response codes are consistent with, and extend, HTTP/1.1 response codes. Not all HTTP/1.1 response
4546 codes are appropriate, and only those that are appropriate are given here. Other HTTP/1.1 response codes
4547 SHOULD NOT be used. Also, SIP defines a new class, 6xx.

4548 **21.1 Provisional 1xx**

4549 Provisional responses, also known as informational responses, indicate that the server contacted is perform-
4550 ing some further action and does not yet have a definitive response. A server sends a 1xx response if it
4551 expects to take more than 200 ms to obtain a final response. Note that 1xx responses are not transmitted
4552 reliably. They never cause the client to send an ACK. Provisional (1xx) responses MAY contain message
4553 bodies, including session descriptions.

4554 **21.1.1 100 Trying**

4555 This response indicates that the request has been received by the next-hop server and that some unspecified
4556 action is being taken on behalf of this call (for example, a database is being consulted). This response, like
4557 all other provisional responses, stops retransmissions of an INVITE by a UAC. The 100 (Trying) response
4558 is different from other provisional responses, in that it is never forwarded upstream by a stateful proxy.

4559 **21.1.2 180 Ringing**

4560 The UA receiving the INVITE is trying to alert the user. This response MAY be used to initiate local ringback.

4561 **21.1.3 181 Call Is Being Forwarded**

4562 A server MAY use this status code to indicate that the call is being forwarded to a different set of destinations.

4563 **21.1.4 182 Queued**

4564 The called party is temporarily unavailable, but the server has decided to queue the call rather than reject it.
4565 When the callee becomes available, it will return the appropriate final status response. The reason phrase
4566 MAY give further details about the status of the call, for example, "5 calls queued; expected waiting time is
4567 15 minutes". The server MAY issue several 182 (Queued) responses to update the caller about the status of
4568 the queued call.

4569 **21.1.5 183 Session Progress**

4570 The 183 (Session Progress) response is used to convey information about the progress of the call that is not
4571 otherwise classified. The Reason-Phrase, header fields, or message body MAY be used to convey more
4572 details about the call progress.

4573 **21.2 Successful 2xx**

4574 The request was successful.

4575 **21.2.1 200 OK**

4576 The request has succeeded. The information returned with the response depends on the method used in the
4577 request.

4578 **21.3 Redirection 3xx**

4579 3xx responses give information about the user's new location, or about alternative services that might be
4580 able to satisfy the call.

4581 **21.3.1 300 Multiple Choices**

4582 The address in the request resolved to several choices, each with its own specific location, and the user (or
4583 UA) can select a preferred communication end point and redirect its request to that location.

4584 The response MAY include a message body containing a list of resource characteristics and location(s)
4585 from which the user or UA can choose the one most appropriate, if allowed by the **Accept** request header
4586 field. However, no MIME types have been defined for this message body.

4587 The choices SHOULD also be listed as **Contact** fields (Section 20.10). Unlike HTTP, the SIP response
4588 MAY contain several **Contact** fields or a list of addresses in a **Contact** field. UAs MAY use the **Contact**
4589 header field value for automatic redirection or MAY ask the user to confirm a choice. However, this specifi-
4590 cation does not define any standard for such automatic selection.

4591 This status response is appropriate if the callee can be reached at several different locations and the server cannot
4592 or prefers not to proxy the request.

4593 **21.3.2 301 Moved Permanently**

4594 The user can no longer be found at the address in the **Request-URI**, and the requesting client SHOULD retry
4595 at the new address given by the **Contact** header field (Section 20.10). The requestor SHOULD update any
4596 local directories, address books, and user location caches with this new value and redirect future requests to
4597 the address(es) listed.

4598 **21.3.3 302 Moved Temporarily**

4599 The requesting client SHOULD retry the request at the new address(es) given by the **Contact** header field
4600 (Section 20.10). The **Request-URI** of the new request uses the value of the **Contact** header field in the
4601 response.

4602 The duration of the validity of the **Contact** URI can be indicated through an **Expires** (Section 20.19)
4603 header field or an **expires** parameter in the **Contact** header field. Both proxies and UAs MAY cache this
4604 URI for the duration of the expiration time. If there is no explicit expiration time, the address is only valid
4605 once for recursing, and MUST NOT be cached for future transactions.

4606 If the URI cached from the **Contact** header field fails, the **Request-URI** from the redirected request
4607 MAY be tried again a single time.

4608 The temporary URI may have become out-of-date sooner than the expiration time, and a new temporary URI
4609 may be available.

4610 **21.3.4 305 Use Proxy**

4611 The requested resource **MUST** be accessed through the proxy given by the **Contact** field. The **Contact** field
4612 gives the URI of the proxy. The recipient is expected to repeat this single request via the proxy. 305 (Use
4613 Proxy) responses **MUST** only be generated by UASs.

4614 **21.3.5 380 Alternative Service**

4615 The call was not successful, but alternative services are possible. The alternative services are described in
4616 the message body of the response. Formats for such bodies are not defined here, and may be the subject of
4617 future standardization.

4618 **21.4 Request Failure 4xx**

4619 4xx responses are definite failure responses from a particular server. The client **SHOULD NOT** retry the same
4620 request without modification (for example, adding appropriate authorization). However, the same request to
4621 a different server might be successful.

4622 **21.4.1 400 Bad Request**

4623 The request could not be understood due to malformed syntax. The **Reason-Phrase** **SHOULD** identify the
4624 syntax problem in more detail, for example, "Missing Call-ID header field".

4625 **21.4.2 401 Unauthorized**

4626 The request requires user authentication. This response is issued by UASs and registrars, while 407 (Proxy
4627 Authentication Required) is used by proxy servers.

4628 **21.4.3 402 Payment Required**

4629 Reserved for future use.

4630 **21.4.4 403 Forbidden**

4631 The server understood the request, but is refusing to fulfill it. Authorization will not help, and the request
4632 **SHOULD NOT** be repeated.

4633 **21.4.5 404 Not Found**

4634 The server has definitive information that the user does not exist at the domain specified in the **Request-**
4635 **URI**. This status is also returned if the domain in the **Request-URI** does not match any of the domains
4636 handled by the recipient of the request.

4637 **21.4.6 405 Method Not Allowed**

4638 The method specified in the **Request-Line** is understood, but not allowed for the address identified by the
4639 **Request-URI**.

4640 The response **MUST** include an **Allow** header field containing a list of valid methods for the indicated
4641 address.

4642 **21.4.7 406 Not Acceptable**

4643 The resource identified by the request is only capable of generating response entities that have content
4644 characteristics not acceptable according to the **Accept** header field sent in the request.

4645 **21.4.8 407 Proxy Authentication Required**

4646 This code is similar to 401 (Unauthorized), but indicates that the client **MUST** first authenticate itself with
4647 the proxy. SIP access authentication is explained in Sections 26 and 22.3.

4648 This status code can be used for applications where access to the communication channel (for example,
4649 a telephony gateway) rather than the callee requires authentication.

4650 **21.4.9 408 Request Timeout**

4651 The server could not produce a response within a suitable amount of time, for example, if it could not
4652 determine the location of the user in time. The client **MAY** repeat the request without modifications at any
4653 later time.

4654 **21.4.10 410 Gone**

4655 The requested resource is no longer available at the server and no forwarding address is known. This
4656 condition is expected to be considered permanent. If the server does not know, or has no facility to determine,
4657 whether or not the condition is permanent, the status code 404 (Not Found) **SHOULD** be used instead.

4658 **21.4.11 413 Request Entity Too Large**

4659 The server is refusing to process a request because the request entity-body is larger than the server is willing
4660 or able to process. The server **MAY** close the connection to prevent the client from continuing the request.

4661 If the condition is temporary, the server **SHOULD** include a **Retry-After** header field to indicate that it is
4662 temporary and after what time the client **MAY** try again.

4663 **21.4.12 414 Request-URI Too Long**

4664 The server is refusing to service the request because the **Request-URI** is longer than the server is willing to
4665 interpret.

4666 **21.4.13 415 Unsupported Media Type**

4667 The server is refusing to service the request because the message body of the request is in a format not
4668 supported by the server for the requested method. The server **MUST** return a list of acceptable formats using
4669 the **Accept**, **Accept-Encoding** or **Accept-Language** header field, depending on the specific problem with
4670 the content. UAC processing of this response is described in Section 8.1.3.5.

4671 **21.4.14 416 Unsupported URI Scheme**

4672 The server cannot process the request because the scheme of the URI in the Request-URI is unknown to
4673 the server. Client processing of this response is described in Section 8.1.3.5.

4674 **21.4.15 420 Bad Extension**

4675 The server did not understand the protocol extension specified in a Proxy-Require (Section 20.29) or Re-
4676 quire (Section 20.32) header field. The server SHOULD include a list of the unsupported extensions in an
4677 Unsupported header field in the response. UAC processing of this response is described in Section 8.1.3.5.

4678 **21.4.16 421 Extension Required**

4679 The UAS needs a particular extension to process the request, but this extension is not listed in a Supported
4680 header field in the request. Responses with this status code MUST contain a Require header field listing the
4681 required extensions.

4682 A UAS SHOULD NOT use this response unless it truly cannot provide any useful service to the client.
4683 Instead, if a desirable extension is not listed in the Supported header field, servers SHOULD process the
4684 request using baseline SIP capabilities and any extensions supported by the client.

4685 **21.4.17 423 Interval Too Brief**

4686 The server is rejecting the request because the expiration time of the resource refreshed by the request is too
4687 short. This response can be used by a registrar to reject a registration whose Contact header field expiration
4688 time was too small. The use of this response and the related Min-Expires header field are described in
4689 Sections 10.2.8, 10.3, and 20.23.

4690 **21.4.18 480 Temporarily Unavailable**

4691 The callee's end system was contacted successfully but the callee is currently unavailable (for example, is
4692 not logged in, logged in but in a state that precludes communication with the callee, or has activated the "do
4693 not disturb" feature). The response MAY indicate a better time to call in the Retry-After header field. The
4694 user could also be available elsewhere (unbeknownst to this server). The reason phrase SHOULD indicate a
4695 more precise cause as to why the callee is unavailable. This value SHOULD be settable by the UA. Status
4696 486 (Busy Here) MAY be used to more precisely indicate a particular reason for the call failure.

4697 This status is also returned by a redirect or proxy server that recognizes the user identified by the
4698 Request-URI, but does not currently have a valid forwarding location for that user.

4699 **21.4.19 481 Call/Transaction Does Not Exist**

4700 This status indicates that the UAS received a request that does not match any existing dialog or transaction.

4701 **21.4.20 482 Loop Detected**

4702 The server has detected a loop (Section 16.3 Item 4).

4703 **21.4.21 483 Too Many Hops**

4704 The server received a request that contains a **Max-Forwards** (Section 20.22) header field with the value
4705 zero.

4706 **21.4.22 484 Address Incomplete**

4707 The server received a request with a **Request-URI** that was incomplete. Additional information **SHOULD**
4708 be provided in the reason phrase.

4709 This status code allows overlapped dialing. With overlapped dialing, the client does not know the length of the
4710 dialing string. It sends strings of increasing lengths, prompting the user for more input, until it no longer receives a
4711 484 (Address Incomplete) status response.

4712 **21.4.23 485 Ambiguous**

4713 The **Request-URI** was ambiguous. The response **MAY** contain a listing of possible unambiguous addresses
4714 in **Contact** header fields. Revealing alternatives can infringe on privacy of the user or the organization. It
4715 **MUST** be possible to configure a server to respond with status 404 (Not Found) or to suppress the listing of
4716 possible choices for ambiguous **Request-URIs**.

4717 Example response to a request with the **Request-URI** `sip:lee@example.com`:

```
4718 SIP/2.0 485 Ambiguous
4719 Contact: Carol Lee <sip:carol.lee@example.com>
4720 Contact: Ping Lee <sip:p.lee@example.com>
4721 Contact: Lee M. Foote <sips:lee.foote@example.com>
```

4722 Some email and voice mail systems provide this functionality. A status code separate from 3xx is used since
4723 the semantics are different: for 300, it is assumed that the same person or service will be reached by the choices
4724 provided. While an automated choice or sequential search makes sense for a 3xx response, user intervention is
4725 required for a 485 (Ambiguous) response.

4726 **21.4.24 486 Busy Here**

4727 The callee's end system was contacted successfully, but the callee is currently not willing or able to take
4728 additional calls at this end system. The response **MAY** indicate a better time to call in the **Retry-After** header
4729 field. The user could also be available elsewhere, such as through a voice mail service. Status 600 (Busy
4730 Everywhere) **SHOULD** be used if the client knows that no other end system will be able to accept this call.

4731 **21.4.25 487 Request Terminated**

4732 The request was terminated by a **BYE** or **CANCEL** request. This response is never returned for a **CANCEL**
4733 request itself.

4734 **21.4.26 488 Not Acceptable Here**

4735 The response has the same meaning as 606 (Not Acceptable), but only applies to the specific resource
4736 addressed by the **Request-URI** and the request may succeed elsewhere.

4737 A message body containing a description of media capabilities MAY be present in the response, which is
4738 formatted according to the Accept header field in the INVITE (or application/sdp if not present), the same
4739 as a message body in a 200 (OK) response to an OPTIONS request.

4740 **21.4.27 491 Request Pending**

4741 The request was received by a UAS that had a pending request within the same dialog. Section 14.2 describes
4742 how such “glare” situations are resolved.

4743 **21.4.28 493 Undecipherable**

4744 The request was received by a UAS that contained an encrypted MIME body for which the recipient does not
4745 possess or will not provide an appropriate decryption key. This response MAY have a single body containing
4746 an appropriate public key that should be used to encrypt MIME bodies sent to this UA. Details of the usage
4747 of this response code can be found in Section 23.2.

4748 **21.5 Server Failure 5xx**

4749 5xx responses are failure responses given when a server itself has erred.

4750 **21.5.1 500 Server Internal Error**

4751 The server encountered an unexpected condition that prevented it from fulfilling the request. The client MAY
4752 display the specific error condition and MAY retry the request after several seconds.

4753 If the condition is temporary, the server MAY indicate when the client may retry the request using the
4754 Retry-After header field.

4755 **21.5.2 501 Not Implemented**

4756 The server does not support the functionality required to fulfill the request. This is the appropriate response
4757 when a UAS does not recognize the request method and is not capable of supporting it for any user. (Proxies
4758 forward all requests regardless of method.)

4759 Note that a 405 (Method Not Allowed) is sent when the server recognizes the request method, but that
4760 method is not allowed or supported.

4761 **21.5.3 502 Bad Gateway**

4762 The server, while acting as a gateway or proxy, received an invalid response from the downstream server it
4763 accessed in attempting to fulfill the request.

4764 **21.5.4 503 Service Unavailable**

4765 The server is temporarily unable to process the request due to a temporary overloading or maintenance of
4766 the server. The server MAY indicate when the client should retry the request in a Retry-After header field.
4767 If no Retry-After is given, the client MUST act as if it had received a 500 (Server Internal Error) response.

4768 A client (proxy or UAC) receiving a 503 (Service Unavailable) SHOULD attempt to forward the request
4769 to an alternate server. It SHOULD NOT forward any other requests to that server for the duration specified in
4770 the **Retry-After** header field, if present.

4771 Servers MAY refuse the connection or drop the request instead of responding with 503 (Service Unavail-
4772 able).

4773 **21.5.5 504 Server Time-out**

4774 The server did not receive a timely response from an external server it accessed in attempting to process the
4775 request. 408 (Request Timeout) should be used instead if there was no response within the period specified
4776 in the **Expires** header field from the upstream server.

4777 **21.5.6 505 Version Not Supported**

4778 The server does not support, or refuses to support, the SIP protocol version that was used in the request. The
4779 server is indicating that it is unable or unwilling to complete the request using the same major version as the
4780 client, other than with this error message.

4781 **21.5.7 513 Message Too Large**

4782 The server was unable to process the request since the message length exceeded its capabilities.

4783 **21.6 Global Failures 6xx**

4784 6xx responses indicate that a server has definitive information about a particular user, not just the particular
4785 instance indicated in the **Request-URI**.

4786 **21.6.1 600 Busy Everywhere**

4787 The callee's end system was contacted successfully but the callee is busy and does not wish to take the call
4788 at this time. The response MAY indicate a better time to call in the **Retry-After** header field. If the callee
4789 does not wish to reveal the reason for declining the call, the callee uses status code 603 (Decline) instead.
4790 This status response is returned only if the client knows that no other end point (such as a voice mail system)
4791 will answer the request. Otherwise, 486 (Busy Here) should be returned.

4792 **21.6.2 603 Decline**

4793 The callee's machine was successfully contacted but the user explicitly does not wish to or cannot partici-
4794 pate. The response MAY indicate a better time to call in the **Retry-After** header field. This status response
4795 is returned only if the client knows that no other end point will answer the request.

4796 **21.6.3 604 Does Not Exist Anywhere**

4797 The server has authoritative information that the user indicated in the **Request-URI** does not exist anywhere.

4798 **21.6.4 606 Not Acceptable**

4799 The user's agent was contacted successfully but some aspects of the session description such as the requested
4800 media, bandwidth, or addressing style were not acceptable.

4801 A 606 (Not Acceptable) response means that the user wishes to communicate, but cannot adequately
4802 support the session described. The 606 (Not Acceptable) response MAY contain a list of reasons in a Warn-
4803 ing header field describing why the session described cannot be supported. Warning reason codes are listed
4804 in Section 20.43.

4805 A message body containing a description of media capabilities MAY be present in the response, which is
4806 formatted according to the Accept header field in the INVITE (or application/sdp if not present), the same
4807 as a message body in a 200 (OK) response to an OPTIONS request.

4808 It is hoped that negotiation will not frequently be needed, and when a new user is being invited to join
4809 an already existing conference, negotiation may not be possible. It is up to the invitation initiator to decide
4810 whether or not to act on a 606 (Not Acceptable) response.

4811 This status response is returned only if the client knows that no other end point will answer the request.

4812 **22 Usage of HTTP Authentication**

4813 SIP provides a stateless, challenge-based mechanism for authentication that is based on authentication in
4814 HTTP. Any time that a proxy server or UA receives a request (with the exceptions given in Section 22.1), it
4815 MAY challenge the initiator of the request to provide assurance of its identity. Once the originator has been
4816 identified, the recipient of the request SHOULD ascertain whether or not this user is authorized to make the
4817 request in question. No authorization systems are recommended or discussed in this document.

4818 The "Digest" authentication mechanism described in this section provides message authentication and
4819 replay protection only, without message integrity or confidentiality. Protective measures above and beyond
4820 those provided by Digest need to be taken to prevent active attackers from modifying SIP requests and
4821 responses.

4822 Note that due to its weak security, the usage of "Basic" authentication has been deprecated. Servers
4823 MUST NOT accept credentials using the "Basic" authorization scheme, and servers also MUST NOT challenge
4824 with "Basic". This is a change from RFC 2543.

4825 **22.1 Framework**

4826 The framework for SIP authentication closely parallels that of HTTP (RFC 2617 [17]). In particular, the
4827 BNF for auth-scheme, auth-param, challenge, realm, realm-value, and credentials is identical (al-
4828 though the usage of "Basic" as a scheme is not permitted). In SIP, a UAS uses the 401 (Unauthorized)
4829 response to challenge the identity of a UAC. Additionally, registrars and redirect servers MAY make use
4830 of 401 (Unauthorized) responses for authentication, but proxies MUST NOT, and instead MAY use the 407
4831 (Proxy Authentication Required) response. The requirements for inclusion of the Proxy-Authenticate,
4832 Proxy-Authorization, WWW-Authenticate, and Authorization in the various messages are identical to
4833 those described in RFC 2617 [17].

4834 Since SIP does not have the concept of a canonical root URL, the notion of protection spaces is in-
4835 terpreted differently in SIP. The realm string alone defines the protection domain. This is a change from
4836 RFC 2543, in which the Request-URI and the realm together defined the protection domain.

4837 This previous definition of protection domain caused some amount of confusion since the Request-URI sent by

4838 the UAC and the Request-URI received by the challenging server might be different, and indeed the final form of
4839 the Request-URI might not be known to the UAC. Also, the previous definition depended on the presence of a SIP
4840 URI in the Request-URI and seemed to rule out alternative URI schemes (for example, the tel URL).

4841 Operators of user agents or proxy servers that will authenticate received requests MUST adhere to the
4842 following guidelines for creation of a realm string for their server:

- 4843 • Realm strings MUST be globally unique. It is RECOMMENDED that a realm string contain a hostname
4844 or domain name, following the recommendation in Section 3.2.1 of RFC 2617 [17].
- 4845 • Realm strings SHOULD present a human-readable identifier that can be rendered to a user.

4846 For example:

```
4847 INVITE sip:bob@biloxi.com SIP/2.0  
4848 Authorization: Digest realm="biloxi.com", <...>
```

4849 Generally, SIP authentication is meaningful for a specific realm, a protection domain. Thus, for Digest
4850 authentication, each such protection domain has its own set of usernames and passwords. If a server does
4851 not require authentication for a particular request, it MAY accept a default username, “anonymous”, which
4852 has no password (password of “”). Similarly, UACs representing many users, such as PSTN gateways, MAY
4853 have their own device-specific username and password, rather than accounts for particular users, for their
4854 realm.

4855 While a server can legitimately challenge most SIP requests, there are two requests defined by this
4856 document that require special handling for authentication: ACK and CANCEL.

4857 Under an authentication scheme that uses responses to carry values used to compute nonces (such as
4858 Digest), some problems come up for any requests that take no response, including ACK. For this reason,
4859 any credentials in the INVITE that were accepted by a server MUST be accepted by that server for the ACK.
4860 UACs creating an ACK message will duplicate all of the Authorization and Proxy-Authorization header
4861 field values that appeared in the INVITE to which the ACK corresponds. Servers MUST NOT attempt to
4862 challenge an ACK.

4863 Although the CANCEL method does take a response (a 2xx), servers MUST NOT attempt to challenge
4864 CANCEL requests since these requests cannot be resubmitted. Generally, a CANCEL request SHOULD be
4865 accepted by a server if it comes from the same hop that sent the request being canceled (provided that some
4866 sort of transport or network layer security association, as described in Section 26.2.1, is in place).

4867 When a UAC receives a challenge, it SHOULD render to the user the contents of the “realm” param-
4868 eter in the challenge (which appears in either a WWW-Authenticate header field or Proxy-Authenticate
4869 header field) if the UAC device does not already know of a credential for the realm in question. A service
4870 provider that pre-configures UAs with credentials for its realm should be aware that users will not have the
4871 opportunity to present their own credentials for this realm when challenged at a pre-configured device.

4872 Finally, note that even if a UAC can locate credentials that are associated with the proper realm, the
4873 potential exists that these credentials may no longer be valid or that the challenging server will not accept
4874 these credentials for whatever reason (especially when “anonymous” with no password is submitted). In
4875 this instance a server may repeat its challenge, or it may respond with a 403 Forbidden. A UAC MUST NOT
4876 re-attempt requests with the credentials that have just been rejected (though the request may be are retried if
4877 the nonce was stale).

4878 22.2 User-to-User Authentication

4879 When a UAS receives a request from a UAC, the UAS MAY authenticate the originator before the request
4880 is processed. If no credentials (in the **Authorization** header field) are provided in the request, the UAS
4881 can challenge the originator to provide credentials by rejecting the request with a 401 (Unauthorized) status
4882 code.

4883 The **WWW-Authenticate** response-header field MUST be included in 401 (Unauthorized) response mes-
4884 sages. The field value consists of at least one challenge that indicates the authentication scheme(s) and
4885 parameters applicable to the realm. See [H14.47] for a definition of the syntax.

4886 An example of the **WWW-Authenticate** header field in a 401 challenge is:

```
4887 WWW-Authenticate: Digest
4888     realm="biloxi.com",
4889     qop="auth,auth-int",
4890     nonce="dcd98b7102dd2f0e8b11d0f600bf0c093",
4891     opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

4892 When the originating UAC receives the 401 (Unauthorized), it SHOULD, if it is able, re-originate the
4893 request with the proper credentials. The UAC may require input from the originating user before proceeding.
4894 Once authentication credentials have been supplied (either directly by the user, or discovered in an internal
4895 keyring), UAs SHOULD cache the credentials for a given value of the **To** header field and “realm” and
4896 attempt to re-use these values on the next request for that destination. UAs MAY cache credentials in any
4897 way they would like.

4898 If no credentials for a realm can be located, UACs MAY attempt to retry the request with a username of
4899 “anonymous” and no password (a password of “”).

4900 Once credentials have been located, any UA that wishes to authenticate itself with a UAS or registrar
4901 – usually, but not necessarily, after receiving a 401 (Unauthorized) response – MAY do so by including an
4902 **Authorization** header field with the request. The **Authorization** field value consists of credentials containing
4903 the authentication information of the UA for the realm of the resource being requested as well as parameters
4904 required in support of authentication and replay protection.

4905 An example of the **Authorization** header field is:

```
4906 Authorization: Digest username="bob",
4907     realm="biloxi.com",
4908     nonce="dcd98b7102dd2f0e8b11d0f600bf0c093",
4909     uri="sip:bob@biloxi.com",
4910     qop=auth,
4911     nc=00000001,
4912     cnonce="0a4f113b",
4913     response="6629fae49393a05397450978507c4ef1",
4914     opaque="5ccc069c403ebaf9f0171e9517f40e41"
4915
```

4916 When a UAC resubmits a request with its credentials after receiving a 401 (Unauthorized) or 407 (Proxy
4917 Authentication Required) response, it MUST increment the **CSeq** header field value as it would normally

4918 when sending an updated request.

4919 **22.3 Proxy-to-User Authentication**

4920 Similarly, when a UAC sends a request to a proxy server, the proxy server MAY authenticate the originator
4921 before the request is processed. If no credentials (in the Proxy-Authorization header field) are provided
4922 in the request, the proxy can challenge the originator to provide credentials by rejecting the request with a
4923 407 (Proxy Authentication Required) status code. The proxy MUST populate the 407 (Proxy Authentication
4924 Required) message with a Proxy-Authenticate header field value applicable to the proxy for the requested
4925 resource.

4926 The use of Proxy-Authentication and Proxy-Authorization parallel that described in [17], with one
4927 difference. Proxies MUST NOT add values to the Proxy-Authorization header field. All 407 (Proxy Au-
4928 thentication Required) responses MUST be forwarded upstream toward the UAC following the procedures
4929 for any other response. It is the UAC's responsibility to add the Proxy-Authorization header field value
4930 containing credentials for the realm of the proxy that has asked for authentication.

4931 If a proxy were to resubmit a request adding a Proxy-Authorization header field value, it would need to in-
4932 crement the CSeq in the new request. However, this would cause the UAC that submitted the original request to
4933 discard a response from the UAS, as the CSeq value would be different.

4934 When the originating UAC receives the 407 (Proxy Authentication Required) it SHOULD, if it is able,
4935 re-originate the request with the proper credentials. It should follow the same procedures for the display of
4936 the "realm" parameter that are given above for responding to 401.

4937 If no credentials for a realm can be located, UACs MAY attempt to retry the request with a username of
4938 "anonymous" and no password (a password of "").

4939 The UAC SHOULD also cache the credentials used in the re-originated request.

4940 The following rule is RECOMMENDED for proxy credential caching:

4941 If a UA receives a Proxy-Authenticate header field value in a 401/407 response to a request with a
4942 particular Call-ID, it should incorporate credentials for that realm in all subsequent requests that contain the
4943 same Call-ID. These credentials MUST NOT be cached across dialogs; however, if a UA is configured with
4944 the realm of its local outbound proxy, when one exists, then the UA MAY cache credentials for that realm
4945 across dialogs. Note that this does mean a future request in a dialog could contain credentials that are not
4946 needed by any proxy along the Route header path.

4947 Any UA that wishes to authenticate itself to a proxy server – usually, but not necessarily, after receiving
4948 a 407 (Proxy Authentication Required) response – MAY do so by including a Proxy-Authorization header
4949 field value with the request. The Proxy-Authorization request-header field allows the client to identify itself
4950 (or its user) to a proxy that requires authentication. The Proxy-Authorization header field value consists of
4951 credentials containing the authentication information of the UA for the proxy and/or realm of the resource
4952 being requested.

4953 A Proxy-Authorization header field value applies only to the proxy whose realm is identified in the
4954 "realm" parameter (this proxy may previously have demanded authentication using the Proxy-Authenticate
4955 field). When multiple proxies are used in a chain, a Proxy-Authorization header field value MUST NOT be
4956 consumed by any proxy whose realm does not match the "realm" parameter specified in that value.

4957 Note that if an authentication scheme that does not support realms is used in the Proxy-Authorization
4958 header field, a proxy server MUST attempt to parse all Proxy-Authorization header field values to determine
4959 whether one of them has what the proxy server considers to be valid credentials. Because this is potentially
4960 very time-consuming in large networks, proxy servers SHOULD use an authentication scheme that supports

4961 realms in the Proxy-Authorization header field.

4962 If a request is forked (as described in Section 16.7), various proxy servers and/or UAs may wish to
4963 challenge the UAC. In this case, the forking proxy server is responsible for aggregating these challenges
4964 into a single response. Each WWW-Authenticate and Proxy-Authenticate value received in responses to
4965 the forked request MUST be placed into the single response that is sent by the forking proxy to the UA; the
4966 ordering of these header field values is not significant.

4967 When a proxy server issues a challenge in response to a request, it will not proxy the request until the UAC has
4968 retried the request with valid credentials. A forking proxy may forward a request simultaneously to multiple proxy
4969 servers that require authentication, each of which in turn will not forward the request until the originating UAC has
4970 authenticated itself in their respective realm. If the UAC does not provide credentials for each challenge, then the
4971 proxy servers that issued the challenges will not forward requests to the UA where the destination user might be
4972 located, and therefore, the virtues of forking are largely lost.

4973 When resubmitting its request in response to a 401 (Unauthorized) or 407 (Proxy Authentication Re-
4974 quired) that contains multiple challenges, a UAC MAY include an Authorization value for each WWW-
4975 Authenticate value and a Proxy-Authorization value for each Proxy-Authenticate value for which the
4976 UAC wishes to supply a credential. As noted above, multiple credentials in a request SHOULD be differen-
4977 tiated by the "realm" parameter.

4978 It is possible for multiple challenges associated with the same realm to appear in the same 401 (Unautho-
4979 rized) or 407 (Proxy Authentication Required). This can occur, for example, when multiple proxies within
4980 the same administrative domain, which use a common realm, are reached by a forking request. When it re-
4981 tries a request, a UAC MAY therefore supply multiple credentials in Authorization or Proxy-Authorization
4982 header fields with the same "realm" parameter value. The same credentials SHOULD be used for the same
4983 realm.

4984 See [H14.34] for a definition of the syntax of Proxy-Authentication and Proxy-Authorization.

4985 22.4 The Digest Authentication Scheme

4986 This section describes the modifications and clarifications required to apply the HTTP Digest authentication
4987 scheme to SIP. The SIP scheme usage is almost completely identical to that for HTTP [17].

4988 Since RFC 2543 is based on HTTP Digest as defined in RFC 2069 [39], SIP servers supporting RFC 2617
4989 MUST ensure they are backwards compatible with RFC 2069. Procedures for this backwards compatibility
4990 are specified in RFC 2617. Note, however, that SIP servers MUST NOT accept or request Basic authentica-
4991 tion.

4992 The rules for Digest authentication follow those defined in [17], with "HTTP/1.1" replaced by "SIP/2.0"
4993 in addition to the following differences:

4994 1. The URI included in the challenge has the following BNF:

4995 URI = SIP-URI / SIPS-URI

4996 2. The BNF in RFC 2617 has an error in that the 'uri' parameter of the Authorization header field
4997 for HTTP Digest authentication is not enclosed in quotation marks. (The example in Section 3.5 of
4998 RFC 2617 is correct.) For SIP, the 'uri' MUST be enclosed in quotation marks.

4999 3. The BNF for digest-uri-value is:

5000 digest-uri-value = Request-URI ; as defined in Section 25

- 5001 4. The example procedure for choosing a nonce based on Etag does not work for SIP.
- 5002 5. The text in RFC 2617 [17] regarding cache operation does not apply to SIP.
- 5003 6. RFC 2617 [17] requires that a server check that the URI in the request line and the URI included in
5004 the **Authorization** header field point to the same resource. In a SIP context, these two URIs may refer
5005 to different users, due to forwarding at some proxy. Therefore, in SIP, a server MAY check that the
5006 **Request-URI** in the **Authorization** header field value corresponds to a user for whom the server is
5007 willing to accept forwarded or direct requests, but it is not necessarily a failure if the two fields are
5008 not equivalent.
- 5009 7. As a clarification to the calculation of the A2 value for message integrity assurance in the Digest
5010 authentication scheme, implementers should assume, when the entity-body is empty (that is, when
5011 SIP messages have no body) that the hash of the entity-body resolves to the MD5 hash of an empty
5012 string, or:

5013
$$H(\text{entity-body}) = \text{MD5}("") = \text{"d41d8cd98f00b204e9800998ecf8427e"}$$

- 5014 8. RFC 2617 notes that a cnonce value MUST NOT be sent in an **Authorization** (and by extension **Proxy-**
5015 **Authorization**) header field if no qop directive has been sent. Therefore, any algorithms that have a
5016 dependency on the cnonce (including "MD5-Sess") require that the qop directive be sent. Use of the
5017 "qop" parameter is optional in RFC 2617 for the purposes of backwards compatibility with RFC 2069;
5018 since RFC 2543 was based on RFC 2069, the "qop" parameter must unfortunately remain optional
5019 for clients and servers to receive. However, servers MUST always send a "qop" parameter in **WWW-**
5020 **Authenticate** and **Proxy-Authenticate** header field values. If a client receives a "qop" parameter in a
5021 challenge header field, it MUST send the "qop" parameter in any resulting authorization header field.

5022 RFC 2543 did not allow usage of the **Authentication-Info** header field (it effectively used RFC 2069).
5023 However, we now allow usage of this header field, since it provides integrity checks over the bodies and
5024 provides mutual authentication. RFC 2617 [17] defines mechanisms for backwards compatibility using the
5025 qop attribute in the request. These mechanisms MUST be used by a server to determine if the client supports
5026 the new mechanisms in RFC 2617 that were not specified in RFC 2069.

5027 **23 S/MIME**

5028 SIP messages carry MIME bodies and the MIME standard includes mechanisms for securing MIME con-
5029 tents to ensure both integrity and confidentiality (including the 'multipart/signed' and 'application/pkcs7-
5030 mime' MIME types, see RFC 1847 [22], RFC 2630 [23] and RFC 2633 [24]). Implementers should note,
5031 however, that there may be rare network intermediaries (not typical proxy servers) that rely on viewing or
5032 modifying the bodies of SIP messages (especially SDP), and that secure MIME may prevent these sorts of
5033 intermediaries from functioning.

5034 This applies particularly to certain types of firewalls.

5035 The PGP mechanism for encrypting the header fields and bodies of SIP messages described in RFC 2543 has
5036 been deprecated.

5037 **23.1 S/MIME Certificates**

5038 The certificates that are used to identify an end-user for the purposes of S/MIME differ from those used
5039 by servers in one important respect - rather than asserting that the identity of the holder corresponds to a
5040 particular hostname, these certificates assert that the holder is identified by an end-user address. This address
5041 is composed of the concatenation of the “userinfo” “@” and “domainname” portions of a SIP or SIPS URI
5042 (in other words, an email address of the form “bob@biloxi.com”), most commonly corresponding to a user’s
5043 address-of-record.

5044 These certificates are also associated with keys that are used to sign or encrypt bodies of SIP messages.
5045 Bodies are signed with the private key of the sender (who may include their public key with the message
5046 as appropriate), but bodies are encrypted with the public key of the intended recipient. Obviously, senders
5047 must have foreknowledge of the public key of recipients in order to encrypt message bodies. Public keys
5048 can be stored within a UA on a virtual keyring.

5049 Each user agent that supports S/MIME **MUST** contain a keyring specifically for end-users’ certificates.
5050 This keyring should map between addresses of record and corresponding certificates. Over time, users
5051 **SHOULD** use the same certificate when they populate the originating URI of signaling (the **From** header
5052 field) with the same address-of-record.

5053 Any mechanisms depending on the existence of end-user certificates are seriously limited in that there is
5054 virtually no consolidated authority today that provides certificates for end-user applications. However, users
5055 **SHOULD** acquire certificates from known public certificate authorities. As an alternative, users **MAY** create
5056 self-signed certificates. The implications of self-signed certificates are explored further in Section 26.4.2.
5057 Implementations may also use pre-configured certificates in deployments in which a previous trust relation-
5058 ship exists between all SIP entities.

5059 Above and beyond the problem of acquiring an end-user certificate, there are few well-known central-
5060 ized directories that distribute end-user certificates. However, the holder of a certificate **SHOULD** publish
5061 their certificate in any public directories as appropriate. Similarly, UACs **SHOULD** support a mechanism
5062 for importing (manually or automatically) certificates discovered in public directories corresponding to the
5063 target URIs of SIP requests.

5064 **23.2 S/MIME Key Exchange**

5065 SIP itself can also be used as a means to distribute public keys in the following manner.

5066 Whenever the CMS SignedData message is used in S/MIME for SIP, it **MUST** contain the certificate
5067 bearing the public key necessary to verify the signature.

5068 When a UAC sends a request containing an S/MIME body that initiates a dialog, or sends a non-
5069 INVITE request outside the context of a dialog, the UAC **SHOULD** structure the body as an S/MIME ‘multi-
5070 part/signed’ CMS SignedData body. If the desired CMS service is EnvelopedData (and the public key of the
5071 target user is known), the UAC **SHOULD** send the EnvelopedData message encapsulated within a SignedData
5072 message.

5073 When a UAS receives a request containing an S/MIME CMS body that includes a certificate, the UAS
5074 **SHOULD** first validate the certificate, if possible, with any available root certificates for certificate authorities.
5075 The UAS **SHOULD** also determine the subject of the certificate (for S/MIME, the SubjectAltName will
5076 contain the appropriate identity) and compare this value to the **From** header field of the request. If the
5077 certificate cannot be verified, because it is self-signed, or signed by no known authority, or if it is verifiable
5078 but its subject does not correspond to the **From** header field of request, the UAS **MUST** notify its user
5079 of the status of the certificate (including the subject of the certificate, its signer, and any key fingerprint

5080 information) and request explicit permission before proceeding. If the certificate was successfully verified
5081 and the subject of the certificate corresponds to the From header field of the SIP request, or if the user (after
5082 notification) explicitly authorizes the use of the certificate, the UAS SHOULD add this certificate to a local
5083 keyring, indexed by the address-of-record of the holder of the certificate.

5084 When a UAS sends a response containing an S/MIME body that answers the first request in a dialog, or
5085 a response to a non-INVITE request outside the context of a dialog, the UAS SHOULD structure the body
5086 as an S/MIME 'multipart/signed' CMS SignedData body. If the desired CMS service is EnvelopedData, the
5087 UAS SHOULD send the EnvelopedData message encapsulated within a SignedData message.

5088 When a UAC receives a response containing an S/MIME CMS body that includes a certificate, the UAC
5089 SHOULD first validate the certificate, if possible, with any appropriate root certificate. The UAC SHOULD
5090 also determine the subject of the certificate and compare this value to the To field of the response; although
5091 the two may very well be different, and this is not necessarily indicative of a security breach. If the certificate
5092 cannot be verified because it is self-signed, or signed by no known authority, the UAC MUST notify its user
5093 of the status of the certificate (including the subject of the certificate, its signator, and any key fingerprint
5094 information) and request explicit permission before proceeding. If the certificate was successfully verified,
5095 and the subject of the certificate corresponds to the To header field in the response, or if the user (after
5096 notification) explicitly authorizes the use of the certificate, the UAC SHOULD add this certificate to a local
5097 keyring, indexed by the address-of-record of the holder of the certificate. If the UAC had not transmitted its
5098 own certificate to the UAS in any previous transaction, it SHOULD use a CMS SignedData body for its next
5099 request or response.

5100 On future occasions, when the UA receives requests or responses that contain a From header field
5101 corresponding to a value in its keyring, the UA SHOULD compare the certificate offered in these messages
5102 with the existing certificate in its keyring. If there is a discrepancy, the UA MUST notify its user of a change
5103 of the certificate (preferably in terms that indicate that this is a potential security breach) and acquire the
5104 user's permission before continuing to process the signaling. If the user authorizes this certificate, it SHOULD
5105 be added to the keyring alongside any previous value(s) for this address-of-record.

5106 Note well however, that this key exchange mechanism does not guarantee the secure exchange of keys
5107 when self-signed certificates, or certificates signed by an obscure authority, are used - it is vulnerable to
5108 well-known attacks. In the opinion of the authors, however, the security it provides is proverbially better
5109 than nothing; it is in fact comparable to the widely used SSH application. These limitations are explored in
5110 greater detail in Section 26.4.2.

5111 If a UA receives an S/MIME body that has been encrypted with a public key unknown to the recipient,
5112 it MUST reject the request with a 493 (Undecipherable) response. This response SHOULD contain a valid
5113 certificate for the respondent (corresponding, if possible, to any address of record given in the To header
5114 field of the rejected request) within a MIME body with a 'certs-only' "smime-type" parameter.

5115 A 493 (Undecipherable) sent without any certificate indicates that the respondent cannot or will not
5116 utilize S/MIME encrypted messages, though they may still support S/MIME signatures.

5117 Note that a user agent that receives a request containing an S/MIME body that is not optional (with
5118 a Content-Disposition header "handling" parameter of "required") MUST reject the request with a 415
5119 Unsupported Media Type response if the MIME type is not understood. A user agent that receives such a
5120 response when S/MIME is sent SHOULD notify its user that the remote device does not support S/MIME,
5121 and it MAY subsequently resend the request without S/MIME, if appropriate; however, this 415 response
5122 may constitute a downgrade attack.

5123 If a user agent sends an S/MIME body in a request, but receives a response that contains a MIME body
5124 that is not secured, the UAC SHOULD notify its user that the session could not be secured. However, if a

5125 user agent that supports S/MIME receives a request with an unsecured body, it SHOULD NOT respond with
5126 a secured body, but if it expects S/MIME from the sender (for example, because the sender's From header
5127 field value corresponds to an identity on its keychain), the UAS SHOULD notify its user that the session
5128 could not be secured.

5129 A number of conditions that arise in the previous text call for the notification of the user when an
5130 anomalous certificate-management event occurs. Users might well ask what they should do under these
5131 circumstances. First and foremost, an unexpected change in a certificate, or an absence of security when
5132 security is expected, are causes for caution but not necessarily indications that an attack is in progress. Users
5133 might abort any connection attempt or refuse a connection request they have received; in telephony parlance,
5134 they could hang up and call back. Users may wish to find an alternate means to contact the other party and
5135 confirm that their key has legitimately changed. Note that users are sometimes compelled to change their
5136 certificates, for example when they suspect that the secrecy of their private key has been compromised.
5137 When their private key is no longer private, users must legitimately generate a new key and re-establish trust
5138 with any users that held their old key.

5139 Finally, if during the course of a dialog a UA receives a certificate in a CMS SignedData message that
5140 does not correspond with the certificates previously exchanged during a dialog, the UA MUST notify its user
5141 of the change, preferably in terms that indicate that this is a potential security breach.

5142 23.3 Securing MIME bodies

5143 There are two types of secure MIME bodies that are of interest to SIP: 'multipart/signed' and 'application/pkcs7-
5144 mime'. The procedures for the use of these bodies should follow the S/MIME specification [24] with a few
5145 variations.

- 5146 • "multipart/signed" MUST be used only with CMS detached signatures.

5147 This allows backwards compatibility with non-S/MIME-compliant recipients.

- 5148 • S/MIME bodies SHOULD have a Content-Disposition header field, and the value of the "handling"
5149 parameter SHOULD be "required."
- 5150 • If a UAC has no certificate on its keyring associated with the address-of-record to which it wants to
5151 send a request, it cannot send an encrypted "application/pkcs7-mime" MIME message. UACs MAY
5152 send an initial request such as an OPTIONS message with a CMS detached signature in order to
5153 solicit the certificate of the remote side (the signature SHOULD be over a "message/sip" body of the
5154 type described in Section 23.4).

5155 Note that future standardization work on S/MIME may define non-certificate based keys.

- 5156 • Senders of S/MIME bodies SHOULD use the "SMIMECapabilities" (see Section 2.5.2 of [24]) at-
5157 tribute to express their capabilities and preferences for further communications. Note especially that
5158 senders MAY use the "preferSignedData" capability to encourage receivers to respond with CMS
5159 SignedData messages (for example, when sending an OPTIONS request as described above).
- 5160 • S/MIME implementations MUST at a minimum support SHA1 as a digital signature algorithm, and
5161 3DES as an encryption algorithm. All other signature and encryption algorithms MAY be supported.
5162 Implementations can negotiate support for these algorithms with the "SMIMECapabilities" attribute.

5203 **23.4.1 Integrity and Confidentiality Properties of SIP Headers**

5204 When the S/MIME integrity or confidentiality mechanisms are used, there may be discrepancies between the
5205 values in the “inner” message and values in the “outer” message. The rules for handling any such differences
5206 for all of the header fields described in this document are given in this section.

5207 Note that for the purposes of loose timestamping, all SIP messages that tunnel “message/sip” SHOULD
5208 contain a **Date** header in both the “inner” and “outer” headers.

5209 **23.4.1.1 Integrity** Whenever integrity checks are performed, the integrity of a header field should be
5210 determined by matching the value of the header field in the signed body with that in the “outer” messages
5211 using the comparison rules of SIP as described in 20.

5212 Header fields that can be legitimately modified by proxy servers are: **Request-URI**, **Via**, **Record-**
5213 **Route**, **Route**, **Max-Forwards**, and **Proxy-Authorization**. If these header fields are not intact end-to-end,
5214 implementations SHOULD NOT consider this a breach of security. Changes to any other header fields defined
5215 in this document constitute an integrity violation; users MUST be notified of a discrepancy.

5216 **23.4.1.2 Confidentiality** When messages are encrypted, header fields may be included in the encrypted
5217 body that are not present in the “outer” message.

5218 Some header fields must always have a plaintext version because they are required header fields in
5219 requests and responses - these include: **To**, **From**, **Call-ID**, **CSeq**, **Contact**. While it is probably not
5220 useful to provide an encrypted alternative for the **Call-ID**, **Cseq**, or **Contact**, providing an alternative to the
5221 information in the “outer” **To** or **From** is permitted. Note that the values in an encrypted body are not used
5222 for the purposes of identifying transactions or dialogs - they are merely informational. If the **From** header
5223 field in an encrypted body differs from the value in the “outer” message, the value within the encrypted
5224 body SHOULD be displayed to the user, but MUST NOT be used in the “outer” header fields of any future
5225 messages.

5226 Primarily, a user agent will want to encrypt header fields that have an end-to-end semantic, including:
5227 **Subject**, **Reply-To**, **Organization**, **Accept**, **Accept-Encoding**, **Accept-Language**, **Alert-Info**, **Error-**
5228 **Info**, **Authentication-Info**, **Expires**, **In-Reply-To**, **Require**, **Supported**, **Unsupported**, **Retry-After**, **User-**
5229 **Agent**, **Server**, and **Warning**. If any of these header fields are present in an encrypted body, they should be
5230 used instead of any “outer” header fields, whether this entails displaying the header field values to users or
5231 setting internal states in the UA. They SHOULD NOT however be used in the “outer” headers of any future
5232 messages.

5233 If present, the **Date** header field MUST always be the same in the “inner” and “outer” headers.

5234 Since MIME bodies are attached to the “inner” message, implementations will usually encrypt MIME-
5235 specific header fields, including: **MIME-Version**, **Content-Type**, **Content-Length**, **Content-Language**,
5236 **Content-Encoding** and **Content-Disposition**. The “outer” message will have the proper MIME header
5237 fields for S/MIME bodies. These header fields (and any MIME bodies they preface) should be treated as
5238 normal MIME header fields and bodies received in a SIP message.

5239 It is not particularly useful to encrypt the following header fields: **Min-Expires**, **Timestamp**, **Autho-**
5240 **rization**, **Priority**, and **WWW-Authenticate**. This category also includes those header fields that can be
5241 changed by proxy servers (described in the preceding section). UAs SHOULD never include these in an
5242 “inner” message if they are not included in the “outer” message. UAs that receive any of these header fields
5243 in an encrypted body SHOULD ignore the encrypted values.

5244 Note that extensions to SIP may define additional header fields; the authors of these extensions should

5245 describe the integrity and confidentiality properties of such header fields. If a SIP UA encounters an un-
5246 known header field with an integrity violation, it MUST ignore the header field.

5247 **23.4.2 Tunneling Integrity and Authentication**

5248 Tunneling SIP messages within S/MIME bodies can provide integrity for SIP header fields if the header
5249 fields that the sender wishes to secure are replicated in a "message/sip" MIME body signed with a CMS
5250 detached signature.

5251 Provided that the "message/sip" body contains at least the fundamental dialog identifiers (To, From,
5252 Call-ID, CSeq), then a signed MIME body can provide limited authentication. At the very least, if the
5253 certificate used to sign the body is unknown to the recipient and cannot be verified, the signature can be used
5254 to ascertain that a later request in a dialog was transmitted by the same certificate-holder that initiated the
5255 dialog. If the recipient of the signed MIME body has some stronger incentive to trust the certificate (they
5256 were able to validate it, they acquired it from a trusted repository, or they have used it frequently) then the
5257 signature can be taken as a stronger assertion of the identity of the subject of the certificate.

5258 In order to eliminate possible confusions about the addition or subtraction of entire header fields, senders
5259 SHOULD replicate all header fields from the request within the signed body. Any message bodies that require
5260 integrity protection MUST be attached to the "inner" message.

5261 If a Date header is present in a message with a signed body, the recipient SHOULD compare the header
5262 field value with its own internal clock, if applicable. If a significant time discrepancy is detected (on the
5263 order of an hour or more), the user agent SHOULD alert the user to the anomaly, and note that it is a potential
5264 security breach.

5265 If an integrity violation in a message is detected by its recipient, the message MAY be rejected with a
5266 403 (Forbidden) response if it is a request, or any existing dialog MAY be terminated. UAs SHOULD notify
5267 users of this circumstance and request explicit guidance on how to proceed.

5268 The following is an example of the use of a tunneled "message/sip" body:

```
5269     INVITE sip:bob@biloxi.com SIP/2.0
5270     Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
5271     To: Bob <sip:bob@biloxi.com>
5272     From: Alice <sip:alice@atlanta.com>;tag=1928301774
5273     Call-ID: a84b4c76e66710
5274     CSeq: 314159 INVITE
5275     Max-Forwards: 70
5276     Date: Thu, 21 Feb 2002 13:02:03 GMT
5277     Contact: <sip:alice@pc33.atlanta.com>
5278     Content-Type: multipart/signed;
5279         protocol="application/pkcs7-signature";
5280         micalg=sha1; boundary=boundary42
5281     Content-Length: 568
5282
5283     --boundary42
5284     Content-Type: message/sip
5285
5286     INVITE sip:bob@biloxi.com SIP/2.0
```

5287 Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
5288 To: Bob <bob@biloxi.com>
5289 From: Alice <alice@atlanta.com>;tag=1928301774
5290 Call-ID: a84b4c76e66710
5291 CSeq: 314159 INVITE
5292 Max-Forwards: 70
5293 Date: Thu, 21 Feb 2002 13:02:03 GMT
5294 Contact: <sip:alice@pc33.atlanta.com>
5295 Content-Type: application/sdp
5296 Content-Length: 147
5297
5298 v=0
5299 o=UserA 2890844526 2890844526 IN IP4 here.com
5300 s=Session SDP
5301 c=IN IP4 pc33.atlanta.com
5302 t=0 0
5303 m=audio 49172 RTP/AVP 0
5304 a=rtpmap:0 PCMU/8000
5305
5306 --boundary42
5307 Content-Type: application/pkcs7-signature; name=smime.p7s
5308 Content-Transfer-Encoding: base64
5309 Content-Disposition: attachment; filename=smime.p7s;
5310 handling=required
5311
5312 ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
5313 4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
5314 n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpFyF4
5315 7GhIGfHfYT64VQbnj756
5316
5317 --boundary42-

5318 23.4.3 Tunneling Encryption

5319 It may also be desirable to use this mechanism to encrypt a “message/sip” MIME body within a CMS
5320 EnvelopedData message S/MIME body, but in practice, most header fields are of at least some use to the
5321 network; the general use of encryption with S/MIME is to secure message bodies like SDP rather than
5322 message headers. Some informational header fields, such as the Subject or Organization could perhaps
5323 warrant end-to-end security. Headers defined by future SIP applications might also require obfuscation.

5324 Another possible application of encrypting header fields is selective anonymity. A request could be con-
5325 structed with a From header field that contains no personal information (for example, sip:anonymous@anonymizer.invalid).
5326 However, a second From header field containing the genuine address-of-record of the originator could be
5327 encrypted within a “message/sip” MIME body where it will only be visible to the endpoints of a dialog.

5328 motivationNote that if this mechanism is used for anonymity, the From header field will no longer
5329 be usable by the recipient of a message as an index to their certificate keychain for retrieving the proper


```

5374 * v=0 *
5375 * o=alice 53655765 2353687637 IN IP4 pc33.atlanta.com *
5376 * s=Session SDP *
5377 * t=0 0 *
5378 * c=IN IP4 pc33.atlanta.com *
5379 * m=audio 3456 RTP/AVP 0 1 3 99 *
5380 * a=rtpmap:0 PCMU/8000 *
5381 *****
5382
5383 --boundary42
5384 Content-Type: application/pkcs7-signature; name=smime.p7s
5385 Content-Transfer-Encoding: base64
5386 Content-Disposition: attachment; filename=smime.p7s;
5387     handling=required
5388
5389 ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
5390 4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
5391 n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpFyF4
5392 7GhIGfHfYT64VQbnj756
5393
5394 --boundary42-

```

5395 24 Examples

5396 In the following examples, we often omit the message body and the corresponding Content-Length and
 5397 Content-Type header fields for brevity.

5398 24.1 Registration

5399 Bob registers on start-up. The message flow is shown in Figure 9. Note that the authentication usually
 5400 required for registration is not shown for simplicity.

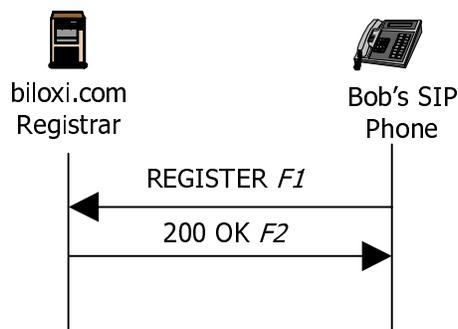


Figure 9: SIP Registration Example

5401
5402 F1 REGISTER Bob -> Registrar
5403
5404 REGISTER sip:registrar.biloxi.com SIP/2.0
5405 Via: SIP/2.0/UDP bobspc.biloxi.com:5060;branch=z9hG4bKnashds7
5406 Max-Forwards: 70
5407 To: Bob <sip:bob@biloxi.com>
5408 From: Bob <sip:bob@biloxi.com>;tag=456248
5409 Call-ID: 843817637684230@998sdasdh09
5410 CSeq: 1826 REGISTER
5411 Contact: <sip:bob@192.0.2.4>
5412 Expires: 7200
5413 Content-Length: 0

5414 The registration expires after two hours. The registrar responds with a 200 OK:

5415
5416 F2 200 OK Registrar -> Bob
5417
5418 SIP/2.0 200 OK
5419 Via: SIP/2.0/UDP bobspc.biloxi.com:5060;branch=z9hG4bKnashds7
5420 ;received=192.0.2.4
5421 To: Bob <sip:bob@biloxi.com>
5422 From: Bob <sip:bob@biloxi.com>;tag=456248
5423 Call-ID: 843817637684230@998sdasdh09
5424 CSeq: 1826 REGISTER
5425 Contact: <sip:bob@192.0.2.4>
5426 Expires: 7200
5427 Content-Length: 0
5428

5429 24.2 Session Setup

5430 This example contains the full details of the example session setup in Section 4. The message flow is shown
5431 in Figure 1. Note that these flows show the minimum required set of header fields - some other header fields
5432 such as Allow and Supported would normally be present.

5433
5434 F1 INVITE Alice -> atlanta.com proxy
5435
5436 INVITE sip:bob@biloxi.com SIP/2.0
5437 Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
5438 Max-Forwards: 70
5439 To: Bob <sip:bob@biloxi.com>
5440 From: Alice <sip:alice@atlanta.com>;tag=1928301774

5441 Call-ID: a84b4c76e66710
5442 CSeq: 314159 INVITE
5443 Contact: <sip:alice@pc33.atlanta.com>
5444 Content-Type: application/sdp
5445 Content-Length: 142
5446
5447 (Alice's SDP not shown)

5448
5449 F2 100 Trying atlanta.com proxy -> Alice
5450
5451 SIP/2.0 100 Trying
5452 Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
5453 ;received=192.0.2.1
5454 To: Bob <sip:bob@biloxi.com>
5455 From: Alice <sip:alice@atlanta.com>;tag=1928301774
5456 Call-ID: a84b4c76e66710
5457 CSeq: 314159 INVITE
5458 Content-Length: 0

5459
5460 F3 INVITE atlanta.com proxy -> biloxi.com proxy
5461
5462 INVITE sip:bob@biloxi.com SIP/2.0
5463 Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
5464 Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
5465 ;received=192.0.2.1
5466 Max-Forwards: 69
5467 To: Bob <sip:bob@biloxi.com>
5468 From: Alice <sip:alice@atlanta.com>;tag=1928301774
5469 Call-ID: a84b4c76e66710
5470 CSeq: 314159 INVITE
5471 Contact: <sip:alice@pc33.atlanta.com>
5472 Content-Type: application/sdp
5473 Content-Length: 142
5474
5475 (Alice's SDP not shown)

5476
5477 F4 100 Trying biloxi.com proxy -> atlanta.com proxy
5478
5479 SIP/2.0 100 Trying
5480 Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
5481 ;received=192.0.2.2
5482 Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

5483 ;received=192.0.2.1
5484 To: Bob <sip:bob@biloxi.com>
5485 From: Alice <sip:alice@atlanta.com>;tag=1928301774
5486 Call-ID: a84b4c76e66710
5487 CSeq: 314159 INVITE
5488 Content-Length: 0

5489
5490 F5 INVITE biloxi.com proxy -> Bob
5491
5492 INVITE sip:bob@192.0.2.4 SIP/2.0
5493 Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bK4b43c2ff8.1
5494 Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
5495 ;received=192.0.2.2
5496 Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
5497 ;received=192.0.2.1
5498 Max-Forwards: 68
5499 To: Bob <sip:bob@biloxi.com>
5500 From: Alice <sip:alice@atlanta.com>;tag=1928301774
5501 Call-ID: a84b4c76e66710
5502 CSeq: 314159 INVITE
5503 Contact: <sip:alice@pc33.atlanta.com>
5504 Content-Type: application/sdp
5505 Content-Length: 142
5506
5507 (Alice's SDP not shown)

5508
5509 F6 180 Ringing Bob -> biloxi.com proxy
5510
5511 SIP/2.0 180 Ringing
5512 Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bK4b43c2ff8.1
5513 ;received=192.0.2.3
5514 Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
5515 ;received=192.0.2.2
5516 Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
5517 ;received=192.0.2.1
5518 To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
5519 From: Alice <sip:alice@atlanta.com>;tag=1928301774
5520 Call-ID: a84b4c76e66710
5521 Contact: <sip:bob@192.0.2.4>
5522 CSeq: 314159 INVITE
5523 Content-Length: 0

5524

5525 F7 180 Ringing biloxi.com proxy -> atlanta.com proxy
5526
5527 SIP/2.0 180 Ringing
5528 Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
5529 ;received=192.0.2.2
5530 Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
5531 ;received=192.0.2.1
5532 To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
5533 From: Alice <sip:alice@atlanta.com>;tag=1928301774
5534 Call-ID: a84b4c76e66710
5535 Contact: <sip:bob@192.0.2.4>
5536 CSeq: 314159 INVITE
5537 Content-Length: 0

5538
5539 F8 180 Ringing atlanta.com proxy -> Alice
5540
5541 SIP/2.0 180 Ringing
5542 Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
5543 ;received=192.0.2.1
5544 To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
5545 From: Alice <sip:alice@atlanta.com>;tag=1928301774
5546 Call-ID: a84b4c76e66710
5547 Contact: <sip:bob@192.0.2.4>
5548 CSeq: 314159 INVITE
5549 Content-Length: 0

5550
5551 F9 200 OK Bob -> biloxi.com proxy
5552
5553 SIP/2.0 200 OK
5554 Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bK4b43c2ff8.1
5555 ;received=192.0.2.3
5556 Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
5557 ;received=192.0.2.2
5558 Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
5559 ;received=192.0.2.1
5560 To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
5561 From: Alice <sip:alice@atlanta.com>;tag=1928301774
5562 Call-ID: a84b4c76e66710
5563 CSeq: 314159 INVITE
5564 Contact: <sip:bob@192.0.2.4>
5565 Content-Type: application/sdp
5566 Content-Length: 131
5567

5568 (Bob's SDP not shown)

5569

5570 F10 200 OK biloxi.com proxy -> atlanta.com proxy

5571

5572 SIP/2.0 200 OK

5573 Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1

5574 ;received=192.0.2.2

5575 Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

5576 ;received=192.0.2.1

5577 To: Bob <sip:bob@biloxi.com>;tag=a6c85cf

5578 From: Alice <sip:alice@atlanta.com>;tag=1928301774

5579 Call-ID: a84b4c76e66710

5580 CSeq: 314159 INVITE

5581 Contact: <sip:bob@192.0.2.4>

5582 Content-Type: application/sdp

5583 Content-Length: 131

5584

5585 (Bob's SDP not shown)

5586

5587 F11 200 OK atlanta.com proxy -> Alice

5588

5589 SIP/2.0 200 OK

5590 Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

5591 ;received=192.0.2.1

5592 To: Bob <sip:bob@biloxi.com>;tag=a6c85cf

5593 From: Alice <sip:alice@atlanta.com>;tag=1928301774

5594 Call-ID: a84b4c76e66710

5595 CSeq: 314159 INVITE

5596 Contact: <sip:bob@192.0.2.4>

5597 Content-Type: application/sdp

5598 Content-Length: 131

5599

5600 (Bob's SDP not shown)

5601

5602 F12 ACK Alice -> Bob

5603

5604 ACK sip:bob@192.0.2.4 SIP/2.0

5605 Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds9

5606 Max-Forwards: 70

5607 To: Bob <sip:bob@biloxi.com>;tag=a6c85cf

5608 From: Alice <sip:alice@atlanta.com>;tag=1928301774

5609 Call-ID: a84b4c76e66710

5610 CSeq: 314159 ACK
5611 Content-Length: 0

5612 The media session between Alice and Bob is now established.

5613 Bob hangs up first. Note that Bob's SIP phone maintains its own CSeq numbering space, which, in
5614 this example, begins with 231. Since Bob is making the request, the To and From URIs and tags have been
5615 swapped.

5616
5617 F13 BYE Bob -> Alice

5618
5619 BYE sip:alice@pc33.atlanta.com SIP/2.0
5620 Via: SIP/2.0/UDP 192.0.2.4;branch=z9hG4bKnashds10
5621 Max-Forwards: 70
5622 From: Bob <sip:bob@biloxi.com>;tag=a6c85cf
5623 To: Alice <sip:alice@atlanta.com>;tag=1928301774
5624 Call-ID: a84b4c76e66710
5625 CSeq: 231 BYE
5626 Content-Length: 0

5627
5628 F14 200 OK Alice -> Bob

5629
5630 SIP/2.0 200 OK
5631 Via: SIP/2.0/UDP 192.0.2.4;branch=z9hG4bKnashds10
5632 From: Bob <sip:bob@biloxi.com>;tag=a6c85cf
5633 To: Alice <sip:alice@atlanta.com>;tag=1928301774
5634 Call-ID: a84b4c76e66710
5635 CSeq: 231 BYE
5636 Content-Length: 0

5637 The SIP Call Flows document [40] contains further examples of SIP messages.

5638 **25 Augmented BNF for the SIP Protocol**

5639 All of the mechanisms specified in this document are described in both prose and an augmented Backus-
5640 Naur Form (BNF) defined in RFC 2234 [10]. Section 6.1 of RFC 2234 defines a set of core rules that are
5641 used by this specification, and not repeated here. Implementers need to be familiar with the notation and
5642 content of RFC 2234 in order to understand this specification. Certain basic rules are in uppercase, such as
5643 SP, LWS, HTAB, CRLF, DIGIT, ALPHA, etc. Angle brackets are used within definitions to clarify the use
5644 of rule names.

5645 In some cases, the BNF for a choice will indicate that some elements are optional through square brack-
5646 ets. For example:

5647 `foo = bar / baz / [boo]`

5648 The use of square brackets is redundant syntactically. It is used as a semantic hint that the specific
5649 parameter is optional to use.

5650 25.1 Basic Rules

5651 The following rules are used throughout this specification to describe basic parsing constructs. The US-
5652 ASCII coded character set is defined by ANSI X3.4-1986.

5653 alphanum = ALPHA / DIGIT

5654 Several rules are incorporated from RFC 2396 [5] but are updated to make them compliant with RFC 2234 [10].
5655 These include:

 reserved = "," / "/" / "?" / ":" / "@" / "&" / "=" / "+"
 / "\$" / ","
 unreserved = alphanum / mark
 mark = "-" / "_" / "." / "!" / "'" / "*" / ""
 / "(" / ")"
5656 escaped = "%" HEXDIG HEXDIG

5657 SIP header field values can be folded onto multiple lines if the continuation line begins with a space or
5658 horizontal tab. All linear white space, including folding, has the same semantics as SP. A recipient MAY
5659 replace any linear white space with a single SP before interpreting the field value or forwarding the message
5660 downstream. This is intended to behave exactly as HTTP/1.1 as described in RFC 2616 [8]. The SWS
5661 construct is used when linear white space is optional, generally between tokens and separators.

 LWS = [*WSP CRLF] 1*WSP ; linear whitespace
5662 SWS = [LWS] ; sep whitespace

5663 To separate the header name from the rest of value, a colon is used, which, by the above rule, allows
5664 whitespace before, but no line break, and whitespace after, including a linebreak. The HCOLON defines
5665 this construct.

5666 HCOLON = *(SP / HTAB) ":" SWS

5667 The TEXT-UTF8 rule is only used for descriptive field contents and values that are not intended to
5668 be interpreted by the message parser. Words of *TEXT-UTF8 contain characters from the UTF-8 charset
5669 (RFC 2279 [7]). The TEXT-UTF8-TRIM rule is used for descriptive field contents that are *not* quoted
5670 strings, where leading and trailing LWS is not meaningful. In this regard, SIP differs from HTTP, which
5671 uses the ISO 8859-1 character set.

 TEXT-UTF8-TRIM = 1*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
5672 UTF8-CONT = %x80-BF

5673 A CRLF is allowed in the definition of TEXT-UTF8-TRIM only as part of a header field continuation.
 5674 It is expected that the folding LWS will be replaced with a single SP before interpretation of the TEXT-
 5675 UTF8-TRIM value.

5676 Hexadecimal numeric characters are used in several protocol elements. Some elements (authentication)
 5677 force hex alphas to be lower case.

5678 LHEX = DIGIT / %x61-66 ;lowercase a-f

5679 Many SIP header field values consist of words separated by LWS or special characters. Unless otherwise
 5680 stated, tokens are case-insensitive. These special characters MUST be in a quoted string to be used within a
 5681 parameter value. The word construct is used in Call-ID to allow most separators to be used.

```

token      = 1*(alphanum / "-" / "." / "!" / "%" / "*" /
              / "_" / "+" / "=" / "()" / "[]" / "[]" / "[]" )
separators = "(" / ")" / "<" / ">" / "@" /
              ; / "," / ":" / "\" / DQUOTE /
              "/" / "[" / "]" / "?" / "=" /
              "{" / "}" / SP / HTAB
word       = 1*(alphanum / "-" / "." / "!" / "%" / "*" /
              / "_" / "+" / "=" / "()" / "[]" / "[]" / "[]" /
              "(" / ")" / "<" / ">" /
              ":" / "\" / DQUOTE /
              "/" / "[" / "]" / "?" /
              "{" / "}" )
  
```

5682
 5683 When tokens are used or separators are used between elements, whitespace is often allowed before or
 5684 after these characters:

```

STAR      = SWS "*" SWS ; asterisk
SLASH     = SWS "/" SWS ; slash
EQUAL     = SWS "=" SWS ; equal
LPAREN    = SWS "(" SWS ; left parenthesis
RPAREN    = SWS ")" SWS ; right parenthesis
RAQUOT    = ">" SWS ; right angle quote
LAQUOT    = SWS "<"; left angle quote
COMMA     = SWS "," SWS ; comma
SEMI      = SWS ";" SWS ; semicolon
COLON     = SWS ":" SWS ; colon
LDQUOT    = SWS DQUOTE; open double quotation mark
RDQUOT    = DQUOTE SWS ; close double quotation mark
  
```

5685
 5686 Comments can be included in some SIP header fields by surrounding the comment text with parentheses.
 5687 Comments are only allowed in fields containing "comment" as part of their field value definition. In all other
 5688 fields, parentheses are considered part of the field value.

```

comment   = LPAREN *(ctext / quoted-pair / comment) RPAREN
ctext     = %x21-27 / %x2A-5B / %x5D-7E / UTF8-NONASCII
           / LWS
  
```

5689

5690 ctext includes all chars except left and right parens and backslash. A string of text is parsed as a single
 5691 word if it is quoted using double-quote marks. In quoted strings, quotation marks (") and backslashes (\)
 5692 need to be escaped.

```

    quoted-string = SWS DQUOTE *(qdtex / quoted-pair) DQUOTE
    qdtex        = LWS / %x21 / %x23-5B / %x5D-7E
    5693          / UTF8-NONASCII
  
```

5694 The backslash character ("`) MAY be used as a single-character quoting mechanism only within quoted-
 5695 string and comment constructs. Unlike HTTP/1.1, the characters CR and LF cannot be escaped by this
 5696 mechanism to avoid conflict with line folding and header separation.

```

    quoted-pair = "\" (%x00-09 / %x0B-0C
    5697          / %x0E-7F)
  
```

```

    SIP-URI      = "sip:" [ userinfo "@" ] hostport
                  uri-parameters [ headers ]
    SIPS-URI     = "sips:" [ userinfo "@" ] hostport
                  uri-parameters [ headers ]
    userinfo     = [ user / telephone-subscriber [ ":" password ] ]
    user         = *( unreserved / escaped / user-unreserved )
    user-unreserved = "&" / "=" / "+" / "$" / "," / ";" / "?" / "/"
    password     = *( unreserved / escaped /
    5698          "&" / "=" / "+" / "$" / "," )
    hostport     = host [ ":" port ]
    host         = hostname / IPv4address / IPv6reference
    hostname     = *( domainlabel "." ) toplabel [ "." ]
    domainlabel  = alphanum
                  / alphanum *( alphanum / "-" ) alphanum
    5699 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*4HEXDIG
    5699 port     = 1*DIGIT
  
```

5700 The BNF for telephone-subscriber can be found in RFC 2806 [9]. Note, however, that any characters
 5701 allowed there that are not allowed in the user part of the SIP URI MUST be escaped.

uri-parameters = *(";" uri-parameter)
 uri-parameter = transport-param / user-param / method-param
 / ttl-param / maddr-param / lr-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
 lr-param = "lr"
 other-param = pname ["=" pvalue]
 pname = 1*paramchar
 pvalue = 1*paramchar
 paramchar = param-unreserved / unreserved / escaped
 5702 param-unreserved = "[/]" / "/" / "." / "&" / "+" / "\$"

 headers = "?" header *("&" header)
 header = hname "=" hvalue
 hname = 1*(hnv-unreserved / unreserved / escaped)
 hvalue = *(hnv-unreserved / unreserved / escaped)
 5703 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-URI / SIPS-URI / absoluteURI
 absoluteURI = scheme ":" (hier-part / opaque-part)
 hier-part = (net-path / abs-path) ["?" query]
 net-path = "//" authority [abs-path]
 abs-path = "/" path-segments
 opaque-part = uric-no-slash *uric
 uric = reserved / unreserved / escaped
 uric-no-slash = unreserved / escaped / "," / "?" / ":" / "@"
 / "&" / "=" / "+" / "\$" / ";"
 path-segments = segment *("/" segment)
 segment = *pchar *(";" param)
 param = *pchar
 pchar = unreserved / escaped /
 "." / "@" / "&" / "=" / "+" / "\$" / ";"
 scheme = ALPHA *(ALPHA / DIGIT / "+" / "-" / ".")
 authority = srvr / reg-name
 srvr = [[userinfo "@"] hostport]
 reg-name = 1*(unreserved / escaped / "\$" / "
 / ";" / ":" / "@" / "&" / "=" / "+")
 query = *uric
 SIP-Version = "SIP" "/" 1*DIGIT "." 1*DIGIT

5704

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
/ Min-Expires
/ Organization
/ Priority
/ Proxy-Authenticate
/ Proxy-Authorization
/ Proxy-Require
/ Record-Route
/ Reply-To
/ Require
/ Retry-After
/ Route
/ Server
/ Subject
/ Supported
/ Timestamp
/ To
/ Unsupported
/ User-Agent
/ Via
/ Warning
/ WWW-Authenticate
/ extension-header) CRLF

5705

INVITE_m = %x49.4E.56.49.54.45 ; INVITE in caps
 ACK_m = %x41.43.4B ; ACK in caps
 OPTIONSm = %x4F.50.54.49.4F.4E.53 ; OPTIONS in caps
 BYE_m = %x42.59.45 ; BYE in caps
 CANCEL_m = %x43.41.4E.43.45.4C ; CANCEL in caps
 REGISTER_m = %x52.45.47.49.53.54.45.52 ; REGISTER in caps
 Method = INVITE_m / ACK_m / OPTIONSm / BYE_m
 / CANCEL_m / REGISTER_m
 / extension-method
 extension-method = token
 Response = Status-Line
 *(message-header)
 CRLF
 5706 [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
 extension-code = 3DIGIT
 Reason-Phrase = *(reserved / unreserved / escaped
 5707 / UTF8-NONASCII / UTF8-CONT / SP / HTAB)

Informational = "100" ; Trying
 / "180" ; Ringing
 / "181" ; Call Is Being Forwarded
 5708 / "182" ; Queued
 / "183" ; Session Progress

5709 Success = "200" ; OK

Redirection = "300" ; Multiple Choices
 / "301" ; Moved Permanently
 / "302" ; Moved Temporarily
 / "305" ; Use Proxy
 5710 / "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
/ "410" ; Gone
/ "413" ; Request Entity Too Large
/ "414" ; Request-URI Too Large
/ "415" ; Unsupported Media Type
/ "416" ; Unsupported URI Scheme
/ "420" ; Bad Extension
/ "421" ; Extension Required
/ "423" ; Interval Too Brief
/ "480" ; Temporarily not available
/ "481" ; Call Leg/Transaction Does Not Exist
/ "482" ; Loop Detected
/ "483" ; Too Many Hops
/ "484" ; Address Incomplete
/ "485" ; Ambiguous
/ "486" ; Busy Here
/ "487" ; Request Terminated
/ "488" ; Not Acceptable Here
/ "491" ; Request Pending
5711 / "493" ; Undecipherable

Server-Error = "500" ; Internal Server Error
/ "501" ; Not Implemented
/ "502" ; Bad Gateway
/ "503" ; Service Unavailable
/ "504" ; Server Time-out
/ "505" ; SIP Version not supported
5712 / "513" ; Message Too Large

Global-Failure = "600" ; Busy Everywhere
/ "603" ; Decline
/ "604" ; Does not exist anywhere
5713 / "606" ; Not Acceptable

Accept = "Accept" HCOLON
 (accept-range *(COMMA accept-range))
 accept-range = media-range *(SEMI accept-param)
 media-range = ("*"/*"
 / (m-type SLASH "*")
 / (m-type SLASH m-subtype)
) *(SEMI m-parameter)
 accept-param = ("q" EQUAL qvalue) / generic-param
 qvalue = ("0" ["." 0*3DIGIT])
 / ("1" ["." 0*3("0")])
 generic-param = token [EQUAL gen-value]
 gen-value = token / host / quoted-string

 Accept-Encoding = "Accept-Encoding" HCOLON
 (encoding *(COMMA encoding))
 encoding = codings *(SEMI accept-param)
 codings = content-coding / "*"/*"
 content-coding = token

 Accept-Language = "Accept-Language" HCOLON
 (language *(COMMA language))
 language = language-range *(SEMI accept-param)
 language-range = ((1*8ALPHA *("-" 1*8ALPHA)) / "*"/*")

 Alert-Info = "Alert-Info" HCOLON alert-param *(COMMA alert-param)
 alert-param = LAQUOT absoluteURI RAQUOT *(SEMI generic-param)

 Allow = "Allow" HCOLON Method *(COMMA Method)

Authorization credentials = "Authorization" HCOLON credentials
 = ("Digest" LWS digest-response)
 / other-response
 digest-response = dig-resp *(COMMA dig-resp)
 dig-resp = 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 LDQUOT digest-uri-value RDQUOT
 digest-uri-value = rquest-uri ; Equal to 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
 nc-value = 8LHEX
 dresponse = "response" EQUAL request-digest
 request-digest = LDQUOT 32LHEX RDQUOT
 auth-param = auth-param-name EQUAL
 (token / quoted-string)
 auth-param-name = token
 other-response = auth-scheme LWS auth-param
 *(COMMA auth-param)
 5719 auth-scheme = token

 Authentication-Info = "Authentication-Info" HCOLON ainfo
 *(COMMA ainfo)
 ainfo = [nextnonce] / [message-qop]
 / [response-auth] / [cnonce]
 / [nonce-count]
 nextnonce = "nextnonce" EQUAL nonce-value
 response-auth = "rspauth" EQUAL response-digest
 5720 response-digest = LDQUOT *LHEX RDQUOT

 Call-ID = ("Call-ID" / "i") HCOLON callid
 5721 callid = word ["@" word]

 Call-Info = "Call-Info" HCOLON info *(COMMA info)
 info = LAQUOT absoluteURI RAQUOT *(SEMI info-param)
 info-param = ("purpose" EQUAL ("icon" / "info"
 5722 / "card" / token)) / generic-param

5723 Contact = ("Contact" / "m") HCOLON
(STAR / (contact-param *(COMMA contact-param)))
contact-param = (name-addr / addr-spec) *(SEMI contact-params)
name-addr = [display-name] LAQUOT addr-spec RAQUOT
addr-spec = SIP-URI / SIPS-URI / absoluteURI
display-name = *(token LWS)/ quoted-string

5724 contact-params = c-p-q / c-p-expires
/ contact-extension
c-p-q = "q" EQUAL qvalue
c-p-expires = "expires" EQUAL delta-seconds
contact-extension = generic-param
delta-seconds = 1*DIGIT

5725 Content-Disposition = "Content-Disposition" HCOLON
disp-type *(SEMI disp-param)
disp-type = "render" / "session" / "icon" / "alert"
/ disp-extension-token
disp-param = handling-param / generic-param
handling-param = "handling" EQUAL
("optional" / "required"
/ other-handling)
other-handling = token
disp-extension-token = token

5726 Content-Encoding = ("Content-Encoding" / "e") HCOLON
content-coding *(COMMA content-coding)

5727 Content-Language = "Content-Language" HCOLON
language-tag *(COMMA language-tag)
language-tag = primary-tag *("-" subtag)
primary-tag = 1*8ALPHA
subtag = 1*8ALPHA

5728 Content-Length = ("Content-Length" / "l") HCOLON 1*DIGIT

Content-Type = ("Content-Type" / "c") HCOLON media-type
 media-type = m-type SLASH m-subtype *(SEMI m-parameter)
 m-type = discrete-type / composite-type
 discrete-type = "text" / "image" / "audio" / "video"
 / "application" / extension-token
 composite-type = "message" / "multipart" / extension-token
 extension-token = ietf-token / x-token
 ietf-token = token
 x-token = "x-" token
 m-subtype = extension-token / iana-token
 iana-token = token
 m-parameter = m-attribute EQUAL m-value
 m-attribute = token
 m-value = token / quoted-string

5729

5730

CSeq = "CSeq" HCOLON 1*DIGIT LWS Method

Date = "Date" HCOLON SIP-date
 SIP-date = rfc1123-date
 rfc1123-date = wkday "," 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"
 / "Sep" / "Oct" / "Nov" / "Dec"

5731

Error-Info = "Error-Info" HCOLON error-uri *(COMMA error-uri)
 error-uri = LAQUOT absoluteURI RAQUOT *(SEMI generic-param)

5732

Expires = "Expires" HCOLON delta-seconds
 From = ("From" / "f") HCOLON from-spec
 from-spec = (name-addr / addr-spec)
 *(SEMI from-param)

from-param = tag-param / generic-param
 tag-param = "tag" EQUAL token

5733

In-Reply-To = "In-Reply-To" HCOLON callid *(COMMA callid)

5734

Max-Forwards = "Max-Forwards" HCOLON 1*DIGIT

5735

MIME-Version = "MIME-Version" HCOLON 1*DIGIT "." 1*DIGIT

5736

5737 Min-Expires = "Min-Expires" HCOLON delta-seconds

5738 Organization = "Organization" HCOLON [TEXT-UTF8-TRIM]

 Priority = "Priority" HCOLON priority-value

 priority-value = "emergency" / "urgent" / "normal"

 / "non-urgent" / other-priority

5739 other-priority = token

 Proxy-Authenticate = "Proxy-Authenticate" HCOLON challenge

 challenge = ("Digest" LWS digest-chn *(COMMA digest-chn))

 / other-challenge

 other-challenge = auth-scheme LWS auth-param

 *(COMMA auth-param)

 digest-chn = realm / [domain] / nonce

 / [opaque] / [stale] / [algorithm]

 / [qop-options] / [auth-param]

 realm = "realm" EQUAL 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 qop-value

 *(" " qop-value) RDQUOT

5740 qop-value = "auth" / "auth-int" / token

5741 Proxy-Authorization = "Proxy-Authorization" HCOLON credentials

 Proxy-Require = "Proxy-Require" HCOLON option-tag

 *(COMMA option-tag)

5742 option-tag = token

 Record-Route = "Record-Route" HCOLON rec-route *(COMMA rec-route)

 rec-route = name-addr *(SEMI rr-param)

5743 rr-param = generic-param

 Reply-To = "Reply-To" HCOLON rplyto-spec

 rplyto-spec = (name-addr / addr-spec)

 *(SEMI rplyto-param)

 rplyto-param = generic-param

5744 Require = "Require" HCOLON option-tag *(COMMA option-tag)

Retry-After = "Retry-After" HCOLON delta-seconds
 [comment] *(SEMI retry-param)
 5745 retry-param = ("duration" EQUAL delta-seconds)
 / generic-param

5746 Route = "Route" HCOLON route-param *(COMMA route-param)
 route-param = name-addr *(SEMI rr-param)

Server = "Server" HCOLON server-val *(LWS server-val)
 server-val = product / comment
 product = token [SLASH product-version]
 5747 product-version = token

5748 Subject = ("Subject" / "s") HCOLON [TEXT-UTF8-TRIM]

5749 Supported = ("Supported" / "k") HCOLON
 [option-tag *(COMMA option-tag)]

5750 Timestamp = "Timestamp" HCOLON 1*(DIGIT)
 ["." *(DIGIT)] [delay]
 delay = *(DIGIT) ["." *(DIGIT)]

5751 To = ("To" / "t") HCOLON (name-addr
 / addr-spec) *(SEMI to-param)
 to-param = tag-param / generic-param

5752 Unsupported = "Unsupported" HCOLON option-tag *(COMMA option-tag)

5753 User-Agent = "User-Agent" HCOLON server-val *(LWS server-val)

Via = ("Via" / "v") HCOLON via-param *(COMMA via-param)
 via-param = sent-protocol LWS sent-by *(SEMI via-params)
 via-params = via-ttl / via-maddr
 / via-received / via-branch
 / via-extension
 via-ttl = "ttl" EQUAL ttl
 via-maddr = "maddr" EQUAL host
 via-received = "received" EQUAL (IPv4address / IPv6address)
 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]
 5754 ttl = 1*3DIGIT ; 0 to 255

Warning = "Warning" HCOLON warning-value *(COMMA warning-value)
warning-value = warn-code SP warn-agent SP warn-text
warn-code = 3DIGIT
warn-agent = hostport / pseudonym
; the name or pseudonym of the server adding
; the Warning header, for use in debugging
warn-text = quoted-string
5755 pseudonym = token

5756 WWW-Authenticate = "WWW-Authenticate" HCOLON challenge

extension-header = header-name HCOLON header-value
header-name = token
5757 header-value = *(TEXT-UTF8char / UTF8-CONT / LWS)

5758 message-body = *OCTET

5759 **26 Security Considerations: Threat Model and Security Usage Recommen-** 5760 **datations**

5761 SIP is not an easy protocol to secure. Its use of intermediaries, its multi-faceted trust relationships, its
5762 expected usage between elements with no trust at all, and its user-to-user operation make security far from
5763 trivial. Security solutions are needed that are deployable today, without extensive coordination, in a wide
5764 variety of environments and usages. In order to meet these diverse needs, several distinct mechanisms
5765 applicable to different aspects and usages of SIP will be required.

5766 Note that the security of SIP signaling itself has no bearing on the security of protocols used in concert
5767 with SIP such as RTP, or with the security implications of any specific bodies SIP might carry (although
5768 MIME security plays a substantial role in securing SIP). Any media associated with a session can be en-
5769 crypted end-to-end independently of any associated SIP signaling. Media encryption is outside the scope of
5770 this document.

5771 The considerations that follow first examine a set of classic threat models that broadly identify the
5772 security needs of SIP. The set of security services required to address these threats is then detailed, followed
5773 by an explanation of several security mechanisms that can be used to provide these services. Next, the
5774 requirements for implementers of SIP are enumerated, along with exemplary deployments in which these
5775 security mechanisms could be used to improve the security of SIP. Some notes on privacy conclude this
5776 section.

5777 **26.1 Attacks and Threat Models**

5778 This section details some threats that should be common to most deployments of SIP. These threats have
5779 been chosen specifically to illustrate each of the security services that SIP requires.

5780 The following examples by no means provide an exhaustive list of the threats against SIP; rather, these
5781 are "classic" threats that demonstrate the need for particular security services that can potentially prevent
5782 whole categories of threats.

5783 These attacks assume an environment in which attackers can potentially read any packet on the network
5784 - it is anticipated that SIP will frequently be used on the public Internet. Attackers on the network may be
5785 able to modify packets (perhaps at some compromised intermediary). Attackers may wish to steal services,
5786 eavesdrop on communications, or disrupt sessions.

5787 **26.1.1 Registration Hijacking**

5788 The SIP registration mechanism allows a user agent to identify itself to a registrar as a device at which a
5789 user (designated by an address of record) is located. A registrar assesses the identity asserted in the **From**
5790 header field of a **REGISTER** message to determine whether this request can modify the contact addresses
5791 associated with the address-of-record in the **To** header field. While these two fields are frequently the same,
5792 there are many valid deployments in which a third-party may register contacts on a user's behalf.

5793 The **From** header field of a SIP request, however, can be modified arbitrarily by the owner of a UA, and
5794 this opens the door to malicious registrations. An attacker that successfully impersonates a party authorized
5795 to change contacts associated with an address-of-record could, for example, de-register all existing contacts
5796 for a URI and then register their own device as the appropriate contact address, thereby directing all requests
5797 for the affected user to the attacker's device.

5798 This threat belongs to a family of threats that rely on the absence of cryptographic assurance of a re-
5799 quest's originator. Any SIP UAS that represents a valuable service (a gateway that interworks SIP requests
5800 with traditional telephone calls, for example) might want to control access to its resources by authenticating
5801 requests that it receives. Even end-user UAs, for example SIP phones, have an interest in ascertaining the
5802 identities of originators of requests.

5803 This threat demonstrates the need for security services that enable SIP entities to authenticate the origi-
5804 nators of requests.

5805 **26.1.2 Impersonating a Server**

5806 The domain to which a request is destined is generally specified in the **Request-URI**. UAs commonly
5807 contact a server in this domain directly in order to deliver a request. However, there is always a possibility
5808 that an attacker could impersonate the remote server, and that the UA's request could be intercepted by some
5809 other party.

5810 For example, consider a case in which a redirect server at one domain, *chicago.com*, impersonates a
5811 redirect server at another domain, *biloxi.com*. A user agent sends a request to *biloxi.com*, but the redirect
5812 server at *chicago.com* answers with a forged response that has appropriate SIP header fields for a response
5813 from *biloxi.com*. The forged contact addresses in the redirection response could direct the originating UA
5814 to inappropriate or insecure resources, or simply prevent requests for *biloxi.com* from succeeding.

5815 This family of threats has a vast membership, many of which are critical. As a converse to the registration
5816 hijacking threat, consider the case in which a registration sent to *biloxi.com* is intercepted by *chicago.com*,
5817 which replies to the intercepted registration with a forged 301 (Moved Permanently) response. This response
5818 might seem to come from *biloxi.com* yet designate *chicago.com* as the appropriate registrar. All future
5819 **REGISTER** requests from the originating UA would then go to *chicago.com*.

5820 Prevention of this threat requires a means by which UAs can authenticate the servers to whom they send
5821 requests.

5822 **26.1.3 Tampering with Message Bodies**

5823 As a matter of course, SIP UAs route requests through trusted proxy servers. Regardless of how that trust is
5824 established (authentication of proxies is discussed elsewhere in this section), a UA may trust a proxy server
5825 to route a request, but not to inspect or possibly modify the bodies contained in that request.

5826 Consider a UA that is using SIP message bodies to communicate session encryption keys for a media
5827 session. Although it trusts the proxy server of the domain it is contacting to deliver signaling properly, it
5828 may not want the administrators of that domain to be capable of decrypting any subsequent media session.
5829 Worse yet, if the proxy server were actively malicious, it could modify the session key, either acting as a
5830 man-in-the-middle, or perhaps changing the security characteristics requested by the originating UA.

5831 This family of threats applies not only to session keys, but to most conceivable forms of content car-
5832 ried end-to-end in SIP. These might include MIME bodies that should be rendered to the user, SDP, or
5833 encapsulated telephony signals, among others. Attackers might attempt to modify SDP bodies, for example,
5834 in order to point RTP media streams to a wiretapping device in order to eavesdrop on subsequent voice
5835 communications.

5836 Also note that some header fields in SIP are meaningful end-to-end, for example, **Subject**. UAs might
5837 be protective of these header fields as well as bodies (a malicious intermediary changing the **Subject** header
5838 field might make an important request appear to be spam, for example). However, since many header fields
5839 are legitimately inspected or altered by proxy servers as a request is routed, not all header fields should be
5840 secured end-to-end.

5841 For these reasons, the UA might want to secure SIP message bodies, and in some limited cases header
5842 fields, end-to-end. The security services required for bodies include confidentiality, integrity, and authen-
5843 tication. These end-to-end services should be independent of the means used to secure interactions with
5844 intermediaries such as proxy servers.

5845 **26.1.4 Tearing Down Sessions**

5846 Once a dialog has been established by initial messaging, subsequent requests can be sent that modify the
5847 state of the dialog and/or session. It is critical that principals in a session can be certain that such requests
5848 are not forged by attackers.

5849 Consider a case in which a third-party attacker captures some initial messages in a dialog shared by two
5850 parties in order to learn the parameters of the session (**To** tag, **From** tag, and so forth) and then inserts a
5851 **BYE** request into the session. The attacker could opt to forge the request such that it seemed to come from
5852 either participant. Once the **BYE** is received by its target, the session will be torn down prematurely.

5853 Similar mid-session threats include the transmission of forged re-**INVITE**s that alter the session (possibly
5854 to reduce session security or redirect media streams as part of a wiretapping attack).

5855 The most effective countermeasure to this threat is the authentication of the sender of the **BYE**. In this
5856 instance, the recipient needs only know that the **BYE** came from the same party with whom the correspond-
5857 ing dialog was established (as opposed to ascertaining the absolute identity of the sender). Also, if the
5858 attacker is unable to learn the parameters of the session due to confidentiality, it would not be possible to
5859 forge the **BYE**. However, some intermediaries (like proxy servers) will need to inspect those parameters as
5860 the session is established.

5861 **26.1.5 Denial of Service and Amplification**

5862 Denial-of-service attacks focus on rendering a particular network element unavailable, usually by directing
5863 an excessive amount of network traffic at its interfaces. A distributed denial-of-service attack allows one
5864 network user to cause multiple network hosts to flood a target host with a large amount of network traffic.

5865 In many architectures, SIP proxy servers face the public Internet in order to accept requests from world-
5866 wide IP endpoints. SIP creates a number of potential opportunities for distributed denial-of-service attacks
5867 that must be recognized and addressed by the implementers and operators of SIP systems.

5868 Attackers can create bogus requests that contain a falsified source IP address and a corresponding *Via*
5869 header field that identify a targeted host as the originator of the request and then send this request to a large
5870 number of SIP network elements, thereby using hapless SIP UAs or proxies to generate denial-of-service
5871 traffic aimed at the target.

5872 Similarly, attackers might use falsified *Route* header field values in a request that identify the target
5873 host and then send such messages to forking proxies that will amplify messaging sent to the target. *Record-
5874 Route* could be used to similar effect when the attacker is certain that the SIP dialog initiated by the request
5875 will result in numerous transactions originating in the backwards direction.

5876 A number of denial-of-service attacks open up if *REGISTER* requests are not properly authenticated
5877 and authorized by registrars. Attackers could de-register some or all users in an administrative domain,
5878 thereby preventing these users from being invited to new sessions. An attacker could also register a large
5879 number of contacts designating the same host for a given address-of-record in order to use the registrar and
5880 any associated proxy servers as amplifiers in a denial-of-service attack. Attackers might also attempt to
5881 deplete available memory and disk resources of a registrar by registering huge numbers of bindings.

5882 The use of multicast to transmit SIP requests can greatly increase the potential for denial-of-service
5883 attacks.

5884 These problems demonstrate a general need to define architectures that minimize the risks of denial-of-
5885 service, and the need to be mindful in recommendations for security mechanisms of this class of attacks.

5886 **26.2 Security Mechanisms**

5887 From the threats described above, we gather that the fundamental security services required for the SIP
5888 protocol are: preserving the confidentiality and integrity of messaging, preventing replay attacks or message
5889 spoofing, providing for the authentication and privacy of the participants in a session, and preventing denial-
5890 of-service attacks. Bodies within SIP messages separately require the security services of confidentiality,
5891 integrity, and authentication.

5892 Rather than defining new security mechanisms specific to SIP, SIP reuses wherever possible existing
5893 security models derived from the HTTP and SMTP space.

5894 Full encryption of messages provides the best means to preserve the confidentiality of signaling - it
5895 can also guarantee that messages are not modified by any malicious intermediaries. However, SIP requests
5896 and responses cannot be naively encrypted end-to-end in their entirety because message fields such as the
5897 *Request-URI*, *Route*, and *Via* need to be visible to proxies in most network architectures so that SIP
5898 requests are routed correctly. Note that proxy servers need to modify some features of messages as well (such
5899 as adding *Via* header field values) in order for SIP to function. Proxy servers must therefore be trusted, to
5900 some degree, by SIP UAs. To this purpose, low-layer security mechanisms for SIP are recommended, which
5901 encrypt the entire SIP requests or responses on the wire on a hop-by-hop basis, and that allow endpoints to
5902 verify the identity of proxy servers to whom they send requests.

5903 SIP entities also have a need to identify one another in a secure fashion. When a SIP endpoint asserts

5904 the identity of its user to a peer UA or to a proxy server, that identity should in some way be verifiable. A
5905 cryptographic authentication mechanism is provided in SIP to address this requirement.

5906 An independent security mechanism for SIP message bodies supplies an alternative means of end-to-end
5907 mutual authentication, as well as providing a limit on the degree to which user agents must trust intermedi-
5908 aries.

5909 **26.2.1 Transport and Network Layer Security**

5910 Transport or network layer security encrypts signaling traffic, guaranteeing message confidentiality and
5911 integrity.

5912 Oftentimes, certificates are used in the establishment of lower-layer security, and these certificates can
5913 also be used to provide a means of authentication in many architectures.

5914 Two popular alternatives for providing security at the transport and network layer are, respectively,
5915 TLS [25] and IPSec [26].

5916 IPSec is a set of network-layer protocol tools that collectively can be used as a secure replacement for
5917 traditional IP (Internet Protocol). IPSec is most commonly used in architectures in which a set of hosts or
5918 administrative domains have an existing trust relationship with one another. IPSec is usually implemented
5919 at the operating system level in a host, or on a security gateway that provides confidentiality and integrity
5920 for all traffic it receives from a particular interface (as in a VPN architecture). IPSec can also be used on a
5921 hop-by-hop basis.

5922 In many architectures IPSec does not require integration with SIP applications; IPSec is perhaps best
5923 suited to deployments in which adding security directly to SIP hosts would be arduous. UAs that have a
5924 pre-shared keying relationship with their first-hop proxy server are also good candidates to use IPSec. Any
5925 deployment of IPSec for SIP would require an IPSec profile describing the protocol tools that would be
5926 required to secure SIP. No such profile is given in this document.

5927 TLS provides transport-layer security over connection-oriented protocols (for the purposes of this docu-
5928 ment, TCP); "tls" (signifying TLS over TCP) can be specified as the desired transport protocol within a Via
5929 header field value or a SIP-URI. TLS is most suited to architectures in which hop-by-hop security is required
5930 between hosts with no pre-existing trust association. For example, Alice trusts her local proxy server, which
5931 after a certificate exchange decides to trust Bob's local proxy server, which Bob trusts, hence Bob and Alice
5932 can communicate securely.

5933 TLS must be tightly coupled with a SIP application. Note that transport mechanisms are specified on
5934 a hop-by-hop basis in SIP, thus a UA that sends requests over TLS to a proxy server has no assurance that
5935 TLS will be used end-to-end.

5936 The TLS_RSA_WITH_AES_128_CBC_SHA ciphersuite MUST be supported at a minimum by imple-
5937 menters when TLS is used in a SIP application. For purposes of backwards compatibility, proxy servers,
5938 redirect servers, and registrars SHOULD support TLS_RSA_WITH_3DES_EDE_CBC_SHA. Implementers
5939 MAY also support any other ciphersuite.

5940 **26.2.2 SIPS URI Scheme**

5941 The SIPS URI scheme adheres to the syntax of the SIP URI (described in 19), although the scheme string is
5942 "sips" rather than "sip". The semantics of SIPS are very different from the SIP URI, however. SIPS allows
5943 resources to specify that they should be reached securely.

5944 A SIPS URI can be used as an address-of-record for a particular user - the URI by which the user is
5945 canonically known (on their business cards, in the From header field of their requests, in the To header field

5946 of REGISTER requests). When used as the Request-URI of a request, the SIPS scheme signifies that
5947 each hop over which the request is forwarded, until the request reaches the SIP entity responsible for the
5948 domain portion of the Request-URI, must be secured with TLS; once it reaches the domain in question it
5949 is handled in accordance with local security and routing policy, quite possibly using TLS for any last hop to
5950 a UAS. When used by the originator of a request (as would be the case if they employed a SIPS URI as the
5951 address-of-record of the target), SIPS dictates that the entire request path to the target domain be so secured.

5952 The SIPS scheme is applicable to many of the other ways in which SIP URIs are used in SIP today in
5953 addition to the Request-URI, including in addresses-of-record, contact addresses (the contents of Contact
5954 headers, including those of REGISTER methods), and Route headers. In each instance, the SIPS URI
5955 scheme allows these existing fields to designate secure resources. The manner in which a SIPS URI is
5956 dereferenced in any of these contexts has its own security properties which are detailed in [4].

5957 The use of SIPS in particular entails that mutual TLS authentication SHOULD be employed, as SHOULD
5958 the ciphersuite TLS_RSA_WITH_AES_128_CBC_SHA. Certificates received in the authentication process
5959 SHOULD be validated with root certificates held by the client; failure to validate a certificate SHOULD result
5960 in the failure of the request.

5961 motivationNote that in the SIPS URI scheme, transport is independent of TLS, and thus “sips:alice@atlanta.com;transport=tl
5962 and “sips:alice@atlanta.com;transport=sctp” are both valid (although note that UDP is not a valid transport
5963 for SIPS). The use of “transport=tl” has consequently been deprecated, partly because it was specific to a
5964 single hop of the request. This is a change since RFC 2543.

5965 Users that distribute a SIPS URI as an address-of-record may elect to operate devices that refuse requests
5966 over insecure transports.

5967 26.2.3 HTTP Authentication

5968 SIP provides a challenge capability, based on HTTP authentication, that relies on the 401 and 407 response
5969 codes as well as header fields for carrying challenges and credentials. Without significant modification, the
5970 reuse of the HTTP Digest authentication scheme in SIP allows for replay protection and one-way authenti-
5971 cation.

5972 The usage of Digest authentication in SIP is detailed in Section 22.

5973 26.2.4 S/MIME

5974 As is discussed above, encrypting entire SIP messages end-to-end for the purpose of confidentiality is not
5975 appropriate because network intermediaries (like proxy servers) need to view certain header fields in order
5976 to route messages correctly, and if these intermediaries are excluded from security associations, then SIP
5977 messages will essentially be non-routable.

5978 However, S/MIME allows SIP UAs to encrypt MIME bodies within SIP, securing these bodies end-to-
5979 end without affecting message headers. S/MIME can provide end-to-end confidentiality and integrity for
5980 message bodies, as well as mutual authentication. It is also possible to use S/MIME to provide a form of
5981 integrity and confidentiality for SIP header fields through SIP message tunneling.

5982 The usage of S/MIME in SIP is detailed in Section 23.

5983 **26.3 Implementing Security Mechanisms**

5984 **26.3.1 Requirements for Implementers of SIP**

5985 Proxy servers, redirect servers, and registrars **MUST** implement TLS, and **MUST** support both mutual and
5986 one-way authentication. It is strongly **RECOMMENDED** that UAs be capable initiating TLS; UAs **MAY**
5987 also be capable of acting as a TLS server. Proxy servers, redirect servers, and registrars **SHOULD** possess
5988 a site certificate whose subject corresponds to their canonical hostname. UAs **MAY** have certificates of
5989 their own for mutual authentication with TLS, but no provisions are set forth in this document for their
5990 use. All SIP elements that support TLS **MUST** have a mechanism for validating certificates received during
5991 TLS negotiation; this entails possession of one or more root certificates issued by certificate authorities
5992 (preferably well-known distributors of site certificates comparable to those that issue root certificates for
5993 web browsers).

5994 All SIP elements that support TLS **MUST** also support the SIPS URI scheme.

5995 Proxy servers, redirect servers, registrars, and UAs **MAY** also implement IPsec or other lower-layer
5996 security protocols.

5997 When a UA attempts to contact a proxy server, redirect server, or registrar, the UAC **SHOULD** initiate a
5998 TLS connection over which it will send SIP messages. In some architectures, UASs **MAY** receive requests
5999 over such TLS connections as well.

6000 Proxy servers, redirect servers, registrars, and UAs **MUST** implement Digest Authorization, encompass-
6001 ing all of the aspects required in 22. Proxy servers, redirect servers, and registrars **SHOULD** be configured
6002 with at least one Digest realm, and at least one “realm” string supported by a given server **SHOULD** corre-
6003 spond to the server’s hostname or domainname.

6004 UAs **MAY** support the signing and encrypting of MIME bodies, and transference of credentials with
6005 S/MIME as described in Section 23. If a UA holds one or more root certificates of certificate authorities
6006 in order to validate certificates for TLS or IPsec, it **SHOULD** be capable of reusing these to verify S/MIME
6007 certificates, as appropriate. A UA **MAY** hold root certificates specifically for validating S/MIME certificates.

6008 Note that it is anticipated that future security extensions may upgrade the normative strength associated with
6009 S/MIME as S/MIME implementations appear and the problem space becomes better understood.

6010 **26.3.2 Security Solutions**

6011 The operation of these security mechanisms in concert can follow the existing web and email security models
6012 to some degree. At a high level, UAs authenticate themselves to servers (proxy servers, redirect servers, and
6013 registrars) with a Digest username and password; servers authenticate themselves to UAs one hop away, or
6014 to another server one hop away (and vice versa), with a site certificate delivered by TLS.

6015 On a peer-to-peer level, UAs trust the network to authenticate one another ordinarily; however, S/MIME
6016 can also be used to provide direct authentication when the network does not, or if the network itself is not
6017 trusted.

6018 The following is an illustrative example in which these security mechanisms are used by various UAs
6019 and servers to prevent the sorts of threats described in Section 26.1. While implementers and network
6020 administrators **MAY** follow the normative guidelines given in the remainder of this section, these are provided
6021 only as example implementations.

6022 **26.3.2.1 Registration** When a UA comes online and registers with its local administrative domain, it
6023 **SHOULD** establish a TLS connection with its registrar (Section 10 describes how the UA reaches its reg-

6024 istrar). The registrar SHOULD offer a certificate to the UA, and the site identified by the certificate MUST
6025 correspond with the domain in which the UA intends to register; for example, if the UA intends to register
6026 the address-of-record 'alice@atlanta.com', the site certificate must identify a host within the atlanta.com
6027 domain (such as sip.atlanta.com). When it receives the TLS Certificate message, the UA SHOULD verify the
6028 certificate and inspect the site identified by the certificate. If the certificate is invalid, revoked, or if it does
6029 not identify the appropriate party, the UA MUST NOT send the REGISTER message and otherwise proceed
6030 with the registration.

6031 When a valid certificate has been provided by the registrar, the UA knows that the registrar is not an attacker
6032 who might redirect the UA, steal passwords, or attempt any similar attacks.

6033 The UA then creates a REGISTER request that SHOULD be addressed to a Request-URI correspond-
6034 ing to the site certificate received from the registrar. When the UA sends the REGISTER request over
6035 the existing TLS connection, the registrar SHOULD challenge the request with a 401 (Proxy Authentication
6036 Required) response. The "realm" parameter within the Proxy-Authenticate header field of the response
6037 SHOULD correspond to the domain previously given by the site certificate. When the UAC receives the
6038 challenge, it SHOULD either prompt the user for credentials or take an appropriate credential from a keyring
6039 corresponding to the "realm" parameter in the challenge. The username of this credential SHOULD corre-
6040 spond with the "userinfo" portion of the URI in the To header field of the REGISTER request. Once the
6041 Digest credentials have been inserted into an appropriate Proxy-Authorization header field, the REGIS-
6042 TER should be resubmitted to the registrar.

6043 Since the registrar requires the user agent to authenticate itself, it would be difficult for an attacker to forge REG-
6044 ISTER requests for the user's address-of-record. Also note that since the REGISTER is sent over a confidential
6045 TLS connection, attackers will not be able to intercept the REGISTER to record credentials for any possible replay
6046 attack.

6047 Once the registration has been accepted by the registrar, the UA SHOULD leave this TLS connection
6048 open provided that the registrar also acts as the proxy server to which requests are sent for users in this
6049 administrative domain. The existing TLS connection will be reused to deliver incoming requests to the UA
6050 that has just completed registration.

6051 Because the UA has already authenticated the server on the other side of the TLS connection, all requests that
6052 come over this connection are known to have passed through the proxy server - attackers cannot create spoofed
6053 requests that appear to have been sent through that proxy server.

6054 **26.3.2.2 Interdomain Requests** Now let's say that Alice's UA would like to initiate a session with a user
6055 in a remote administrative domain, namely "bob@biloxi.com". We will also say that the local administrative
6056 domain (atlanta.com) has a local outbound proxy.

6057 The proxy server that handles inbound requests for an administrative domain MAY also act as a local
6058 outbound proxy; for simplicity's sake we'll assume this to be the case for atlanta.com (otherwise the user
6059 agent would initiate a new TLS connection to a separate server at this point). Assuming that the client has
6060 completed the registration process described in the preceding section, it SHOULD reuse the TLS connection
6061 to the local proxy server when it sends an INVITE request to another user. The UA SHOULD reuse cached
6062 credentials in the INVITE to avoid prompting the user unnecessarily.

6063 When the local outbound proxy server has validated the credentials presented by the UA in the INVITE,
6064 it SHOULD inspect the Request-URI to determine how the message should be routed (see [4]). If the
6065 "domainname" portion of the Request-URI had corresponded to the local domain (atlanta.com) rather than
6066 biloxi.com, then the proxy server would have consulted its location service to determine how best to reach
6067 the requested user.

6068 Had "alice@atlanta.com" been attempting to contact, say, "alex@atlanta.com", the local proxy would have
6069 proxied to the request to the TLS connection Alex had established with the registrar when he registered. Since
6070 Alex would receive this request over his authenticated channel, he would be assured that Alice's request had been
6071 authorized by the proxy server of the local administrative domain.

6072 However, in this instance the Request-URI designates a remote domain. The local outbound proxy
6073 server at atlanta.com SHOULD therefore establish a TLS connection with the remote proxy server at biloxi.com.
6074 Since both of the participants in this TLS connection are servers that possess site certificates, mutual TLS
6075 authentication SHOULD occur. Each side of the connection SHOULD verify and inspect the certificate of
6076 the other, noting the domain name that appears in the certificate for comparison with the header fields of
6077 SIP messages. The atlanta.com proxy server, for example, SHOULD verify at this stage that the certificate
6078 received from the remote side corresponds with the biloxi.com domain. Once it has done so, and TLS ne-
6079 gotiation has completed, resulting in a secure channel between the two proxies, the atlanta.com proxy can
6080 forward the INVITE request to biloxi.com.

6081 The proxy server at biloxi.com SHOULD inspect the certificate of the proxy server at atlanta.com in turn
6082 and compare the domain asserted by the certificate with the "domainname" portion of the From header field
6083 in the INVITE request. The biloxi proxy MAY have a strict security policy that requires it to reject requests
6084 that do not match the administrative domain from which they have been proxied.

6085 Such security policies could be instituted to prevent the SIP equivalent of SMTP 'open relays' that are frequently
6086 exploited to generate spam.

6087 This policy, however, only guarantees that the request came from the domain it ascribes to itself; it
6088 does not allow biloxi.com to ascertain how atlanta.com authenticated Alice. Only if biloxi.com has some
6089 other way of knowing atlanta.com's authentication policies could it possibly ascertain how Alice proved her
6090 identity. biloxi.com might then institute an even stricter policy that forbids requests that come from domains
6091 that are not known administratively to share a common authentication policy with biloxi.com.

6092 Once the INVITE has been approved by the biloxi proxy, the proxy server SHOULD identify the existing
6093 TLS channel, if any, associated with the user targeted by this request (in this case "bob@biloxi.com"). The
6094 INVITE should be proxied through this channel to Bob. Since the request is received over a TLS connection
6095 that had previously been authenticated as the biloxi proxy, Bob knows that the From header field was not
6096 tampered with and that atlanta.com has validated Alice, although not necessarily whether or not to trust
6097 Alice's identity.

6098 Before they forward the request, both proxy servers SHOULD add a Record-Route header field to the
6099 request so that all future requests in this dialog will pass through the proxy servers. The proxy servers can
6100 thereby continue to provide security services for the lifetime of this dialog. If the proxy servers do not add
6101 themselves to the Record-Route, future messages will pass directly end-to-end between Alice and Bob
6102 without any security services (unless the two parties agree on some independent end-to-end security such
6103 as S/MIME). In this respect the SIP trapezoid model can provide a nice structure where conventions of
6104 agreement between the site proxies can provide a reasonably secure channel between Alice and Bob.

6105 An attacker preying on this architecture would, for example, be unable to forge a BYE request and insert it into
6106 the signaling stream between Bob and Alice because the attacker has no way of ascertaining the parameters of the
6107 session and also because the integrity mechanism transitively protects the traffic between Alice and Bob.

6108 **26.3.2.3 Peer to Peer Requests** Alternatively, consider a UA asserting the identity "carol@chicago.com"
6109 that has no local outbound proxy. When Carol wishes to send an INVITE to "bob@biloxi.com", her UA
6110 SHOULD initiate a TLS connection with the biloxi proxy directly (using the mechanism described in [4]
6111 to determine how to best to reach the given Request-URI). When her UA receives a certificate from the

6112 biloxi proxy, it SHOULD be verified normally before she passes her INVITE across the TLS connection.
6113 However, Carol has no means of proving her identity to the biloxi proxy, but she does have a CMS-detached
6114 signature over a “message/sip” body in the INVITE. It is unlikely in this instance that Carol would have any
6115 credentials in the biloxi.com realm, since she has no formal association with biloxi.com. The biloxi proxy
6116 MAY also have a strict policy that precludes it from even bothering to challenge requests that do not have
6117 biloxi.com in the “domainname” portion of the From header field - it treats these users as unauthenticated.

6118 The biloxi proxy has a policy for Bob that all non-authenticated requests should be redirected to the
6119 appropriate contact address registered against 'bob@biloxi.com', namely <sip:bob@192.0.2.4>. Carol
6120 receives the redirection response over the TLS connection she established with the biloxi proxy, so she
6121 trusts the veracity of the contact address.

6122 Carol SHOULD then establish a TCP connection with the designated address and send a new INVITE
6123 with a Request-URI containing the received contact address (recomputing the signature in the body as
6124 the request is readied). Bob receives this INVITE on an insecure interface, but his UA inspects and, in
6125 this instance, recognizes the From header field of the request and subsequently matches a locally cached
6126 certificate with the one presented in the signature of the body of the INVITE. He replies in similar fashion,
6127 authenticating himself to Carol, and a secure dialog begins.

6128 Sometimes firewalls or NATs in an administrative domain could preclude the establishment of a direct TCP
6129 connection to a UA. In these cases, proxy servers could also potentially relay requests to UAs in a way that has no
6130 trust implications (for example, forgoing an existing TLS connection and forwarding the request over cleartext TCP)
6131 as local policy dictates.

6132 **26.3.2.4 DoS Protection** In order to minimize the risk of a denial-of-service attack against architectures
6133 using these security solutions, implementers should take note of the following guidelines.

6134 When the host on which a SIP proxy server is operating is routable from the public Internet, it SHOULD
6135 be deployed in an administrative domain with defensive operational policies (blocking source-routed traffic,
6136 preferably filtering ping traffic). Both TLS and IPSec can also make use of bastion hosts at the edges of
6137 administrative domains that participate in the security associations to aggregate secure tunnels and sockets.
6138 These bastion hosts can also take the brunt of denial-of-service attacks, ensuring that SIP hosts within the
6139 administrative domain are not encumbered with superfluous messaging.

6140 No matter what security solutions are deployed, floods of messages directed at proxy servers can lock up
6141 proxy server resources and prevent desirable traffic from reaching its destination. There is a computational
6142 expense associated with processing a SIP transaction at a proxy server, and that expense is greater for
6143 stateful proxy servers than it is for stateless proxy servers. Therefore, stateful proxies are more susceptible
6144 to flooding than stateless proxy servers.

6145 UAs and proxy servers SHOULD challenge questionable requests with only a *single* 401 (Unauthorized)
6146 or 407 (Proxy Authentication Required), forgoing the normal response retransmission algorithm, and thus
6147 behaving statelessly towards unauthenticated requests.

6148 Retransmitting the 401 (Unauthorized) or 407 (Proxy Authentication Required) status response amplifies the
6149 problem of an attacker using a falsified header field value (such as Via) to direct traffic to a third party.

6150 In summary, the mutual authentication of proxy servers through mechanisms such as TLS significantly
6151 reduces the potential for rogue intermediaries to introduce falsified requests or responses that can deny
6152 service. This commensurately makes it harder for attackers to make innocent SIP nodes into agents of
6153 amplification.

6154 **26.4 Limitations**

6155 Although these security mechanisms, when applied in a judicious manner, can thwart many threats, there are
6156 limitations in the scope of the mechanisms that must be understood by implementers and network operators.

6157 **26.4.1 HTTP Digest**

6158 One of the primary limitations of using HTTP Digest in SIP is that the integrity mechanisms in Digest do
6159 not work very well for SIP. Specifically, they offer protection of the Request-URI and the method of a
6160 message, but not for any of the header fields that UAs would most likely wish to secure.

6161 The existing replay protection mechanisms described in RFC 2617 also have some limitations for SIP.
6162 The next-nonce mechanism, for example, does not support pipelined requests. The nonce-count mechanism
6163 should be used for replay protection.

6164 Another limitation of HTTP Digest is the scope of realms. Digest is valuable when a user wants to
6165 authenticate themselves to a resource with which they have a pre-existing association, like a service provider
6166 of which the user is a customer (which is quite a common scenario and thus Digest provides an extremely
6167 useful function). By way of contrast, the scope of TLS is interdomain or multirealm, since certificates are
6168 often globally verifiable, so that the UA can authenticate the server with no pre-existing association.

6169 **26.4.2 S/MIME**

6170 The largest outstanding defect with the S/MIME mechanism is the lack of a prevalent public key infrastruc-
6171 ture for end users. If self-signed certificates (or certificates that cannot be verified by one of the participants
6172 in a dialog) are used, the SIP-based key exchange mechanism described in Section 23.2 is susceptible to a
6173 man-in-the-middle attack with which an attacker can potentially inspect and modify S/MIME bodies. The
6174 attacker needs to intercept the first exchange of keys between the two parties in a dialog, remove the exist-
6175 ing CMS-detached signatures from the request and response, and insert a different CMS-detached signature
6176 containing a certificate supplied by the attacker (but which seems to be a certificate for the proper address-
6177 of-record). Each party will think they have exchanged keys with the other, when in fact each has the public
6178 key of the attacker.

6179 It is important to note that the attacker can only leverage this vulnerability on the first exchange of keys
6180 between two parties - on subsequent occasions, the alteration of the key would be noticeable to the UAs. It
6181 would also be difficult for the attacker to remain in the path of all future dialogs between the two parties
6182 over time (as potentially days, weeks, or years pass).

6183 SSH is susceptible to the same man-in-the-middle attack on the first exchange of keys; however, it is
6184 widely acknowledged that while SSH is not perfect, it does improve the security of connections. The use of
6185 key fingerprints could provide some assistance to SIP, just as it does for SSH. For example, if two parties use
6186 SIP to establish a voice communications session, each could read off the fingerprint of the key they received
6187 from the other, which could be compared against the original. It would certainly be more difficult for the
6188 man-in-the-middle to emulate the voices of the participants than their signaling (a practice that was used
6189 with the Clipper chip-based secure telephone).

6190 The S/MIME mechanism allows UAs to send encrypted requests without preamble if they possess a
6191 certificate for the destination address-of-record on their keyring. However, it is possible that any particular
6192 device registered for an address-of-record will not hold the certificate that has been previously employed by
6193 the device's current user, and that it will therefore be unable to process an encrypted request properly, which
6194 could lead to some avoidable error signaling. This is especially likely when an encrypted request is forked.

6195 The keys associated with S/MIME are most useful when associated with a particular user (an address-
6196 of-record) rather than a device (a UA). When users move between devices, it may be difficult to transport
6197 private keys securely between UAs; how such keys might be acquired by a device is outside the scope of
6198 this document.

6199 Another, more prosaic difficulty with the S/MIME mechanism is that it can result in very large messages,
6200 especially when the SIP tunneling mechanism described in Section 23.4 is used. For that reason, it is
6201 RECOMMENDED that TCP should be used as a transport protocol when S/MIME tunneling is employed.

6202 26.4.3 TLS

6203 The most commonly voiced concern about TLS is that it cannot run over UDP; TLS requires a connection-
6204 oriented underlying transport protocol, which for the purposes of this document means TCP.

6205 It may also be arduous for a local outbound proxy server and/or registrar to maintain many simultaneous
6206 long-lived TLS connections with numerous UAs. This introduces some valid scalability concerns, especially
6207 for intensive ciphersuites. Maintaining redundancy of long-lived TLS connections, especially when a UA is
6208 solely responsible for their establishment, could also be cumbersome.

6209 TLS only allows SIP entities to authenticate servers to which they are adjacent; TLS offers strictly
6210 hop-by-hop security. Neither TLS, nor any other mechanism specified in this document, allows clients to
6211 authenticate proxy servers to whom they cannot form a direct TCP connection.

6212 26.4.4 SIPS URIs

6213 Actually using TLS on every segment of a request path entails that the terminating UAS must be reachable
6214 over TLS (perhaps registering with a SIPS URI as a contact address). This is the preferred use of SIPS. Many
6215 valid architectures, however, use TLS to secure part of the request path, but rely on some other mechanism
6216 for the final hop to a UAS, for example. Thus SIPS cannot guarantee that TLS usage will be truly end-to-
6217 end. Note that since many UAs will not accept incoming TLS connections, even those UAs that do support
6218 TLS may be required to maintain persistent TLS connections as described in the TLS limitations section
6219 above in order to receive requests over TLS as a UAS.

6220 Location services are not required to provide a SIPS binding for a SIPS Request-URI. Although loca-
6221 tion services are commonly populated by user registrations (as described in Section 10.2.1), various other
6222 protocols and interfaces could conceivably supply contact addresses for an AOR, and these tools are free to
6223 map SIPS URIs to SIP URIs as appropriate. When queried for bindings, a location service returns its contact
6224 addresses without regard for whether it received a request with a SIPS Request-URI. If a redirect server is
6225 accessing the location service, it is up to the entity that processes the Contact header field of a redirection
6226 to determine the propriety of the contact addresses.

6227 Ensuring that TLS will be used for all of the request segments up to the target domain is somewhat com-
6228 plex. It is possible that cryptographically authenticated proxy servers along the way that are non-compliant
6229 or compromised may choose to disregard the forwarding rules associated with SIPS (and the general for-
6230 warding rules in Section 16.6). Such malicious intermediaries could, for example, retarget a request from a
6231 SIPS URI to a SIP URI in an attempt to downgrade security.

6232 Alternatively, an intermediary might legitimately retarget a request from a SIP to a SIPS URI. Recipi-
6233 ents of a request whose Request-URI uses the SIPS URI scheme thus cannot assume on the basis of the
6234 Request-URI alone that SIPS was used for the entire request path (from the client onwards).

6235 To address these concerns, it is RECOMMENDED that recipients of a request whose Request-URI con-
6236 tains a SIP or SIPS URI inspect the To header field value to see if it contains a SIPS URI (though note that

6237 it does not constitute a breach of security if this URI has the same scheme but is not equivalent to the URI
6238 in the **To** header field). Although clients may choose to populate the **Request-URI** and **To** header field of
6239 a request differently, when SIPS is used this disparity could be interpreted as a possible security violation,
6240 and the request could consequently be rejected by its recipient. Recipients **MAY** also inspect the **Via** header
6241 chain in order to double-check whether or not TLS was used for the entire request path until the local ad-
6242 ministrative domain was reached. **S/MIME** may also be used by the originating UAC to help ensure that the
6243 original form of the **To** header field is carried end-to-end.

6244 If the UAS has reason to believe that the scheme of the **Request-URI** has been improperly modified in
6245 transit, the UA **SHOULD** notify its user of a potential security breach.

6246 As a further measure to prevent downgrade attacks, entities that accept only SIPS requests **MAY** also
6247 refuse connections on insecure ports.

6248 End users will undoubtedly discern the difference between SIPS and SIP URIs, and they may manually
6249 edit them in response to stimuli. This can either benefit or degrade security. For example, if an attacker
6250 corrupts a DNS cache, inserting a fake record set that effectively removes all SIPS records for a proxy
6251 server, then any SIPS requests that traverse this proxy server may fail. When a user, however, sees that
6252 repeated calls to a SIPS AOR are failing, they could on some devices manually convert the scheme from
6253 SIPS to SIP and retry. Of course, there are some safeguards against this (if the destination UA is truly
6254 paranoid it could refuse all non-SIPS requests), but it is a limitation worth noting. On the bright side, users
6255 might also divine that 'SIPS' would be valid even when they are presented only with a SIP URI.

6256 **26.5 Privacy**

6257 SIP messages frequently contain sensitive information about their senders - not just what they have to say, but
6258 with whom they communicate, when they communicate and for how long, and from where they participate
6259 in sessions. Many applications and their users require that this sort of private information be hidden from
6260 any parties that do not need to know it.

6261 Note that there are also less direct ways in which private information can be divulged. If a user or service
6262 chooses to be reachable at an address that is guessable from the person's name and organizational affiliation
6263 (which describes most addresses-of-record), the traditional method of ensuring privacy by having an unlisted
6264 "phone number" is compromised. A user location service can infringe on the privacy of the recipient of a
6265 session invitation by divulging their specific whereabouts to the caller; an implementation consequently
6266 **SHOULD** be able to restrict, on a per-user basis, what kind of location and availability information is given
6267 out to certain classes of callers. This is a whole class of problem that is expected to be studied further in
6268 ongoing SIP work.

6269 In some cases, users may want to conceal personal information in header fields that convey identity. This
6270 can apply not only to the **From** and related headers representing the originator of the request, but also the
6271 **To** - it may not be appropriate to convey to the final destination a speed-dialing nickname, or an unexpanded
6272 identifier for a group of targets, either of which would be removed from the **Request-URI** as the request is
6273 routed, but not changed in the **To** header field if the two were initially identical. Thus it **MAY** be desirable
6274 for privacy reasons to create a **To** header field that differs from the **Request-URI**.

6275 **27 IANA Considerations**

6276 All method names, header field names, status codes, and option tags used in SIP applications are registered
6277 with IANA through instructions in an IANA Considerations section in an RFC.

6278 The specification instructs the IANA to create four new sub- registries under [http://www.iana.org/assignments/sip-](http://www.iana.org/assignments/sip-parameters)
6279 [parameters](http://www.iana.org/assignments/sip-parameters): Option Tags, Warning Codes (warn-codes), Methods and Response Codes, added to the sub-
6280 registry of Header Fields that is already present there.

6281 27.1 Option Tags

6282 This specification establishes the Option Tags sub-registry under <http://www.iana.org/assignments/sip-parameters>.

6283

6284 Option tags are used in header fields such as Require, Supported, Proxy-Require, and Unsupported
6285 in support of SIP compatibility mechanisms for extensions (Section 19.2). The option tag itself is a string
6286 that is associated with a particular SIP option (that is, an extension). It identifies the option to SIP endpoints.

6287 Option tags are registered by the IANA when they are published in standards track RFCs. The IANA
6288 Considerations section of the RFC must include the following information, which appears in the IANA
6289 registry along with the RFC number of the publication.

6290 • Name of the option tag. The name MAY be of any length, but SHOULD be no more than twenty
6291 characters long. The name MUST consist of alphanum (Section 25) characters only.

6292 • Descriptive text that describes the extension.

6293 27.2 Warn-Codes

6294 This specification establishes the Warn-codes sub-registry under <http://www.iana.org/assignments/sip-parameters>
6295 and initiates its population with the warn-codes listed in Section 20.43. Additional warn-codes are registered
6296 by RFC publication.

6297 The descriptive text for the table of warn-codes is:

6298 Warning codes provide information supplemental to the status code in SIP response messages when the
6299 failure of the transaction results from a Session Description Protocol (SDP) (RFC 2327 [1]) problem.

6300 The "warn-code" consists of three digits. A first digit of "3" indicates warnings specific to SIP. Until a
6301 future specification describes uses of warn-codes other than 3xx, only 3xx warn-codes may be registered.

6302 Warnings 300 through 329 are reserved for indicating problems with keywords in the session description,
6303 330 through 339 are warnings related to basic network services requested in the session description, 370
6304 through 379 are warnings related to quantitative QoS parameters requested in the session description, and
6305 390 through 399 are miscellaneous warnings that do not fall into one of the above categories.

6306 27.3 Header Field Names

6307 This obsoletes the IANA instructions about the header sub-registry under [http://www.iana.org/assignments/sip-](http://www.iana.org/assignments/sip-parameters)
6308 [parameters](http://www.iana.org/assignments/sip-parameters).

6309 The following information needs to be provided in an RFC publication in order to register a new header
6310 field name:

- 6311 • The RFC number in which the header is registered;
- 6312 • the name of the header field being registered;
- 6313 • a compact form version for that header field, if one is defined;

6314 Some common and widely used header fields MAY be assigned one-letter compact forms (Section 7.3.3).
 6315 Compact forms can only be assigned after SIP working group review, followed by RFC publication.

6316 27.4 Method and Response Codes

6317 This specification establishes the Method and Response-Code sub- registries under [http://www.iana.org/assignments/sip-](http://www.iana.org/assignments/sip-parameters)
 6318 parameters and initiates their population as follows. The initial Methods table is:

6319	INVITE	[RFCxxxx]
6320	ACK	[RFCxxxx]
6321	BYE	[RFCxxxx]
6322	CANCEL	[RFCxxxx]
6323	REGISTER	[RFCxxxx]
6324	OPTIONS	[RFCxxxx]
6325	INFO	[RFC2976]

6326 The response code table is initially populated from Section 21, the portions labeled Informational, Suc-
 6327 cess, Redirection, Client-Error, Server-Error, and Global-Failure. The table has the following format:

6328	Type (e.g. Informational)			
6329	Number	Default Reason Phrase		[RFCxxx]

6330 The following information needs to be provided in an RFC publication in order to register a new response
 6331 code or method:

- 6332 • The RFC number in which the method or response code is registered;
- 6333 • the number of the response code or name of the method being registered;
- 6334 • the default reason phrase for that response code, if applicable;

6335 27.5 The “message/sip” MIME type.

6336 This document registers the “message/sip” MIME media type in order to allow SIP messages to be tunneled
 6337 as bodies within SIP, primarily for end-to-end security purposes. This media type is defined by the following
 6338 information:

```

6339 Media type name: message
6340 Media subtype name: sip
6341 Required parameters: none
6342 Optional parameters: version
6343     version: The SIP-Version number of the enclosed message
6344     (e.g., "2.0"). If not present, the version defaults to "2.0".
6345 Encoding scheme: SIP messages consist of an 8-bit header optionally
6346 followed by a binary MIME data object. As such, SIP messages
  
```

6347 must be treated as binary. Under normal circumstances SIP
6348 messages are transported over binary-capable transports, no
6349 special encodings are needed.

6350 Security considerations: see below

6351 Motivation and examples of this usage as a security mechanism in concert with S/MIME are given
6352 in 23.4.

6353 27.6 New Content-Disposition Parameter Registrations

6354 This document also registers four new Content-Disposition header “disposition-types”: alert, icon, ses-
6355 sion and render. The authors request that these values be recorded in the IANA registry for Content-
6356 Dispositions.

6357 Descriptions of these “disposition-types”, including motivation and examples, are given in Section 20.11.
6358 Short descriptions suitable for the IANA registry are:

6359 alert	the body is a custom ring tone to alert the user
6360 icon	the body is displayed as an icon to the user
6361 render	the body should be displayed to the user
6362 session	the body describes a communications session, for example, 6363 an RFC2327 SDP body

6364 28 Changes From RFC 2543

6365 This RFC revises RFC 2543. It is mostly backwards compatible with RFC 2543. The changes described here
6366 fix many errors discovered in RFC 2543 and provide information on scenarios not detailed in RFC 2543.
6367 The protocol has been presented in a more cleanly layered model here.

6368 We break the differences into functional behavior that is a substantial change from RFC 2543, which has
6369 impact on interoperability or correct operation in some cases, and functional behavior that is different from
6370 RFC 2543 but not a potential source of interoperability problems. There have been countless clarifications
6371 as well, which are not documented here.

6372 28.1 Major Functional Changes

- 6373 • When a UAC wishes to terminate a call before it has been answered, it sends CANCEL. If the original
6374 INVITE still returns a 2xx, the UAC then sends BYE. BYE can only be sent on an existing call leg
6375 (now called a dialog in this RFC), whereas it could be sent at any time in RFC 2543.
- 6376 • The SIP BNF was converted to be RFC 2234 compliant.
- 6377 • SIP URL BNF was made more general, allowing a greater set of characters in the user part. Fur-
6378 thermore, comparison rules were simplified to be primarily case-insensitive, and detailed handling of
6379 comparison in the presence of parameters was described. The most substantial change is that a URI
6380 with a parameter with the default value does not match a URI without that parameter.

- 6381 • Removed **Via** hiding. It had serious trust issues, since it relied on the next hop to perform the obfus-
6382 cation process. Instead, **Via** hiding can be done as a local implementation choice in stateful proxies,
6383 and thus is no longer documented.
- 6384 • In RFC 2543, **CANCEL** and **INVITE** transactions were intermingled. They are separated now. When
6385 a user sends an **INVITE** and then a **CANCEL**, the **INVITE** transaction still terminates normally. A
6386 UAS needs to respond to the original **INVITE** request with a 487 response.
- 6387 • Similarly, **CANCEL** and **BYE** transactions were intermingled; RFC 2543 allowed the UAS not to
6388 send a response to **INVITE** when a **BYE** was received. That is disallowed here. The original **INVITE**
6389 needs a response.
- 6390 • In RFC 2543, UAs needed to support only UDP. In this RFC, UAs need to support both UDP and
6391 TCP.
- 6392 • In RFC 2543, a forking proxy only passed up one challenge from downstream elements in the event
6393 of multiple challenges. In this RFC, proxies are supposed to collect all challenges and place them into
6394 the forwarded response.
- 6395 • In Digest credentials, the URI needs to be quoted; this is unclear from RFC 2617 and RFC 2069 which
6396 are both inconsistent on it.
- 6397 • SDP processing has been split off into a separate specification [13], and more fully specified as a
6398 formal offer/answer exchange process that is effectively tunneled through SIP. SDP is allowed in
6399 **INVITE/200** or **200/ACK** for baseline SIP implementations; RFC 2543 alluded to the ability to use it
6400 in **INVITE**, **200**, and **ACK** in a single transaction, but this was not well specified. More complex SDP
6401 usages are allowed in extensions.
- 6402 • Added full support for IPv6 in URIs and in the **Via** header field. Support for IPv6 in **Via** has required
6403 that its header field parameters allow the square bracket and colon characters. These characters were
6404 previously not permitted. In theory, this could cause interop problems with older implementations.
6405 However, we have observed that most implementations accept any non-control ASCII character in
6406 these parameters.
- 6407 • DNS SRV procedure is now documented in a separate specification [4]. This procedure uses both SRV
6408 and NAPTR resource records and no longer combines data from across SRV records as described in
6409 RFC 2543.
- 6410 • Loop detection has been made optional, supplanted by a mandatory usage of **Max-Forwards**. The
6411 loop detection procedure in RFC 2543 had a serious bug which would report “spirals” as an error
6412 condition when it was not. The optional loop detection procedure is more fully and correctly specified
6413 here.
- 6414 • Usage of tags is now mandatory (they were optional in RFC 2543), as they are now the fundamental
6415 building blocks of dialog identification.
- 6416 • Added the **Supported** header field, allowing for clients to indicate what extensions are supported to
6417 a server, which can apply those extensions to the response, and indicate their usage with a **Require** in
6418 the response.

- 6419 • Extension parameters were missing from the BNF for several header fields, and they have been added.
- 6420 • Handling of **Route** and **Record-Route** construction was very underspecified in RFC 2543, and also
6421 not the right approach. It has been substantially reworked in this specification (and made vastly
6422 simpler), and this is arguably the largest change. Backwards compatibility is still provided for de-
6423 ployments that do not use “pre-loaded routes”, where the initial request has a set of **Route** header
6424 field values obtained in some way outside of **Record-Route**. In those situations, the new mechanism
6425 is not interoperable.
- 6426 • In RFC 2543, lines in a message could be terminated with CR, LF, or CRLF. This specification only
6427 allows CRLF.
- 6428 • Usage of **Route** in **CANCEL** and **ACK** was not well defined in RFC 2543. It is now well specified; if
6429 a request had a **Route** header field, its **CANCEL** or **ACK** for a non-2xx response to the request need
6430 to carry the same **Route** header field values. **ACKs** for 2xx responses use the **Route** values learned
6431 from the **Record-Route** of the 2xx responses.
- 6432 • RFC 2543 allowed multiple requests in a single UDP packet. This usage has been removed.
- 6433 • Usage of absolute time in the **Expires** header field and parameter has been removed. It caused inter-
6434 operability problems in elements that were not time synchronized, a common occurrence. Relative
6435 times are used instead.
- 6436 • The branch parameter of the **Via** header field value is now mandatory for all elements to use. It now
6437 plays the role of a unique transaction identifier. This avoids the complex and bug-laden transaction
6438 identification rules from RFC 2543. A magic cookie is used in the parameter value to determine if
6439 the previous hop has made the parameter globally unique, and comparison falls back to the old rules
6440 when it is not present. Thus, interoperability is assured.
- 6441 • In RFC 2543, closure of a TCP connection was made equivalent to a **CANCEL**. This was nearly
6442 impossible to implement (and wrong) for TCP connections between proxies. This has been eliminated,
6443 so that there is no coupling between TCP connection state and SIP processing.
- 6444 • RFC 2543 was silent on whether a UA could initiate a new transaction to a peer while another was in
6445 progress. That is now specified here. It is allowed for non-INVITE requests, disallowed for INVITE.
- 6446 • PGP was removed. It was not sufficiently specified, and not compatible with the more complete PGP
6447 MIME. It was replaced with S/MIME.
- 6448 • Added the “sips” URI scheme for end-to-end TLS. This scheme is not backwards compatible with
6449 RFC 2543. Existing elements that receive a request with a SIPS URI scheme in the **Request-URI**
6450 will likely reject the request. This is actually a feature; it ensures that a call to a SIPS URI is only
6451 delivered if all path hops can be secured.
- 6452 • Additional security features were added with TLS, and these are described in a much larger and
6453 complete security considerations section.
- 6454 • In RFC 2543, a proxy was not required to forward provisional responses from 101 to 199 upstream.
6455 This was changed to **MUST**. This is important, since many subsequent features depend on delivery of
6456 all provisional responses from 101 to 199.

- 6457 ● Little was said about the 503 response code in RFC 2543. It has since found substantial use in indicat-
6458 ing failure or overload conditions in proxies. This requires somewhat special treatment. Specifically,
6459 receipt of a 503 should trigger an attempt to contact the next element in the result of a DNS SRV
6460 lookup. Also, 503 response is only forwarded upstream by a proxy under certain conditions.
- 6461 ● RFC 2543 defined, but did not sufficiently specify, a mechanism for UA authentication of a server.
6462 That has been removed. Instead, the mutual authentication procedures of RFC 2617 are allowed.
- 6463 ● A UA cannot send a BYE for a call until it has received an ACK for the initial INVITE. This was
6464 allowed in RFC 2543 but leads to a potential race condition.
- 6465 ● A UA or proxy cannot send CANCEL for a transaction until it gets a provisional response for the
6466 request. This was allowed in RFC 2543 but leads to potential race conditions.
- 6467 ● The action parameter in registrations has been deprecated. It was insufficient for any useful services,
6468 and caused conflicts when application processing was applied in proxies.
- 6469 ● RFC 2543 had a number of special cases for multicast. For example, certain responses were sup-
6470 pressed, timers were adjusted, and so on. Multicast now plays a more limited role, and the protocol
6471 operation is unaffected by usage of multicast as opposed to unicast. The limitations as a result of that
6472 are documented.
- 6473 ● Basic authentication has been removed entirely and its usage forbidden.
- 6474 ● Proxies no longer forward a 6xx immediately on receiving it. Instead, they CANCEL pending
6475 branches immediately. This avoids a potential race condition that would result in a UAC getting a
6476 6xx followed by a 2xx. In all cases except this race condition, the result will be the same - the 6xx is
6477 forwarded upstream.
- 6478 ● RFC 2543 did not address the problem of request merging. This occurs when a request forks at a
6479 proxy and later rejoins at an element. Handling of merging is done only at a UA, and procedures are
6480 defined for rejecting all but the first request.

6481 28.2 Minor Functional Changes

- 6482 ● Added the Alert-Info, Error-Info, and Call-Info header fields for optional content presentation to
6483 users.
- 6484 ● Added the Content-Language, Content-Disposition and MIME-Version header fields.
- 6485 ● Added a “glare handling” mechanism to deal with the case where both parties send each other a
6486 re-INVITE simultaneously. It uses the new 491 (Request Pending) error code.
- 6487 ● Added the In-Reply-To and Reply-To header fields for supporting the return of missed calls or mes-
6488 sages at a later time.
- 6489 ● Added TLS and SCTP as valid SIP transports.
- 6490 ● There were a variety of mechanisms described for handling failures at any time during a call; those
6491 are now generally unified. BYE is sent to terminate.

- 6492 • RFC 2543 mandated retransmission of INVITE responses over TCP, but noted it was really only
6493 needed for 2xx. That was an artifact of insufficient protocol layering. With a more coherent transaction
6494 layer defined here, that is no longer needed. Only 2xx responses to INVITEs are retransmitted over
6495 TCP.
- 6496 • Client and server transaction machines are now driven based on timeouts rather than retransmit counts.
6497 This allows the state machines to be properly specified for TCP and UDP.
- 6498 • The **Date** header field is used in REGISTER responses to provide a simple means for auto-configuration
6499 of dates in user agents.
- 6500 • Allowed a registrar to reject registrations with expirations that are too short in duration. Defined the
6501 423 response code and the **Min-Expires** for this purpose.

6502 **29 Acknowledgments**

6503 We wish to thank the members of the IETF MMUSIC and SIP WGs for their comments and suggestions. De-
6504 tailed comments were provided by Ofir Arkin, Brian Bidulock, Jim Buller, Neil Deason, Dave Devanathan,
6505 Keith Drage, Bill Fenner, Cedric Fluckiger, Yaron Goland, John Hearty, Bernie Hoeneisen, Jo Hornsby,
6506 Phil Hoffer, Christian Huitema, Hisham Khartabil, Jean Jervis, Gadi Karmi, Peter Kjellerstedt, Anders Kris-
6507 tensen, Jonathan Lennox, Gethin Liddell, Allison Mankin, William Marshall, Rohan Mahy, Keith Moore,
6508 Vern Paxson, Bob Penfield, Moshe J. Sambol, Chip Sharp, Igor Slepchin, Eric Tremblay, and Rick Work-
6509 man.

6510 Brian Rosen provided the compiled BNF.

6511 Jean Mahoney provided technical writing assistance.

6512 This work is based, inter alia, on [41, 42].

6513 **30 Authors' Addresses**

6514 Authors addresses are listed alphabetically for the editors, the writers, and then the original authors of
6515 RFC 2543. All listed authors actively contributed large amounts of text to this document.

6516 Jonathan Rosenberg
6517 dynamicsoft
6518 72 Eagle Rock Ave
6519 East Hanover, NJ 07936
6520 USA
6521 electronic mail: jdrosen@dynamicsoft.com

6522 Henning Schulzrinne
6523 Dept. of Computer Science
6524 Columbia University
6525 1214 Amsterdam Avenue
6526 New York, NY 10027
6527 USA
6528 electronic mail: schulzrinne@cs.columbia.edu

6529 Gonzalo Camarillo
6530 Ericsson
6531 Advanced Signalling Research Lab.
6532 FIN-02420 Jorvas
6533 Finland
6534 electronic mail: Gonzalo.Camarillo@ericsson.com

6535 Alan Johnston
6536 WorldCom
6537 100 South 4th Street
6538 St. Louis, MO 63102
6539 USA
6540 electronic mail: alan.johnston@wcom.com

6541 Jon Peterson
6542 NeuStar, Inc
6543 1800 Sutter Street, Suite 570
6544 Concord, CA 94520
6545 USA
6546 electronic mail: jon.peterson@neustar.com

6547 Robert Sparks
6548 dynamicsoft, Inc.
6549 5100 Tennyson Parkway
6550 Suite 1200
6551 Plano, Texas 75024
6552 USA
6553 electronic mail: rsparks@dynamicsoft.com

6554 Mark Handley
6555 ACIRI
6556 electronic mail: mjh@aciri.org

6557 Eve Schooler
6558 Computer Science Department 256-80
6559 California Institute of Technology
6560 Pasadena, CA 91125
6561 USA
6562 electronic mail: schooler@cs.caltech.edu

6563 **Normative References**

- 6564 [1] M. Handley and V. Jacobson, "SDP: session description protocol," Request for Comments 2327, Inter-
6565 net Engineering Task Force, Apr. 1998.
- 6566 [2] S. Bradner, "Key words for use in RFCs to indicate requirement levels," Request for Comments 2119,
6567 Internet Engineering Task Force, Mar. 1997.

- 6568 [3] P. Resnick and Ed, "Internet message format," Request for Comments 2822, Internet Engineering Task
6569 Force, Apr. 2001.
- 6570 [4] H. Schulzrinne and J. Rosenberg, "SIP: Locating SIP servers," Internet Draft, Internet Engineering
6571 Task Force, Feb. 2002. Work in progress.
- 6572 [5] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform resource identifiers (URI): generic syntax,"
6573 Request for Comments 2396, Internet Engineering Task Force, Aug. 1998.
- 6574 [6] T. Berners-Lee, L. Masinter, and M. McCahill, "Uniform resource locators (URL)," Request for Com-
6575 ments 1738, Internet Engineering Task Force, Dec. 1994.
- 6576 [7] F. Yergeau, "UTF-8, a transformation format of ISO 10646," Request for Comments 2279, Internet
6577 Engineering Task Force, Jan. 1998.
- 6578 [8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext
6579 transfer protocol – HTTP/1.1," Request for Comments 2616, Internet Engineering Task Force, June
6580 1999.
- 6581 [9] A. Vaha-Sipila, "URLs for telephone calls," Request for Comments 2806, Internet Engineering Task
6582 Force, Apr. 2000.
- 6583 [10] D. Crocker, Ed., and P. Overell, "Augmented BNF for syntax specifications: ABNF," Request for
6584 Comments 2234, Internet Engineering Task Force, Nov. 1997.
- 6585 [11] N. Freed and N. Borenstein, "Multipurpose internet mail extensions (MIME) part two: Media types,"
6586 Request for Comments 2046, Internet Engineering Task Force, Nov. 1996.
- 6587 [12] D. Eastlake, S. Crocker, and J. Schiller, "Randomness recommendations for security," Request for
6588 Comments 1750, Internet Engineering Task Force, Dec. 1994.
- 6589 [13] J. Rosenberg and H. Schulzrinne, "An offer/answer model with SDP," Internet Draft, Internet Engi-
6590 neering Task Force, Feb. 2002. Work in progress.
- 6591 [14] J. Postel, "User datagram protocol," Request for Comments 768, Internet Engineering Task Force,
6592 Aug. 1980.
- 6593 [15] J. Postel, "DoD standard transmission control protocol," Request for Comments 761, Internet Engi-
6594 neering Task Force, Jan. 1980.
- 6595 [16] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang,
6596 and V. Paxson, "Stream control transmission protocol," Request for Comments 2960, Internet Engi-
6597 neering Task Force, Oct. 2000.
- 6598 [17] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart, "HTTP
6599 authentication: Basic and digest access authentication," Request for Comments 2617, Internet Engi-
6600 neering Task Force, June 1999.
- 6601 [18] R. Troost, S. Dorner, K. Moore, and Ed, "Communicating presentation information in internet mes-
6602 sages: The content-disposition header field," Request for Comments 2183, Internet Engineering Task
6603 Force, Aug. 1997.

- 6604 [19] E. Zimmerer, J. Peterson, A. Vemuri, L. Ong, F. Audet, M. Watson, and M. Zonoun, "MIME media
6605 types for ISUP and QSIG objects," Request for Comments 3204, Internet Engineering Task Force,
6606 Dec. 2001.
- 6607 [20] R. Braden and Ed, "Requirements for internet hosts - application and support," Request for Comments
6608 1123, Internet Engineering Task Force, Oct. 1989.
- 6609 [21] H. Alvestrand, "IETF policy on character sets and languages," Request for Comments 2277, Internet
6610 Engineering Task Force, Jan. 1998.
- 6611 [22] J. Galvin, S. Murphy, S. Crocker, and N. Freed, "Security multipart for MIME: multipart/signed and
6612 multipart/encrypted," Request for Comments 1847, Internet Engineering Task Force, Oct. 1995.
- 6613 [23] R. Housley, "Cryptographic message syntax," Request for Comments 2630, Internet Engineering Task
6614 Force, June 1999.
- 6615 [24] B. Ramsdell and Ed, "S/MIME version 3 message specification," Request for Comments 2633, Internet
6616 Engineering Task Force, June 1999.
- 6617 [25] T. Dierks and C. Allen, "The TLS protocol version 1.0," Request for Comments 2246, Internet Engi-
6618 neering Task Force, Jan. 1999.
- 6619 [26] S. Kent and R. Atkinson, "Security architecture for the internet protocol," Request for Comments 2401,
6620 Internet Engineering Task Force, Nov. 1998.

6621 Informative References

- 6622 [27] R. Pandya, "Emerging mobile and personal communication systems," *IEEE Communications Maga-*
6623 *zine*, Vol. 33, pp. 44-52, June 1995.
- 6624 [28] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: a transport protocol for real-time
6625 applications," Request for Comments 1889, Internet Engineering Task Force, Jan. 1996.
- 6626 [29] H. Schulzrinne, A. Rao, and R. Lanphier, "Real time streaming protocol (RTSP)," Request for Com-
6627 ments 2326, Internet Engineering Task Force, Apr. 1998.
- 6628 [30] F. Cuervo, N. Greene, A. Rayhan, C. Huitema, B. Rosen, and J. Segers, "Megaco protocol version
6629 1.0," Request for Comments 3015, Internet Engineering Task Force, Nov. 2000.
- 6630 [31] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: session initiation protocol," Request
6631 for Comments 2543, Internet Engineering Task Force, Mar. 1999.
- 6632 [32] P. Hoffman, L. Masinter, and J. Zawinski, "The mailto URL scheme," Request for Comments 2368,
6633 Internet Engineering Task Force, July 1998.
- 6634 [33] E. M. Schooler, "A multicast user directory service for synchronous rendezvous," Master's Thesis CS-
6635 TR-96-18, Department of Computer Science, California Institute of Technology, Pasadena, California,
6636 Aug. 1996.

- 6637 [34] S. Donovan, "The SIP INFO method," Request for Comments 2976, Internet Engineering Task Force,
6638 Oct. 2000.
- 6639 [35] R. Rivest, "The MD5 message-digest algorithm," Request for Comments 1321, Internet Engineering
6640 Task Force, Apr. 1992.
- 6641 [36] F. Dawson and T. Howes, "vcard MIME directory profile," Request for Comments 2426, Internet
6642 Engineering Task Force, Sept. 1998.
- 6643 [37] G. Good, "The LDAP data interchange format (LDIF) - technical specification," Request for Com-
6644 ments 2849, Internet Engineering Task Force, June 2000.
- 6645 [38] J. Palme, "Common internet message headers," Request for Comments 2076, Internet Engineering
6646 Task Force, Feb. 1997.
- 6647 [39] J. Franks, P. Hallam-Baker, J. Hostetler, P. Leach, A. Luotonen, E. Sink, and L. Stewart, "An exten-
6648 sion to HTTP : Digest access authentication," Request for Comments 2069, Internet Engineering Task
6649 Force, Jan. 1997.
- 6650 [40] A. Johnston, S. Donovan, R. Sparks, C. Cunningham, D. Willis, J. Rosenberg, K. Summers, and
6651 H. Schulzrinne, "SIP telephony call flow examples," Internet Draft, Internet Engineering Task Force,
6652 Apr. 2001. Work in progress.
- 6653 [41] E. M. Schooler, "Case study: multimedia conference control in a packet-switched teleconferencing
6654 system," *Journal of Internetworking: Research and Experience*, Vol. 4, pp. 99–120, June 1993. ISI
6655 reprint series ISI/RS-93-359.
- 6656 [42] H. Schulzrinne, "Personal mobility for multimedia services in the Internet," in *European Workshop on*
6657 *Interactive Distributed Multimedia Systems and Services (IDMS)*, (Berlin, Germany), Mar. 1996.

6658 **A Table of Timer Values**

6659 Table 4 summarizes the meaning and defaults of the various timers used by this specification.

6660 **Full Copyright Statement**

6661 Copyright (c) The Internet Society (2002). All Rights Reserved.

6662 This document and translations of it may be copied and furnished to others, and derivative works that
6663 comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and
6664 distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and
6665 this paragraph are included on all such copies and derivative works. However, this document itself may not
6666 be modified in any way, such as by removing the copyright notice or references to the Internet Society or
6667 other Internet organizations, except as needed for the purpose of developing Internet standards in which case
6668 the procedures for copyrights defined in the Internet Standards process must be followed, or as required to
6669 translate it into languages other than English.

6670 The limited permissions granted above are perpetual and will not be revoked by the Internet Society or
6671 its successors or assigns.

Timer	Value	Section	Meaning
T1	500ms default	Section 17.1.1.1	RTT Estimate
T2	4s	Section 17.1.2.2	The maximum retransmit interval for non-INVITE requests and INVITE responses
T4	5s	Section 17.1.2.2	Maximum duration a message will remain in the network
Timer A	initially T1	Section 17.1.1.2	INVITE request retransmit interval, for UDP only
Timer B	64*T1	Section 17.1.1.2	INVITE transaction timeout timer
Timer C	> 3min	Section Section 16.6 bullet 11	proxy INVITE transaction timeout
Timer D	> 32s for UDP 0s for TCP/SCTP	Section 17.1.1.2	Wait time for response retransmits
Timer E	initially T1	Section 17.1.2.2	non-INVITE request retransmit interval, UDP only
Timer F	64*T1	Section 17.1.2.2	non-INVITE transaction timeout timer
Timer G	initially T1	Section 17.2.1	INVITE response retransmit interval
Timer H	64*T1	Section 17.2.1	Wait time for ACK receipt
Timer I	T4 for UDP 0s for TCP/SCTP	Section 17.2.1	Wait time for ACK retransmits
Timer J	64*T1 for UDP 0s for TCP/SCTP	Section 17.2.2	Wait time for non-INVITE request retransmits
Timer K	T4 for UDP 0s for TCP/SCTP	Section 17.1.2.2	Wait time for response retransmits

Table 4: Summary of timers

6672 This document and the information contained herein is provided on an "AS IS" basis and THE IN-
6673 TERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WAR-
6674 RANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT
6675 THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED
6676 WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.