

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 (2001). 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 which allow participants to agree on a set of compatible media types. SIP makes use of elements called proxy servers to help route requests to the users current location, assist in firewall traversal, and provide features to users. SIP also provides a registration function that allows them to upload their current location for use by proxy servers. SIP runs ontop of several different transport protocols.

Contents

1	Introduction	7
2	Overview of SIP Functionality	7
3	Terminology	8
4	Overview of Operation	8
5	Structure of the Protocol	13
6	Definitions	15
7	SIP Messages	18
7.1	Requests	18
7.2	Responses	19
7.3	Header Fields	20

34	7.3.1	Header Field Format	20
35	7.3.2	Header Field Classification	22
36	7.3.3	Compact Form	22
37	7.4	Bodies	22
38	7.4.1	Message Body Type	22
39	7.4.2	Message Body Length	22
40	7.5	Framing SIP messages	23
41	8	General User Agent Behavior	23
42	8.1	UAC Behavior	23
43	8.1.1	Generating the Request	23
44	8.1.2	Sending the Request	26
45	8.1.3	Processing Responses	26
46	8.2	UAS Behavior	27
47	8.2.1	Authentication/Authorization	27
48	8.2.2	Method Inspection	27
49	8.2.3	Header Inspection	28
50	8.2.4	Content Processing	29
51	8.2.5	Applying Extensions	29
52	8.2.6	Processing the Request	29
53	8.2.7	Generating the Response	29
54	8.3	Redirect Servers	30
55	9	Canceling a Request	31
56	9.1	Client Behavior	31
57	9.2	Server Behavior	32
58	10	Registrations	32
59	10.1	Overview of Usage	32
60	10.2	Construction of the REGISTER request	34
61	10.2.1	Adding Bindings with REGISTER	34
62	10.2.2	Removing Bindings with REGISTER	35
63	10.2.3	Fetching Bindings with REGISTER	36
64	10.2.4	Refreshing Registrations	36
65	10.2.5	Discovering a Registrar	36
66	10.3	Processing of REGISTER at the Registrar	36
67	11	Querying for Capabilities	38
68	11.1	Construction of OPTIONS Request	38
69	11.2	Processing of OPTIONS Request	39
70	12	Dialogs	40
71	12.1	Creation of a Dialog	41
72	12.2	Requests within a Dialog	42
73	12.2.1	UAC Behavior	42
74	12.2.2	UAS behavior	44

75	12.3 Termination of a Dialog	44
76	13 Initiating a Session	44
77	13.1 Overview	44
78	13.2 Caller Processing	45
79	13.2.1 Creating the Initial INVITE	45
80	13.2.2 Processing INVITE Responses	46
81	13.3 Callee Processing	47
82	13.3.1 Processing of the INVITE	47
83	14 Modifying an Existing Session	49
84	14.1 UAC Behavior	50
85	14.2 UAS Behavior	50
86	15 Terminating a Session	51
87	15.1 Terminating a Dialog with a BYE	51
88	15.1.1 UAC Behavior	51
89	15.1.2 UAS Behavior	52
90	16 Proxy Behavior	52
91	16.1 Overview	52
92	16.2 Stateful Proxy	53
93	16.3 Request Validation	54
94	16.4 Making a Routing Decision	55
95	16.5 Request Processing	57
96	16.6 Response Processing	60
97	16.7 Handling transport errors	64
98	16.8 CANCEL Processing	64
99	16.9 Stateless proxy	64
100	17 Transactions	65
101	17.1 Client transaction	67
102	17.1.1 INVITE Client Transaction	67
103	17.1.2 non-INVITE Client Transaction	70
104	17.1.3 Matching Responses to Client Transactions	72
105	17.1.4 Handling Transport Errors	72
106	17.2 Server Transaction	73
107	17.2.1 INVITE Server Transaction	73
108	17.2.2 non-INVITE Server Transaction	75
109	17.2.3 Matching Requests to Server Transactions	75
110	17.3 RTT Estimation	76
111	18 Reliability of Provisional Responses	77
112	19 Transport	77
113	19.1 Clients	77

114	19.1.1	Sending Requests	77
115	19.1.2	Receiving Responses	78
116	19.2	Servers	79
117	19.2.1	Receiving Requests	79
118	19.2.2	Sending Responses	79
119	19.3	Framing	80
120	19.4	Error Handling	80
121	20	Security Considerations	80
122	20.1	Transport and Network Layer Security	81
123	20.2	SIP Authentication	82
124	20.2.1	Framework	82
125	20.2.2	User to User Authentication	83
126	20.2.3	Proxy to User Authentication	83
127	20.2.4	Authentication Schemes	84
128	20.3	SIP Encryption	85
129	20.4	Denial of Service	86
130	21	Common Message Components	87
131	21.1	SIP Uniform Resource Locators	87
132	21.1.1	SIP URL components	87
133	21.1.2	Character escaping requirements	89
134	21.1.3	Example SIP URLs	90
135	21.1.4	SIP URL Comparison	90
136	21.2	Option Tags	92
137	21.3	Tags	92
138	22	Header Fields	92
139	22.1	Accept	95
140	22.2	Accept-Encoding	95
141	22.3	Accept-Language	96
142	22.4	Alert-Info	96
143	22.5	Allow	96
144	22.6	Authentication-Info	96
145	22.7	Authorization	97
146	22.8	Call-ID	97
147	22.9	Call-Info	97
148	22.10	Contact	98
149	22.11	Content-Disposition	98
150	22.12	Content-Encoding	98
151	22.13	Content-Language	99
152	22.14	Content-Length	99
153	22.15	Content-Type	99
154	22.16	CSeq	100
155	22.17	Date	100

156	22.18 Error-Info	100
157	22.19 Expires	101
158	22.20 From	101
159	22.21 In-Reply-To	101
160	22.22 Max-Forwards	101
161	22.23 MIME-Version	102
162	22.24 Organization	102
163	22.25 Priority	102
164	22.26 Proxy-Authenticate	102
165	22.27 Proxy-Authorization	103
166	22.28 Proxy-Require	103
167	22.29 Record-Route	103
168	22.30 Require	103
169	22.31 Retry-After	104
170	22.32 Route	104
171	22.33 Server	104
172	22.34 Subject	104
173	22.35 Supported	105
174	22.36 Timestamp	105
175	22.37 To	105
176	22.38 Unsupported	105
177	22.39 User-Agent	106
178	22.40 Via	106
179	22.41 Warning	106
180	22.42 WWW-Authenticate	107
181	23 Response Codes	108
182	23.1 Provisional 1xx	108
183	23.1.1 100 Trying	108
184	23.1.2 180 Ringing	108
185	23.1.3 181 Call Is Being Forwarded	108
186	23.1.4 182 Queued	108
187	23.1.5 183 Session Progress	109
188	23.2 Successful 2xx	109
189	23.2.1 200 OK	109
190	23.3 Redirection 3xx	109
191	23.3.1 300 Multiple Choices	109
192	23.3.2 301 Moved Permanently	109
193	23.3.3 302 Moved Temporarily	109
194	23.3.4 305 Use Proxy	110
195	23.3.5 380 Alternative Service	110
196	23.4 Request Failure 4xx	110
197	23.4.1 400 Bad Request	110
198	23.4.2 401 Unauthorized	110
199	23.4.3 402 Payment Required	110

200	23.4.4	403 Forbidden	110
201	23.4.5	404 Not Found	110
202	23.4.6	405 Method Not Allowed	111
203	23.4.7	406 Not Acceptable	111
204	23.4.8	407 Proxy Authentication Required	111
205	23.4.9	408 Request Timeout	111
206	23.4.10	410 Gone	111
207	23.4.11	413 Request Entity Too Large	111
208	23.4.12	414 Request-URI Too Long	111
209	23.4.13	415 Unsupported Media Type	111
210	23.4.14	420 Bad Extension	112
211	23.4.15	421 Extension Required	112
212	23.4.16	480 Temporarily Unavailable	112
213	23.4.17	481 Call/Transaction Does Not Exist	112
214	23.4.18	482 Loop Detected	112
215	23.4.19	483 Too Many Hops	112
216	23.4.20	484 Address Incomplete	112
217	23.4.21	485 Ambiguous	113
218	23.4.22	486 Busy Here	113
219	23.4.23	487 Request Terminated	113
220	23.4.24	488 Not Acceptable Here	113
221	23.5	Server Failure 5xx	113
222	23.5.1	500 Server Internal Error	113
223	23.5.2	501 Not Implemented	114
224	23.5.3	502 Bad Gateway	114
225	23.5.4	503 Service Unavailable	114
226	23.5.5	504 Server Time-out	114
227	23.5.6	505 Version Not Supported	114
228	23.5.7	513 Message Too Large	114
229	23.6	Global Failures 6xx	114
230	23.6.1	600 Busy Everywhere	115
231	23.6.2	603 Decline	115
232	23.6.3	604 Does Not Exist Anywhere	115
233	23.6.4	606 Not Acceptable	115
234	24	Locating a SIP Server	115
235	24.1	Computing the List of Next Hops	116
236	24.1.1	Numeric Destination Address	116
237	24.1.2	SRV Resolution of Host Name	116
238	24.1.3	Address Record Resolution of Host Name	117
239	24.2	Contacting the Next Hops	117
240	25	Examples	118
241	25.1	Registration	118
242	25.2	Session Setup	119

243	26 Augmented BNF for the SIP Protocol	124
244	26.1 Basic Rules	125
245	27 IANA Considerations	138
246	27.1 Option Tags	138
247	27.2 Warn-Codes	138
248	27.3 Header Field Names	139
249	27.4 Method and Response Codes	139
250	28 Changes Made in Version 00	139
251	29 Changes Made in Version 01	144
252	30 Changes Made in Version 02	145
253	31 Changes Made in Version 03	146
254	32 Changes Made in Version 04	148
255	33 Changes Made in Version 05	150
256	34 Acknowledgments	153
257	35 Authors' Addresses	153

258 **1 Introduction**

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

271 **2 Overview of SIP Functionality**

272 The Session Initiation Protocol (SIP) is an application-layer control protocol that can establish, modify and
273 terminate multimedia sessions (conferences) such as Internet telephony calls. SIP can also invite participants
274 to already existing sessions. A SIP entity issuing an invitation for an already existing session does not
275 necessarily have to be a member of the session to which it is inviting. Media can be added to (and removed

276 from) an existing session. SIP transparently supports name mapping and redirection services, which supports
277 *personal mobility* [1, p. 44] - users can maintain a single externally visible identifier (SIP URI) regardless
278 of their network location.

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

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

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

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

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

284 **Session handling:** including transfer and termination of sessions, modifying session parameters, and in-
285 voking services.

286 SIP is not a vertically integrated communications system. SIP is rather a component of the overall IETF
287 multimedia data and control architecture which incorporates protocols such as RSVP (RFC 2205 [2]) for re-
288 serving network resources, the real-time transport protocol (RTP) (RFC 1889 [3]) for transporting real-time
289 data and providing QOS feedback, the real-time streaming protocol (RTSP) (RFC 2326 [4]) for controlling
290 delivery of streaming media, the session announcement protocol (SAP) [5] for advertising multimedia ses-
291 sions via multicast and the session description protocol (SDP) (RFC 2327 [6]) for describing multimedia
292 sessions. Therefore, SIP should be used in conjunction with other protocols in order to provide complete
293 services to the users. However, the basic functionality and operation of SIP does not depend on any of these
294 protocols.

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

302 SIP does not offer conference control services such as floor control or voting and does not prescribe how
303 a conference is to be managed, but SIP can be used to initiate a session that uses some other conference
304 control protocol. SIP does not allocate multicast addresses and does not reserve network resources.

305 **3 Terminology**

306 In this document, the key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”,
307 “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” are to be interpreted as described in RFC
308 2119 [7] and indicate requirement levels for compliant SIP implementations.

309 **4 Overview of Operation**

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

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

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

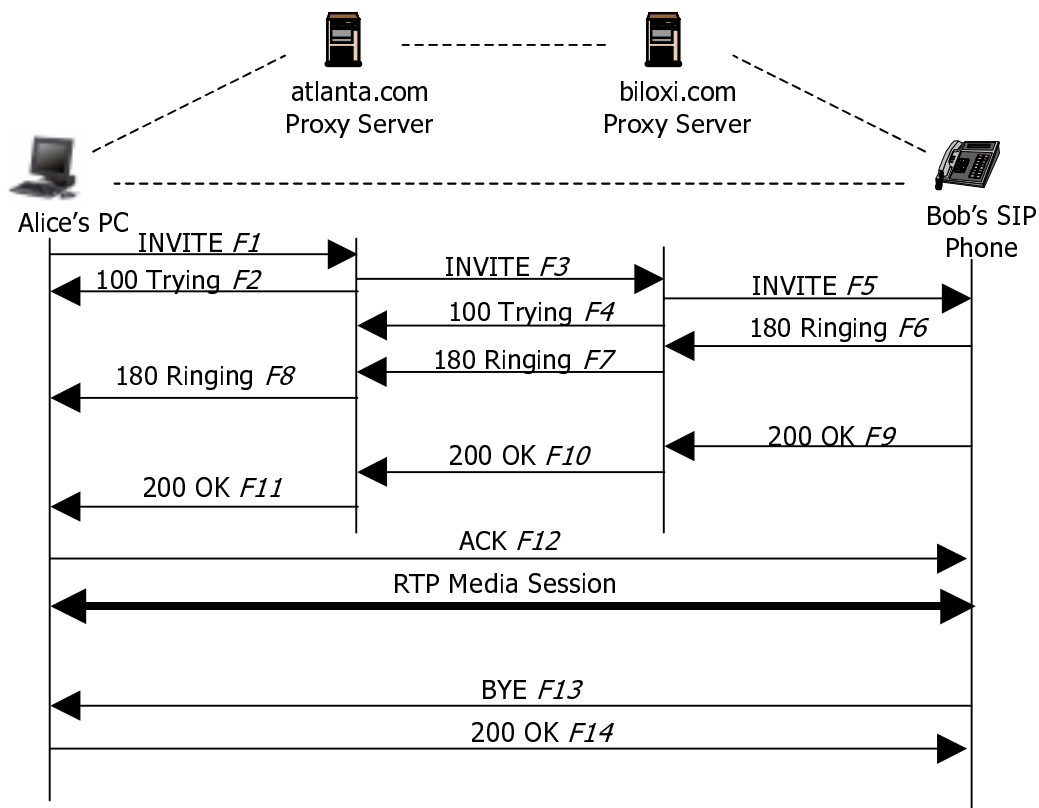


Figure 1: SIP session setup example with SIP trapezoid

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

326 SIP is based on an HTTP-like request/response transaction model. Each transaction consists of a request
 327 that invokes a particular “Method”, or function, on the server, and at least one response. In this example, the
 328 transaction begins with Alice’s softphone sending an INVITE request addressed to Bob’s SIP URI. INVITE
 329 is an example of a SIP method which specifies the action that the requestor (Alice) wants the server (Bob) to

330 take. The INVITE request contains a number of header fields. Header fields are additional named attributes
331 which provide additional information about a message. The ones present in an INVITE include a unique
332 identifier for the call, the destination address, Alice's address, and information about the type of session that
333 Alice wishes to establish with Bob. The INVITE (message F1 in Figure 1) might look like this:

```
334   INVITE sip:bob@biloxi.com SIP/2.0
335   Via: SIP/2.0/UDP 10.1.3.3:5060
336   To: Bob <sip:bob@biloxi.com>
337   From: Alice <sip:alice@atlanta.com>;tag=1928301774
338   Call-ID: a84b4c76e66710@10.1.3.3
339   CSeq: 314159 INVITE
340   Contact: <sip:alice@10.1.3.3>
341   Content-Type: application/sdp
342   Contact-Length: 142
343
344   (Alice's SDP not shown)
```

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

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

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

352 **From** also contains a display name (Alice) and a SIP URI (sip:alice@atlanta.com) that indicate the
353 originator of the request. This header field also has a **tag** parameter which contains a pseudorandom string
354 (1928301774) which was added to the URI by the softphone. It is used for identification purposes.

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

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

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

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

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

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

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

371 Since the softphone has no knowledge of Bob's exact location, or how to locate the SIP server in
372 the biloxi.com domain, the softphone sends the INVITE to the SIP server that serves Alice's domain, at-

373 lanta.com. The IP address of the atlanta.com SIP server could have been configured in Alice's softphone, or
374 it could have been discovered by DHCP, for example.

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

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

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

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

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

```
417   CSeq: 314159 INVITE
418   Contact: <sip:bob@10.4.1.4>
419   Content-Type: application/sdp
420   Contact-Length: 131
421
422   (Bob's SDP not shown)
```

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

431 In addition to DNS and location service lookups shown in this example, proxy servers can make arbi-
432 trarily complex "routing decisions" in order to decide where to send a request. For example, if Bob's SIP
433 phone returned a 486 Busy Here response, the biloxi.com proxy server could proxy the **INVITE** to Bob's
434 voicemail server. A proxy server can also send an **INVITE** to a number of locations at the same time. This
435 type of parallel search is known as "forking".

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

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

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

456 At the end of the call, Bob disconnects (hangs up) first, and generates a **BYE** message. This **BYE** is
457 routed directly to Alice's softphone, again bypassing the proxies. Alice confirms receipt of the **BYE** with
458 a 200 OK response, which terminates the session and the **BYE** transaction. Note that no **ACK** is sent - an
459 **ACK** is only sent in response to a response to an **INVITE** request. The reasons for this special handling for
460 **INVITE** will be discussed later, but relate to the reliability mechanisms in SIP, the length of time it can take

461 for a ringing phone to be answered, and forking. For this reason, request handling in SIP is often classified
462 as either INVITE or non- INVITE, referring to all other methods besides INVITE. Full details on session
463 termination is in Section 15.

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

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

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

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

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

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

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

495 Additional operations in SIP include querying for the capabilities of a SIP server or client using **OP-**
496 **TIONS**, and canceling a pending request using **CANCEL** will be introduced in later sections.

497 **5 Structure of the Protocol**

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

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

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

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

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

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

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

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

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

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

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

548 The most important method in SIP is the INVITE method, which is used to establish a session between

549 participants. A session is a collection of participants, and streams of media between them, for the purposes
550 of communication. Section 13 discusses how sessions are initiated, resulting in one or more SIP dialogs.
551 Section 14 discusses how characteristics of that session are modified, through the use of an INVITE request
552 within a dialog. Finally, section 15 discusses how a session is terminated.

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

555 6 Definitions

556 This specification uses a number of terms to refer to the roles played by participants in SIP communications.
557 The definitions of client, server and proxy are similar to those used by the Hypertext Transport Protocol
558 (HTTP) (RFC 2616 [8]). The terms and generic syntax of URI and URL are defined in RFC 2396 [9]. The
559 following terms have special significance for SIP.

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

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

567 **Call leg:** Another name for a dialog.

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

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

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

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

577 **Downstream:** A direction of message forwarding within a transaction which refers to the direction that
578 requests flow from the user agent client to user agent server.

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

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

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

584 **Invitation:** An INVITE request.

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

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

593 **Location server:** See *location service*.

594 **Location service:** A location service is used by a SIP redirect or proxy server to obtain information about
595 a callee's possible location(s). It is an abstract database, sometimes referred to as a location server.
596 The contents of the database can be populated in many ways, including being written by registrars.

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

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

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

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

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

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

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

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

- 622 **Ringback:** Ringback is the signaling tone produced by the calling party's application indicating that a
623 called party is being alerted (ringing).
- 624 **Server:** A server is a network element that receives requests in order to service them, and sends back
625 responses to those requests. Examples of servers are proxies, user agent servers, redirect servers, and
626 registrars.
- 627 **Sequential search:** In a sequential search, a proxy server attempts each contact address in sequence, pro-
628 ceeding to the next one only after the previous has generated a non-2xx final response.
- 629 **Session:** From the SDP specification: "A multimedia session is a set of multimedia senders and receivers
630 and the data streams flowing from senders to receivers. A multimedia conference is an example of a
631 multimedia session." (RFC 2327 [6]) (A session as defined for SDP can comprise one or more RTP
632 sessions.) As defined, a callee can be invited several times, by different calls, to the same session.
633 If SDP is used, a session is defined by the concatenation of the *user name*, *session id*, *network type*,
634 *address type* and *address* elements in the origin field.
- 635 **(SIP) transaction:** A SIP transaction occurs between a client and a server and comprises all messages from
636 the first request sent from the client to the server up to a final (non-1xx) response sent from the server
637 to the client, and the ACK for the response in the case the response was a 2xx. The ACK for a 2xx
638 response is a separate transaction.
- 639 **Spiral:** A spiral is a SIP request which is routed to a proxy, forwarded onwards, and arrives once again
640 at that proxy, but this time, differs in a way which will result in a different processing decision than
641 the original request. Typically, this means that it has a Request-URI that differs from the previous
642 arrival. A spiral is not an error condition, unlike a loop.
- 643 **Stateless proxy:** A logical entity that does not maintain the client or server transaction state machines
644 defined in this specification when it processes requests. A stateless proxy forwards every request it
645 receives downstream and every response it receives upstream.
- 646 **Stateful proxy:** A logical entity that maintains the client and server transaction state machines defined by
647 this specification during the processing of a request. Also known as a transaction stateful proxy. The
648 behavior of a stateful proxy is further defined in Section 16. A stateful proxy is not the same as a call
649 stateful proxy.
- 650 **Transaction User (TU):** The layer of protocol processing that resides above the transaction layer. Trans-
651 action users include the UAC core, UAS core, and proxy core.
- 652 **Upstream:** A direction of message forwarding within a transaction which refers to the direction that re-
653 sponses flow from the user agent server to user agent client.
- 654 **URL-encoded:** A character string encoded according to RFC 1738, Section 2.2 [10].
- 655 **User agent client (UAC):** A user agent client is a logical entity that creates a new request, and then uses
656 the client transaction state machinery to send it. The role of UAC lasts only for the duration of that
657 transaction. In other words, if a piece of software initiates a request, it acts as a UAC for the duration
658 of that transaction. If it receives a request later on, it takes on the role of a User Agent Server for the
659 processing of that transaction.

660 **UAC Core:** The set of processing functions required of a UAC that reside above the transaction and trans-
661 port layers.

662 **User agent server (UAS):** A user agent server is a logical entity that generates a response to a SIP request.
663 The response accepts, rejects or redirects the request. This role lasts only for the duration of that
664 transaction. In other words, if a piece of software responds to a request, it acts as a UAS for the
665 duration of that transaction. If it generates a request later on, it takes on the role of a User agent client
666 for the processing of that transaction.

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

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

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

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

677 7 SIP Messages

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

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

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

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

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

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

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

690 7.1 Requests

691 SIP Requests are distinguished by having a **Request-Line** for a start-line. A **Request-Line** begins with
692 a method token, followed by the **Request-URI** and the protocol version, and ending with CRLF. The ele-
693 ments are separated by SP characters. No CR or LF are allowed except in the end-of-line CRLF sequence.
694 No LWS is allowed in any of the elements.

Method Request-URI SIP-Version

- Method

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

- Request-URI

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

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

- SIP Version

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

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

7.2 Responses

SIP Responses are distinguished by having a Status-Line for a start-line. A Status-Line, consists of the protocol version followed by a numeric Status-Code and its associated textual phrase, with each element separated by SP characters. No CR or LF is allowed except in the final CRLF sequence.

SIP-version Status-Code Reason-Phrase

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

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

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

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

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

731 **4xx:** Client Error – the request contains bad syntax or cannot be fulfilled at this server;

732 **5xx:** Server Error – the server failed to fulfill an apparently valid request;

733 **6xx:** Global Failure – the request cannot be fulfilled at any server.

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

735 **7.3 Header Fields**

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

740 **7.3.1 Header Field Format**

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

743 `field-name: field-value`

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

747 `Subject: lunch`

748 `Subject : lunch`

749 `Subject :lunch`

750 `Subject: lunch`

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

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

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

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

757 `pick up the phone`

758 `and talk to me!`

759 The relative order of header fields with different field names is not significant. The relative order of those
760 with the same field name is important. Multiple header fields with the same field-name may be present in a
761 message if and only if the entire field-value for that header field is defined as a comma-separated list (i.e.,
762 `#(values)`). It MUST be possible to combine the multiple header fields into one "field-name: field-value"
763 pair, without changing the semantics of the message, by appending each subsequent field-value to the first,
764 each separated by a comma.

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

767 The following blocks of headers are valid and equivalent:

```
768 Route: sip:alice@atlanta.com
769 Subject: Lunch
770 Route: sip:bob@biloxi.com
771 Route: sip:carol@chicago.com
772
773 Route: sip:alice@atlanta.com, sip:bob@biloxi.com
774 Route: sip:carol@chicago.com
775 Subject: Lunch
776
777 Subject: Lunch
778 Route: sip:alice@atlanta.com, sip:bob@biloxi.com, sip:carol@chicago.com
```

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

```
780 Route: sip:alice@atlanta.com
781 Route: sip:bob@biloxi.com
782 Route: sip:carol@chicago.com
783
784 Route: sip:bob@biloxi.com
785 Route: sip:alice@atlanta.com
786 Route: sip:carol@chicago.com
787
788 Route: sip:alice@atlanta.com, sip:carol@chicago.com, sip:bob@biloxi.com
```

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

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

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

797 The following are equivalent:

```
798 Contact: <sip:alice@atlanta.com>;expires=3600
799 CONTACT: <sip:alice@atlanta.com>;ExPiReS=3600
800
801 Contact-Disposition: session;handling=optional
802 contact-disposition: Session;HANDLING=OPTIONAL
803
```

804 The following are not equivalent;

805 Warning: 370 devnull "Choose a bigger pipe"

806 Warning: 370 devnull "CHOOSE A BIGGER PIPE"

807 **7.3.2 Header Field Classification**

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

812 **7.3.3 Compact Form**

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

820 **7.4 Bodies**

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

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

825 **7.4.1 Message Body Type**

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

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

834 **7.4.2 Message Body Length**

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

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

7.5 Framing SIP messages

Unlike HTTP, SIP MAY use UDP or other unreliable datagram protocols. Each such datagram carries one request or response. Datagrams, including all headers, SHOULD NOT be larger than the path maximum transmission unit (MTU) if the MTU is known, or 1500 bytes if the MTU is unknown. However, implementations MUST be able to handle messages up to the maximum datagram packet size. For UDP, this size is 65,535 bytes, including headers.

The MTU of 1500 bytes accommodates encapsulation within the "typical" ethernet MTU without IP fragmentation. Recent studies [15, p. 154] indicate that an MTU of 1500 bytes is a reasonable assumption. The next lower common MTU values are 1006 bytes for SLIP and 296 for low-delay PPP (RFC 1191 [16]). Thus, another reasonable value would be a message size of 950 bytes, to accommodate packet headers within the SLIP MTU without fragmentation.

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

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

8 General User Agent Behavior

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

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

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

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

8.1 UAC Behavior

8.1.1 Generating the Request

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

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

878 **8.1.1.1 To** The **To** general-header field first and foremost specifies the desired “logical” recipient of the
879 request, or the address of record of the user or resource that is the target of this request. This may or may
880 not be the ultimate recipient of the request. The **To** header **MAY** contain a SIP URI, but it may also make
881 use of other URI schemes (for example as the tel URL [13]) when appropriate. The **To** header field allows
882 for a display name; this is meant to contain a descriptive version of the URI, and is intended to be displayed
883 to a user interface.

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

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

889 For further information on the **To** header see Section 22.37.

890 The following is an example of valid **To** header:

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

892 **8.1.1.2 From** The **From** general-header field indicates the logical identity of the initiator of the request,
893 possibly the user’s address of record. Like the **To** field, it contains a URI and optionally a display name.
894 It is used by SIP elements to determine processing rules to apply to a request (for example, automatic call
895 rejection). As such, it is very important that the URI not contain IP addresses or host names, since these are
896 not logical names.

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

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

905 The **From** field **MUST** contain a new “tag” parameter, chosen by the UAC. See Section 21.3 for details
906 on choosing a tag.

907 For further information on the **From** header see Section 22.20.

908 Examples:

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

912 **8.1.1.3 Call-ID** The **Call-ID** general-header field acts as a unique identifier to group together series of
913 messages. It is always the same for all requests and responses sent by either UA in a dialog. It is also the
914 same in each registration from a UA within a single boot cycle.

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

919 Use of cryptographically random identifiers [17] in the generation of Call-IDs is RECOMMENDED. Im-
920 plementations MAY use the form "localid@host". Call-IDs are case-sensitive and are simply compared
921 byte-by-byte.

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

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

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

927 Example:

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

929 **8.1.1.4 CSeq** The Cseq header serves as a way to identify and order transactions. It consists of a
930 sequence number and a method. The method MUST match that of the request. The sequence number value
931 is arbitrary, but MUST be expressible as a 32-bit unsigned integer and MUST be less than 2**31.

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

934 For further information on the CSeq header see Section 22.16.

935 Example:

936 CSeq: 4711 INVITE

937 **8.1.1.5 Via** The Via header is used to determine the transport to use for sending a request, and for
938 identifying the IP address and port where the response is to be sent. Rules for setting and using the values
939 in this header are described in Section 19.

940 For further information on the Via header see Section 22.40.

941 **8.1.1.6 Contact** The Contact header provides a SIP URI that can be used to contact that specific in-
942 stance of the user agent for subsequent requests. The Contact header MUST be present in any request that
943 can result in the establishment of a dialog. For the methods defined in this specification, that includes only
944 the INVITE request. For these requests, the scope of the Contact is the dialog. That is, the Contact header
945 refers to the URL that the UA would like to receive requests at, for requests that are part of that dialog only.
946 Only a single URI MUST be present.

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

948 **8.1.1.7 Request-URI** The initial Request-URI of the message SHOULD be set to the value of the URI
949 in the To field. One notable exception is the REGISTER method; behavior for setting the Request-URI
950 of register is given in Section 10. Another exception is the case of pre-existing Route headers; in that case,
951 the procedures of Section 12.2.1.1 as they pertain to the Request- URI are followed, even though there is
952 no dialog.

953 **8.1.1.8 Supported and Require** If the UAC supports extensions to SIP that can be applied by the
954 server to the response, the UAC SHOULD include a Supported header in the request listing the option tags
955 for those extensions.

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

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

964 **8.1.1.9 Additional Message Components** After a new request has been created, the headers described
965 above have been properly constructed, any additional optional headers are added, as are any headers specific
966 to the method.

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

970 **8.1.2 Sending the Request**

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

977 **8.1.3 Processing Responses**

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

981 **8.1.3.1 Unrecognized Responses** A UAC MUST treat any response they do not recognize as being
982 equivalent to the x00 response code of that class, and MUST be able to process the x00 response code for
983 all classes. For example, if a UAC receives an unrecognized response code of 431, it can safely assume that
984 there was something wrong with its request and treat the response as if it had received a 400 (Bad Request)
985 response code.

986 **8.1.3.2 Vias** If more than one **Via** header field is present in a response, the UAC SHOULD discard the
987 message.

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

990 **8.1.3.3 Processing 3xx responses** Upon receipt of a redirection response (e.g. a 3xx response status
991 code), clients SHOULD use the URI(s) in the **Contact** header field to formulate a new request.

992 To do that, the client copies all but the “method-param” and “header” elements of the addr-spec part
993 of the Contact header field into the Request-URI of the request. It uses the “header” parameters to create
994 headers for the request, replacing any default headers normally used.

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

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

999 **8.1.3.4 Processing 4xx responses** Certain 4xx response codes require specific UA processing, indepen-
1000 dent of the method.

1001 If a 401 or 407 response is received, the UAC SHOULD follow the authorization procedures of Section
1002 20.2.2 and Section 20.2.3 to retry the request with credentials.

1003 If a 413 response is received (Section 23.4.11), it means that the request contained a body that was
1004 longer than the UAS was willing to accept. If possible, the UAC SHOULD retry the request, either omitting
1005 the body or using one of a smaller length.

1006 If a 415 response is received (Section 23.4.13), it means the request contained media types not supported
1007 by the UAS. The UAC SHOULD retry sending the request, this time only using content with types listed in
1008 the Accept header in the response, with encodings listed in the Accept-Encoding header in the response,
1009 and with languages listed in the Accept-Language in the response.

1010 If a 420 response is received (Section 23.4.14), it means the request contained a Require or Proxy-
1011 Require header listing an option-tag for a feature not supported by a proxy or UAS. The UAC SHOULD
1012 retry the request, this time omitting any extensions listed in the Unsupported header in the response.

1013 In all of the above cases, retrying the request is accomplished by creating a new request with the appro-
1014 priate modifications. This new request SHOULD have the same value of the Call-ID, To, and From of the
1015 previous request, but the CSeq should contain a new sequence number that is one higher than the previous.

1016 With other 4xx responses, a retry may or may not be possible depending on the method and the use case.

1017 **8.2 UAS Behavior**

1018 When a request outside of a dialog is processed by a UAS, there are a set of processing rules which are
1019 followed, independent of the method. Section 12 gives guidance on how a UAS can tell whether a request
1020 is inside or outside of a dialog.

1021 **8.2.1 Authentication/Authorization**

1022 A UAS MAY authenticate the originator of a request, and this process may require the server to issue a
1023 challenge for credentials. The required behavior is independent of the method of the request, and is detailed
1024 in Section 20.2.

1025 **8.2.2 Method Inspection**

1026 Once a request is authenticated (or no authentication was desired), the UAS MUST inspect the method of the
1027 request. If the UAS does not support the method of a request it MUST generate a 405 (Method Not Allowed)
1028 response. Procedures for generation of responses are described in Section 8.2.7. The UAS MUST also add
1029 an Allow header to the 405 response. The Allow header field MUST list the set of methods supported by the
1030 UAS generating the message.

1031 The Allow header is presented in Section 22.5.
1032 If the method is one supported by the server, processing continues.

1033 8.2.3 Header Inspection

1034 If a UAS does not understand a header field in a request (i.e. the header is not defined in this specification
1035 or in any supported extension), the server MUST ignore that header and continue processing the message. A
1036 UAS SHOULD ignore any malformed headers which are not necessary for processing requests.

1037 **8.2.3.1 To and Request-URI** The To header field identifies the original recipient of the request design-
1038 nated by the user identified in the From field. The original recipient may or may not be the UAS processing
1039 the request, do to call forwarding or other proxy operations. A UAS MAY apply any policy it wishes in
1040 determination of whether to accept requests when the To field is not the identity of the UAS. However, it is
1041 RECOMMENDED that a UAS accept requests even if they do not recognize the URI scheme (e.g., a tel:
1042 URI) in the To header, or if the To header does not address a known or current user of this UAS. If, on the
1043 other hand, the UAS decides to reject the request, it SHOULD generate a response with a 403 status code and
1044 send it to the server transaction for transmission.

1045 However, the Request-URI identifies the UAS that is to process the request. If the Request-URI does
1046 not identify an address that the UAS is willing to accept requests for, it SHOULD reject the request with
1047 a 404 (Not Found) response. If the Request-URI does not provide sufficient information for the UAS to
1048 determine whether it is willing to process the request, it SHOULD return a 485 (Ambiguous) response. This
1049 response SHOULD contain a Contact header field containing URIs of new addresses to be tried. Typically,
1050 a UA which uses the REGISTER method to bind its address of record to a specific contact address, will see
1051 requests whose Request-URI equals those contact addresses.

1052 **8.2.3.2 Require** Assuming the UAS decides that it is the proper element to process the request, it ex-
1053 amines the Require header field, if present.

1054 The Require general-header field is used by UAC to tell UAS about SIP extensions that the UAC expects
1055 the UAS to support in order to properly process the request. If a UAS does not understand an option listed
1056 in a Require header field, it MUST respond by generating a response with status code 420 (Bad Extension).
1057 The UAS MUST add a Unsupported, and list in it those options it does not understand amongst those in
1058 the Require header of the request. Upon receipt of the 420 the client SHOULD retry the request, this time
1059 without using those extensions listed in the Unsupported header in the response.

1060 Example:

```
1061 UACC->UAS:  INVITE sip:watson@bell-telephone.com SIP/2.0
1062             Require: com.example.billing
1063             Payment: sheep_skins, conch_shells
1064
1065 UASS->UAC:  SIP/2.0 420 Bad Extension
1066             Unsupported: com.example.billing
```

1067 This is to make sure that the client-server interaction will proceed without delay when all options are understood
1068 by both sides, and only slow down if options are not understood (as in the example above). For a well-matched
1069 client-server pair, the interaction proceeds quickly, saving a round-trip often required by negotiation mechanisms.

1070 In addition, it also removes ambiguity when the client requires features that the server does not understand. Some
1071 features, such as call handling fields, are only of interest to end systems.

1072 **8.2.4 Content Processing**

1073 Assuming the UAS understands any extensions required by the client, the UAS examines the body of the
1074 message, and the headers that describe it. If there are any bodies whose type (indicated by the **Content-**
1075 **Type**), language (indicated by the **Content-Language**) or encoding (indicated by the **Content-Encoding**)
1076 are not understood, and that body part is not optional (as indicated by the **Content-Disposition**) header, the
1077 UAS **MUST** reject the request with a 415 (Unsupported Media Type) response. The response **MUST** contain
1078 a **Accept** header listing the types of all bodies it understands, in the event the request contained bodies of
1079 types not supported by the UAS. If the request contained content encodings not understood by the UAS,
1080 the response **MUST** contain an **Accept-Encoding** header listing the encodings understood by the UAS. If
1081 the request contained content with languages not understood by the UAS, the response **MUST** contain an
1082 **Accept-Language** header indicating the languages understood by the UAS.

1083 Beyond these checks, body handling is method and type specific.

1084 For further information on the processing of Content-specific headers see Section 7.4.

1085 **8.2.5 Applying Extensions**

1086 A UAS that wishes to apply some extension when generating the response **MUST** only do so if support for
1087 that extension is indicated in the **Supported** header in the request. If the desired extension is not supported,
1088 the server **SHOULD** rely only on baseline SIP and any other extensions supported by the client. To ensure
1089 that the **SHOULD** can be fulfilled, any specification of a new extension **MUST** include discussion of how
1090 to gracefully return to baseline SIP when the extension is not present. In rare circumstances, where the
1091 server cannot process the request without the extension, the server **MAY** send a 421 (Extension Required)
1092 response. This response indicates that the proper response cannot be generated without support of a specific
1093 extension. The needed extension(s) **MUST** be included in a **Require** header in the response. This behavior
1094 is **NOT RECOMMENDED**, as it will generally break interoperability.

1095 Any extensions applied to a non-421 response **MUST** be listed in a **Require** header included in the
1096 response. Of course, the server **MUST NOT** apply extensions not listed in the **Supported** header in the
1097 request. As a result of this, the **Require** header in a response will only ever contain option tags defined in
1098 standards track RFCs.

1099 **8.2.6 Processing the Request**

1100 Assuming all of the checks in the previous subsections are passed, the UAS processing becomes method
1101 specific. Section 10 deals with the **REGISTER** request, section 11 deals with the **OPTIONS** request,
1102 section 13 deals with the **INVITE** request, and section 15 deals with the **BYE** request.

1103 **8.2.7 Generating the Response**

1104 When a UAS wishes to construct a response to a request, it follows these procedures. Additional procedures
1105 may be needed depending on the status code of the response and the circumstances of its construction. These
1106 additional procedures are documented elsewhere.

1107 The **From** field of the response **MUST** equal the **From** field of the request. The **Call-ID** field of the
1108 response **MUST** equal the **Call-ID** field of the request. The **Cseq** field of the response **MUST** equal the **Cseq**
1109 field of the request. The **Via** headers in the response **MUST** equal the **Via** headers in the request, and **MUST**
1110 maintain the same ordering.

1111 If a request contained a **To** tag in the request, the **To** field in the response **MUST** equal that of the request.
1112 However, if the **To** field in the request did not contain a tag, the URI in the **To** field in the response **MUST**
1113 equal the URI in the **To** field in the request. Additionally, the UAS **MUST** add a tag to the **To** field in the
1114 response. This serves to identify the UAS that is responding, possibly resulting in a component of a dialog
1115 ID. The same tag **MUST** be used for all responses to that request, both provisional and final. Procedures for
1116 generation of tags are defined in Section 21.3.

1117 **8.3 Redirect Servers**

1118 In some architectures it may be desirable to reduce the processing load on proxy servers that are responsible
1119 for routing requests by relying on redirection. Redirection allows servers to push routing information for a
1120 request back in a response to the client, thereby taking themselves out of the loop of further messaging for
1121 this transaction while still aiding in locating the target of the request. When the originator of the request
1122 receives the redirection it will send a new request based on the routing information it has received. By
1123 propagating routing information from the core of the network to its edges, redirection allows for considerable
1124 network scalability.

1125 A redirect server is logically constituted of a server transaction layer and a transaction user that has
1126 access to a location service of some kind (see Section 10 for more on registrars and location services). This
1127 location service is effectively a database containing mappings between a single URI and a set of one or more
1128 alternative locations at which the target of that URI can be found.

1129 A redirect server does not issue any SIP requests of its own. After receiving a request other than **CAN-**
1130 **CEL**, the server gathers the list of alternative locations from the location service and either returns a final
1131 response of class 3xx or it refuses the request. For well-formed **CANCEL** requests, it **SHOULD** return a
1132 2xx response. This response ends the SIP transaction. The redirect server maintains transaction state for an
1133 entire SIP transaction. It is the responsibility of clients to detect forwarding loops between redirect servers.

1134 When a redirect server returns a 3xx response to a request, it populates the list of (one or more) alterna-
1135 tive locations into **Contact** headers. An “**expires**” parameter to the **Contact** header may also be supplied
1136 to indicate the lifetime of the **Contact** data.

1137 The **Contact** header field contains URIs giving the new locations or user names to try, or may simply
1138 specify additional transport parameters. A 301 or 302 response may also give the same location and user-
1139 name that was targeted by the initial request but specify additional transport parameters such as a different
1140 server or multicast address to try, or a change of SIP transport from UDP to TCP or vice versa.

1141 Note that the **Contact** header field **MAY** also refer to a different entity than the one originally called. For
1142 example, a SIP call connected to GSTN gateway may need to deliver a special informational announcement
1143 such as “The number you have dialed has been changed.”

1144 A **Contact** response header field can contain any suitable URI indicating where the called party can be
1145 reached, not limited to SIP URIs. For example, it could contain URL’s for phones, fax, or irc (if they were
1146 defined) or a **mailto:** (RFC 2368, [18]) URL.

1147 The “**expires**” parameter of the **Contact** header field indicates how long the URI is valid. The parameter
1148 is either a number indicating seconds or a quoted string containing a **SIP-date**. If this parameter is not
1149 provided, the value of the **Expires** header field determines how long the URI is valid. Implementations

1150 MAY treat values larger than 2**32-1 (4294967295 seconds or 136 years) as equivalent to 2**32-1.

1151 Redirect servers MUST ignore features that are not understood (including unrecognized headers, Re-
1152 quired extensions, or even method names) and proceed with the redirection of the session in question. If
1153 a particular extension requires that intermediate devices support it, the extension MUST be tagged in the
1154 Proxy-Require field as well (see Section 22.28).

1155 9 Canceling a Request

1156 The previous section has discussed general UA behavior for generating requests, and processing responses,
1157 for requests of all methods. In this section, we discuss a general purpose method, called CANCEL.

1158 The CANCEL request, as the name implies, is used to cancel a previous request sent by a client. Specif-
1159 ically, it asks the user agent server to cease processing the request, and generate an error response to that
1160 request. CANCEL has no effect on a request that has already been responded to. Because of this, it is most
1161 useful to CANCEL requests which can take a long time to respond to. For this reason, CANCEL is most
1162 useful for INVITE requests, which can take a long time to generate a response. In that usage, a UAS that
1163 receives a CANCEL request for an INVITE, but has not yet sent a response, would “stop ringing”, and then
1164 respond to the INVITE with a specific error response (a 487).

1165 Cancel requests can be constructed and sent by any type of client, including both proxies and user
1166 agent servers. Section 15 discusses under what conditions a UAC would CANCEL an INVITE request, and
1167 Section 16 discusses proxy usage of INVITE.

1168 Because a stateful proxy can generate its own CANCEL, a stateful proxy also responds to a CANCEL,
1169 rather than simply forwarding a response it would receive from a downstream element. For that reason,
1170 CANCEL is referred to as a “hop-by-hop” request, since it is responded to at each stateful proxy hop.

1171 9.1 Client Behavior

1172 The following procedures are used to construct a CANCEL request. The Request-URI, Call-ID, To, the
1173 numeric part of CSeq and From header fields in the CANCEL request MUST be identical to those in the
1174 request being cancelled, including tags. A CANCEL constructed by a client MUST have only a single Via
1175 header, whose value matches the top Via in the request being cancelled. Using the same values for these
1176 headers allows the CANCEL to be matched with the request it cancels (Section 9.2 indicates how such
1177 matching occurs). However, the method part of the CSeq header MUST have a value of CANCEL. This
1178 allows it to be identified and processed as a transaction in its own right (See Section 17).

1179 Once the CANCEL is constructed, the client SHOULD check whether any response (provisional or final)
1180 has been received for the request being cancelled (herein referred to as the “original request”). The CANCEL
1181 request MUST NOT be sent if no provisional response has been received, rather, the client MUST wait for the
1182 arrival of a provisional response before sending the request. If the original request has generated a final
1183 response, the CANCEL SHOULD NOT be sent, as it is an effective no-op, since CANCEL has no effect on
1184 requests which have already generated a final response. When the client decides to send the CANCEL, it
1185 creates a client transaction for the CANCEL, and passes it the CANCEL request along with the destination
1186 address, port and transport. The destination address, port, and transport for the CANCEL MUST be identical
1187 to those used to send the original request.

1188 If it was allowed to send the CANCEL before receiving a response for the previous request the server could
1189 receive the CANCEL before the original request.

1190 Note that both the transaction corresponding to the original request and the CANCEL transaction will
1191 complete independently. However, a UAC canceling a request cannot rely on receiving a 487 (Request
1192 Terminated) response for the original request, as an RFC 2543-compliant UAS will not generate such a
1193 response. If there is no final response for the original request in 64*T1 seconds for an INVITE transaction,
1194 and T3 seconds for a non-INVITE transaction, the client SHOULD then consider the original transaction
1195 cancelled and SHOULD destroy the client transaction handling the original request.

1196 9.2 Server Behavior

1197 The CANCEL method requests that the TU at the server side cancel a pending request with the same Call-
1198 ID, To, From, top Via header and Request-URI and CSeq (sequence number only) header field values.

1199 The processing of a CANCEL request at a server depends on the type of server. A stateless proxy will
1200 forward it, a stateful proxy might respond to it and generate some CANCEL requests of its own, and a UAS
1201 will respond to it. See Section 16.8 for proxy treatment of CANCEL.

1202 When a UAS receives a CANCEL, it looks for any server transactions which were created by requests
1203 with the same To, From, Call-ID, Cseq numeric value, Request-URI and top Via header. If no matching
1204 transactions are found, the CANCEL is responded to with a 481 (Call Leg/Transaction Does Not Exist). If
1205 the transaction for the original request still exists, the behavior of the UAS on receiving a CANCEL request
1206 depends on whether it has already sent a final response for original request. If it has, the CANCEL request
1207 has no effect on the processing of the original request, no effect on any session state, and no effect on the
1208 responses generated for the original request. If the UAS has not issued a final response for the original
1209 request, it immediately responds to the original request with a 487 (Request Terminated).

1210 The CANCEL request itself is answered with a 200 (OK) response in either case. Once the response is
1211 constructed it is passed to the server transaction for the CANCEL request.

1212 10 Registrations

1213 10.1 Overview of Usage

1214 SIP is a protocol that offers a discovery capability. For one user to initiate a session with another, SIP must
1215 discover the current host(s) that the called user is reachable at. This discovery process is accomplished
1216 by SIP proxy servers, which are responsible for receiving a request, determining where to send it based
1217 on knowledge of the location of the user, and then sending it there. To do this, proxies consult an abstract
1218 service known as a *location service*, which provides address bindings for a particular domain. These address
1219 bindings map an incoming SIP URL, sip:bob@Biloxi.com, for example, to one or more SIP URLs
1220 which are somehow "closer" to the desired user, sip:bob@engineering.Biloxi.com, for example.
1221 Ultimately, a proxy will consult a location service which maps a received URL to the current host(s) that a
1222 user is logged in to.

1223 There are many ways by which the contents of the location service can be established. One way is
1224 administratively. In the above example, Bob is known to be a member of the engineering department through
1225 access to a corporate database. SIP provides a mechanism, however, for a user agent to explicitly create a
1226 binding in the location service of a proxy. This mechanism is known as registration.

1227 The process of registration entails sending a REGISTER message to a special type of UAS known as a
1228 registrar. The registrar acts as a front end to the location service for a domain, reading and writing mappings
1229 based on the contents of the REGISTER messages. This location service will then be consulted by a proxy

1230 server that is responsible for routing requests for that domain.

1231 SIP does not mandate a particular mechanism for implementing the location service. The only require-
 1232 ment is that a registrar for some domain MUST be capable of reading and writing data to the location service,
 1233 and a proxy for that domain MUST be capable of reading that same data. A registrar MAY be co-located with
 1234 a particular SIP proxy server for the same domain, allowing usage of an in memory database for the location
 1235 service. Usage of a shared database is another implementation choice. The choice depends entirely on the
 1236 architectural requirements (redundancy, scalability, etc) of a particular deployment.

1237 Registration creates bindings in a location service for a particular domain that associate an “address of
 1238 record” URI with one or more “contact addresses”. This means that when a proxy for that domain receives a
 1239 request whose request URI matches the address of record, the proxy will forward the request to the contact
 1240 addresses registered to that address of record. Generally, it only makes sense to register an address of record
 1241 at a location service for a domain when requests for that address of record would be routed to that domain.
 1242 In most cases, this means that the domain of the registration will need to match the domain in the URI of
 1243 the address of record.

1244 The most important usage of the registration mechanism is to inform a proxy of the mapping between
 1245 the address of record and the current host on which the UA resides. However, the registration process is a
 1246 general mechanism for establishing bindings, and can be used for other purposes (for example, to set up call
 1247 forwarding).

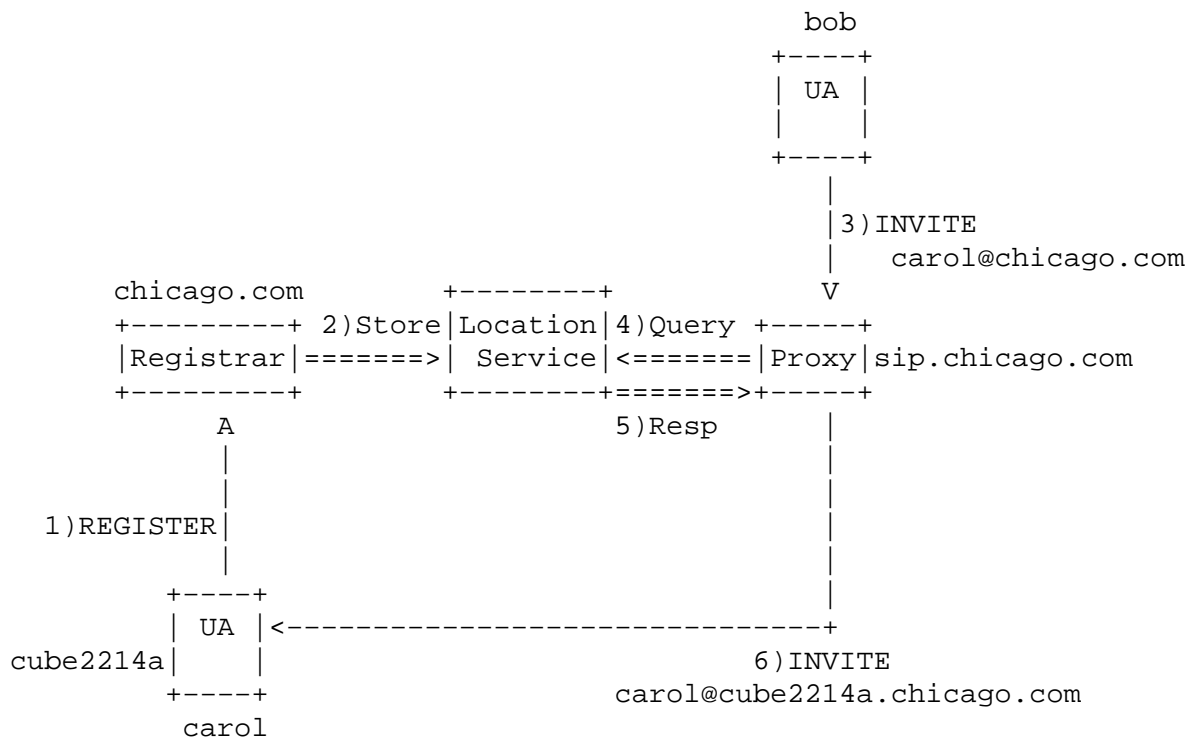


Figure 2: REGISTER example

1248 10.2 Construction of the REGISTER request

1249 Several operations can be performed with a REGISTER method with respect to a registrar. One of these is
1250 the basic registration operation that is described above, which provides a new binding between an address
1251 of record and one or more contact addresses. Registration on behalf of a particular address of record may be
1252 performed by a third party if they are authorized to do so. A client may also remove previous bindings, or
1253 query to determine which bindings are currently in place for an address of record.

1254 Aside from the exceptions noted in this and the following sections, the construction of the REGISTER
1255 method, and behavior of clients sending a REGISTER is identical to the general UAC behavior described in
1256 Section 8.1 and Section 17.1. Regardless of the operation that is performed by a REGISTER, the following
1257 header fields MUST be formulated as follows:

1258 **Request-URI:** The Request-URI names the domain of the location service that the registration is meant
1259 for (e.g. "chicago.com"). The user name MUST be empty.

1260 **To:** The To header field contains the address of record whose registration is to be created or modified.
1261 Note that the initial To header field and the Request-URI field SHOULD therefore be different in a
1262 REGISTER message.

1263 **From:** The From header field contains the address of record of the person responsible for the registration,
1264 which MAY be identical to the value of the To header field. For third-party registrations the From
1265 header field and To header field are different.

1266 **Call-ID:** All registrations from a user agent client SHOULD use the same Call-ID header value, at least
1267 within the same reboot cycle.

1268 If different Call-IDs were used for overlapping REGISTER messages coming from the same client, the
1269 registrar might have trouble determining their ordering.

1270 **Contact:** REGISTER requests MAY contain one or more Contact header fields. Contact addresses are
1271 presented in the Contact header fields of REGISTER requests.

1272 Note that user agents MUST NOT send a new registration (containing new Contact header fields, as
1273 opposed to a retransmission) until they have received a response from the registrar for the previous one.

1274 The following optional Contact header parameters also contain behavior specific to the registration
1275 process.

1276 **action:** The "action" parameter has been deprecated. UACs SHOULD NOT use the "action" parameter.

1277 **expires:** The "expires" parameter indicates how long the UAC would like the binding to be valid. The
1278 parameter is either a number indicating seconds or a quoted string containing a SIP-date. If this
1279 parameter is not provided, the value of the Expires header field determines how long the binding is
1280 valid. Implementations MAY treat values larger than 2**32-1 (4294967295 seconds or 136 years) as
1281 equivalent to 2**32-1.

1282 10.2.1 Adding Bindings with REGISTER

1283 For a simple registration, a REGISTER request sent to a registrar includes contact addresses to which
1284 requests should be forward for the originating user's address of record. The address of record itself (i.e.

1285 'sip:carol@chicago.com') MUST populate the To header of the REGISTER. The Contact header fields of
1286 the request typically contain SIP URIs that identify particular SIP endpoints (i.e. 'sip:carol@cube2214a.chicago.com'),
1287 but they MAY use any URI scheme; this way a SIP UA can choose to register telephone numbers (with the
1288 tel URL, [13]) or email addresses (with a mailto URL, [18]) as Contacts for an address of record.

1289 For example, if Carol, whose address of record is 'sip:carol@chicago.com', needed to register, she would
1290 typically want to register with the registrar associated with the location service of chicago.com. This location
1291 service would then be accessed by a proxy server that receives requests targeting users in the chicago.com
1292 domain, and hence new requests for Carol's address of record will be routed to her SIP endpoint.

1293 Once a client has established bindings at a registrar, it MAY send subsequent registrations containing
1294 new bindings or modifications to pre-existing bindings as necessary. The 2xx response to the REGISTER
1295 message will contain (in Contact header fields) a complete list of bindings that have been registered for this
1296 address of record at this registrar.

1297 **10.2.1.1 Setting the Expiration Interval of Contact Addresses** When a client sends a REGISTER
1298 request, it MAY suggest an expiration interval that indicates how long the client would like the registration
1299 to be valid (although as is detailed in Section 10.3, the registrar has the ultimate say).

1300 There are two ways in which a client can suggest an expiration interval for a binding: through an Expires
1301 header, or an "expires" Contact header parameter. The latter allows expiration intervals to be suggested
1302 on a per-binding basis when more than one binding is given in a single REGISTER, whereas the former
1303 suggests an expiration interval for all Contact header fields that do not contain the "expires" parameter.

1304 If neither mechanism for expressing a suggested expiration time is present in a REGISTER, a default
1305 suggestion of one hour is assumed.

1306 **10.2.1.2 Setting Preference among Contact Addresses** If more than one Contact is sent in a REGIS-
1307 TER, then the registering UA intends to associate all of the URIs given in these Contact headers with the
1308 address of record present in the To field. This list can be prioritized with the "q" mechanism.

1309 **q:** The "q" parameter indicates a relative preference for the particular Contact header field compared to
1310 other bindings present in this REGISTER message or existing within the location service of the
1311 registrar. For an example of how a proxy server uses "q" values, see Section 16.5.

1312 **10.2.2 Removing Bindings with REGISTER**

1313 Registrations are removed from the registrar through an expiration process; registrations are soft state and
1314 need to be refreshed periodically. A client may attempt to influence the expiration intervals selected by the
1315 registrar as described in Section 10.2.1.

1316 A registering user agent requests the immediate removal of a binding by specifying an expiration in-
1317 terval of "0" for that contact address in a REGISTER. It is RECOMMENDED that user agents support this
1318 mechanism so that bindings can be removed (for whatever reason) before their expiration interval has passed.

1319 The REGISTER-specific Contact header field value of "*" applies to all registrations, but it MUST only
1320 be used when the Expires header is present with a value of "0".

1321 Use of the "*" Contact header field value allows a registering user agent to remove all of its bindings expediently.

1322 **10.2.3 Fetching Bindings with REGISTER**

1323 If no **Contact** headers are present in a **REGISTER**, then the UA is not in fact registering any new bindings,
1324 and the list of bindings is therefore left unchanged. As noted above, in a successful response to this **REG-**
1325 **ISTER** message, the complete list of existing bindings is returned, and thus a **REGISTER** without **Contact**
1326 headers serves as a fetch operation.

1327 **10.2.4 Refreshing Registrations**

1328 When a 2xx response has been received by the client for a **REGISTER** request, the client **MUST** determine
1329 when each of the bindings enumerated in the response needs to be refreshed. This may include bindings that
1330 were registered in previous **REGISTER** transactions.

1331 Since the list of bindings returned in the response to a **REGISTER** may contain bindings that were not
1332 included in this **REGISTER** transaction, the client must correlate **Contact** header fields in the response
1333 with the **Contact** header fields it sent in the request in order to establish proper expiration timers. This
1334 correlation should be performed in accordance with the URI comparison rules given in Section 21.1.4.

1335 The registering UA **MUST** re-register each contact address at least as often as the mandated expiration
1336 interval. A **REGISTER** that refreshes a binding **SHOULD** have the same **Call-ID** as the request which
1337 created the binding. The **CSeq** header **SHOULD** have a numeric sequence number that is one higher than
1338 the value sent in the last request with the same **Call-ID**.

1339 Note that a UA **MUST** update its expiration timers for refreshing each binding every time it receives
1340 a response to a registration request.

1341 Registration refreshes **SHOULD** be sent to the same address as the original registration, unless redirected.

1342 **10.2.5 Discovering a Registrar**

1343 Depending on the policy of their administrative domain, SIP UAs can be configured with the address of a
1344 local registrar. Some UAs may be equipped with protocol tools (outside the scope of SIP) that allow them
1345 to discover their local registrar dynamically.

1346 Note that as an alternate means of discovering a registrar if no local registrar is configured in the user
1347 agent, clients **MAY** register via multicast. Multicast registrations are addressed to the well-known “all SIP
1348 servers” multicast address “sip.mcast.net” (224.0.1.75). This request **MUST** be scoped to ensure it is not
1349 forwarded beyond the boundaries of the administrative system. This **MAY** be done with either TTL or
1350 administrative scopes (see [19]), depending on what is implemented in the network. SIP user agents **MAY**
1351 listen to that address and use it to become aware of the location of other local users (see [20]); however, they
1352 do not respond to the request.

1353 Multicast registration may be inappropriate in some environments, for example, if multiple businesses share the
1354 same local area network.

1355 If a SIP UA knows of an appropriate registrar it **SHOULD** attempt to register with this server periodically
1356 - management of registration intervals is detailed below.

1357 **10.3 Processing of REGISTER at the Registrar**

1358 A registrar is a UAS that responds to a **REGISTER** request, and stores the information gathered from that
1359 request in a location service that is in turn accessible to proxy servers within its administrative domain. A
1360 registrar handles requests as a UAS (in conformity with Section 8.2 and Section 17.2) but it accepts only the

1361 REGISTER method and generates only the responses detailed in this section. Note that the REGISTER
1362 method also does not support the Record-Route or Route header, and that proxy servers MUST NOT add
1363 Record-Route headers to REGISTER requests.

1364 A registrar must know (through provisioning or some other mechanism) the set of administrative do-
1365 main(s) for which its associated location service(s) are responsible. REGISTER requests MUST be pro-
1366 cessed by a registrar in the order that they are received.

1367 Upon the arrival of a REGISTER message, the registrar MUST inspect the Request-URI to determine
1368 whether it has access to a location service responsible for the domain to which this request is addressed.
1369 If this message is for some other administrative domain, then if the registrar can act as a proxy server, it
1370 SHOULD forward the request to the addressed domain (following the general behavior for proxying messages
1371 described in Section 16).

1372 When a registrar receives a REGISTER message, it is RECOMMENDED that the registrar authenticate
1373 the user agent client. Mechanisms for the authentication of SIP user agents are described in Section 20.2;
1374 registration behavior in no way overrides the generic authentication framework for SIP. If no authentication
1375 mechanism is available, the registrar MAY take the From address as the asserted identity of the originator of
1376 the request.

1377 Once the identity of the registering user has been ascertained, it is RECOMMENDED that the registrar
1378 determine if the authenticated user agent is authorized to request and/or modify registrations for this address
1379 of record. For example, a registrar might consult a authorization database (directly or through an appropriate
1380 protocol) that maps credentials or other tokens of identity resulting from authentication to one or more
1381 addresses of record for which this identity is responsible.

1382 Note that in architectures that support third-party registration, one entity may be responsible for updating the
1383 registrations associated with multiple addresses of record.

1384 When the registrar has determined that the client is permitted to make the request, the registrar MUST
1385 extract the address of record from the To header field of the REGISTER. Note that the registrar MUST
1386 extract the entire To header field URI in order to use it as an index in the location service.

1387 Next, the registrar MUST query its location service (the repository of previously registered bindings)
1388 for the set of bindings associated with this address of record. If the address of record is not valid for this
1389 administrative domain (for example, because the username is not assigned), then the registration attempt
1390 fails (see below). A full URI comparison (as described in Section 21.1.4) MUST be performed to determine
1391 whether a given binding matches this address of record.

1392 The registrar now MUST extract all the Contact header fields from the REGISTER message (note that
1393 there may be no Contact header field).

1394 Each contact address in a REGISTER MUST now be compared to all existing registrations at this loca-
1395 tion service according to the rules in Section 21.1.4. Note that URIs other than SIP URIs in contact addresses
1396 MUST be compared according to the standard URI equivalency rules for the URI schema in question.

1397 If a match is found among pre-existing registrations, the registrar MUST copy all parameters associated
1398 with the current Contact header field from the REGISTER message into the pre-existing binding in its
1399 location service (overwriting with changed values any existing parameters as necessary, with the exception
1400 of “expires”). Expiration intervals for this contact address MUST also be reset, based on any suggested
1401 expiration in the REGISTER (remember that this can be “0”).

1402 If no match is found among the set of pre-existing registrations, the registrar MUST create a new binding
1403 in its location service between the address of record and the current Contact header field. All Contact
1404 header field parameters are copied verbatim into this new binding (again with the exception of “expires”).
1405 An expiration interval MUST be selected by the registrar, taking into account any suggested expiration for

1406 this contact address in the REGISTER.

1407 Allowing the registrar to set the registration interval protects it against excessively frequent registration refreshes
1408 while limiting the state that it needs to maintain and decreasing the likelihood of registrations going stale.

1409 The expiration interval mandated by the registrar may be either longer or shorter than the interval sug-
1410 gested by the sender of the REGISTER, though the registrar SHOULD abide by the registering client's
1411 suggestion.

1412 A server MAY decide to lengthen the expiration interval if the refresh rate of a particular client exceeds a thresh-
1413 old, for example.

1414 After the expiration interval selected by the registrar for a binding has passed, if the binding has not been
1415 refreshed (increasing the expiration interval), the registrar SHOULD silently discard the binding.

1416 Once all bindings in the location service have been updated to reflect any changes present to contact
1417 addresses in the REGISTER message, the registrar MUST remove any bindings that expire immediately.

1418 The REGISTER might have set the expiration interval for some bindings to "0" to remove them before their
1419 expiration interval passes.

1420 Finally, the registrar must generate a response. If the address of record given in the To header field of
1421 the REGISTER method is valid for its administrative domain, then a 200 response MUST be sent, which
1422 MUST contain a complete list (within Contact header fields) of the currently valid bindings in the location
1423 service associated with the address of record contained in the To field of the REGISTER request. This list
1424 MAY be empty (in which case the 200 would not contain any Contact headers).

1425 In a successful response to a REGISTER, wherein the bindings for this address of record are enumerated
1426 as described above, the registrar MUST supply an expiration interval for each contact address in either an
1427 "expires" parameter of a Contact header or an Expires header. This interval specifies the expiration interval
1428 that has been mandated by the registrar (taking into account the registering UA's suggestion).

1429 If the registration failed because the address of record contained in the To field of the REGISTER is not
1430 valid for this domain, then a 404 MUST be sent.

1431 **11 Querying for Capabilities**

1432 The SIP method OPTIONS allows a client to query another client or server as to its capabilities. This
1433 allows a client to discover information about the methods, content types, extensions, codecs etc. supported
1434 without actually "ringing" the other party. For example, before a client inserts a Require header field into
1435 an INVITE listing an option that it is not certain the destination UAS supports, the client can query the
1436 destination UAS with an OPTIONS to see if this option is returned in a Supported header field.

1437 The target of the OPTIONS request is identified by the Request-URI, which could identify another
1438 User Agent or a SIP Server. Alternatively, a server receiving an OPTIONS request with a Max-Forwards
1439 header value of 0 MAY respond to the request regardless of the Request-URI.

1440 This behavior is common with HTTP/1.1.

1441 An OPTIONS request sent as part of an established dialog does not have any impact on the dialog.

1442 **11.1 Construction of OPTIONS Request**

1443 An OPTIONS request is constructed using the standard rules for a SIP request as discussed Section 8.1.1.

1444 A Contact header field MAY be present in an OPTIONS.

1445 OPEN ISSUE #197: What is the semantic of this Contact

1446 An **Accept** header field **SHOULD** be included to indicate the type of message body the UAC wishes to
1447 receive in the response.

1448 Example **OPTIONS** request:

```
1449 OPTIONS sip:carol@chicago.com SIP/2.0
1450 Via: SIP/2.0/UDP 10.1.1.1:5060;branch=23411513a6
1451 Via: SIP/2.0/UDP 10.1.3.3:5060
1452 To: <sip:carol@chicago.com>
1453 From: Alice <sip:alice@atlanta.com>;tag=1928301774
1454 Call-ID: a84b4c76e66710@10.1.3.3
1455 CSeq: 63104 OPTIONS
1456 Contact: <sip:alice@10.1.3.3>
1457 Accept: application/sdp
1458 Contact-Length: 0
```

1459 11.2 Processing of **OPTIONS** Request

1460 The response to an **OPTIONS** is constructed using the standard rules for a SIP response as discussed in
1461 Section 8.2.7. The response code chosen is the same that would have been chosen had the request been an
1462 **INVITE**. That is, a 200 (OK) would be returned if the UAS is ready to accept a call, a 486 (Busy Here)
1463 would be returned if the UAS is busy, etc. This allows an **OPTIONS** request to be used to determine the
1464 basic state of a UAS, which can be an indication of whether the UAC will accept an **INVITE** request.

1465 Note that this use of **OPTIONS** has limitations due the differences in proxy handling of **OPTIONS** and
1466 **INVITE** requests. While a forked **INVITE** can result in multiple 200 OK responses being returned, a forked
1467 **OPTIONS** will only result in a single 200 OK response, since it is treated by proxies using the non-**INVITE**
1468 handling. See Section 13.2.1 for the normative details.

1469 **Allow**, **Accept**, **Accept-Encoding**, **Accept-Language**, and **Supported** header fields **SHOULD** be
1470 present in a 200 OK response to an **OPTIONS** request.

1471 A **Contact** header field **MAY** be present in a 200 OK response.

1472 A **Warning** header field **MAY** be present.

1473 A message body **MAY** be sent, the type of which is determined by the **Accept** header in the **OPTIONS**
1474 request.

1475 Example **OPTIONS** response (corresponding to the request in Section 11.1):

```
1476 SIP/2.0 200 OK
1477 Via: SIP/2.0/UDP 10.1.1.1:5060;branch=23411513a6
1478 Via: SIP/2.0/UDP 10.1.3.3:5060
1479 To: <sip:carol@chicago.com>;tag=93810874
1480 From: Alice <sip:alice@atlanta.com>;tag=1928301774
1481 Call-ID: a84b4c76e66710@10.1.3.3
1482 CSeq: 63104 OPTIONS
1483 Contact: <sip:carol@10.3.6.6>
1484 Allow: INVITE, ACK, CANCEL, OPTIONS, BYE
1485 Accept: application/sdp
```

```
1486   Accept-Encoding: gzip
1487   Accept-Language: en
1488   Supported: foo
1489   Content-Type: application/sdp
1490   Contact-Length: 274
1491
1492   v=0
1493   o=carol 28908764872 28908764872 IN IP4 10.3.6.6
1494   s=-
1495   t=0 0
1496   c=IN IP4 10.3.6.6
1497   m=audio 0 RTP/AVP 0 1 3 99
1498   a=rtpmap:0 PCMU/8000
1499   a=rtpmap:1 1016/8000
1500   a=rtpmap:3 GSM/8000
1501   a=rtpmap:99 SX7300/8000
1502   m=video 0 RTP/AVP 31 34
1503   a=rtpmap:31 H261/90000
1504   a=rtpmap:34 H263/90000
```

1505 12 Dialogs

1506 A key concept for a user agent is that of a dialog. A dialog represents a peer- to-peer SIP relationship between
1507 a two user agents that persists for some time. The dialog facilitates sequencing of messages between the
1508 user agents, and proper routing of requests between both them. The dialog represents a context in which to
1509 interpret SIP messages. The previous section discussed method independent UA processing for requests and
1510 responses outside of a dialog. This section discusses how those requests and responses are used to construct
1511 a dialog, and then how subsequent requests and responses are sent within a dialog.

1512 A dialog is identified at each UA with a dialog ID, which consists of a **Call-ID** value, a local URI and
1513 local tag (together called the local address), and a remote URI and remote tag (together called the remote
1514 address). The dialog ID at each UA involved in the dialog is not the same. Specifically, the local URI and
1515 local tag at one UA are identical to the remote URI and remote tag at the peer UA. The tags are opaque
1516 tokens that facilitate the generation of unique dialog IDs.

1517 A dialog ID is also associated with all responses, and with any request that contains a tag in the **To** field.
1518 The rules for computing the dialog ID of a message depend on whether the entity is a UAC or UAS. For a
1519 UAC, the **Call-ID** value of the dialog ID is set to the **Call-ID** of the message, the remote address is set to the
1520 **To** field of the message, and the local address is set to the **From** field of the message (these rules apply to
1521 both requests and responses). As one would expect, for a UAS, the **Call-ID** value of the dialog ID is set to
1522 the **Call-ID** of the message, the remote address is set to the **From** field of the message, and the local address
1523 is set to the **To** field of the message.

1524 A dialog contains certain pieces of state needed for further message transmissions within the dialog.
1525 This state consists of the **Call-ID**, a local sequence number (used to order requests from the UA to its peer),
1526 a remote sequence number (used to order requests from its peer to the UA), and a route set, which is an
1527 ordered list of URIs. The route set is the set of servers that need to be traversed to send a request to the peer.
1528 A dialog can also be in the “early” state, which occurs when it is created with a provisional response, and

1529 then transition to the “established” state when the final response comes.

1530 **12.1 Creation of a Dialog**

1531 Dialogs are created through the generation of non-failure responses to requests with specific methods.
1532 Within this specification, only the 2xx and 1xx responses to INVITE establish a dialog. A dialog estab-
1533 lished by a non-final response to a request is called an early dialog. Extensions MAY define other means for
1534 creating dialogs. Section 13 gives more details that are specific to the INVITE method. Here, we describe
1535 the process for creation of dialog state that is not dependent on the method.

1536 **12.1.0.1 UAS** When a UAS responds to a request with a response that establishes a dialog (such as a
1537 2xx to INVITE), the UAS MUST copy all Record-Route headers from the request into the response, and
1538 MUST maintain the order of those headers. This includes the URIs, URI parameters, and any Record-
1539 Route header parameters, whether they are known or unknown to the UAS. The UAS MUST add a Contact
1540 header field to the response. The Contact header field contains an address where the UAS would like to
1541 be contacted for subsequent requests in the dialog (which includes the ACK for a 2xx response in the case
1542 of an INVITE). Generally, the host portion of this URI is the IP address of the host, or its FQDN. The URI
1543 provided in the Contact header MUST be a SIP URL.

1544 The UAS then constructs the state of the dialog. This state MUST be maintained for the duration of the
1545 dialog. First, the route set MUST be computed by following these steps:

- 1546 1. The list of URIs in the Record-Route headers in the request, if present, are taken, including any URI
1547 parameters.
- 1548 2. The URI in the Contact header from the request if present, is taken, including any URI parameters.
1549 The URI is appended to the bottom of the list of URIs from the previous step.

1550 Contact was not mandatory in RFC2543. Thus, if the UAS is talking to an older UAC, the UAC might not
1551 have inserted the Contact header.

- 1552 3. The resulting list of URIs is called the *route set*.

1553 These rules clearly imply that a UA MUST be able to parse and process Record-Route header fields. This is a
1554 change from RFC2543, where all record-route and route processing was optional for user agents.

1555 It is possible for the *route set* to be empty. This will occur if neither Record-Route headers nor a
1556 Contact header were present in the request. The UAS MUST also remember whether the bottom-most entry
1557 in the *route set* was constructed from a Contact header or not. This is effectively a boolean value, which we
1558 refer to as CONTACT_SET. This is needed in order for the UA to determine whether the bottom most value
1559 can be updated from subsequent requests; if it was constructed from a Contact, it can be updated.

1560 The remote sequence number sequence number MUST be set to the value of the sequence number in the
1561 Cseq header of the request. The local sequence number MUST be empty. The call identifier component
1562 of the dialog ID MUST be set to the value of the Call-ID in the request. The local address component of
1563 the dialog ID MUST be set to the To field in the response to the request (which therefore includes the tag),
1564 and the remote address component of the dialog ID MUST be set to the From field in the request. A UAS
1565 MUST be prepared to receive a request without a tag in the From field, in which case the tag is considered
1566 to effectively have a value of null.

1567 This is to maintain backwards compatibility with RFC2543, which did not mandate From tags.

1568 **12.1.0.2 UAC** When a UAC receives a response that establishes a dialog, it constructs the state of the
1569 dialog. This state **MUST** be maintained for the duration of the dialog. First, the route set **MUST** be computed
1570 by following these steps:

- 1571 1. The list of URIs present in the **Record-Route** headers in the response are taken, if present, including
1572 all URI parameters, and their order is reversed.
- 1573 2. The URI in the **Contact** header from the response, if present, is taken, including all URI parameters,
1574 and appended to the end of the list from the previous step.
- 1575 3. The list of URIs resulting from the above two operations is referred to as the *route set*.

1576 It is possible for the *route set* to be empty. This will occur if neither **Record-Route** headers nor a
1577 **Contact** header were present in the response. The UAC **MUST** also remember whether the bottom-most
1578 entry in the *route set* was constructed from a **Contact** header or not. This is effectively a boolean value,
1579 which we refer to as **CONTACT.SET**. This is needed in order for the UA to determine whether the bottom
1580 most value can be updated from subsequent requests; if it was constructed from a **Contact**, it can be updated.

1581 The local sequence number **MUST** be set to the value of the sequence number in the
1582 **Cseq** header of the request. The remote sequence number **MUST** be empty (it is established when the UA
1583 sends a request within the dialog). The call identifier component of the dialog ID **MUST** be set to the value
1584 of the **Call-ID** in the request. The local address component of the dialog ID **MUST** be set to the **From**
1585 field in the request, and the remote address component of the dialog ID **MUST** be set to the **To** field of the
1586 response. A UAC **MUST** be prepared to receive a response without a tag in the **To** field, in which case the
1587 tag is considered to effectively have a value of null.

1588 This is to maintain backwards compatibility with RFC2543, which did not mandate **To** tags.

1589 12.2 Requests within a Dialog

1590 Once a dialog has been established between two UAs either of them **MAY** initiate new transactions as needed
1591 within the dialog. However, a dialog imposes some restrictions on the use of simultaneous transactions.

1592 A TU **MUST NOT** initiate a new regular transaction within a dialog while a regular transaction is in
1593 progress (in either direction) within that dialog.

1594 OPEN ISSUE #113: Should we relax the constraint on non-overlapping regular transactions?

1595 A refresh request sent within a dialog is defined as a request that can modify the *route set* of the dialog.
1596 For dialogs that have been established with an **INVITE**, the only refresh request defined is re-**INVITE** (see
1597 Section 14). Other extensions may define different refresh requests for dialogs established in other ways.

1598 Note that an **ACK** is *NOT* a refresh request.

1599 12.2.1 UAC Behavior

1600 **12.2.1.1 Generating the Request** A request within a dialog is constructed by using many of the com-
1601 ponents of the state stored as part of the dialog.

1602 The **To** header field of the request **MUST** be set to the remote address, and the **From** header field **MUST**
1603 be set to the local address (both including tags, assuming the tags are not null).

1604 The **Call-ID** of the request **MUST** be set to the **Call-ID** of the dialog. Requests within a dialog **MUST**
1605 contain strictly monotonically increasing and contiguous **CSeq** sequence numbers (increasing-by-one) in
1606 each direction. Therefore, if the local sequence number is not empty, the value of the local sequence number
1607 **MUST** be incremented by one, and this value **MUST** be placed into the **Cseq** header. If the local sequence

1608 number is empty, an initial value **MUST** be chosen using the guidelines of Section 8.1.1.4. The method field
1609 in the **Cseq** header **MUST** match the method of the request.

1610 With a length of 32 bits, a client could generate, within a single call, one request a second for about 136 years
1611 before needing to wrap around. The initial value of the sequence number is chosen so that subsequent requests within
1612 the same call will not wrap around. A non-zero initial value allows clients to use a time-based initial sequence
1613 number. A client could, for example, choose the 31 most significant bits of a 32-bit second clock as an initial
1614 sequence number.

1615 The **Request-URI** of requests is determined according to the following rules:

1616 The **UAC** takes the list of **URI** in the *route set*. The top **URI** **MUST** be inserted into the request **URI** of
1617 the request, including all **URI** parameters. Any **URI** parameters not allowed in the request **URI** **MUST** then
1618 be stripped. Each of the remaining **URIs** (if any) from the *route set*, including all **URI** parameters, **MUST** be
1619 placed into a **Route** header field into the request, in order.

1620 A **TU** **SHOULD** follow the rules just mentioned to build the **Request-URI** of the request, regardless of
1621 whether the **UA** uses an outbound proxy server or not. However, in some instances, a **UA** may not be willing
1622 or capable of sending the request to the top element in the *route set*. One example is a **UA** that is not capable
1623 of **DNS**, and therefore may not be able to follow those procedures. In these cases, the **UA** **MAY** send the
1624 request to a local outbound server. In this case, it **MUST NOT** remove the top **Route** header.

1625 In dialogs created by an **INVITE**, if the **UA** is the caller, it sets the **Request-URI** to the same value it used for
1626 the initial request, and sends it to its local outbound server.

1627 Bug#161: Which **Request-URI** does the callee use?

1628 A **UAC** **SHOULD** include a **Contact** header in any refresh requests within a dialog, and unless there is a
1629 need to change it, the **URI** **SHOULD** be the same as used in previous requests within the dialog. As discussed
1630 in Section 12.2.2, a **Contact** header in a refresh request updates the route set. This allows a **UA** to provide
1631 a new contact address, should its address change during the duration of the dialog.

1632 However, requests that are not refresh requests do not affect the *route set* for the dialog.

1633 Once the request has been constructed, the address of the server is computed and the request is sent,
1634 using the same procedures for requests outside of a dialog (Section 8.1.1).

1635 **12.2.1.2 Processing the Responses** The **UAC** will receives responses to the request from the transaction
1636 layer.

1637 The behavior of a **UAC** that receives a 3xx response for a request sent within a dialog is the same as if
1638 the request would have been sent outside a dialog. This behavior is described in Section 13.2.2.

1639 Note however that when the **UAC** tries alternative locations it still uses the *route set* for the dialog to build the
1640 **Route** header of the request.

1641 If a **UAC** has a *route set* for a dialog, and receives a 2xx response to a refresh it sent, the **Contact** header
1642 field of the response is examined. If not present, the *route set* remains unchanged. If the response had a
1643 **Contact** header field, and the boolean variable **CONTACT_SET** is false, the **URL** in the **Contact** header
1644 field in the response is added to the bottom of the *route set*, and **CONTACT_SET** is set to true. If the refresh
1645 request response had a **Contact** header field, and **CONTACT_SET** is true, the **URL** in the **Contact** header
1646 field of the response to the refresh request replaces the bottom value in the *route set*. If a refresh request is
1647 responded with a non-2xx final response the *route set* remains unchanged as if no refresh request had been
1648 issued.

1649 If the response for the a request within a dialog is a 481 (Call/Transaction Does Not Exist) or a 408
1650 (Request Timeout) the **UAC** **SHOULD** terminate the dialog.

1651 For **INVITE** initiated dialogs terminating the dialog consists of sending a **BYE**.

1652 12.2.2 UAS behavior

1653 The UAS will receive the request from the transaction layer. If the request has a tag in the To header field,
1654 the UAS core computes the dialog identifier corresponding to the request and compares it with existing
1655 dialogs. If there is a match, this is a mid-dialog request. In that case, the same processing rules for requests
1656 outside of a dialog, discussed in Section 8.2, are applied by the UAS once the request is received from the
1657 transaction layer.

1658 Requests that do not change in any way the state of a dialog may be received within a dialog (e.g., an
1659 OPTIONS request). They are processed as if they had been received outside the dialog.

1660 Requests within a dialog MAY contain Record-Route and Contact header fields. However, requests
1661 that are not refresh requests do not update the *route set* for the dialog. This specification only defines one
1662 refresh request: re-INVITE (see Section 14).

1663 Special rules apply when updated Record-Route or Contact header fields are received inside a refresh
1664 request. If a UAS has a *route set* for a dialog, and receives a refresh for that dialog containing Record-
1665 Route header fields, it MUST copy those header fields into any 2xx response to that request. If the boolean
1666 variable CONTACT_SET is true, the Contact header field in the request (if present) replaces the last entry in
1667 the *route set*. If the boolean variable CONTACT_SET is false, the UAS MUST add the URL in the Contact
1668 header field in the re- INVITE to the bottom of the *route set*, and then set CONTACT_SET to true. If the
1669 request did not contain a Contact header field, the route-set at the UAS remains unchanged.

1670 If the remote sequence number is empty, it MUST be set to the value of the sequence number in the Cseq
1671 header in the request. If the remote sequence number was not empty, but the sequence number of the request
1672 is lower than the remote sequence number, the request is out of order and MUST be rejected with a 500
1673 response. If the remote sequence number was not empty, and the sequence number of the request is greater
1674 than the remote sequence number, the request is in order. It is possible for the CSeq header to be higher
1675 than the remote sequence number by more than one. This is not an error condition, and a UAS SHOULD be
1676 prepared to receive and process requests with CSeq values more than one higher than the previous received
1677 request. The UAS MUST then set the remote sequence number to the value of the sequence number in the
1678 Cseq header in the request.

1679 12.3 Termination of a Dialog

1680 Dialogs can end in several different ways, depending on the method. When a dialog is established with
1681 INVITE, it is terminated with a BYE. No other means to terminate a dialog are described in this specification,
1682 but extensions can define other ways.

1683 13 Initiating a Session

1684 13.1 Overview

1685 When a user agent client desires to initiate a session (for example, audio, video, or a game), it formulates
1686 an INVITE request. The INVITE request asks a server to establish a session. This request is forwarded by
1687 proxies, eventually arriving at one or more UAS which can potentially accept the invitation. These UAS's
1688 will frequently need to query the user about whether to accept the invitation. After some time, those UAS can
1689 accept the invitation (meaning the session is to be established) by sending a 2xx response. If the invitation
1690 is not accepted, a 3xx,4xx,5xx or 6xx response is sent, depending on the reason for the rejection. Before

1691 sending a final response, the UAS can also send a provisional response (1xx) to advise the UAC of progress
1692 in contacting the called user.

1693 After possibly receiving one or more provisional responses, the UA will get one or more 2xx responses or
1694 one non-2xx final response. Because of the protracted amount of time it can take to receive final responses
1695 to INVITE, the reliability mechanisms for INVITE transactions differ from those of other requests (like
1696 OPTIONS). Once it receives a final response, the UAC needs send an ACK for every final response it
1697 receives. The procedure for sending this ACK depends on the type of response. For final responses between
1698 300 and 699, the ACK processing is done in the transaction layer, and follows one set of rules (See Section
1699 17). For 2xx responses, the ACK is generated by the UAC core.

1700 A 2xx response to an INVITE establishes a session, and it also creates a dialog between the UA that
1701 issued the INVITE and the UA that generated the 2xx response. Therefore, when multiple 2xx responses are
1702 received from different remote UAs (because the INVITE forked), each 2xx establishes a different dialog.
1703 All these dialogs are part of the same call.

1704 This section provides details on the establishment of a session using INVITE.

1705 **13.2 Caller Processing**

1706 **13.2.1 Creating the Initial INVITE**

1707 Since the initial INVITE represents a request outside of a dialog, its construction follows the procedures of
1708 Section 8.1.1. Additional processing is required for the specific case of INVITE.

1709 An Allow header field (Section 22.5) SHOULD be present in the INVITE. It indicates what methods can
1710 be invoked within a dialog, on the UA sending the INVITE, for the duration of the dialog. For example, a
1711 UA capable of receiving INFO requests within a dialog [21] SHOULD include an Allow header listing the
1712 INFO method.

1713 A Supported header field (Section 22.35) SHOULD be present in the INVITE. It enumerates all the
1714 extensions understood by the UAC.

1715 An Accept (Section 22.1) header field MAY be present in the INVITE. It indicates which content-types
1716 are acceptable to the UA, in both the response received by it, and in any subsequent requests sent to it within
1717 dialogs established by the INVITE. The Accept header is especially useful for indicating support of various
1718 session description formats.

1719 The UA MAY add an Expires header field (Section 22.19) to limit the validity of the invitation. If the
1720 time indicated in the Expires header field is reached and no final answer for the INVITE has been received
1721 the UAC core SHOULD generate a CANCEL request for the original INVITE.

1722 A UAC MAY also find useful to add, among others, Subject (Section 22.34), Organization (Section
1723 22.24) and User-Agent (Section 22.39) header fields. They all contain useful information related to the
1724 INVITE.

1725 The UAC MAY choose to add a message body to the INVITE. Section 8.1.1.9 deals with how to construct
1726 the header fields- Content-Type among others- needed to describe the message body.

1727 There are special rules for message bodies that contain a session description - their corresponding
1728 Content-Disposition is "session". SIP uses an offer/answer model where one UA sends a session de-
1729 scription, called the offer, which contains a proposed description of the session. The offer indicates the
1730 desired communications means (audio, video, games), parameters of those means (such as codec types) and
1731 addresses for receiving media from the offerer. The other UA responds with another session description,
1732 called the answer, which indicates which communications means are accepted, the parameters which ap-
1733 ply to those means, and addresses for receiving media from the answerer. The offer/answer model can be

1734 mapped into the INVITE transaction in two ways. The first, which is the most intuitive, is that the INVITE
1735 contains the offer, the 2xx response contains the answer, and no session description is provided in the ACK.
1736 In this model, the UAC is the offerer, and the UAS is the answerer. A second model is that the INVITE con-
1737 tains no session description, the 2xx response contains the offer, and the ACK contains the answer. In this
1738 model, the UAS is the offerer, and the UAC is the answerer. The second model is useful for gateways from
1739 H.323v1 to SIP, where the H.323 media characteristics are not known until the call is established. This is
1740 also useful for sessions that use third-party call control. As a result of these models, if the INVITE contains
1741 a session description, the ACK MUST NOT contain one. Conversely, if the caller chooses to omit the session
1742 description in the INVITE, the ACK MUST contain one (if a 2xx response is received). 2xx responses to
1743 an INVITE MUST always contain a session description. All user agents that support INVITE MUST support
1744 both models.

1745 The Session Description Protocol (SDP) [6] MUST be supported by all user agents as a means to describe
1746 sessions, and its usage for construction offers and answers MUST follow the procedures defined in [22].

1747 Note that the restrictions of the offer-answer model (session description only in the INVITE OR in
1748 the ACK, but not in both) just described only apply to bodies whose Content-Disposition header field
1749 is "session". Therefore, it is possible that both the INVITE and the ACK contain a body message (e.g.,
1750 the INVITE carries a photo (Content-Disposition: render) and the ACK a session description (Content-
1751 Disposition: session)).

1752 If the Content-Disposition header field is missing, bodies of Content-Type application/sdp imply the
1753 disposition "session", while other content types imply "render".

1754 Once the INVITE has been created, the UAC follows the procedures defined for sending requests outside
1755 of a dialog (Section 8). This results in the construction of a client transaction that will ultimately send the
1756 request and deliver responses to the UAC.

1757 If a UA *A* sends an INVITE request to *B* and receives an INVITE request from *B* before it has received
1758 the response to its request from *B*, *A* MAY return a 500 (Internal Server Error), which SHOULD include a
1759 Retry-After header field specifying when the request should be resubmitted.

1760 13.2.2 Processing INVITE Responses

1761 Once the INVITE has been passed to the INVITE client transaction, the UAC waits for responses for the IN-
1762 VITE. Responses are matched to their corresponding INVITE because they have the same Call-ID, the same
1763 From header field, the same To header field, excluding the tag, and the same CSeq. Rules for comparisons
1764 of these headers are described in Section 22.

1765 **13.2.2.1 1xx responses** Zero, one or multiple provisional responses may arrive before one or more
1766 final responses are received. Provisional responses for an INVITE request can create "early dialogs". If a
1767 provisional response has a tag in the To field, and if the dialog ID of the response does not match an existing
1768 dialog, one is constructed using the procedures defined in Section 12.1.0.2.

1769 The early dialog will only be needed if the UAC needs to send a request to its peer within the dialog
1770 before the initial INVITE transaction completes. Header fields present in a provisional response are appli-
1771 cable for the duration of the early dialog (e.g., an Allow header field in a provisional response contains the
1772 methods that can be used in the early dialog).

1773 **13.2.2.2 3xx responses** A 3xx response may contain a Contact header field providing new addresses
1774 where the callee might be reachable. Depending on the status code of the 3xx response (see Section 23.3)
1775 the UAC MAY choose to try those new addresses.

1776 **13.2.2.3 4xx, 5xx and 6xx responses** A single non-2xx final response may be received for the IN-
1777 VITE. 4xx, 5xx and 6xx responses may contain a **Contact** header field indicating the location where addi-
1778 tional information about the error can be found.

1779 All early dialogs are considered terminated upon reception of the non-2xx final response.

1780 After having received the non-2xx final response the UAC core considers the INVITE transaction com-
1781 pleted. The INVITE client transaction handles generation of ACKs for the response (see Section 17).

1782 **13.2.2.4 2xx responses** Multiple 2xx responses may arrive at the UAC for a single INVITE request
1783 due to a forking proxy. Each response is distinguished by the **tag** parameter in the **To** header field, and each
1784 represents a distinct dialog, with a distinct dialog identifier.

1785 If the dialog identifier in the 2xx response matches the dialog identifier of an existing dialog, the dialog
1786 MUST be transitioned to the “established”, and the route set for the dialog MUST be recomputed based on the
1787 2xx response using the procedures of Section 12.1.0.2. Otherwise, a new established dialog is constructed
1788 in the same fashion.

1789 The route set only is recomputed for backwards compatibility. RFC 2543 did not mandate mirroring of **Record-**
1790 **Route** headers in a 1xx, only 2xx. However, we cannot update the entire state of the dialog, since mid-dialog
1791 requests may have been sent within the early call leg, modifying the sequence numbers, for example.

1792 The UAC core MUST generate an **ACK** request for each 2xx received from the transaction layer. The
1793 header fields of the **ACK** are constructed in the same way as for any request sent within a dialog (see Section
1794 12) with the exception of the **CSeq**. The sequence number of the **CSeq** header field MUST be the same as
1795 the INVITE being acknowledged, but the **CSeq** method MUST be **ACK**. If the INVITE did not contain an
1796 offer, the 2xx will contain one, and therefore the **ACK** MUST carry an answer in its body.

1797 Once the **ACK** has been constructed, the procedures of Section 24 are used to send it. However, the
1798 request is passed to the transport layer directly for transmission, rather than a client transaction. This is
1799 because the UAC core handles retransmissions of the **ACK**, not the transaction layer. The **ACK** MUST be
1800 passed to the client transport every time a retransmission of the 2xx final response that triggered the **ACK**
1801 arrives.

1802 The UAC core considers the INVITE transaction completed $62 * T1$ seconds after the reception of the
1803 first 2xx response. At this point all the early dialogs that have not transitioned to established dialogs are
1804 terminated. Once the INVITE transaction is considered completed by the UAC core, no more new 2xx
1805 responses are expected to arrive.

1806 If, after acknowledging any 2xx response to an INVITE, the caller does not want to continue with that
1807 dialog, then the caller MUST terminate the dialog by sending a **BYE** request as described in Section 15.

1808 **13.3 Callee Processing**

1809 **13.3.1 Processing of the INVITE**

1810 The UAS core will receive INVITE requests from the transaction layer. It first performs the request process-
1811 ing procedures of Section 8.2, which are applied for both requests inside and outside of a dialog.

1812 Assuming these processing states complete without generating a response, the UAS core performs the
1813 additional processing steps:

- 1814 1. If the request is an INVITE that contains an **Expires** header field the UAS core inspects this header
1815 field. If the INVITE has already expired a 487 response is generated.

- 1816 2. If the request has no tag in the **To** the UAS core checks ongoing transactions. If the **To**, **From**, **Call-ID**,
1817 **CSeq** exactly match (including tags) those of any request received previously, but the **branch-ID** in
1818 the topmost **Via** is different from those received previously, the UAS core SHOULD generate a 482
1819 (Loop detected) response and pass it to the server transaction.

1820 The same request that was generated by the UAC has arrived to the UAS more than once following different
1821 paths. The UAS processes the request that was received first and responds with 482 (Loop detected) to the rest
1822 of them.

1823 If no match is found, the request does not belong to any existing dialog. If the request is an **INVITE**
1824 the UAS core follows the procedures described in this section.

- 1825 3. If the request is a mid-dialog request, the method-independent processing described in Section 12.2.2
1826 is first applied. It might also modify the session; Section 14 provides details.

- 1827 4. If the request has a tag in the **To** header field but the dialog identifier does not match any of the
1828 existing dialogs, the UAS may have crashed and restarted, or may have received a request for a
1829 different (possibly failed) UAS. The UAS MAY either accept or reject the request. Accepting the
1830 request provides robustness, so that dialogs can persist even through crashes. UAs wishing to support
1831 this capability must choose monotonically increasing **CSeq** sequence numbers even across reboots.
1832 This is because subsequent requests from the crashed-and-rebooted UA towards the other UA need to
1833 have a **CSeq** sequence number higher than previous requests in that direction.

1834 Note also that the crashed-and-rebooted UA will have lost any **Route** headers which would need to
1835 be inserted into a subsequent request. Therefore, it is possible that the requests may not be properly
1836 forwarded by proxies.

1837 RTP media agents allowing restarts need to be robust by accepting out-of-range timestamps and sequence
1838 numbers.

1839 If the UAS wishes to reject the request, because it does not wish to recreate the dialog, it MUST
1840 respond to the request with a 481 (Call/Transaction Does Not exist) status code and pass that to the
1841 server transaction.

1842 Processing from here forward assumes that the **INVITE** is outside of a dialog, and is thus for the purposes
1843 of establishing a new session.

1844 The **INVITE** may contain a session description, in which case the UAS is being presented with an offer
1845 for that session. It is possible that the user is already a participant in that session, even though the **INVITE**
1846 is outside of a dialog. This can happen when a user is invited to the same multicast conference by multiple
1847 other participants. If desired, the UAS MAY use identifiers within the session description to detect this
1848 duplication. For example, **SDP** contains a session id and version number in the origin (**O**) field. If the user
1849 is already a member of the session and the session parameters contained in the session description have not
1850 changed, the UAS MAY silently accept the **INVITE**

1851 The **INVITE** may not contain a session description at all, in which case the UAS is being asked to
1852 participate in a session, but the UAC has asked that the UAS provide the offer of the session.

1853 The callee can indicate progress, accept, redirect, or reject the invitation. In all of these cases, it formu-
1854 lates a response using the procedures described in Section 8.2.7.

1855 **13.3.1.1 Progress** The UAS may not be able to answer the invitation immediately, and might choose
1856 to indicate some kind of progress to the caller (for example, an indication that a phone is ringing). This is
1857 accomplished with a provisional response between 101 and 199. These provisional responses establish early
1858 dialogs and therefore follow the procedures of Section 12.1.0.1 in addition to those of Section 8.2.7. A UAS
1859 MAY send as many provisional responses as it likes. Each of these MUST indicate the same dialog ID. SIP,
1860 however, does not guarantee that these provisional responses are reliably delivered to the UAC.

1861 **13.3.1.2 The INVITE is redirected** If the UAS decides to redirect the call, a 3xx response is sent. A
1862 300 (Multiple Choices), 301 (Moved Permanently) or 302 (Moved Temporarily) response SHOULD contain
1863 a **Contact** header field containing URIs of new addresses to be tried. The response is passed to the INVITE
1864 server transaction, which will deal with its retransmissions.

1865 **13.3.1.3 The INVITE is rejected** A common scenario occurs when the callee is currently not willing
1866 or able to take additional calls at this end system. A 486 (Busy Here) SHOULD be returned in such scenario.
1867 If the UAS knows that no other end system will be able to accept this call a 600 (Busy Everywhere) response
1868 SHOULD be sent instead. However, it is unlikely that a UAS will be able to know this in general, and thus
1869 this response will not usually be used. The response is passed to the INVITE server transaction, which will
1870 deal with its retransmissions.

1871 **13.3.1.4 The INVITE is accepted** The UAS core generates a 2xx response. This response establishes
1872 a dialog, and therefore follows the procedures of Section 12.1.0.1 in addition to those of Section 8.2.7.

1873 A 2xx response to an INVITE SHOULD contain the **Allow** header field and the **Supported** header field,
1874 and MAY contain the **Accept** header field. Including these header fields allows the UAC to determine the
1875 features and extensions supported by the UAS for the duration of the call, without probing.

1876 If the INVITE request contained an offer, the 2xx MUST contain an answer. If the INVITE did not contain
1877 an offer, the 2xx MUST contain an offer.

1878 Once the response has been constructed it is passed to the INVITE server transaction. Note, however,
1879 that the INVITE server transaction does not retransmit 2xx responses to an INVITE. Therefore, it is neces-
1880 sary to pass periodically the response to the server transaction until the ACK arrives. The 2xx response is
1881 resubmitted to the server transaction with an interval that starts at T1 seconds and doubles for each retrans-
1882 mission until it reaches T2 seconds (T1 and T2 are defined in Section 17). Response retransmissions cease
1883 when an ACK request is received with the same dialog ID as the response. This is independent of whatever
1884 transport protocols are used to send the response.

1885 Since 2xx is retransmitted end-to-end, there may be hops between UAS and UAC which are UDP. To ensure
1886 reliable delivery across these hops, the response is retransmitted periodically even if the transport at the UAS is
1887 reliable.

1888 If the server retransmits the 2xx response for $64 * T1$ seconds without receiving an ACK, it considers the
1889 dialog completed, the session terminated, and therefore it SHOULD send a BYE.

1890 14 Modifying an Existing Session

1891 A successful INVITE request (see Section 13) establishes both a dialog between two user agents and a
1892 session (using the offer/answer model). Section 12 explains how to modify an existing dialog using a
1893 refresh request (e.g., changing the *route set* of the dialog). This section describes how to modify the actual

1894 session. This modification can involve changing addresses or ports, adding a media stream, deleting a media
1895 stream, and so on. This is accomplished by sending a new INVITE request within the same dialog that
1896 established the session. An INVITE request sent within an existing dialog is known as a re-INVITE.

1897 Note that a single re-INVITE can modify at the same time the dialog and the parameters of the session.

1898 Either the caller or callee can modify an existing session.

1899 14.1 UAC Behavior

1900 The same offer-answer model that applies to session descriptions in INVITEs (Section 13.2.1) applies to
1901 re-INVITEs. As a result, a UAC that wants to add a media stream, for example, will create a new offer that
1902 contains this media stream, and send that in an INVITE request to its peer. It is important to note that the
1903 full description of the session, not just the change, is sent. This maintains the idempotency of SIP, supports
1904 stateless session processing in various elements, and supports failover and recovery capabilities. Of course,
1905 a UAC MAY send a re-INVITE with no session description, in which case the response to the re-INVITE will
1906 contain the offer.

1907 If the session description format has the capability for version numbers, the offerer SHOULD indicate
1908 that the version of the session description has changed.

1909 The To, From, Call-ID, CSeq, and Request-URI of a re-INVITE are set following the same rules as
1910 for regular requests within an existing dialog, described in Section 12.

1911 Note that, as opposed to initial INVITEs (see Section 13), re-INVITEs contain tags in the To header
1912 field and are sent using the *route set* for the dialog. Therefore, a single final (2xx or non-2xx) response is
1913 received for re-INVITEs.

1914 Note that a UAC MUST NOT initiate a new INVITE transaction within a dialog while another transaction
1915 (INVITE or non-INVITE) is in progress. However, a UA MAY initiate a regular transaction within an early
1916 dialog - while an INVITE transaction is in progress.

1917 If a re-INVITE is responded with a non-2xx final response the session parameters MUST remain un-
1918 changed, as if no re-INVITE had been issued.

1919 The rules for transmitting a re-INVITE and for generating an ACK for a 2xx response to re-INVITE are
1920 the same as for an INVITE (Section 13.2.1).

1921 14.2 UAS Behavior

1922 Section 13.3.1 describes the steps to follow in order to distinguish incoming re-INVITEs from incoming
1923 initial INVITEs. This Section describes the procedures to follow upon reception of a re-INVITE for an
1924 existing dialog.

1925 A UAS that receives a second INVITE before it sent the final response to a first INVITE with a lower
1926 CSeq sequence number on the same dialog MUST return a 500 response to the second INVITE and MUST
1927 include a Retry-After header field with a randomly chosen value of between 0 and 10 seconds. Similarly,
1928 a UAS that receives an INVITE on a dialog while an INVITE it had sent on that dialog is in progress MUST
1929 return a 500 response to the received INVITE and MUST include a Retry-After header field with a randomly
1930 chosen value of between 0 and 10 seconds.

1931 If a user agent receives a re-INVITE for an existing dialog it MUST check any version identifiers in the
1932 session description or, if there are no version identifiers, the content of the session description to see if it has
1933 changed. If the session description has changed, the user agent server MUST adjust the session parameters
1934 accordingly, possibly after asking the user for confirmation.

1935 Versioning of the session description can be used to accommodate the capabilities of new arrivals to a conference,
1936 add or delete media or change from a unicast to a multicast conference.

1937 If a UAS generates a 2xx response and never receives an ACK, it SHOULD generate a re-INVITE itself
1938 with an offer equal to the last session description sent to the peer. The purpose of this is to ensure that both
1939 caller and callee have a consistent view of the session parameters.

1940 A UAS providing an offer in a 2xx (because the INVITE did not contain an offer) MUST offer the same
1941 session description as last provided to the peer, with the exception of being able to change the IP address/port
1942 if so desired.

1943 Under error conditions (e.g., the UAS has crashed and restarted) the session description in the 2xx response for
1944 an empty re-INVITE may be different than the one in use at that moment. If the new session description is not
1945 acceptable for the UAC it SHOULD then send a BYE (after ACKing the 2xx response).

1946 **15 Terminating a Session**

1947 Terminating a session is done either with the BYE request, or the CANCEL request, depending on the state
1948 of the dialog. Either caller or callee can terminate, and may do so for any reason. Sections 13 and 12
1949 document some cases where call termination is normative behavior. As a general rule, if a UA decides that
1950 the session is to be terminated, it MUST follow the procedures here to initiate signaling action to convey that.

1951 Note that both the session and the dialog between both user agents will be terminated.

1952 When a UAC sends an INVITE request to create a session, if a 1xx response with a tag in the To field
1953 is received, an early dialog is created. When a 2xx response is received, the dialog becomes established.
1954 For either state of the dialog, if the UAC desires to terminate the session, the UAC SHOULD follow the
1955 procedures described in Section 15.1.1 to terminate the session. If the callee for a new session wishes to
1956 terminate the dialog, it uses the procedures of Section 15.1.1, but MUST NOT do so until it has receive an
1957 ACK or until the server transaction times out.

1958 This does not mean a user can't hang up right away; it just means that the software in their phone needs to
1959 maintain state for a short while in order to properly clean up.

1960 OPEN ISSUE #202: Is this the right solution.

1961 If the UAC desires to end the session before any type of dialog has been created, it SHOULD send a
1962 CANCEL for the INVITE request that requested establishment of the session that is to be terminated. The
1963 UAC constructs and sends the CANCEL following the procedures described in Section 9. This CANCEL
1964 will normally result in a 487 response to be returned to the INVITE, indicating successful cancellation.
1965 However, it is possible that the CANCEL and a 2xx response to the INVITE "pass on the wire". In this case,
1966 the UAC will receive a 2xx to the INVITE. It SHOULD then terminate the call by following the procedures
1967 described in Section 15.1.1.

1968 **15.1 Terminating a Dialog with a BYE**

1969 **15.1.1 UAC Behavior**

1970 A user agent client uses BYE request, sent within a dialog, to indicate to the server that it wishes to terminate
1971 the session. This will also terminate the dialog. A BYE request MAY be issued by either caller or callee. A
1972 BYE request SHOULD NOT be sent before the creation of a dialog (either early or established). In that case
1973 the UAC SHOULD follow the procedures described in Section 9 instead.

1974 Proxies ensure that a CANCEL request is routed in the same way as the INVITE was. However, a proxy
1975 performing load balancing may route a BYE without a Route header field in a different way than the INVITE, since
1976 both requests have different CSeq sequence numbers.

1977 The To, From, Call-ID, CSeq, and Request-URI of a BYE are set following the same rules as for
1978 regular requests sent within a dialog, described in Section 12.

1979 Once the BYE is constructed, it creates a new non-INVITE client transaction, and passes it the BYE
1980 request. The user agent SHOULD stop sending media as soon as the BYE request is passed to the client
1981 transaction.

1982 15.1.2 UAS Behavior

1983 A UAS core receiving a BYE request checks to see if it matches an existing dialog. If the BYE does
1984 not match an existing dialog, the UAS core SHOULD generate a 481 response and pass that to the server
1985 transaction.

1986 A UAS core receiving a BYE request for an existing dialog MUST follow the procedures of Section
1987 12.2.2 to process the request. Once done, the UAS MUST cease transmitting media streams for the session
1988 being terminated. The UAS core MUST generate a 2xx response to the BYE, and MUST pass that to the
1989 server transaction for transmission.

1990 The UAS MUST still respond to any pending requests received for that dialog, (which can only be an
1991 INVITE). It is RECOMMENDED that a 487 (Request Terminated) response is generated to those pending
1992 requests.

1993 16 Proxy Behavior

1994 16.1 Overview

1995 SIP proxies are elements that route SIP requests to user agent servers and SIP responses to user agent clients.
1996 A request may traverse several proxies on its way to a UAS. Each will make routing decisions, modifying
1997 the request before forwarding it to the next element. Responses will route through the same set of proxies
1998 traversed by the request in the reverse order.

1999 It is important to note that being a proxy is a logical role for a SIP element. When a request arrives, an
2000 element that can play the role of a proxy must first decide if it needs to respond to the request on its own.
2001 For instance, the request could be malformed or the element may need credentials from the client before
2002 acting as a proxy. The element MAY respond with any appropriate error code. When responding directly to
2003 a request, the element is playing the role of a UAS and MUST behave as described in Section 8.2.

2004 A proxy can operate in either a stateful or stateless mode for each new request.

2005 When stateless, a proxy acts as a simple forwarding element. It forwards each request downstream to
2006 a single element determined by making a routing decision based on the request. It simply forwards every
2007 response it receives upstream. A stateless proxy discards information about a message once it has been
2008 forwarded.

2009 On the other hand, a stateful proxy remembers information (specifically, transaction state) about each
2010 incoming request and any requests it sends as a result of processing the incoming request. It uses this
2011 information to affect the processing of future messages associated with that request. A stateful proxy MAY
2012 chose to “fork” a request, routing it to multiple destinations. Any request that is forwarded to more than
2013 one location MUST be handled statefully. Any request processed using TCP (or any other mechanism that is
2014 inherently stateful), MUST be handled statefully.

2015 Much of the processing involved when acting statelessly or statefully for a request is identical. The next
2016 several subsections are written from the point of view of a stateful proxy. The last section calls out those

2017 places where a stateless proxy behaves differently.

2018 **16.2 Stateful Proxy**

2019 When stateful, a proxy is purely a SIP transaction processing engine. Its behavior is modeled here in terms of
 2020 the Server and Client Transactions defined in Section 17. A stateful proxy has a server transaction associated
 2021 with one or more client transactions by a higher layer proxy processing component (see figure 3), known as
 2022 a proxy core. An incoming request is processed by a server transaction. Requests from the server transaction
 2023 are passed to a proxy core. The proxy core determines where to route the request, choosing one or more
 2024 next-hop locations. An outgoing request for each next-hop location is processed by its own associated
 2025 client transaction. The proxy core collects the responses from the client transactions and uses them to send
 2026 responses to the server transaction.

2027 A stateful proxy creates a new server transaction for each new request received. Any retransmissions of
 2028 the request will then be handled by that server transaction per Section 17.

2029 Note that this is a model of proxy behavior, not of software. An implementation is free to take any
 2030 approach that replicates the external behavior this model defines.

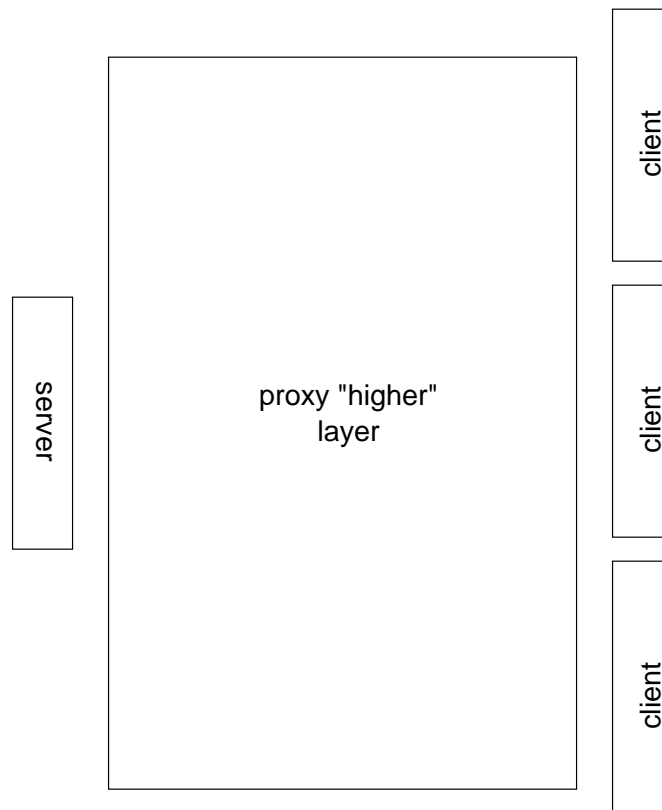


Figure 3: Stateful Proxy Model

2031 For all new requests, including any with unknown methods, an element intending to proxy the request
 2032 MUST:

- 2033 1. Validate the request (Section 16.3)

- 2034 2. Make a routing decision (Section 16.4)
- 2035 3. Forward the request to each chosen destination (Section 16.5)
- 2036 4. Process all responses (Section 16.6)

2037 **16.3 Request Validation**

2038 Before an element can proxy a request, it **MUST** verify the message's validity. A valid message must pass
2039 the following checks:

- 2040 1. Reasonable Syntax
- 2041 2. Max-Forwards
- 2042 3. Loop Detection
- 2043 4. Proxy-Require
- 2044 5. Proxy-Authorization

2045 If any of these checks fail, the element **MUST** behave as a user agent server (see Section 8.2) and respond
2046 with an error code.

2047 1. Reasonable Syntax check

2048 The request **MUST** be well-formed enough to be handled with a server transaction. Any components
2049 involved in the remainder of these Request Validation steps or the Request Processing section **MUST**
2050 be well-formed. Any other components, well-formed or not, **SHOULD** be ignored. For instance, an
2051 element **SHOULD NOT** reject a request because of a malformed **Date** header field.

2052 This protocol is designed to be extended. Future extensions may define new methods and header fields
2053 at any time. An element **MUST NOT** refuse to proxy a request because it contains a method or header
2054 field it does not know about.

2055 2. Max-Forwards check

2056 The **Max-Forwards** header (Section 22.22) is used to limit the number of elements a SIP request can
2057 traverse.

2058 If the request does not contain a **Max-Forwards** header field, this check is passed.

2059 If the request contains a **Max-Forwards** header field with a field value greater than zero, the check is
2060 passed.

2061 If the request contains a **Max-Forwards** header field with a field value of zero (0), the element **MUST**
2062 **NOT** forward the request. If the request was for **OPTIONS**, the element **MAY** act as the final recipient
2063 and respond per Section 11. Otherwise, the element **MUST** return a 483 (Too many hops) response.

2064 3. Loop Detection check

2065 An element **MUST** check for forwarding loops before forwarding a request. If the request contains a
2066 **Via** header field value with A sent-by value that equals a value placed into previous requests by the

2067 proxy, the request has been forwarded by this element before. The request has either looped or is
2068 legitimately spiraling through the element. To determine if the request has looped, the element MUST
2069 perform the **branch** parameter calculation described in Section 3 on this message and compare it to
2070 the parameter received in that **Via** field value. If the parameters match, the request has looped. If
2071 they differ, the request is spiraling, and processing continues. If a loop is detected, the element MUST
2072 return a 482 (Loop Detected) response.

2073 An element MUST NOT forward a request to a multicast group which already appears in any of the
2074 **Via** headers.

2075 4. Proxy-Require check

2076 Future extensions to this protocol may introduce features that require special handling by proxies.
2077 Endpoints will include a **Proxy-Require** header in requests that use these features, telling the proxy
2078 it should not process the request unless the feature is understood.

2079 If the request contains a **Proxy-Require** header (Section 22.28) with one or more option-tags this
2080 element does not understand, the element MUST return a 420 (Bad Extension) response. The response
2081 MUST include an **Unsupported** (Section 22.38) header field listing those option-tags the element did
2082 not understand.

2083 5. Proxy-Authorization check

2084 If an element requires credentials before forwarding a request, the request MUST be inspected as
2085 described in Section 20.2.3. That section also defines what the element must do if the inspection fails.

2086 16.4 Making a Routing Decision

2087 At this point, the proxy must decide where to forward the request. This can be modeled as computing a set
2088 of destinations for the request. This set will either be predetermined by the contents of the request or will
2089 be obtained from an abstract location service. Each destination is represented as a URI and an optional IP
2090 address, port and transport. This combination is referred to as a “next-hop location”.

2091 First, the proxy core checks the received request for **Route** headers. If any **Route** header fields are
2092 present in the request, the element MUST use the URL (including all of its parameters) from the topmost
2093 **Route** header field as only next hop URI in the destination set, with no IP address, port and transport set for
2094 that next hop. The destination set is complete, containing **only** this URL, and the proxy MUST proceed to
2095 the Request Processing of Section 16.5.

2096 The **Route** mechanism is used to control the path a request takes through SIP elements, much like strict
2097 IP source routing. The UAC will insert **Route** header fields (see Section 12), usually based on information
2098 provided by proxies through **Record-Route** header fields (see Section 6).

2099 Assuming there were no **Route** headers in the received request, the proxy checks the **Request-URI** of
2100 the received request. If it has an **maddr** parameter, and that parameter does not indicate an interface the
2101 proxy is listening on, the **Request-URI** MUST be placed into the destination set as the only next hop URI,
2102 with no IP address, port and transport set for that next hop, and the proxy MUST proceed to Section 16.5.
2103 If the **maddr** parameter was present, but did indicate an interface the proxy is listening on, the proxy MUST
2104 strip the **maddr** and continue processing as if no **maddr** were present.

2105 OPEN ISSUE #213: Do we strip just the **maddr**, or the port and transport as well?

2106 OPEN ISSUE #218: Are we really sure this ordering of precedence of Route, maddr, and domain is correct??
2107 It is not yet clear. This needs resolution asap finally, since it affects things like loose source routing, outbound proxy
2108 processing at a UA, and so on.

2109 If the domain of the Request-URI indicates a domain this element is not responsible for, it SHOULD set
2110 the next hop URI to the Request-URI, and leave the IP address, port and transport of the next hop empty.
2111 That next hops MUST be placed into the destination set as the only next hop, and the element MUST proceed
2112 to the task of Request Processing (Section 16.5).

2113 There are many circumstances in which a proxy might receive a request for a domain it is not responsible for.
2114 A firewall proxy handling outgoing calls (the way HTTP proxies handle outgoing requests) is an example of where
2115 this is likely to occur.

2116 If the destination set for the request has not been predetermined as described above, this implies that the
2117 element is responsible for the domain in the Request-URI, and the element MAY use whatever mechanism
2118 it desires to determine where to send the request. Any of these mechanisms can be modeled as accessing
2119 an abstract Location Service. This may consist of obtaining information from a location service created
2120 by a SIP Registrar, reading a database, consulting a presence server, utilizing other protocols, or simply
2121 performing an algorithmic substitution on the Request-URI. The output of these mechanisms is used to
2122 construct the destination set.

2123 Any information in or about the request or the current environment of the element MAY be used in the
2124 construction of the destination set. For instance, different sets may be constructed depending contents or
2125 presence of header fields and bodies, the time of day of the request's arrival, the interface on which the
2126 request arrived, failure of previous requests, or even the element's current level of utilization.

2127 As potential destinations are located through these services, their next hops are added to the destination
2128 set. Next-hop locations may only be placed in the destination set once. If a next-hop location is already
2129 present in the set (based on the definition of equality for the URI type and equality of the optional parame-
2130 ters), it MUST NOT be added again.

2131 A proxy MAY continue to add destinations to the set after beginning Request Processing. It MAY use any
2132 information obtained during that processing to determine new locations. For instance, a proxy may choose
2133 to incorporate contacts obtained in a redirect response (3xx class) into the destination set. If a proxy uses a
2134 dynamic source of information while building the destination set (for instance, if it consults a SIP Registrar),
2135 it SHOULD monitor that source for the duration of processing the request. New locations SHOULD be added
2136 to the destination set as they become available. As above, any given URI MUST NOT be added to the set
2137 more than once.

2138 Allowing a URI to be added to the set only once reduces unnecessary network traffic, and in the case of incor-
2139 porating contacts from redirect requests prevents infinite recursion.

2140 An example trivial location service is achieved by configuring an element with a default outbound des-
2141 tination. All requests are forwarded to this location. The Request-URI of the request is placed in the
2142 destination set with the optional next-hop IP address, port and transport parameters set to the default out-
2143 bound destination. The destination set is complete, containing **only** this URI, and the element proceeds to
2144 the task of Request Processing.

2145 If the Request-URI indicates a resource at this proxy that does not exist, the proxy MUST return a 404
2146 (Not Found) response.

2147 If the destination set remains empty after applying all of the above, the proxy MUST return an error
2148 response, which SHOULD be the 480 (Temporarily Unavailable) response.

2149 16.5 Request Processing

2150 As soon as the destination set is non-empty, a proxy MAY begin forwarding the request. A stateful proxy
2151 MAY process the set in any order. It MAY process multiple destinations serially, allowing each client transac-
2152 tion to complete before starting the next. It MAY start client transactions with every destination in parallel. It
2153 also MAY arbitrarily divide the set into groups, processing the groups serially and processing the destinations
2154 in each group in parallel.

2155 A common ordering mechanism is to use the qvalue parameter of destinations obtained from Contact
2156 header fields (see Section 22.10). Destinations are processed from highest qvalue to lowest. Destinations
2157 with equal qvalues may be processed in parallel.

2158 A stateful proxy must have a mechanism to maintain the destination set as responses are received and
2159 associate the responses to each forwarded request with the original request. For the purposes of this model,
2160 this mechanism is a "response context" created by the proxy layer before forwarding the first request.

2161 For each destination, the proxy forwards the request following these steps:

- 2162 1. Make a copy of the received request
- 2163 2. Update the Request-URI
- 2164 3. Add a Via header field value
- 2165 4. Update the Max-Forwards field if present
- 2166 5. Update the Route header field if present
- 2167 6. Optionally add a Record-route header field value
- 2168 7. Optionally add additional headers
- 2169 8. send the new request

2170 Each of these steps is detailed below:

2171 1. Copy request

2172 The proxy starts with a copy of the received request. The copy MUST initially contain all of the header
2173 fields from the received request. Only those fields detailed in the processing described below may be
2174 removed. The copy SHOULD maintain the ordering of the header fields as in the received request. The
2175 proxy MUST NOT reorder field values with a common field name (See Section 7.3.1).

2176 An actual implementation need not perform a copy; the primary requirement is that the processing of each
2177 next hop begin with the same request.

2178 2. Request-URI

2179 The Request-URI in the copy's start line MUST be replaced with the URI for this destination. If the
2180 URI contains any parameters not allowed in a Request-URI, they MUST be removed.

2181 This is the essence of a proxy's role. This is the mechanism through which a proxy routes a request
2182 toward its destination.

2183 3. Via

2184 The proxy MUST insert a **Via** header field into the copy before the existing **Via** header fields. The **Via**
2185 header maddr, ttl, and sent-by components will be set when the request is processed by the transport
2186 layer (Section 19). The **Via** headers ensure that responses will follow the same set of elements that
2187 the request traversed.

2188 The proxy MUST include a “branch” parameter (Section 22.40) in the **Via** header. When the path of
2189 a request through one or more forking proxies is graphed, the result is a tree. The branch parameter
2190 identifies the “branch” each request was forwarded on. The branch parameter value MUST be unique
2191 for each client transaction to which the request is forwarded. The precise format of the branch. token
2192 is implementation-defined. In order to be able to both detect loops and associate responses with the
2193 corresponding request, the parameter SHOULD consist of two parts separable by the implementation.
2194 The first part is used to detect loops and distinguish loops from spirals. The second is used to match
2195 responses to requests.

2196 Loop detection is performed by verifying that those fields having an impact on the routing decision
2197 have not changed. The value placed in the this part of the branch parameter SHOULD reflect all of
2198 those fields (which include any **Proxy-Require** and **Proxy-Authorization** headers). This is to ensure
2199 that if the request is routed back to the proxy, and one of those fields changes, it is treated as a spiral
2200 and not a loop (Section 3). A common way to create this value is to compute a cryptographic hash
2201 of the **To**, **From**, **Call-ID** header fields, the **Request-URI** of the request received (before translation)
2202 and the sequence number from the **CSeq** header field, in addition to any **Proxy-Require** and **Proxy-**
2203 **Authorization** fields that may be present. The algorithm used to compute the hash is implementation-
2204 dependent, but MD5 [23], expressed in hexadecimal, is a reasonable choice. (Note that base64 is not
2205 permissible for a token.)

2206 In order to correctly match responses to requests (Section 17.1.3), the value SHOULD also contain a
2207 part that is a globally unique function of of the branch on which this request will be forwarded. One
2208 example is a hash of a sequence number, local IP address and **request-URI** of the request

2209 For example: 7a83e5750418bce23d5106b4c06cc632.1

2210 The “branch” parameter MUST depend on all information used for routing decisions, including the incom-
2211 ing **request-URI** and any header values affecting the routing choices. This is necessary to distinguish looped
2212 requests from requests whose routing parameters have changed before returning to this server.

2213 Note that the request method MUST NOT be included in the calculation of the branch parameter.
2214 In particular, **CANCEL** and **ACK** requests MUST have the same branch value as the corresponding
2215 request they cancel or acknowledge. The branch parameter is used in correlating those requests at
2216 server handling them (see Section 17.2.3 and 9.2).

2217 4. Max-Forwards

2218 If the copy contains a **Max-Forwards** header field, the proxy must decrement its value by one (1).

2219 5. Route

2220 If the copy contains a **Route** header field, the proxy must remove the first (topmost) value. Note that
2221 this value was placed in the destination set and then into the **Request-URI** of this copy in previous
2222 steps.

2223 6. Record-Route

2224 If this proxy wishes to request to remain on the path of future requests in this dialog, it MUST insert a
2225 **Record-Route** header value (Section refsec:record-route) into the copy before any existing **Record-**
2226 **Route** header values. See Section 12 for details on whether this request will be honored. Each proxy
2227 in the path of a request makes this request independently the presence of a **Record-Route** header does
2228 not obligate this proxy to add a value.

2229 If the request is honored, the information the proxy places in the **Record-Route** header value will be
2230 used at the endpoints to construct **Route** headers. As shown in the processing steps above, **Route**
2231 headers determine forwarding destinations much like strict IP source routing.

2232 The URL placed in the **Record-Route** header value MUST be a SIP URL. This URL MAY be dif-
2233 ferent for each destination the request is forwarded to. The URL SHOULD NOT contain the transport
2234 parameter unless the proxy has knowledge (such as in a private network) that the next downstream
2235 element that will be in the path of subsequent requests supports that transport.

2236 The URL this proxy provides will be used by some other element to make a routing decision. This proxy, in
2237 general, has no way to know what the capabilities of that element are, so it must restrict itself to the mandatory
2238 elements of a SIP implementation: SIP URLs and UDP transports.

2239 The URL placed in the **Record-Route** header value MUST resolve to this element when the server
2240 location procedures of Section 24 are applied to it. This ensures subsequent requests are routed back
2241 to this element.

2242 The URL placed in the **Record-Route** header value SHOULD be such that if a subsequent request is
2243 received with this URL in the **Request-URI**, the proxy's normal request processing will cause it to be
2244 forwarded to one of the previous elements, including the originating client, traversed by the original
2245 request. This improves robustness, ensuring that the **Request-URI** contains enough information to
2246 forward subsequent requests to a reasonable destination even in the absence of **Route** headers.

2247 The URL placed in the **Record-Route** header value MUST vary with the **Request-URI** in the received
2248 request. A request may legitimately pass through this proxy more than once on the way to its final
2249 destination (this is called a spiraling request). The **Request-URI** will be different each time the
2250 request passes through. If this proxy places the same URL in the **Record-Route** header field each
2251 time, subsequent requests will be rejected as looped requests. It is insufficient to simply copy the
2252 **Request-URI** from each request into the **Record-Route** header. Some modification, such as adding
2253 an **maddr** parameter, is necessary.

2254 URLs satisfying the above paragraphs can be constructed in many ways. One way is to use a URL
2255 that is nearly the same as the **Contact** header in the initial request (if present, else the **From** field),
2256 but with the **maddr** and **port** set to resolve to the proxy, and with a transaction identifier added to the
2257 user part of the request-URI (in order to meet the requirement that the URL in the **Record-Route**
2258 be different for each distinct **Request-URI**). A call stateful proxy could use a URL of the form
2259 sip:proxy.example.com and use information from the stored call state to meet the requirements.

2260 The proxy MAY include **Record-Route** header parameters in the value it provides. These will be
2261 returned in some responses to the request (200 responses to **INVITE** for example) and may be useful
2262 for pushing state into the message.

2263 The **Record-Route** process is designed to work for any SIP request that initiates a dialog. The only
2264 such request in this specification is **INVITE**. Extensions to the protocol MAY define others, and the

2265 mechanisms described here will apply. The request that initiates a dialog and all refreshes (re-INVITE
2266 for example) MUST have **Record-Route** header values added to them if the proxy wishes to remain
2267 in the request path. This means a proxy will often need to record-route requests that contain **Route**
2268 headers. Section 12 describes how this will affect a dialog.

2269 Including **Record-Route** even when **Route** headers already exist in a request improves robustness in the
2270 presence of a preloaded **Route** header field and recovery from endpoint failure.

2271 If a proxy needs to be in the path of any type of dialog (such as one straddling a firewall), it SHOULD
2272 add a **Record-Route** header value to every request with a method it doesn't understand.

2273 Generally, the choice about whether to record-route or not is a tradeoff of features vs. performance.
2274 Faster request processing and higher scalability is achieved when proxies do not record route. How-
2275 ever, provision of certain services may require a proxy to observe all messages in a dialog. It is
2276 RECOMMENDED that proxies do not automatically record route. They should do so only if specifi-
2277 cally required.

2278 7. Adding Additional Headers

2279 The proxy MAY add any other appropriate headers to the copy at this point.

2280 8. Forward Request

2281 A stateful proxy creates a new client transaction for this request as described in Section 17.1. If
2282 the next-hop location used in building this request contains the optional addressing parameters, the
2283 transaction is instructed to send the request based on those parameters. Otherwise, the proxy uses
2284 the procedures of Section 24 to compute an ordered set of addresses from the **Request-URI**, and
2285 as described there, attempts to contact the first one by instructing the client transaction to send the
2286 request there. If this fails, the stateful proxy continues down the list. Each attempt is a new client
2287 transaction, and therefore represents a new branch, so that the processing described above for each
2288 branch would need to be repeated. This results in a requirement to use a different branch ID parameter
2289 for each attempt.

2290 16.6 Response Processing

2291 When a response is received by an element, it first tries to locate a client transaction (Section 17.1.3) match-
2292 ing the response. If none is found, the element MUST process the response (even if it is an informational
2293 response) as a stateless proxy (described below). If a match is found, the response is handed to the client
2294 transaction.

2295 Forwarding responses for which a client transaction (or more generally any knowledge of having sent an asso-
2296 ciated request) is not found improves robustness. In particular, it ensures that "late" 2xx class responses to INVITE
2297 requests are forwarded properly.

2298 As client transactions pass responses to the proxy layer, the following processing MUST take place:

- 2299 1. Find the appropriate response context
- 2300 2. Remove the topmost **Via**
- 2301 3. Add the response to the response context

2302 4. Check to see if this response should be forwarded

2303 The following processing **MUST** be performed on each response that is forwarded. Note that more than
2304 one response to each request will likely be forwarded - each provisional and one final at the least.

2305 1. Aggregate authorization header fields if necessary

2306 2. Forward the response

2307 3. Generate any necessary **CANCEL** requests

2308 If no final response has been forwarded after every client transaction associated with the response context
2309 has been terminated, the proxy must choose and forward the “best” response from those it has seen so far.

2310 Each of the above steps are detailed below:

2311 1. Find Context

2312 The proxy locates the “response context” it created before forwarding the original request using the
2313 key described in Section 16.5. The remaining processing steps take place in this context.

2314 2. Via

2315 The proxy removes the topmost **Via** field value from the response. The address in this value necessar-
2316 ily matches the proxy since the response matched a client transaction above. The branch parameter
2317 from this value can be used to determine which branch the response corresponds to.

2318 If no **Via** field values remain in the response, the response was meant for this element and **MUST**
2319 **NOT** be forwarded. The remainder of the processing described in this section is not performed on this
2320 message. This will happen, for instance, when the element generates **CANCEL** requests as described
2321 in Section sec:proxy-response-processing-cancel.

2322 3. Add response to context

2323 Final responses received are stored in the response context until a final response is generated on
2324 the server transaction associated with this context. The response may a candidate for the best final
2325 response to be returned on that server transaction. Information from this response may be needed in
2326 forming the best response even if this response is not chosen.

2327 If the proxy chooses to recurse on a 3xx class response, it **MUST NOT** add the response to the response
2328 context

2329 4. Check response for forwarding

2330 Until a final response has been sent on the server transaction, the following responses **MUST** be for-
2331 forwarded immediately:

- 2332 • Any provisional response other than 100 Trying
- 2333 • Any 2xx response

2334 If a 6xx response is received, it is not immediately forwarded, but the stateful proxy **SHOULD** cancel
2335 all pending transactions as described in Section 9.

2336 This is a change from RFC2543, which mandated that the 6xx be forwarded immediately. The problem
2337 with this is that it is possible for a 2xx to arrive on another branch, in which case the proxy would have to
2338 forward that in the case of an INVITE transaction. The result is that the UAC could receive a 6xx followed by
2339 a 2xx, which should never be allowed to happen. So, instead, upon receiving a 6xx, a proxy will CANCEL,
2340 which will generally result in 487s to all outstanding client transactions, and then at that point the 6xx is
2341 forwarded upstream.

2342 After a final response has been sent on the server transaction, the following responses MUST be for-
2343 warded immediately:

- 2344 • Any 2xx class response to an INVITE request

2345 A stateful proxy MUST NOT immediately forward any other responses. In particular, a stateful proxy
2346 MUST NOT forward any 100 Trying response. Those responses that are candidates for forwarding later
2347 as the “best” response have been gathered as described in step “Add Response to Context”.

2348 Any response chosen for immediate forwarding MUST be processed as described in steps “Aggregate
2349 authorization headers” through “Record-Route”.

2350 5. Choosing the best response

2351 A stateful proxy MUST send a final response to a response context’s server transaction if no final
2352 responses have been immediately forwarded by the above rules and all client transactions in this
2353 response context have been terminated.

2354 The stateful proxy MUST choose the “best” final response among those received and stored in the
2355 response context.

2356 If there are no final responses in the context, the proxy MUST send a 408 (Request Timeout) response
2357 to the server transaction.

2358 Otherwise, the proxy MUST forward one of the responses from the lowest response class stored in the
2359 response context. The proxy MAY select any response within that lowest class. The proxy SHOULD
2360 give preference to responses that provide information affecting resubmission of this request, such as
2361 401, 407, 415, 420, and 484.

2362 A proxy which receives a 503 response SHOULD NOT forward it upstream unless it can determine that
2363 any subsequent requests it might proxy will also generate a 503. In other words, forwarding a 503
2364 means that the proxy knows it cannot service any requests, not just the one for the Request-URI in
2365 the request which generated the 503.

2366 The forwarded response MUST be processed as described in steps “Aggregate authorization headers”
2367 through “Record-Route”.

2368 For example, if a proxy forwarded a request to 4 locations, and received 503, 407, 501, and 404
2369 responses, it may choose to forward the 407 response.

2370 The tag in the To header field serves to distinguish responses at the UAC. If the forwarded response
2371 did not have one, it MUST NOT be inserted into the response by the proxy.

2372 6. Aggregate authorization headers

2373 If the selected response is a 401 or 407, the proxy MUST collect any WWW-Authenticate and Proxy-
2374 Authenticate header fields from all other 401 and 407 responses received so far in this response
2375 context and add them to this response before forwarding.

2376 This is necessary because any or all of the destinations the request was forwarded to may have re-
2377 requested credentials. The client must receive all of those challenges and supply credentials for each of
2378 them when it retries the request. Motivation for this behavior is provided in Section 20.

2379 7. Record-Route

2380 If the selected response contains a **Record-Route** header field value originally provided by this proxy,
2381 the proxy MAY chose to rewrite the value before forwarding the response. This allows the proxy to
2382 provide different URLs for itself to the next upstream and downstream elements. A proxy may choose
2383 to use this mechanism for any reason. For instance, it is useful for multi-homed hosts.

2384 The new URL provided by the proxy MUST satisfy the same constraints on URLs placed in **Record-**
2385 **Route** header fields in requests (see Section 6) with the following modifications:

2386 The URL SHOULD NOT contain the transport parameter unless the proxy has knowledge that the next
2387 upstream (as opposed to downstream) element that will be in the path of subsequent requests supports
2388 that transport.

2389 The URL placed in the **Record-Route** header value SHOULD be such that if a subsequent request is
2390 received with this URL in the **Request-URI**, the proxy's normal request processing will cause it to
2391 be forwarded to the same next-hop element (as opposed to some previous element) as the originally
2392 forwarded request.

2393 When a proxy does decide to modify the **Record-Route** header in the response, one of the operations
2394 it must perform is to locate the **Record-Route** that it had inserted. If the request spiraled, and the
2395 proxy inserted a **Record-Route** in each iteration of the spiral, locating the correct header in the
2396 response (which must be the proper iteration in the reverse direction) is tricky. Note that the rules
2397 above dictate that a proxy insert a different URI into the **Record-Route** for each distinct **Request-**
2398 **URI** received. The two issues can be solved jointly. A RECOMMENDED mechanism is for the proxy
2399 to append a piece of data to the user portion of the URL. This piece of data is a hash of the transaction
2400 key for the incoming request, concatenated with a unique identifier for the proxy instance. Since the
2401 transaction key includes the **Request-URI**, this key will be unique for each distinct **Request-URI**.
2402 When the response arrives, the proxy modifies the first **Record-Route** whose identifier matches the
2403 proxy instance. The modification results in a URI without this piece of data appended to the user
2404 portion of the URI. Upon the next iteration, the same algorithm (find the topmost **Record-Route**
2405 header with the parameter) will correctly extract the next **Record-Route** header inserted by that
2406 proxy.

2407 8. Forward response

2408 After performing the processing described in steps "Aggregate authorization headers" through "Record-
2409 Route", the proxy may perform any feature specific manipulations on the selected response. Unless
2410 otherwise specified, the proxy MUST NOT remove the message body or any header values other than
2411 the **Via** header value discussed in Section refsec:proxy-response-processing-via. The proxy MUST
2412 pass the response to the server transaction associated with the response context. This will result in
2413 the response being sent to the location now indicated in the topmost **Via** field value. If the server
2414 transaction is no longer available to handle the transmission, the element MUST forward the response
2415 statelessly by sending it to the server transport.

2416 Even after forwarding a final response, the proxy MUST maintain the response context until all of its
2417 associated transactions have been terminated.

2418 9. Generate CANCELs

2419 OPEN ISSUE #7: If CANCEL is restricted to INVITE only, this behavior must restrict itself to
2420 INVITE requests.

2421 OPEN ISSUE #122: The MUST below reflects list discussion, but the question of how strong this
2422 requirement should be was not formally closed.

2423 If the forwarded response was a final response, the proxy MUST generate a CANCEL request for all
2424 pending client transactions associated with this response context. A proxy SHOULD also generate a
2425 CANCEL request for all pending client transactions associated with this response context when it
2426 receives a 6xx response. A pending client transaction is one that has received a provisional response,
2427 but no final response and has not had an associated CANCEL generated for it. Generating CANCEL
2428 requests is described in Section 9.1.

2429 16.7 Handling transport errors

2430 If the transport layer notifies a proxy of an error when it tries to forward a request (see Section 19.4), the
2431 proxy MUST behave as if the forwarded request received a 400 response.

2432 If the proxy is notified of an error when forwarding a response, it drops the response. The proxy SHOULD
2433 NOT cancel any outstanding client transactions associated with this response context due to this notification.

2434 If a proxy cancels its outstanding client transactions, a single malicious or misbehaving client can cause all
2435 transactions to fail through its Via header field.

2436 16.8 CANCEL Processing

2437 A stateful proxy may generate a CANCEL to any other request it has generated at any time. For instance,
2438 it may choose to generate CANCELs based on having a transaction exceed the time specified in the Ex-
2439 pire header of certain requests, or as a result of any logic it applies while forwarding requests. A proxy
2440 MUST cancel any pending client transactions associated with a response context when it receives a matching
2441 CANCEL request.

2442 OPEN ISSUE #185: Should generating CANCEL at a proxy based on Expires in INVITE be deprecated?

2443 While a CANCEL request is handled in a stateful proxy by its own server transaction, a new response
2444 context is not created for it. Instead, the proxy layer searches its existing response contexts for the server
2445 transaction handling the request associated with this CANCEL. If a matching response context is found, the
2446 element MUST immediately return a 200 OK response to the CANCEL request. In this case, the element is
2447 acting as a user agent server as defined in Section 8.2. Furthermore, the element MUST generate CANCEL
2448 requests for all pending client transactions in the context as described in Section 9.

2449 If a response context is not found, the element does not have any knowledge of the request to apply
2450 the CANCEL to. It MUST forward the CANCEL request statelessly (it may have statelessly forwarded the
2451 associated request previously).

2452 16.9 Stateless proxy

2453 When acting statelessly, a proxy is a simple message forwarder. Much of the processing performed when
2454 acting statelessly is the same as when behaving statefully. The differences are detailed here.

2455 A stateless proxy does not have any notion of a transaction, or of the response context used to describe
2456 stateful proxy behavior. Instead, the stateless proxy takes messages, both requests and responses, directly
2457 from the transport layer (See section 19). As a result, stateless proxies do not retransmit messages on their
2458 own. They do, however, forward all retransmission they receive (they do not have the ability to distinguish
2459 a retransmission from the original message). Furthermore, when handling a request statelessly, an element
2460 MUST NOT generate its own 100 Trying (or any other provisional) response.

2461 A stateless proxy must validate a request as described in Section 16.3

2462 A stateless proxy must make a routing decision as described in Section 16.4 with the following excep-
2463 tion:

- 2464 • A stateless proxy MUST choose one and only one destination from the destination set. This choice
2465 MUST only rely on fields in the message and time-invariant properties of the server. In particular, a
2466 retransmitted request MUST be forwarded to the same destination each time it is processed. Further-
2467 more, CANCEL and non-Routed ACK requests MUST generate the same choice as their associated
2468 INVITE.

2469 A stateless proxy must process the request before forwarding as described in Section 16.5 with the
2470 following exceptions:

- 2471 • The **branch** parameter on the inserted **Via** header field MUST be the same each time a retransmitted
2472 request is forwarded. Thus for a stateless proxy, the **branch** parameter calculation MUST **only** depend
2473 on message parameters affecting the routing of the request which are invariant on retransmission.
- 2474 • The request is sent directly to the transport layer instead of through a client transaction. If the next-
2475 hop destination parameters don't provide an explicit destination, the element applies the procedures
2476 of Section 24 to the Request-URI to determine where to send the request.

2477 Stateless proxies MUST NOT perform special processing for CANCEL requests. They are processed by
2478 the above rules as any other requests.

2479 Response processing as described in Section 16.6 does not apply to a proxy behaving statelessly. When
2480 a response arrives at a stateless proxy, the proxy inspects the address in the first (topmost) **Via** header value.
2481 If that address matches the proxy, the proxy MUST remove that value from the response and forward the
2482 result to the location indicated in the next **Via** header value. Unless specified otherwise, the proxy MUST
2483 NOT remove any other header values or the message body. If the address does not match the proxy, the
2484 message MUST be silently discarded.

2485 17 Transactions

2486 SIP is fundamentally a transactional protocol. This means that interactions between components take place
2487 in a series of independent message exchanges. Specifically, a SIP transaction consists of a single request,
2488 and any responses to that request (which include zero or more provisional responses and one or more final
2489 responses). In the case of a transaction where the request was an INVITE (known as an INVITE transaction),
2490 the transaction also includes the ACK only if the final response was not a 2xx response. If the response was
2491 a 2xx, the ACK is not considered part of the transaction.

2492 The reason for this separation is rooted in the importance of delivering all 200 OK responses to an INVITE to
2493 the UAC. To deliver them all to the UAC, the UAS alone takes responsibility for retransmitting them, and the UAC
2494 alone takes responsibility for acknowledging them with ACK. Since this ACK is retransmitted only by the UAC, it
2495 is effectively considered its own transaction.

2496 Transactions have a client side and a server side. The client side is known as a client transaction, and the
 2497 server side, as a server transaction. The client transaction sends the request, and the server transaction sends
 2498 the response. The client and server transactions are logical functions that are embedded in any number of
 2499 elements. Specifically, they exist within user agents and stateful proxy servers. Consider the example of
 2500 Section 4. In this example, the UAC executes the client transaction, and its outbound proxy executes the
 2501 server transaction. The outbound proxy also executes a client transaction, which sends the request to a
 2502 server transaction in the inbound proxy. That proxy also executes a client transaction, which in turn, sends
 2503 the request to a server transaction in the UAS. This is shown pictorially in Figure 4.

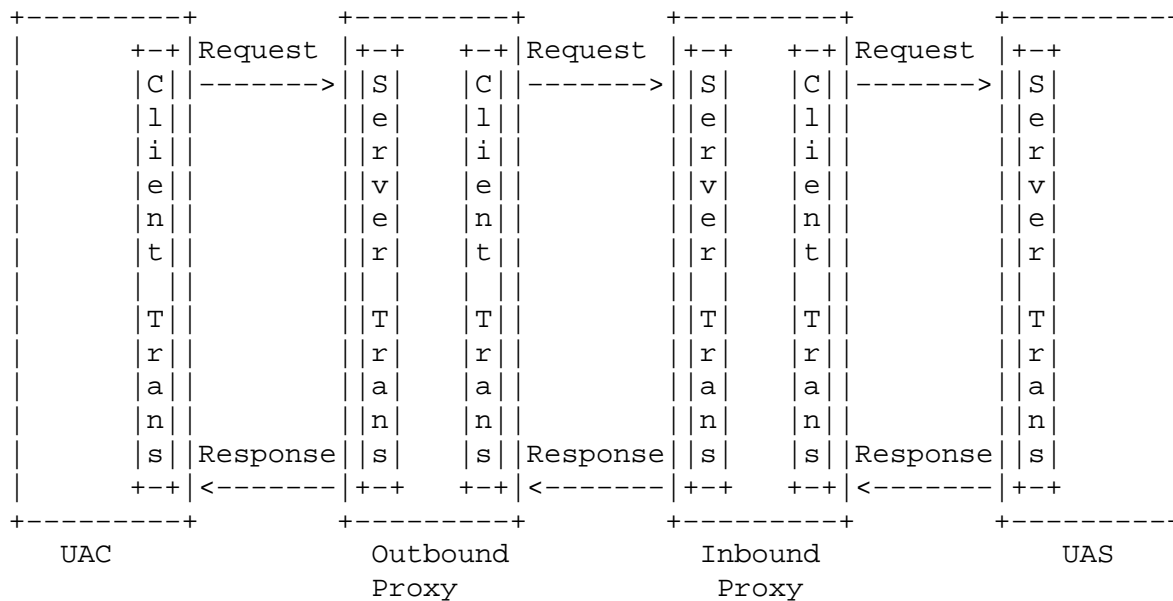


Figure 4: Transaction relationships

2504 A stateless proxy does not contain a client or server transaction. The transaction exists between the
 2505 UA or stateful proxy on one side of the stateless proxy, and the UA or stateful proxy on the other side.
 2506 As far as SIP transactions are concerned, stateless proxies are effectively transparent. The purpose of the
 2507 client transaction is to receive a request from the element the client is embedded in (call this element the
 2508 “Transaction User” or TU; it can be a UA or a stateful proxy), and reliably deliver the request to that server
 2509 transaction. The client transaction is also responsible for receiving responses, and delivering them to the
 2510 TU, filtering out any retransmissions or disallowed responses (such as a response to ACK). In the case of
 2511 an INVITE transaction, that includes generation of the ACK request for any final response excepting a 2xx
 2512 response.

2513 Similarly, the purpose of the server transaction is to receive requests from the transport layer, and deliver
 2514 them to the TU. The server transaction filters any request retransmissions from the network. The server
 2515 transaction accepts responses from the TU, and delivers them to the transport layer for transmission over the
 2516 network. In the case of an INVITE transaction, it absorbs the ACK request for any final response excepting
 2517 a 2xx response.

2518 The 2xx response, and the ACK for it, have special treatment. This response is retransmitted only by a

2519 UAS, and its ACK generated only by the UAC. This end-to-end treatment is needed so that a caller knows
2520 the entire set of users that have accepted the call. Because of this special handling, retransmissions of the
2521 2xx response are handled by the UA core, not the transaction layer. Similarly, generation of the ACK for the
2522 2xx is handled by the UA core. Each proxy along the path merely forwards each 2xx response to INVITE,
2523 and its corresponding ACK.

2524 17.1 Client transaction

2525 The client transaction provides its functionality through the maintenance of a state machine.

2526 The TU communicates with the client transaction through a simple interface. When the TU wishes to
2527 initiate a new transaction, it creates a client transaction, and passes it the SIP request to send, a value for
2528 timer C (described below), and an IP address, port, and transport to send it to. The client transaction begins
2529 execution of its state machine. Valid responses are passed up to the TU from the client transaction.

2530 There are two types of client transaction state machines, depending on the method the request passed
2531 by the TU. One handles client transactions for INVITE request. This type of machine is referred to as an
2532 INVITE client transaction. Another type handles client transactions for all requests except INVITE and
2533 ACK. This is referred to as a non-INVITE client transaction. There is no client transaction for ACK. If the
2534 TU wishes to send an ACK, it passes one directly to the transport layer for transmission.

2535 The INVITE transaction is different from those of other methods because of its extended duration. Nor-
2536 mally, human input is required in order to respond to an INVITE. The long delays expected for sending a
2537 response argue for a three way handshake. Requests of other methods, on the other hand, are expected to
2538 completely rapidly. In fact, because of its reliance on just a two way handshake, TUs SHOULD respond
2539 immediately to non-INVITE requests. Protocol extensions which require longer durations for generation of
2540 a response (such as a new method that does require human interaction) SHOULD instead use two transactions
2541 - one to send the request, and another in the reverse direction to convey the result of the request.

2542 17.1.1 INVITE Client Transaction

2543 **17.1.1.1 Overview of INVITE Transaction** The INVITE transaction consists of a three-way handshake.
2544 The client transaction sends an INVITE, the server transaction sends responses, and the client transaction
2545 sends an ACK. For unreliable transports (such as UDP), the client transaction will retransmit requests at an
2546 interval that starts at T1 seconds and doubles after every retransmission. The request is not retransmitted over
2547 reliable transports. After receiving a 1xx response, any retransmissions cease altogether, and the client waits
2548 for further responses. The server transaction can send additional 1xx responses, which are not transmitted
2549 reliably. Eventually, the server transaction decides to send a final response. For unreliable transports, that
2550 response is retransmitted periodically, and for reliable transports, its sent once. For each final response that
2551 is received at the client transaction, the client transaction sends an ACK, the purpose of which is to quench
2552 retransmissions of the response.

2553 **17.1.1.2 Formal Description** The state machine for the INVITE client transaction is shown in Figure 5.
2554 The initial state, "calling", MUST be entered when the TU initiates a new client transaction with an INVITE
2555 request. The client transaction MUST pass the request to the transport layer for transmission (see Section
2556 19). If an unreliable transport is being used, the client transaction SHOULD start timer A with a value
2557 of T1, and SHOULD NOT start timer A when a reliable transport is being used (Timer A controls request
2558 retransmissions). For any transport, the client transaction MUST start timer B with a value of 64*T1 seconds

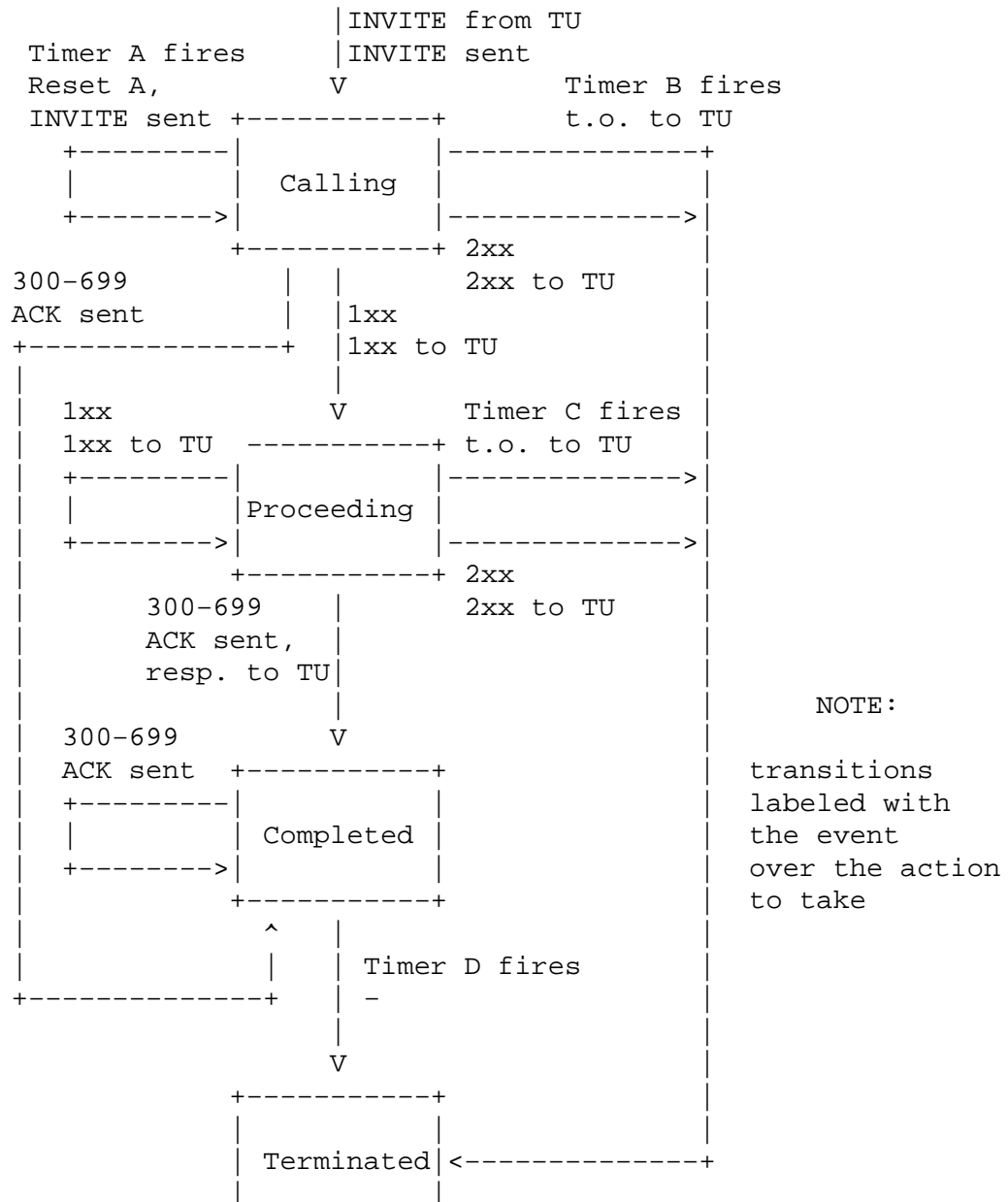


Figure 5: INVITE client transaction

2559 (Timer B controls transaction timeouts).

2560 When timer A fires, the client transaction SHOULD retransmit the request by passing it to the transport
 2561 layer, and SHOULD reset the timer with a value of 2*T1. When the timer fires 2*T1 seconds later, the
 2562 request SHOULD be retransmitted again (assuming the client transaction is still in this state). This process
 2563 SHOULD continue, so that the request is retransmitted with intervals that double after each transmission.
 2564 These retransmissions SHOULD only be done while the client transaction is in the “calling” state.

2565 The default value for T1 is 500ms. T1 is an estimate of the RTT between the client and server transac-

2566 tions. The optional RTT estimation procedure of Section 17.3 MAY be followed, in which case the resulting
2567 estimate MAY be used instead of 500ms. If no RTT estimation is used, other values MAY be used in private
2568 networks where it is known that RTT has a different value. On the public Internet, T1 MAY be chosen larger,
2569 but SHOULD NOT be smaller.

2570 If the client transaction is still in the "calling" when timer B fires, the client transaction SHOULD inform
2571 the TU that a timeout has occurred. The client transaction MUST NOT generate an ACK. The value of $64 * T1$
2572 is equal to the amount of time required to send seven requests in the case of an unreliable transport.

2573 If the client transaction receives a provisional response while in the "calling" state, it transitions to
2574 the "proceeding" state. Upon entering this state, the client transaction MUST start timer C with the value
2575 provided by the TU when the client transaction was created. This timeout dictates how long the client
2576 transaction waits for a final response before giving up (i.e., roughly how long does it "let the phone ring"). In
2577 the "proceeding" state, the client transaction SHOULD NOT retransmit the request any longer. Furthermore,
2578 the provisional response MUST be passed to the TU. Any further provisional responses MUST be passed up
2579 to the TU while in the "proceeding" state. When timer C fires, the client transaction MUST transition to the
2580 terminated state, and it MUST inform the TU of the timeout.

2581 When in either the "calling" or "proceeding" states, reception of a response with status code from 300-
2582 699 MUST cause the client transaction to transition to "completed". The client transaction MUST pass the
2583 received response up to the TU, and it MUST generate an ACK request, even if the transport is reliable
2584 (guidelines for constructing the ACK from the response are given in Section 17.1.1.3) and then pass the ACK
2585 to the transport layer for transmission. The ACK MUST be sent to the same address, port and transport that
2586 the original request was sent to. The client transaction SHOULD start timer D when it enters the "completed"
2587 state, with a value of T3 seconds for unreliable transports, and zero seconds for reliable transports. T3 is
2588 the total amount of time that the server transaction can remain in the "completed" state when unreliable
2589 transports are used. For the default values of the timers below, this is 16 seconds.

2590 OPEN ISSUE #210: Timer D should be based on the values of the timers selected at the server, but these values
2591 aren't known by the client. We could alternatively specify an absolute minimum.

2592 Any retransmissions of the final response that are received while in the "completed" state SHOULD cause
2593 the ACK to be re-passed to the transport layer for retransmission, but the newly received response MUST
2594 NOT be passed up to the TU. A retransmission of the response is defined as any response which would match
2595 the same client transaction, based on the rules of Section 17.1.3.

2596 If timer D fires while the client transaction is in the "completed" state, the client transaction MUST move
2597 to the terminated state, and it MUST inform the TU of the timeout.

2598 When in either the "calling" or "proceeding" states, reception of a 2xx response MUST cause the client
2599 transaction to enter the terminated state, and the response MUST be passed up to the TU. The handling of
2600 this response depends on whether the TU is a proxy core or a UAC core. A UAC core will handle generation
2601 of the ACK for this response, while a proxy core will always forward the 200 OK upstream. The differing
2602 treatment of 200 OK between proxy and UAC is the reason that handling of it does not take place in the
2603 transaction layer.

2604 The client transaction MUST be destroyed the instant it enters the terminated state. This is actually nec-
2605 essary to guarantee correct operation. The reason is that 2xx responses to an INVITE are treated differently;
2606 each one is forwarded by proxies, and the ACK handling in a UAC is different. Thus, each 2xx needs to be
2607 passed to a proxy core (so that it can be forwarded) and to a UAC core (so it can be acknowledged). No
2608 transaction layer processing takes place. Whenever a response is received by the transport, if the transport
2609 layer finds no matching client transaction (using the rules of Section 17.1.3, the response is passed directly
2610 to the core. Since the matching client transaction is destroyed by the first 2xx, subsequent 2xx will find no

2611 match and therefore be passed to the core.

2612 **17.1.1.3 Construction of the ACK Request** The ACK request constructed by the client transaction
2613 MUST contain values for the Call-ID, From, and Request-URI which are equal to the values of those
2614 headers in the request that created the client transaction (call this the “original request”). The To field in the
2615 ACK MUST equal the To field in the response being acknowledged, and will therefore usually differ from
2616 the To field in the original request by the addition of the tag parameter. The ACK MUST contain a single Via
2617 header, and this MUST be equal to the top Via header of the original request. The ACK request MUST NOT
2618 contain any Route headers. The CSeq header in the ACK MUST contain the same value for the sequence
2619 number as was present in the original request, but the method parameter MUST be equal to “ACK”.

2620 These rules for construction of ACK only apply to the client transaction. A UAC core which generates
2621 an ACK for 2xx MUST instead follow the rules described in Section 13.

2622 For example, consider the following request:

```
2623 INVITE sip:bob@biloxi.com SIP/2.0
2624 Via: SIP/2.0/UDP 10.1.3.3
2625 To: Bob <sip:bob@biloxi.com>
2626 From: Alice <sip:alice@atlanta.com>;tag=88sja8x
2627 Call-ID: 987asjd97y7atg@10.1.3.3
2628 CSeq: 986759 INVITE
```

2629 The ACK request for a non-2xx final response to this request would look like:

```
2630 ACK sip:bob@biloxi.com SIP/2.0
2631 Via: SIP/2.0/UDP 10.1.3.3
2632 To: Bob <sip:bob@biloxi.com>;tag=99sa0xk
2633 From: Alice <sip:alice@atlanta.com>;tag=88sja8x
2634 Call-ID: 987asjd97y7atg@10.1.3.3
2635 CSeq: 986759 ACK
```

2636 17.1.2 non-INVITE Client Transaction

2637 **17.1.2.1 Overview of the non-INVITE Transaction** non-INVITE transactions do not make use of ACK.
2638 They are a simple request-response interaction. For unreliable transports, requests are retransmitted at an
2639 interval which starts at T1, and doubles until it hits T2. If a provisional response is received, retransmis-
2640 sions continue for unreliable transports, but at an interval of T2. The server transaction retransmits the last
2641 response it sent (which can be a provisional or final response) only when a retransmission of the request is
2642 received. This is why request retransmissions need to continue even after a provisional response, they are
2643 what ensure reliable delivery of the final response.

2644 Unlike an INVITE transaction, a non-INVITE transaction has no special handling for the 2xx response.
2645 The result is that only a single 2xx response to a non-INVITE is ever delivered to a UAC.

2646 **17.1.2.2 Formal Description** The state machine for the non-INVITE client transaction is shown in Fig-
2647 ure 6. It is very similar to the state machine for INVITE.

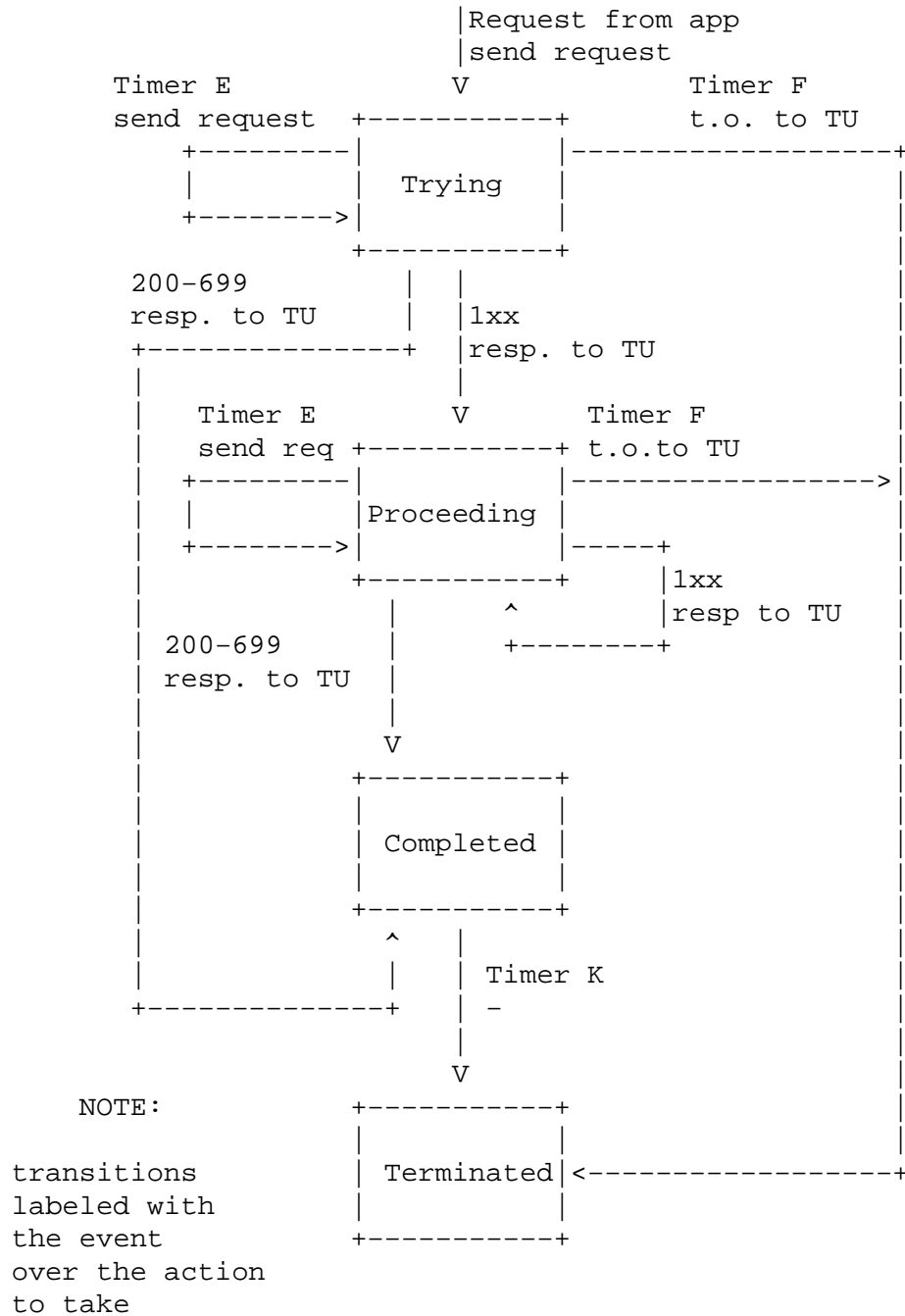


Figure 6: non-INVITE client transaction

2648 The “Trying” state is entered when the TU initiates a new client transaction with a request. When
 2649 entering this state, the client transaction SHOULD set Timer F to fire in T3 seconds. The request MUST be
 2650 passed to the transport layer for transmission. If an unreliable transport is in use, the client transaction MUST
 2651 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

2652 value of $\text{MIN}(2*T1, T2)$. When the timer fires again, it is reset to a $\text{MIN}(4*T1, T2)$. This process continues,
2653 so that retransmissions occur with an exponentially increasing interval that caps at $T2$. The default value
2654 of $T2$ is 4s, and it represents the amount of time a non-INVITE server transaction will take to respond to a
2655 request, if it does not respond immediately. For the default values of $T1$ and $T2$, this results in intervals of
2656 500 ms, 1 s, 2 s, 4 s, 4 s, 4s, etc.

2657 If Timer F fires while the client transaction is still in the "Trying" state, the client transaction SHOULD
2658 inform the TU about the timeout, and then it SHOULD enter the "Terminated" state. If a provisional response
2659 is received while in the "Trying" state, the response MUST be passed to the TU, and then the client transaction
2660 SHOULD move to the "Proceeding" state. If a final response (status codes 200-699) is received while in the
2661 "Trying" state, the response MUST be passed to the TU, and the client transaction MUST transition to the
2662 "Completed" state.

2663 If Timer E fires while in the "Proceeding" state, the request MUST be passed to the transport layer
2664 for retransmission, and Timer E MUST be reset with a value of $T2$ seconds. If timer F fires while in the
2665 "Proceeding" state, the TU MUST be informed of a timeout, and the client transaction MUST transition to the
2666 terminated state. If a final response (status codes 200-699) is received while in the "Proceeding" state, the
2667 response MUST be passed to the TU, and the client transaction MUST transition to the "Completed" state.

2668 Once the client transaction enters the "Completed" state, it MUST set Timer K to fire in $T4$ seconds for
2669 unreliable transports, and zero seconds for reliable transports. The "Completed" state exists to buffer any
2670 additional response retransmissions that may be received (which is why the client transaction remains there
2671 only for unreliable transports). $T4$ represents the amount of time the network will take to clear messages
2672 between client and server transactions. The default value of $T4$ is 5s. A response is a retransmission when it
2673 matches the same transaction, using the rules specified in Section 17.1.3. If Timer K fires while in this state,
2674 the client transaction MUST transition to the "Terminated" state.

2675 OPEN ISSUE #211: This special treatment for reliable transports, where the state machine transactions directly
2676 to terminated, is new.

2677 Once the transaction is in the terminated state, it MUST be destroyed. As with client transactions, this is
2678 needed to ensure reliability of the 2xx responses to INVITE.

2679 **17.1.3 Matching Responses to Client Transactions**

2680 When the transport layer in the client receives a response, it has to figure out which client transaction will
2681 handle the response, so that the processing of Sections 17.1.1 and 17.1.2 can take place.

2682 A response matches a client transaction through a comparison process with fields in the request that
2683 created the transaction. Specifically, the From, Call-ID, CSeq, and the topmost Via header MUST match
2684 the same fields in the request, using the matching operations for those headers defined in Section 22. If
2685 the To field in the request had a tag, the To field in the response MUST match the To field in the request,
2686 as described in Section 22.37. However, if the To field in the request did not contain a tag, the To field in
2687 the response MUST match that in the request, except that the tag MUST NOT be considered as part of the
2688 matching process. This is needed since a UAS will add a tag to the To field of the response.

2689 **17.1.4 Handling Transport Errors**

2690 When the client transaction sends a request to the transport layer to be sent, the following procedures are
2691 followed if the transport layer indicates a failure.

2692 The client transaction SHOULD inform the TU that a transport failure has occurred, and the client trans-
2693 action SHOULD transition directly to the terminated state.

2694 17.2 Server Transaction

2695 The server transaction is responsible for the delivery of requests to the TU, and the reliable transmission of
2696 responses. It accomplishes this through a state machine. Server transactions are created by the core when a
2697 request is received, and transaction handling is desired for that request (this won't always be the case).

2698 As with the client transactions, the state machine depends on whether the received request is an INVITE
2699 request or not.

2700 17.2.1 INVITE Server Transaction

2701 The state diagram for the INVITE server transaction is shown in Figure 7.

2702 When a server transaction is constructed with a request, it enters the "Proceeding" state. The server
2703 transaction MUST generate a 100 response (not any status code - the specific value of 100) unless it knows
2704 that the TU will generate a provisional or final response within 200 ms, in which case it MAY generate a 100
2705 response. This provisional response is needed to rapidly quench request retransmissions in order to avoid
2706 network congestion. The request MUST be passed to the TU.

2707 The TU passes any number of provisional responses to the server transaction. So long as the server
2708 transaction is in the "Proceeding" state, each of these MUST be passed to the transport layer for transmis-
2709 sion. They are not sent reliably (they are not retransmitted), and do not cause a change in the state of the
2710 server transaction. If a request retransmission is received while in the "Proceeding" state, the most recent
2711 provisional response that was received from the TU MUST be passed to the transport layer for retransmis-
2712 sion. A request is a retransmission if it matches the same server transaction based on the rules of Section
2713 17.2.3.

2714 If, while in the "proceeding" state, the TU passes a 2xx Response to the server transaction, the server
2715 transaction MUST pass this response to the transport layer for transmission. It is not retransmitted by the
2716 server transaction; retransmissions of 2xx responses are handled by the TU. The server transaction MUST
2717 then transition to the "terminated" state.

2718 While in the "Proceeding" state, if the TU passes a response with status code from 300 to 699 to the
2719 server transaction, the response MUST be passed to the transport layer for transmission, and the state machine
2720 MUST enter the "Completed" state. For unreliable transports, timer G is set to fire in T1 seconds, and is not
2721 set to fire for reliable transports.

2722 This is a change from RFC2543, where responses were always retransmitted, even over reliable transports.

2723 When the "Completed" state is entered, timer H MUST be set to fire in $64 * T1$ seconds, for all transports.
2724 Timer H determines when the server transaction gives up retransmitting the response. Its value is chosen to
2725 equal Timer B, the amount of time a client transaction will continue to retry sending a request. If timer G
2726 fires, the response is passed to the transport layer once more for retransmission, and timer G is set to fire in
2727 $\text{MIN}(2 * T1, T2)$ seconds. From then on, when timer G fires, the response is passed to the transport again for
2728 transmission, and timer G is reset with a value that doubles, unless that value exceeds T2, in which case it
2729 is reset with the value of T2. This is identical to the retransmit behavior for requests in the "Trying" state of
2730 the non- INVITE client transaction. Furthermore, while in the "completed" state, if a request retransmission
2731 is received, the server SHOULD pass the response to the transport for retransmission.

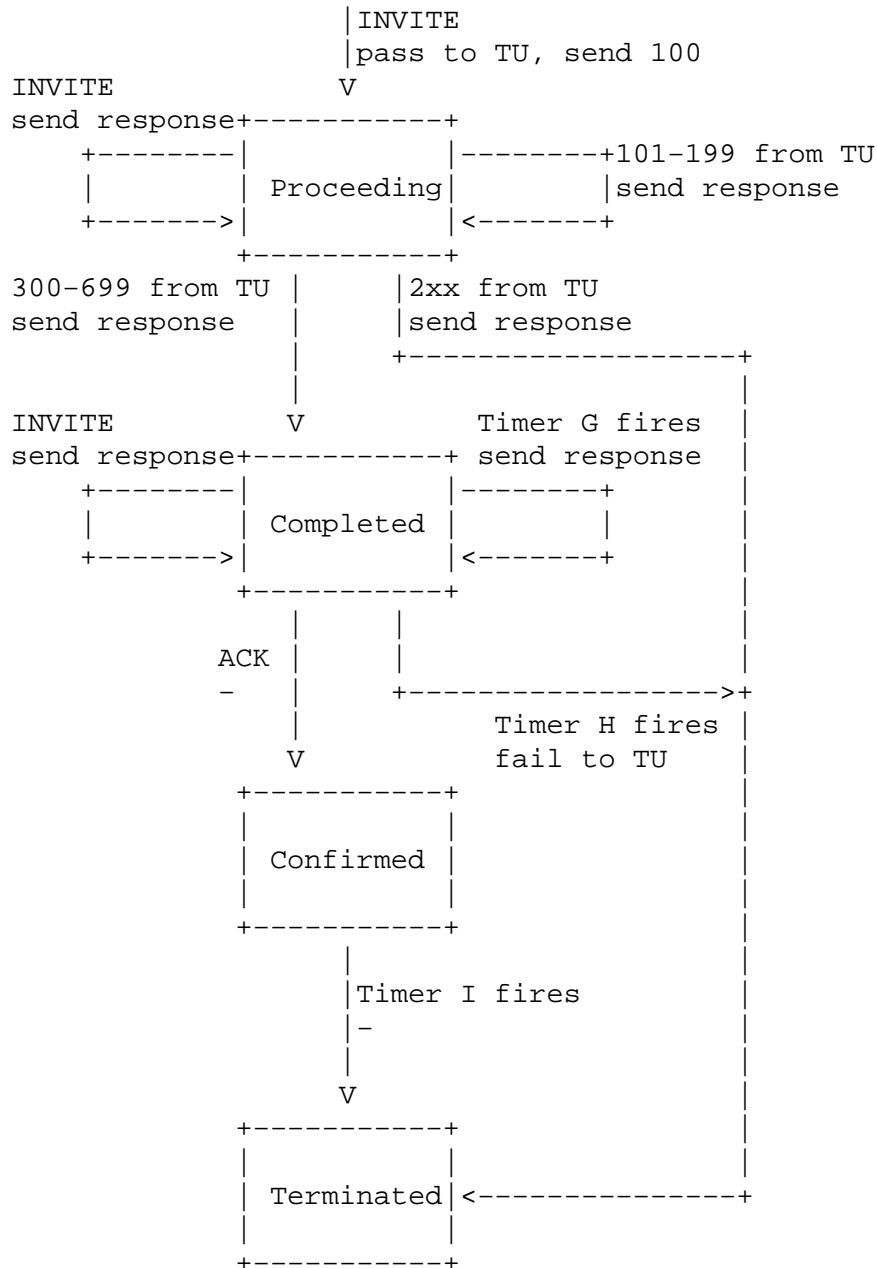


Figure 7: INVITE server transaction

2732 If an **ACK** is received while the server transaction is in the “Completed” state, the server transaction
 2733 MUST transition to the “confirmed” state. As Timer G is ignored in this state, any retransmissions of the
 2734 response will cease.

2735 If timer H fires while in the “Completed” state, it implies that the **ACK** was never received. In this case,
 2736 the server transaction MUST transition to the terminated state, and MUST indicate to the TU that a transaction
 2737 failure has occurred.

2738 The purpose of the “confirmed” state is to absorb any additional ACK messages that arrive, triggered
2739 from retransmissions of the final response. When this state is entered, timer I is set to fire in T4 seconds for
2740 unreliable transports, and zero seconds for reliable transports. Once timer I fires, the server MUST transition
2741 to the “Terminated” state.

2742 Once the transaction is in the terminated state, it MUST be destroyed. As with client transactions, this is
2743 needed to ensure reliability of the 2xx responses to INVITE.

2744 17.2.2 non-INVITE Server Transaction

2745 The state machine for the non-INVITE server transaction is shown in Figure 8.

2746 The state machine is initialized in the “Trying” state, and is passed a request other than INVITE or
2747 ACK when initialized. This request is passed up to the TU. Once in the “Trying” state, any further request
2748 retransmissions are discarded. A request is a retransmission if it matches the same server transaction, using
2749 the rules specified in Section 17.2.3.

2750 While in the “Trying” state, if the TU passes a provisional response to the server transaction, the server
2751 transaction MUST enter the “Proceeding” state. The response MUST be passed to the transport layer for
2752 transmission. Any further provisional responses that are received from the TU while in the “Proceeding”
2753 state MUST be passed to the transport layer for transmission. If a retransmission of the request is received
2754 while in the “Proceeding” state, the most recently sent provisional response MUST be passed to the transport
2755 layer for retransmission. If the TU passes a final response (status codes 200-699) to the server while in the
2756 “Proceeding” state, the transaction MUST enter the “Completed” state, and the response MUST be passed to
2757 the transport layer for transmission.

2758 When the server transaction enters the “Completed” state, it MUST set Timer J to fire in T3 seconds for
2759 unreliable transports, and zero seconds for reliable transports. While in the “Completed” state, the server
2760 transaction MUST pass the final response to the transport layer for retransmission whenever a retransmission
2761 of the request is received. Any other final responses passed by the TU to the server transaction MUST be
2762 discarded while in the “Completed” state. The server transaction remains in this state until Timer J fires, at
2763 which point it MUST transition to the “Terminated” state.

2764 The server transaction MUST be destroyed the instant it enters the “Terminated” state.

2765 17.2.3 Matching Requests to Server Transactions

2766 When an INVITE or ACK request is received from the network by the server, it has to be matched to an
2767 existing INVITE transaction. The INVITE request matches a transaction if the Request-URI, To, From,
2768 Call-ID, CSeq, and top Via header match those of the INVITE request which created the transaction. The
2769 ACK request matches a transaction if the Request-URI, From, Call-ID, CSeq method (not the number),
2770 and top Via header match those of the INVITE request which created the transaction, and the To field of
2771 the ACK matches the To field of the response sent by the server transaction (which then includes the tag).
2772 Matching is done based on the matching rules defined for each of those headers. The usage of the tag in
2773 the To field helps disambiguate ACK for 2xx from ACK for other responses at a proxy which may have
2774 forwarded both responses (which can occur in unusual conditions).

2775 For all other request methods, a request is matched to a transaction if the Request-URI, To, From,
2776 Call-ID and Cseq (including the method) and top Via header match those of the request which created the
2777 transaction. Matching is done based on the matching rules defined for each of those headers.

2778 Because the matching rules include the Request-URI, the server cannot match a response to a transac-
2779 tion. When the TU passes a response to the server, it must inform the TU which transaction the response is

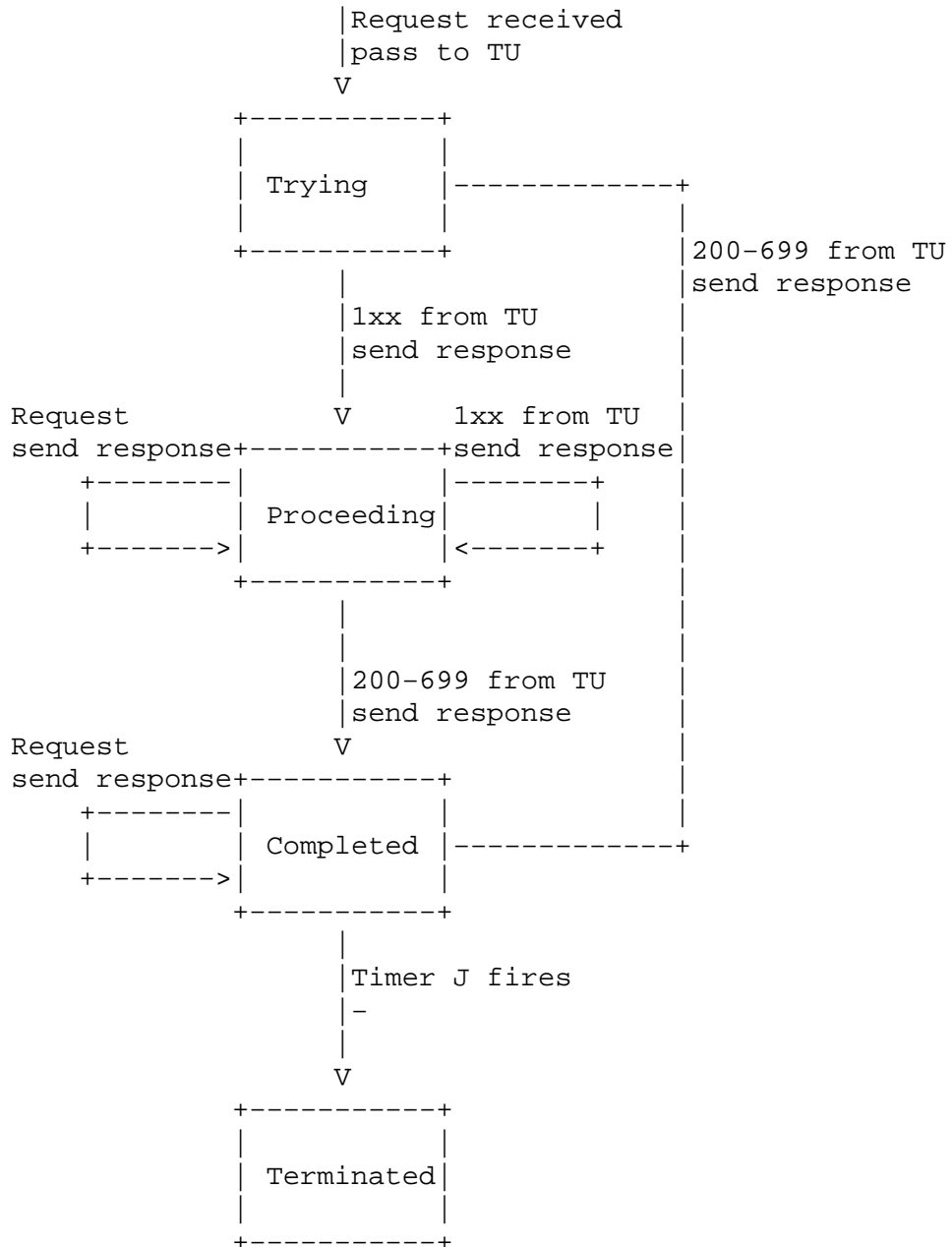


Figure 8: non-INVITE server transaction

2780 for.

2781 **17.3 RTT Estimation**

2782 Most of the timeouts used in the transaction state machines derive from T1, which is an estimate of the RTT
 2783 between the client and server transactions. This subsection defines optional procedures that a client can use

2784 to build up estimates of the RTT to a particular IP address. To perform this procedure, the client MUST
2785 maintain a table of variables for each destination IP address to which an RTT estimate is being made.

2786 OPEN ISSUE #212: Is destination IP address the right index for an RTT estimate? How about Request-URI?

2787 If a client wishes to measure RTT for a particular IP address, it MUST include a **Timestamp** header into
2788 a request containing the time when the request is initially created and passed to a new client transaction,
2789 which transmits the request. If a 100 response (not any 1xx, only the 100 response) is received before the
2790 client transaction generates a retransmission, an RTT estimate is made. This is consistent with the RFC
2791 2988 requirements on TCP for using Karn's algorithm in RTT estimation.

2792 The estimate, called R, is made by computing the difference between the current time and the value of
2793 **Timestamp** header in the 100 response. The value of R is applied to the estimation of RTO as described
2794 in Section 2 of RFC 2988 [24], with the following differences. First, the initial value of RTO is 500 ms for
2795 SIP, not 3 s as is used for TCP. Second, there is no minimum value for the RTO, as there is for TCP, if SIP
2796 is being run on a private network. When run on the public Internet, the minimum is 500 ms, as opposed to
2797 1 s for TCP. This difference is because of the expected usage of SIP in private networks where rapid call
2798 setup times are service critical. Once RTO is computed, the timer T1 is set to the value of RTO, and all other
2799 timers scale proportionally as described above.

2800 **18 Reliability of Provisional Responses**

2801 Placeholder.

2802 Reliability of provisional responses will be incorporated into bis. This is a heads up on that.

2803 **19 Transport**

2804 The transport layer is responsible for the actual transmission of requests and responses over network trans-
2805 ports. This includes determination of the connection to use for a request or response, in the case of connec-
2806 tion oriented transports.

2807 The transport layer is responsible for managing any persistent connections (for transports like TCP, TLS
2808 and SCTP) including ones it opened, as well as ones opened to it. This includes connections opened by the
2809 client or server transports, so that connections are shared between client and server transport functions. It is
2810 RECOMMENDED that connections be kept open for some implementation defined time after the last message
2811 was sent or received over that connection. This time SHOULD be at least 16 seconds in order to ensure with
2812 high probability that responses can be sent over the same connection a request was sent.

2813 All SIP elements MUST support UDP at a minimum.

2814 **19.1 Clients**

2815 **19.1.1 Sending Requests**

2816 The client side of the transport layer is responsible for sending the request and receiving responses. The
2817 user of the transport layer passes the client transport the request, an IP address, port, transport, and possibly
2818 TTL for multicast destinations.

2819 A client that sends a request to a multicast address MUST add the "maddr" parameter to its **Via** header
2820 field, and SHOULD add the "ttl" parameter. (In that case, the **maddr** parameter SHOULD contain the des-
2821 tination multicast address, although under exceptional circumstances it MAY contain a unicast address.)

2822 Requests sent to multicast groups SHOULD be scoped to ensure that they are not forwarded beyond the
2823 administrative domain to which they were targeted. This scooping MAY be done with either TTL or admin-
2824 istrative scopes [19], depending on what is implemented in the network.

2825 It is important to note that the layers above the transport layer do not operate differently for multicast
2826 as opposed to unicast requests. This means that SIP treats multicast more like anycast, assuming that there
2827 is a single recipient generating responses to requests. If this is not the case, the first response will end
2828 up “winning”, based on the client transaction rules. Any other responses from different UA will appear
2829 as retransmissions and be discarded. This limits the utility of multicast to cases where an anycast type of
2830 function is desired, such as registrations.

2831 OPEN ISSUE #7: This is a proposed resolution to whether or not multicast should be removed entirely.

2832 Before a request is sent, the client transport MUST insert a value of the sent-by field into the Via header.
2833 This field contains an IP address or host name, and port. In certain cases discussed in Section 19.2.2, this
2834 IP address and port are used to construct a SIP URL for sending the response. The transport layer MUST
2835 be prepared to receive incoming connections (and receive responses sent over such connections) on any IP
2836 addresses and ports that this SIP URL might resolve to using the procedures defined in Section 24. The
2837 transport layer MUST also be prepared to receive an incoming connection on the source IP address that the
2838 request was sent from, and port number in the sent-by field. The client transport MUST also be prepared to
2839 receive the response on the same connection used to send the request.

2840 For unreliable unicast transports, the client transport MUST be prepared to receive responses on the
2841 source IP address that the request is sent from (as responses are sent back to the source address), but the
2842 port number in the sent-by field. Furthermore, as with reliable transports, in certain cases the IP address and
2843 port are used to construct a URL for sending the response. The client transport MUST be prepared to receive
2844 responses on any IP address/port combinations that this SIP URL might resolve to using the procedures of
2845 Section 24.

2846 For multicast, the client transport MUST be prepared to receive responses on the same multicast group
2847 and port that the request is sent to.

2848 If a request is destined to an IP address, port, and transport to which an existing connection is open, it
2849 is RECOMMENDED that this connection be used to send the request, but another connection MAY be opened
2850 and used.

2851 If a request is sent using multicast, it is sent to the group address, port, and TTL provided by the transport
2852 user. If a request is sent using unicast unreliable transports, it is sent to the IP address and port provided by
2853 the transport user.

2854 19.1.2 Receiving Responses

2855 When a response is received, the client transport examines the top Via header. If the value of the sent-by
2856 parameter in that header does not correspond to a value that the client transport is configured to insert into
2857 requests, the response MUST be rejected.

2858 If there are any client transactions in existence, the client transport uses the matching procedures of Sec-
2859 tion 17.1.3 to attempt to match the response to an existing transaction. If there is a match, the response MUST
2860 be passed to that transaction. Otherwise, the response MUST be passed to the core (whether it be stateless
2861 proxy, stateful proxy, or UA) for further processing. Handling of these “stray” responses is dependent on
2862 the core (a stateless proxy will forward all responses, for example).

2863 19.2 Servers

2864 19.2.1 Receiving Requests

2865 When the server transport receives a request over any transport, it MUST examine the value of the sent-by
2866 parameter in the top *Via* header field. If the host portion of the sent-by parameter contains a domain name,
2867 or if it contains an IP address that differs from the packet source address, the server MUST add a “received”
2868 attribute to that *Via* header field. This attribute MUST contain the source address that the packet was received
2869 from. This is to assist the server transport layer in sending the response, since it must be sent to the source
2870 IP address that the request came from.

2871 Consider a request received by the server transport which looks like, in part:

```
2872 INVITE sip:bob@biloxi.com SIP/2.0  
2873 Via: SIP/2.0/UDP bobspc.biloxi.com:5060
```

2874 The request is received with a source IP address of 1.2.3.4. Before passing the request up, the transport
2875 would add a received parameter, so that the request would look like, in part:

```
2876 INVITE sip:bob@biloxi.com SIP/2.0  
2877 Via: SIP/2.0/UDP bobspc.biloxi.com:5060
```

2878 Next, the client transport attempts to match the request to the client transaction. It does so using the
2879 matching rules described in Section 17.2.3. If a matching server transaction is found, the request is passed
2880 to that transaction for processing. If no match is found, the request is passed to the core, which may decide
2881 to construct a new server transaction for that request.

2882 19.2.2 Sending Responses

2883 The server transport uses the value of the top *Via* header in order to determine where to send a response. It
2884 MUST follow the following process:

- 2885 ● If the “sent-protocol” is a reliable transport protocol such as TCP, TLS or SCTP, the response MUST
2886 be sent using the existing connection to the source of the original request that created the transaction, if
2887 that connection is still open. This does require the server transport to maintain an association between
2888 server transactions and transport connections. If that connection is no longer open, the server MAY
2889 open a connection to the IP address in the *received* parameter, if present, using the port in the *sent-by*
2890 value, or the default port for that transport, if no port is specified (5060 for UDP and TCP, 5061 for
2891 TLS and SSL). If that connection attempt fails, the server SHOULD construct a SIP URL of the form
2892 “sip:;sent-by host;transport=;sent-protocol;” and then use the procedures defined in Section 24 to
2893 determine the IP address and port to open the connection and send the response to.
- 2894 ● Otherwise, if the *Via* header field contains a “maddr” parameter, forward the response to the address
2895 listed there, using the port indicated in “sent-by”, or port 5060 if none is present. If the address is
2896 a multicast address, the response SHOULD be sent using the TTL indicated in the “ttl” parameter, or
2897 with a TTL of 1 if that parameter is not present.

- 2898 • Otherwise (for unreliable unicast transports), if the top Via has a received parameter, send the re-
2899 sponse to the address in the “received” parameter, using the port indicated in the “sent-by” value, or
2900 using port 5060 if none is specified explicitly. If this fails, e.g., elicits an ICMP “port unreachable”
2901 response, send the response to the address in the “sent-by” parameter. The address to send to is de-
2902 termined by constructing a SIP URL of the form “sip:sent-by;”, and then using the DNS procedures
2903 defined in Section 24 to send the response.
- 2904 • Otherwise, if it is not receiver-tagged, send the response to the address indicated by the “sent-by”
2905 value.

2906 **19.3 Framing**

2907 In the case of message oriented transports (such as UDP), if the message has a Content-Length header, the
2908 message body is assumed to contain that many bytes. If there are additional bytes in the transport packet
2909 below the end of the body, they MUST be discarded. If the transport packet ends before the end of the
2910 message body, this is considered an error. If the message is a response, it MUST be discarded. If its a
2911 request, the element SHOULD generate a 400 class response. If the message has no Content-Length header,
2912 the message body is assumed to end at the end of the transport packet.

2913 In the case of stream oriented transports (such as TCP), the Content-Length header indicates the size
2914 of the body. The Content-Length header MUST be used with stream oriented transports.

2915 **19.4 Error Handling**

2916 Error handling is independent of whether the message was a request or response.

2917 If the transport user asks for a message to be sent over an unreliable transport, and the result is an ICMP
2918 error, the behavior depends on the type of ICMP error. A host, network, port or protocol unreachable errors,
2919 or parameter problem errors SHOULD cause the transport layer to inform the transport user of a failure in
2920 sending. Source quench and TTL exceeded ICMP errors SHOULD be ignored.

2921 If the transport user asks for a request to be sent over a reliable transport, and the result is a connection
2922 failure, the transport layer SHOULD inform the transport user of a failure in sending.

2923 **20 Security Considerations**

2924 The fundamental security issues confronting SIP are: preserving the confidentiality and integrity of messag-
2925 ing, preventing replay attacks or message spoofing, ensuring the privacy of the participants in a session, and
2926 preventing denial of service attacks.

2927 SIP messages frequently contain sensitive information about their senders not just what they have to
2928 say, but with whom they communicate, when they communicate and for how long, and from where they
2929 participate in sessions. Many applications and their users require that this sort of private information be
2930 hidden from any parties that do not need to know it.

2931 Encryption provides the best means to preserve the confidentiality of signaling it can also guarantee
2932 that messages are not modified by any malicious intermediaries. However, SIP requests and responses
2933 cannot be encrypted end-to-end (that is, between a pair of distinct user agents who share encryption keys)
2934 in their entirety because message fields such as the Request-URI, Route and Via need, in most network
2935 architectures, to be visible to proxies so that SIP requests are routed correctly. Note that proxy servers need

2936 to modify signaling as well (adding *Via* headers) in order for SIP to function. Proxy servers must therefore
2937 be a part of trust relationships in SIP networks.

2938 Note that there are also less direct ways in which private information can be divulged. If a user or service
2939 chooses to be reachable at an address that is guessable from the person's name and organizational affiliation
2940 (which describes most addresses of record), the traditional method of ensuring privacy by having an unlisted
2941 "phone number" is compromised. A user location service can infringe on the privacy of the recipient of a
2942 session invitation by divulging their specific whereabouts to the caller; an implementation consequently
2943 SHOULD be able to restrict, on a per-user basis, what kind of location and availability information is given
2944 out to certain classes of callers.

2945 SIP entities also have a need to identify one another in a secure fashion. Ordinarily a SIP UA asserts
2946 an identity for the initiator of a request in the *From* header field, but in many systems this information
2947 is controlled directly by the end user, and thus spoofing the contents of the *From* is trivial. When a SIP
2948 endpoint asserts the identity of its user to a peer user agent or to a proxy server, that identity should in some
2949 way be verifiable. A cryptographic authentication mechanism is provided in SIP to address this requirement.

2950 The most comprehensive mechanisms for securing SIP messages (providing confidentiality and integrity
2951 guarantees for signaling as well as authentication) make use of transport or network layer encryption. en-
2952 cryption encrypts the entire SIP request or response on the wire so that packet sniffers or other eavesdroppers
2953 cannot see who is calling whom.

2954 Note that the security of SIP signaling itself has no bearing on the security of protocols used in concert
2955 with SIP such as RTP, or with any MIME types carried as SIP bodies, such as SDP. Any media associated
2956 with a session can be encrypted end-to-end without any of the problems associated with encrypting SIP
2957 signaling. Media encryption is outside the scope of this document.

2958 **20.1 Transport and Network Layer Security**

2959 SIP requests and responses MAY be protected by security mechanisms at the transport or network layer. No
2960 particular mechanism is recommended by this document, but two popular alternatives are briefly examined:
2961 protection at the transport layer can be afforded by TLS [25], and network layer security is provided by
2962 IPsec [26].

2963 Transport or network layer security encrypts signaling traffic, guaranteeing message confidentiality and
2964 integrity (note however that the originator and recipient of a session may be deducible by observers per-
2965 forming a network traffic analysis). The keys used to establish encrypt traffic can also be used to verify an
2966 asserted identity in many architectures, and therefore provide a means of authentication.

2967 IPsec is a network layer protocol essentially, a secure replacement for traditional IP (Internet Protocol).
2968 IPsec is most suited to VPN (virtual private network) architectures in which a set of SIP hosts (mingled user
2969 agents and proxy servers) or bridged administrative domains have a trust relationship with one another.

2970 TLS is a transport protocol and hence, like TCP and UDP, TLS can be specified as the desired transport
2971 protocol within a *Via* header field or a SIP-URI. TLS is most suited to architectures in which a chain of trust
2972 joins together a set of hosts (e.g. Alice trusts her local proxy server, which in turn trust Bob's local proxy
2973 server, which Bob trusts, hence Bob and Alice can communicate securely).

2974 TLS must be tightly coupled with a SIP application. Note that transport mechanisms are specified on
2975 a hop-by-hop basis in SIP, and that in some networks TLS might be used for only certain portions of the
2976 signaling path.

2977 It is RECOMMENDED that SIP endpoints support TLS as a secure transport for SIP.

2978 **20.2 SIP Authentication**

2979 SIP provides a stateless challenged-based mechanism for authentication. Any time that a proxy server or
2980 user agent receives a request, they MAY challenge the initiator of the request to provide assurance of their
2981 identity. Once the originator has been identified, the recipient of the request SHOULD ascertain whether or
2982 not this user is authorized to make the request in question. No authorization systems are recommended or
2983 discussed in this document.

2984 The “basic” and “digest” authentication mechanisms described in this section provide message authen-
2985 tication only, without message integrity or confidentiality. Protective measures above and beyond authen-
2986 tication need to be taken to prevent active attackers from modifying and/or replaying SIP requests and
2987 responses.

2988 Due to its weak security, the usage of “basic” authentication is NOT RECOMMENDED. However, servers
2989 MAY support it to handle older RFC 2543 clients that might still use it.

2990 **20.2.1 Framework**

2991 The framework for SIP authentication closely parallels that of HTTP (RFC 2617 [27]). In particular, the
2992 BNF for auth- scheme, auth-param, challenge, realm, realm-value, and credentials is identical. The
2993 401 response is used by user agent servers in SIP to challenge the identity of a user agent client. Additionally,
2994 registrars and redirect servers MAY make use of 401 (Unauthorized) responses for authentication, but proxies
2995 MUST NOT, and instead MAY use the 407 (Proxy Authentication Required) response. The requirements for
2996 inclusion of the Proxy-Authenticate, Proxy- Authorization, WWW-Authenticate, and Authorization in
2997 the various messages are identical to those described in RFC 2617 [27].

2998 Since SIP does not have the concept of a canonical root URL, the notion of protection spaces is inter-
2999 preted differently in SIP. The realm is a protection domain for all SIP URIs with the same value for the
3000 userinfo, host and port part of the SIP Request-URI. For example:

```
3001     INVITE sip:bob@biloxi.com SIP/2.0  
3002     WWW-Authenticate: Basic realm="business"
```

3003 and

```
3004     INVITE sip:robert@biloxi.com SIP/2.0  
3005     WWW-Authenticate: Basic realm="business"
```

3006 Generally, SIP authentication is for a specific request Request-URI and realm, a protection domain.
3007 Thus, for basic and digest authentication, each such protection domain has its own set of user names and
3008 secrets. If a user agent does not care about different Request-URIs, it makes sense to establish a “global”
3009 user name, secret and realm that is the default challenge if a particular Request-URI does not have its own
3010 realm or set of user names (e.g. an INVITE to ‘sip:10.3.6.6’). Similarly, SIP entities representing many
3011 users, such as PSTN gateways, MAY try a pre- configured global user name and secret when challenged,
3012 independent of the Request-URI.

3013 20.2.2 User to User Authentication

3014 When a UAS receives a request from a UAC, the UAS MAY authenticate the originator before the request
3015 is processed. If no credentials (in the Authorization header field are provided in the request, the UAS can
3016 challenge the originator to provide credentials by rejecting the request with a 401 (Unauthorized) status
3017 code.

3018 The WWW-Authenticate response-header field MUST be included in 401 (Unauthorized) response mes-
3019 sages. The field value consists of at least one challenge that indicates the authentication scheme(s) and
3020 parameters applicable to the Request-URI. See [H14.47] for a definition of the syntax.

3021 An example of the WWW-Authenticate in a 401 challenge is:

```
3022 WWW-Authenticate: Basic realm="business"
```

3023 When the originating UAC receives the 401 it SHOULD, if it is able, re-originate the request with the
3024 proper credentials. The UAC may require input from the originating user before proceeding. The content
3025 of the "realm" parameter of the WWW-Authenticate header SHOULD be displayed to the user. Once
3026 authentication credentials have been supplied (either directly by the user, or discovered in a keyring), user
3027 agents SHOULD cache the credentials for a given value of the Request-URI and "realm" and attempt to
3028 re-use these values on the next request for that destination.

3029 Any user agent that wishes to authenticate itself with a UAS or registrar – usually, but not necessarily,
3030 after receiving a 401 response – MAY do so by including an Authorization header field with the request.
3031 The Authorization field value consists of credentials containing the authentication information of the user
3032 agent for the realm of the resource being requested.

3033 An example of the Authorization header is:

```
3034 Authorization: Basic QWxhZGRpbjpvcmVudHluc2FtZQ==
```

3035 When a UAC resubmits a request with its credentials after receiving a 401 (or 407) response, it MUST
3036 increment the CSeq header field as it would normally do when sending an updated request.

3037 20.2.3 Proxy to User Authentication

3038 Similarly, when a UAC sends a request to a proxy server, the proxy server MAY authenticate the originator
3039 before the request is processed. If no credentials (in the Proxy-Authorization header field) are provided
3040 in the request, the UAS can challenge the originator to provide credentials by rejecting the request with a
3041 407 (Proxy Authentication Required) status code. The proxy MUST populate the 407 (Proxy Authentication
3042 Required) message with a Proxy-Authenticate header applicable to the proxy for the requested resource.

3043 The use of the Proxy-Authentication and Proxy-Authorization parallel that described in [27, Sec-
3044 tion 3.6], with one difference. Proxies MUST NOT add the Proxy-Authorization header. 407 (Proxy Au-
3045 thentication Required) responses MUST be forwarded upstream towards the UAC following the procedures
3046 for any other response. It is the client's responsibility to add the Proxy-Authorization header containing
3047 credentials for the realm of the proxy which has asked for authentication.

3048 If a proxy were to resubmit a request with a Proxy-Authorization header field, it would need to increment the
3049 CSeq in the new request. However, this would mean that the UAC which submitted the original request would
3050 discard a response from the UAS, as the CSeq value would be different.

3051 When the originating UAC receives the 407 it SHOULD, if it is able, re-originate the request with the
3052 proper credentials. It should follow the same procedures for the display of the “realm” parameter that are
3053 given above for responding to 401.

3054 Any user agent that wishes to authenticate itself to a proxy server – usually, but not necessarily, after
3055 receiving a 407 response – MAY do so by including an Proxy-Authorization header field with the request.
3056 The Proxy-Authorization request-header field allows the client to identify itself (or its user) to a proxy
3057 which requires authentication. The Proxy-Authorization field value consists of credentials containing the
3058 authentication information of the user agent for the proxy and/or realm of the resource being requested.

3059 A Proxy-Authorization header field applies only to the proxy whose realm is identifier in the “realm”
3060 parameter (this proxy may previously have demanded authentication using the Proxy-Authenticate field).
3061 When multiple proxies are used in a chain, the Proxy-Authorization header field MUST NOT be consumed
3062 by any proxy whose realm does not match the “realm” parameter specified in the Proxy-Authorization
3063 header.

3064 Note that if an authentication scheme is used in the Proxy- Authorization that does not support realms,
3065 a proxy server MUST attempt to parse all Proxy-Authorization headers to determine whether or not one
3066 of them has what it considers to be valid credentials. Because this is potentially very time consuming in
3067 large networks, proxy servers SHOULD use an authentication scheme that supports realms in the Proxy-
3068 Authorization header.

3069 It is also possible that a 401 or 407 response will contain several challenges, from a mixture of proxies
3070 and user agent servers, if the request was forked. If at least one user agent responds to a request with a
3071 challenge, than a 401 should be used; otherwise a 407 should be used. When resubmitting its request in
3072 response to the challenge, the UAC needs to include an Authorization for each WWW-Authenticate and
3073 Proxy- Authorization for each Proxy-Authenticate.

3074 See [H14.34] for a definition of the syntax of Proxy- Authentication and Proxy-Authorization.

3075 **20.2.4 Authentication Schemes**

3076 SIP implementations MAY use HTTP’s basic and digest authentication mechanisms ([27]) to provide a rudi-
3077 mentary form of security. This section overviews usage of these mechanisms in SIP. The scheme usage is
3078 almost completely identical to that for HTTP [27]. This section outlines this operation, pointing to RFC
3079 2617 ([27]) for details and noting the differences that arise when using SIP. Since RFC 2543 is based on
3080 HTTP basic and digest as defined in RFC 2069 [28], SIP servers supporting RFC 2617 MUST ensure they
3081 are backwards compatible with RFC 2069. Procedures for this backwards compatibility are specified in
3082 RFC 2617.

3083 **20.2.4.1 HTTP Basic** The rules for basic authentication follow those defined in [27, Section 2] but with
3084 the words “origin server” replaced with “user agent server, redirect server , or registrar”.

3085 Since SIP URIs are not hierarchical, the paragraph in [27, Section 2] that states that “all paths at or
3086 deeper than the depth of the last symbolic element in the path field of the Request-URI also are within the
3087 protection space specified by the Basic realm value of the current challenge” does not apply for SIP. SIP
3088 clients MAY preemptively send the corresponding Authorization header with requests for SIP URIs within
3089 the same protection realm (as defined above) without receipt of another challenge from the server.

3090 **20.2.4.2 HTTP Digest** The rules for digest authentication follow those defined in [27, Section 3], with
3091 “HTTP 1.1” replaced by “SIP/2.0” in addition to the following differences:

3092 1. The URI included in the challenge has the following BNF:

3093 URI = SIP-URL

3094 2. The BNF in RFC 2617 has an error in that the URI is not enclosed in quotation marks. (The example
3095 in Section 3.5 is correct.) For SIP, the URI **MUST** be enclosed in quotation marks.

3096 3. The BNF for digest-uri-value is:

3097 digest-uri-value = Request-URI ; as defined in Section 26

3098 4. The example procedure for choosing a nonce based on Etag does not work for SIP.

3099 5. The text in RFC 2617 [27] regarding cache operation does not apply to SIP.

3100 6. RFC 2617 [27] requires that a server check that the URI in the request line, and the URI included in
3101 the Authorization header, point to the same resource. In a SIP context, these two URI's may actually
3102 refer to different users, due to forwarding at some proxy. Therefore, in SIP, a server **MAY** check
3103 that the Request-URI in the Authorization header corresponds to a user for whom that the server is
3104 willing to accept forwarded or direct calls.

3105 RFC2543 did not allow usage of the Authentication-Info header (it effectively used RFC 2069). How-
3106 ever, we now allow usage of this header, since it provides integrity checks over the bodies and provides
3107 mutual authentication. RFC2617 [27] defines mechanisms for backwards compatibility using the qop at-
3108 tribute in the request. These mechanisms **MUST** be used by a server to determine if the client supports the
3109 new mechanisms in RFC 2617 that were not specified in RFC 2069.

3110 **20.3 SIP Encryption**

3111 No mechanism is currently specified for encrypting entire SIP messages end-to-end for the purpose of con-
3112 fidentiality. This is a hard problem because network intermediaries (like proxy servers) need to view certain
3113 headers in order to route messages correctly, and if these intermediaries are excluded from security associa-
3114 tions then SIP messages will essentially be unroutable.

3115 That much said, SIP messages carry MIME bodies and the MIME standard includes mechanisms for
3116 securing MIME contents to ensure both integrity and confidentiality (including the 'multipart/encrypted'
3117 MIME type, see [29]), but detailed description of the use of secure MIME types are outside the scope of this
3118 document. Implementors should note, however, that there may be rare network intermediaries (not typical
3119 proxy servers) that rely on viewing or modifying the bodies of SIP messages (especially SDP), and that
3120 secure MIME may prevent these sorts of intermediaries from functioning.

3121 This applies particularly to certain types of firewalls.

3122 End-to-end encryption relies on keys shared by the two user agents involved in the request. Typically,
3123 the message is sent encrypted with the public key of the recipient, so that only that recipient can read the
3124 message. SIP does not define any mechanism for end-to-end key exchange.

3125 Note that the PGP mechanism for encrypting the headers and bodies of SIP messages described in RFC2543 has
3126 been deprecated.

3127 20.4 Denial of Service

3128 Denial of service attacks focus on rendering a particular network element unavailable, usually by directing
3129 an excessive amount of network traffic at its interfaces. A distributed denial of service attack allows one
3130 network user to cause multiple network hosts to flood a target host with a large amount of network traffic.

3131 In many architectures SIP proxy servers face the public Internet in order to accept requests from world-
3132 wide IP endpoints. When the host on which a SIP proxy server is operating is routable from the public
3133 Internet, it should be deployed in an administrative domain with secure routing policies (blocking source-
3134 routed traffic, preferably filtering ping traffic).

3135 SIP creates a number of potential opportunities for distributed denial of service attacks that must be
3136 recognized and addressed by the implementors and operators of SIP systems.

3137 Floods of messages directed at proxy servers can lock up proxy server resources and prevent desirable
3138 traffic from reaching its destination. There is a computational expense associated with processing a SIP
3139 transaction at a proxy server, and that expense is greater for stateful proxy servers than it is for stateless
3140 proxy servers. Therefore stateful proxies are more susceptible to flooding than stateless proxy servers.

3141 Attackers can create bogus requests that contain a falsified *Via* header field which identifies a targeted
3142 host as the originator of the message and then send this message to a large number of SIP network elements,
3143 thereby using hapless SIP UAs or proxies to generate denial of service traffic aimed at the target.

3144 Similarly, attackers might use falsified *Route* headers in a request that identify the target host and then
3145 send such messages to forking proxies that will amplify messaging sent to the target. *Record-Route* could
3146 be used to similar effect when the attacker is certain that the SIP dialog initiated by the request will result in
3147 numerous transactions originating in the backwards direction.

3148 One could prevent one's host from being commandeered for such an attack by disallowing requests that
3149 do not make use of a persistent security association established through a transport or network layer security
3150 instrument such as TLS or IPsec. This could be an appropriate security solution for two proxy servers that
3151 trust one another and exchange significant amounts of signaling traffic with one another, or between a user
3152 agent and its outbound proxy.

3153 Both TLS and IPSec can also make use of bastion hosts at the edges of administrative domains that
3154 participate in the security associations to aggregate secure tunnels and sockets. These bastion hosts can also
3155 take the brunt of denial of service attacks, ensuring that SIP hosts within the administrative domain are not
3156 encumbered with superfluous messaging.

3157 If such a persistent security association is not feasible, user agents and proxy servers SHOULD chal-
3158 lenge questionable requests with only a *single* 401 (Unauthorized) or 407 (Proxy Authentication Required)
3159 forgoing the normal response retransmission algorithm.

3160 Retransmitting the 401 or 407 status response amplifies the problem of an attacker using a falsified header (such
3161 as *Via*) to direct traffic to a third party.

3162 A number of denial of service attacks open up if *REGISTER* requests are not properly authenticated
3163 and authorized by registrars. Attackers could de-register some or all users in an administrative domain,
3164 thereby preventing these users from being invited to new sessions. An attacker could also register a large
3165 number of contacts designating the same host for a given address of record in order to use the registrar and
3166 any associated proxy servers as amplifiers in a denial of service attack. Attackers might also attempt to
3167 deplete available memory and disk resources of a registrar by registering huge numbers of bindings.

3168 With either TCP or UDP, a denial of service attack exists by a rogue proxy sending 6xx responses.
3169 Although a client SHOULD choose to ignore such responses if it requested authentication, a proxy cannot do
3170 so. It is obliged to forward the 6xx response back to the client. The client can then ignore the response, but
3171 if it repeats the request it will probably reach the same rogue proxy again, and the process will repeat.

3172 The use of multicast to transmit SIP requests can greatly increase the potential for denial of service
3173 attacks.

3174 21 Common Message Components

3175 There are certain components of SIP messages that appear in various places within SIP messages (and
3176 sometimes, outside of them), which merit separate discussion.

3177 21.1 SIP Uniform Resource Locators

3178 A SIP URL identifies a communications resource. Like all URLs, SIP URLs may be placed in web pages,
3179 email messages or printed literature. They contain sufficient information to initiate and maintain a commu-
3180 nication session with the resource.

3181 Examples of communications resources include

- 3182 • a user of an online service
- 3183 • an appearance on a multiline phone
- 3184 • a mailbox on a messaging system
- 3185 • a PSTN phone number at a gateway service
- 3186 • a group (such as “sales” or “helpdesk”) in an organization

3187 21.1.1 SIP URL components

3188 The “sip:” scheme follows the guidelines in RFC 2396 [9]. It uses a form similar to the mailto URL, al-
3189 lowing the specification of SIP request-header fields and the SIP message- body. This makes it possible
3190 to specify the subject, media type, or urgency of sessions initiated by using a URL on a web page or in an
3191 email message. The formal syntax for a SIP URL is presented in Section 26. Its general form is

3192 sip:user:password@host:port;url-parameters?headers

3193 These tokens, and some of the tokens in their expansion, have the following meanings.

3194 **user:** The identifier of a particular resource at the host being addressed. Note that “host” as used here may,
3195 and frequently does, refer to a domain.

3196 The “userpart” of a URL consists of this user field, the password field and the @ sign following them.
3197 The userpart of a URL is optional and MAY be absent when the destination host does not have a notion
3198 of users or when the host itself is the resource being identified. If the @ sign is present in a SIP URL,
3199 the user field MUST NOT be empty.

3200 If the host being addressed is capable of processing telephone numbers, an Internet telephony gateway
3201 for instance, a telephone- subscriber field defined in RFC 2806 [13] MAY be used to populate the
3202 user field. There are special escaping rules for encoding telephone-subscriber fields in SIP URLs
3203 described in Section 21.1.2.

3204 **password:** A password associated with the user

3205 While the SIP URL syntax allows this field to be present, its use is NOT RECOMMENDED, because
3206 the passing of authentication information in clear text (such as URIs) has proven to be a security risk
3207 in almost every case where it has been used. For instance, transporting a PIN number in this field
3208 exposes the PIN.

3209 **host:** The entity hosting the SIP resource

3210 The **host** part contains either a fully-qualified domain name or numeric IPv4 or IPv6 address. Using
3211 the fully-qualified domain name form is RECOMMENDED whenever possible.

3212 **port:** The port number where the request is to be sent.

3213 **URL parameters:** Parameters affecting a request constructed from the URL.

3214 URL parameters are added after the **hostport** component and are separated by semi-colons. This
3215 extensible mechanism includes the **transport**, **maddr**, **ttl**, **user**, and **method** parameters.

3216 The **transport** parameter determines the transport mechanism to be used for sending SIP messages.
3217 SIP can use any network transport protocol. Parameter names are defined for UDP [30], TCP [31],
3218 TLS [25], and SCTP [32].

3219 The **maddr** parameter indicates the server address to be contacted for this user, overriding any address
3220 derived from the **host** field. Section 24 describes the proper interpretation of the **transport**, **maddr**
3221 and **hostport** in order to obtain the destination address, port and transport for sending a request.

3222 The **maddr** field can be used as a simple form of loose source routing. It allows a URL to specify a specific
3223 proxy that must be traversed en-route to the destination. This capability is useful for a roaming user that is
3224 forced to use an outbound proxy, but wishes to force requests through their home proxy.

3225 The **ttl** parameter determines the time-to-live value of the UDP multicast packet and MUST only
3226 be used if **maddr** is a multicast address and the transport protocol is UDP. The **user** parameter
3227 was described above. For example, to specify to call `alice@atlanta.com` using multicast to
3228 `239.255.255.1` with a **ttl** of 15, the following URL would be used:

3229 `sip:alice@atlanta.com;maddr=239.255.255.1;ttl=15`

3230 The set of valid **telephone-subscriber** strings is a subset of valid **user** strings. The **user** URL
3231 parameter exists to distinguish telephone numbers from user names that happen to look like telephone
3232 numbers. If the user string contains a telephone number formatted as a **telephone-subscriber**, the
3233 **user** parameter value "phone" SHOULD be present. Even without this parameter, recipients of SIP
3234 URLs MAY interpret the **pre-@** part as a telephone number if local restrictions on the name space for
3235 user name allow it.

3236 The method of the SIP request constructed from the URL can be specified with the **method** parameter.

3237 Since the url-parameter mechanism is extensible, SIP elements MUST silently ignore any url-parameters
3238 that they do not understand.

3239 **Headers:** Headers to be included in a request constructed from the URL.

3240 Headers fields in the SIP request can be specified with the “?” mechanism within a SIP URL. The
3241 header names and values are encoded in ampersand separated `hname = hvalue` pairs. The special
3242 `hname` “body” indicates that the associated `hvalue` is the message-body of the SIP request.

3243 Table 1 summarizes the use of SIP URL components based on the context in which the URL appears.
3244 The external column describes URLs appearing anywhere outside of a SIP message, for instance on a web
3245 page or business card. Entries marked “m” are mandatory, those marked “o” are optional, and those marked
3246 “-” are not allowed. Elements processing URLs SHOULD ignore any disallowed components if they are
3247 present. The second column indicates the default value of an optional element if it is not present. “-”
3248 indicates that the element is either not optional, or has no default value.

3249 SIP URLs in `Contact` header fields have different restrictions depending on the context in which the
3250 header field appears. One set applies to messages that establish and maintain dialogs (INVITE and its 200
3251 OK response). The other applies to registration and redirection messages (REGISTER, its 200 OK response,
3252 and 3xx class responses to any method).

3253 OPEN ISSUE #203: `maddr` is disallowed in `To/From`, but not port. Should port be disallowed?

3254 OPEN ISSUE #204: Password is disallowed in `From`, but not `To`. Why?

3255 OPEN ISSUE #205: Should we allow method and header URL components in registration/redirect
3256 `Contacts`. What do they mean?

	default	Req.-URI	To	From	reg./redir. Contact	dialog Contact/ R-R/Route	external
user	–	o	o	o	o	o	o
password	–	o	o	-	o	o	o
host	–	m	m	m	m	m	m
port	5060	o	o	o	o	o	o
user-param	ip	o	o	o	o	o	o
method	INVITE	-	-	-	o	-	o
<code>maddr-param</code>	–	o	-	-	o	o	o
<code>ttl-param</code>	1	o	-	-	o	-	o
<code>transp.-param</code>	udp	o	-	-	o	o	o
<code>other-param</code>	–	o	o	o	o	o	o
headers	–	-	-	-	o	-	o

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

3257 21.1.2 Character escaping requirements

3258 SIP follows the requirements and guidelines of RFC 2396 when defining the set of characters that must be
3259 escaped in a SIP URL, and uses its “”%” HEX HEX” mechanism for escaping. From RFC 2396:

3260 The set of characters actually reserved within any given URI component is defined by that com-
3261 ponent. In general, a character is reserved if the semantics of the URI changes if the character
3262 is replaced with its escaped US-ASCII encoding. [9].

3263 Excluded US-ASCII characters [9, Sec. 2.4.3], such as space and control characters and characters used as
3264 URL delimiters, also **MUST** be escaped. URLs **MUST NOT** contain unescaped space and control characters.

3265 For each component, the set of valid BNF expansions defines exactly which characters may appear
3266 unescaped. All other characters **MUST** be escaped.

3267 For example, “@” is not in the set of characters in the user component, so the user “j@s0n” must have
3268 at least the @ sign encoded, as in “j%40s0n”.

3269 Expanding the hname and hvalue tokens in Section 26 show that all URL reserved characters in header
3270 names and values **MUST** be escaped.

3271 The `telephone-subscriber` subset of the `user` component has special escaping considerations. The set
3272 of characters not reserved in the RFC 2806 [13] description of `telephone-subscriber` contains a number
3273 of characters in various syntax elements that need to be escaped when used in SIP URLs. Any characters
3274 occurring in a `telephone-subscriber` that do not appear in an expansion of the BNF for the `user` rule **MUST**
3275 be escaped.

3276 21.1.3 Example SIP URLs

```
3277 sip:alice@atlanta.com  
3278 sip:alice:secretword@atlanta.com;transport=tcp  
3279 sip:alice@atlanta.com?subject=project%20x&priority=urgent  
3280 sip:+1-212-555-1212:1234@gateway.com;user=phone  
3281 sip:1212@gateway.com  
3282 sip:alice@10.1.1.1  
3283 sip:atlanta.com;method=REGISTER?to=alice%40atlanta.com  
3284 sip:alice;day=tuesday@atlanta.com
```

3285 The last example URL above has a `user` field value of “alice;day=tuesday”. The escaping rules defined
3286 above allow a semicolon to appear unescaped in this field. Note, however, that for the purposes of this
3287 protocol, the field is opaque. The apparent structure in that value is only useful to the entity responsible for
3288 the resource.

3289 21.1.4 SIP URL Comparison

3290 SIP URLs are compared for equality according to the following rules:

- 3291 • Comparisons of scheme name (“sip”), domain names, parameter names and header names are case-
3292 insensitive, all other comparisons are case-sensitive. (OPEN ISSUE #100 : There is a proposal to
3293 make only quoted string comparisons case-sensitive.)
- 3294 • The ordering of parameters and headers is not significant in comparing SIP URLs.
- 3295 • Characters other than those in the “reserved” and “unsafe” sets (see RFC 2396 [9]) are equivalent to
3296 their “”%” HEX HEX” encoding.
- 3297 • An IP address that is the result of a DNS lookup of a host name does **not** match that host name.
- 3298 • For two URLs to be equal, the `user`, `password`, `host`, and `port` components must match. A URL
3299 omitting the optional port component will match a URL explicitly declaring port 5060. A URL

3300 omitting the user component will **not** match a URL that includes one. A URL omitting the password
3301 component will **not** match a URL that includes one.

- 3302 • URL url-parameter components are compared as follows
 - 3303 – Any url-parameter appearing in both URLs must match.
 - 3304 – A user, transport, ttl, or method url-parameter appearing in only one URL must contain its
3305 default value or the URLs do not match.
 - 3306 – All other url-parameters appearing in only one URL are ignored when comparing the URLs.
- 3307 • URL header components are never ignored. Any present header component **MUST** be present in
3308 both URLs and match for the URLs to match. The matching rules are defined for each header in
3309 Section sec:header-fields.

3310 The URLs within each of the following sets are equivalent:

3311 sip:alice@%61tlanta.com:5060
3312 sip:alice@AtLanTa.CoM;Transport=udp

3313 sip:carol@chicago.com
3314 sip:carol@chicago.com;newparam=5
3315 sip:carol@chicago.com;security=on

3316 sip:biloxi.com;transport=tcp;method=REGISTER?to=sip:bob%40biloxi.com
3317 sip:biloxi.com;method=REGISTER;transport=tcp?to=sip:bob%40biloxi.com

3318 sip:alice@atlanta.com?subject=project%20x&priority=urgent
3319 sip:alice@atlanta.com?priority=urgent&subject=project%20x

3320 The URLs within each of the following sets are **not** equivalent:

3321 SIP:ALICE@AtLanTa.CoM;Transport=udp (different usernames)
3322 sip:alice@AtLanTa.CoM;Transport=UDP

3323 sip:bob@biloxi.com (different port and transport)
3324 sip:bob@biloxi.com:6000;transport=tcp

3325 sip:carol@chicago.com (different header component)
3326 sip:carol@chicago.com?Subject=next%20meeting

3327 sip:bob@phone21.bboxesbybob.com (even though that's what
3328 sip:bob@10.4.1.4 phone21.bboxesbybob.com resolves to)

3329 Note that equality is not transitive:

3330 sip:carol@chicago.com and sip:carol@chicago.com;security=on are equivalent

3331 and sip:carol@chicago.com and sip:carol@chicago.com;security=off are equivalent

3332 But sip:carol@chicago.com;security=on and sip:carol@chicago.com;security=off are **not** equivalent

3333 Comparing URLs is a major part of comparing several SIP headers (see Section 22).

3334 21.2 Option Tags

3335 Option tags are unique identifiers used to designate new options (extensions) in SIP. These tags are used in
3336 Require (Section 22.30), Proxy-Require (Section 22.28, Supported (Section 22.35) and Unsupported
3337 (Section 22.38) header fields. Note that these options appear as parameters in those headers in an option-tag
3338 = token form (see Section 26 for the definition of token).

3339 The creator of a new SIP option **MUST** either prefix the option with their reverse domain name or register
3340 the new option with the Internet Assigned Numbers Authority (IANA) (See Section 27).

3341 An example of a reverse-domain-name option is “com.foo.mynewfeature”, whose inventor can be reached
3342 at “foo.com”. For these features, individual organizations are responsible for ensuring that option names do
3343 not collide within the same domain. The host name part of the option **MUST** use lower-case; the option name
3344 is case-sensitive.

3345 Options registered with IANA do not contain periods and are globally unique. IANA option tags are
3346 case-sensitive.

3347 21.3 Tags

3348 The “tag” parameter is used in the To and From fields of SIP messages. It serves as a general mechanism
3349 to identify a particular instance of a user agent for a particular SIP URI.

3350 As proxies can fork requests, the same request can reach multiple instances of a user (mobile and home
3351 phones, for example). Since each can respond, there needs to be a means for the originator of a session to
3352 distinguish the responses. Tag fields in the To and From disambiguate these multiple instances of the same
3353 user.

3354 This situation also arises with multicast requests.

3355 When a tag is generated by a UA for insertion into a request or response, it **MUST** be globally unique and
3356 cryptographically random with at least 32 bits of randomness. A property of this selection requirement is
3357 that a UA will place a different tag into the From header of an INVITE as it would place into the To header
3358 of the response to the same INVITE. This is needed in order for a UA to invite itself to a session, a common
3359 case for “hairpinning” of calls in PSTN gateways.

3360 Besides the requirement for global uniqueness, the algorithm for generating a tag is implementation
3361 specific. Tags are helpful in fault tolerant systems, where a dialog is to be recovered on an alternate server
3362 after a failure. A UAS can select the tag in such a way that a backup can recognize a request as part of a
3363 dialog on the failed server, and therefore determine that it should attempt to recover the dialog and any other
3364 state associated with it.

3365 22 Header Fields

3366 The general syntax for header fields is covered in Section 7.3. This section lists the full set of header fields
3367 along with notes on syntax, meaning, and usage. Throughout this section, we use [HX.Y] to refer to Section
3368 X.Y of the current HTTP/1.1 specification RFC 2617 [27]. Examples of each header field are given.

3369 Information about header fields in relation to methods and proxy processing is summarized in Ta-
3370 bles 2 and 3.

3371 The “where” column describes the request and response types in which the header field can be used.
3372 Values in this column are:

3373 **R:** refers to header fields that can be used in requests.

3374 **r:** designates a header field as applicable to all responses, while a list of numeric values indicates the status
3375 codes with which the header field can be used.

3376 **c:** indicates a header field is copied from the request to the response.

3377 The “proxy” column describes the operations a proxy may perform on a header.

3378 **c:** indicates that a proxy can add (concatenate) comma-separated elements to the header

3379 **m:** indicates that a proxy can modify the header

3380 **a:** indicates that a proxy can add the header if not present

3381 **r:** indicates that a proxy must be able to read the header. Headers that need to be read cannot be en-
3382 crypted.

3383 The next six columns relate to the presence of a header field in a method, with the contents indicating:

3384 **o:** for optional

3385 **m:** for mandatory

3386 **m*:** indicates a header that SHOULD be sent, but servers need to be prepared to receive messages without
3387 that header field.

3388 ***:** indicates that the header fields are required if the message body is not empty. See sections 22.14, 22.15
3389 and 7.4 for details.

3390 **-:** for not applicable.

3391 “Optional” means that a UA MAY include the header field in a request or response, and a UA MAY ignore
3392 the header field if present in the request or response (The exception to this rule is the **Require** header field
3393 discussed in 22.30). A “mandatory” header field MUST be present in a request, and MUST be understood by
3394 the UAS receiving the request. A mandatory response header field MUST be present in the response, and the
3395 header field MUST be understood by the UAC processing the response. “Not applicable” means for header
3396 fields that the header field MUST NOT be present in a request. If one is placed in a request by mistake, it
3397 MUST be ignored by the UAS receiving the request. Similarly, a header field labeled “not applicable” for a
3398 response means that the UAS MUST NOT place the header in the response, and the UAC MUST ignore the
3399 header in the response.

3400 A compact form of some common header fields is also defined for use when overall message size is an
3401 issue.

3402 The **Contact**, **From** and **To** header fields contain a URL. If the URL contains a comma, question mark
3403 or semicolon, the URL MUST be enclosed in angle brackets (< and >). Any URL parameters are contained
3404 within these brackets. If the URL is not enclosed in angle brackets, any semicolon-delimited parameters are
3405 header-parameters, not URL parameters.

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG
Accept	R		-	o	-	m*	o	o
Accept	2xx		-	-	-	m*	o	o
Accept	415		-	o	-	o	o	o
Accept-Encoding	R		-	o	-	m*	o	o
Accept-Encoding	2xx		-	-	-	m*	o	o
Accept-Encoding	415		-	o	-	o	o	o
Accept-Language	R		-	o	-	m*	o	o
Accept-Language	2xx		-	-	-	m*	o	o
Accept-Language	415		-	o	-	o	o	o
Alert-Info	R	am	-	-	-	o	-	-
Alert-Info	180	am	-	-	-	o	-	-
Allow	R		o	o	o	o	o	o
Allow	2xx		-	o	o	m*	m*	o
Allow	r		-	o	o	o	o	o
Allow	405		-	m	m	m	m	m
Authentication-Info	2xx		-	o	-	o	o	o
Authorization	R		o	o	o	o	o	o
Call-ID	c	r	m	m	m	m	m	m
Call-Info		am	-	-	-	o	o	o
Contact	R		o	-	-	m	o	o
Contact	1xx		-	-	-	o	o	-
Contact	2xx		-	-	-	m	o	o
Contact	3xx		-	o	-	o	o	o
Contact	485		-	o	-	o	o	o
Content-Disposition			o	o	-	o	o	o
Content-Encoding			o	o	-	o	o	o
Content-Language			o	o	-	o	o	o
Content-Length		r	m*	m*	m*	m*	m*	m*
Content-Type			*	*	-	*	*	*
CSeq	c	r	m	m	m	m	m	m
Date		a	o	o	o	o	o	o
Error-Info	300-699		-	o	o	o	o	o
Expires			-	-	-	o	-	o
From	c	r	m	m	m	m	m	m
In-Reply-To	R		-	-	-	o	-	-
Max-Forwards	R	rm	o	o	o	o	o	o
MIME-Version			o	o	o	o	o	o
Organization		am	-	-	-	o	o	o

Table 2: Summary of header fields, A–O

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

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

3406 22.1 Accept

3407 The Accept header follows the syntax defined in [H14.1]. The semantics are also identical, with the excep-
 3408 tion that if no Accept header is present, the server SHOULD assume a default value of `application/sdp`.

3409 Example:

3410 `Accept: application/sdp;level=1, application/x-private, text/html`

3411 22.2 Accept-Encoding

3412 The Accept-Encoding header field is similar to Accept, but restricts the content-codings [H3.5] that are
 3413 acceptable in the response. See [H14.3]. The syntax of this header is defined in [H14.3]. The semantics in
 3414 SIP are identical to those defined in [H14.3].

3415 An empty Accept-Encoding header field is permissible, even though the syntax in [H14.3] does not
 3416 provide for it. It is equivalent to `Accept-Encoding: identity`, i.e., only the identity encoding, meaning no
 3417 encoding, is permissible. If this header is not present, the default value is `identity`. This differs slightly
 3418 from the HTTP definition, which indicates that when not present, any encoding can be used, but the identity
 3419 encoding is preferred.

3420 Example:

3421 Accept-Encoding: gzip

3422 **22.3 Accept-Language**

3423 The Accept-Language header follows the syntax defined in [H14.4]. The rules for ordering the languages
3424 based on the “q” parameter apply to SIP as well.

3425 The Accept-Language header is used in requests to indicate the preferred languages for reason phrases,
3426 session descriptions or status responses carried as message bodies in the response. If no Accept-Language
3427 header field is present in a request, the server assumes all languages are acceptable to the client.

3428 Example:

3429 Accept-Language: da, en-gb;q=0.8, en;q=0.7

3430 **22.4 Alert-Info**

3431 When present in an INVITE request, the Alert-Info header field specifies an alternative ring tone to the UAS.
3432 When present in a 180 (Ringing) response, the Alert-Info header field specifies an alternative ringback tone
3433 to the UAC. A typical usage is for a proxy to insert this header to provide a distinctive ring feature.

3434 The Alert-Info header can introduce security risks. These risks, and the ways to handle them, are
3435 discussed in Section 22.9 which discusses the Call-Info header, as the risks are identical.

3436 In addition, a user SHOULD be able to disable this feature selectively.

3437 This helps prevent disruptions that could result from the use of this header by untrusted elements.

3438 Example:

3439 Alert-Info: <http://www.example.com/sounds/moo.wav>

3440 **22.5 Allow**

3441 The Allow header field lists the set of methods supported by the user agent generating the message.

3442 All methods, including ACK and CANCEL, understood by the UA MUST be included in the list of
3443 methods in the Allow header, when present. The absence of an Allow header MUST NOT be interpreted to
3444 mean that the UA sending the message supports no methods. Rather, it implies that the UA is not providing
3445 any information on what methods it supports.

3446 Supplying an Allow header in responses to methods other than OPTIONS cuts down on the number of
3447 messages needed.

3448 Example:

3449 Allow: INVITE, ACK, OPTIONS, CANCEL, BYE

3450 **22.6 Authentication-Info**

3451 The Authentication-Info header provides for mutual authentication with HTTP Digest. A UAS MAY include
3452 this header in a 2xx response to a request that was successfully authenticated using digest based on the
3453 Authorization header.

3454 Syntax and semantics follow those specified in RFC2617 [27].

3455 Example:

3456 Authentication-Info: nextnonce="47364c23432d2e131a5fb210812c"

3457 22.7 Authorization

3458 The Authorization header field contains authentication credentials of a UA. Section 20.2.2 overviews the
3459 use of the Authorization header field, and Section 20.2.4 describes the syntax and semantics when used
3460 with HTTP Basic and Digest authentication.

3461 Note that this header field, along with Proxy-Authorization breaks the general rules about multiple
3462 header fields. Although not a comma-separated list, this header field may be present multiple times, and
3463 MUST NOT be combined into a single header using the usual rules described in Section 7.3.

3464 Example:

```
3465 Authorization: Digest username="Alice", realm="Bob's Friends",  
3466 nonce="84a4cc6f3082121f32b42a2187831a9e",  
3467 response="7587245234b3434cc3412213e5f113a5432"
```

3468 22.8 Call-ID

3469 The Call-ID header field uniquely identifies a particular invitation or all registrations of a particular client.
3470 Note that a single multimedia conference can give rise to several calls with different Call-IDs, e.g., if a user
3471 invites a single individual several times to the same (long-running) conference. Call-IDs are case-sensitive
3472 and are simply compared byte-by-byte.

3473 The compact form of the Call-ID header field is i.

3474 Examples:

```
3475 Call-ID: f81d4fae-7dec-11d0-a765-00a0c91e6bf6@biloxi.com  
3476 i:f81d4fae-7dec-11d0-a765-00a0c91e6bf6@10.4.1.4
```

3477 22.9 Call-Info

3478 The Call-Info header field provides additional information about the caller or callee, depending on whether
3479 it is found in a request or response. The purpose of the URI is described by the "purpose" parameter.
3480 "icon" designates an image suitable as an iconic representation of the caller or callee; "info" describes the
3481 caller or callee in general, e.g., through a web page; "card" provides a business card (e.g., in vCard [33] or
3482 LDIF [34] formats). Additional tokens can be registered using IANA and the procedures in Section 27.

3483 Usage of the Call-Info header can pose a security risk. If a callee fetches the URLs provided by an
3484 malicious caller, the callee may be at risk for displaying inappropriate or offensive content, dangerous or
3485 illegal content, and so on. Therefore, it is RECOMMENDED that a UA only render the information in the
3486 Call-Info header if it can verify the authenticity of the element which originated the header, and trusts that
3487 element. This need not be the peer UA; a proxy can insert this header into requests.

3488 The use of this header is important in converged applications.

3489 Example:

```
3490 Call-Info: <http://www.example.com/alice/photo.jpg> ;purpose=icon,  
3491 <http://www.example.com/alice/> ;purpose=info
```

3492 22.10 Contact

3493 The Contact header field provides a URL whose meaning depends on the the type of request or response it
3494 is in.

3495 Parameters defined for Contact include “q” and “expires”. Additional parameters may be defined in
3496 other specifications. Even if the “display-name” is empty, the “name-addr” form MUST be used if the
3497 “addr-spec” contains a comma, semicolon or question mark. Note that there may or may not be LWS
3498 between the display-name and the “<”.

3499 The Contact header field fulfills functionality similar to the Location header field in HTTP. However, the HTTP
3500 header only allows one address, unquoted. Since URIs can contain commas and semicolons as reserved characters,
3501 they can be mistaken for header or parameter delimiters, respectively. The current syntax corresponds to that for the
3502 To and From header, which also allows the use of display names.

3503 The compact form of the Contact header field is m (for “moved”).

3504 Examples:

```
3505 Contact: "Mr. Watson" <sip:watson@worchester.bell-telephone.com>  
3506         ;q=0.7; expires=3600,  
3507         "Mr. Watson" <mailto:watson@bell-telephone.com> ;q=0.1  
3508 m: <sip:bob@10.5.1.5>
```

3509 22.11 Content-Disposition

3510 The Content-Disposition header field describes how the message body or, in the case of multipart mes-
3511 sages, a message body part is to be interpreted by the UAC or UAS. The SIP header extends the MIME
3512 Content-Type (RFC 1806 [35]).

3513 The value “session” indicates that the body part describes a session, for either calls or early (pre-call)
3514 media. The value “render” indicates that the body part should be displayed or otherwise rendered to the
3515 user. For backward-compatibility, if the Content-Disposition header is not missing, bodies of Content-
3516 Type application/sdp imply the disposition “session”, while other content types imply “render”.

3517 The disposition type “icon” indicates that the body part contains an image suitable as an iconic repre-
3518 sentation of the caller or callee. The value “alert” indicates that the body part contains information, such as
3519 an audio clip, that should be rendered instead of ring tone.

3520 The handling parameter, handling-param, describes how the UAS should react if it receives a message
3521 body whose content type or disposition type it does not understand. The parameter has defined values of
3522 “optional” and “required”. If the handling parameter is missing, the value “required” is to be assumed.
3523 If this header field is missing, the MIME type determines the default content disposition. If there is none,
3524 “render” is assumed.

3525 Example:

```
3526 Content-Disposition: session
```

3527 22.12 Content-Encoding

3528 The Content-Encoding header field is used as a modifier to the “media-type”. When present, its value
3529 indicates what additional content codings have been applied to the entity-body, and thus what decoding
3530 mechanisms MUST be applied in order to obtain the media-type referenced by the Content-Type header

3531 field. Content-Encoding is primarily used to allow a body to be compressed without losing the identity of
3532 its underlying media type.

3533 If multiple encodings have been applied to an entity, the content codings MUST be listed in the order in
3534 which they were applied.

3535 All content-coding values are case-insensitive. The Internet Assigned Numbers Authority (IANA) acts
3536 as a registry for content-coding value tokens. See [H3.5] for a definition of the syntax for content-coding.

3537 Clients MAY apply content encodings to the body in requests. A server MAY apply content encodings to
3538 the bodies in responses. The server MUST only use encodings listed in the Accept-Encoding header in the
3539 request.

3540 The compact form of the Content-Encoding header field is e.

3541 Examples:

3542 Content-Encoding: gzip

3543 e: tar

3544 22.13 Content-Language

3545 See [H14.12].

3546 Example:

3547 Content-Language: fr

3548 22.14 Content-Length

3549 The Content-Length header field indicates the size of the message-body, in decimal number of octets, sent
3550 to the recipient.

3551 Applications SHOULD use this field to indicate the size of the message-body to be transferred, regardless
3552 of the media type of the entity. (The size of the message-body does *not* include the CRLF separating headers
3553 and body.) Any Content-Length greater than or equal to zero is a valid value. If no body is present in a
3554 message, then the Content-Length header field MUST be set to zero.

3555 The ability to omit Content-Length simplifies the creation of cgi-like scripts that dynamically generate re-
3556 sponses.

3557 The short form of the header is l.

3558 Examples:

3559 Content-Length: 349

3560 l: 173

3561 22.15 Content-Type

3562 The Content-Type header field indicates the media type of the message-body sent to the recipient. The
3563 “media-type” element is defined in [H3.7]. The Content-Type header MUST be present if the body is not
3564 empty. If the body is empty, and a Content-Length header is present, it indicates that the body of the
3565 specific type has zero length (for example, if it is an empty audio file).

3566 The short form of the header is c.

3567 Examples:

3568 Content-Type: application/sdp
3569 c: text/html; charset=ISO-8859-4

3570 22.16 CSeq

3571 A CSeq header field in a request contains a single decimal sequence number and the request method. The
3572 sequence number MUST be expressible as a 32-bit unsigned integer. The CSeq header serves to order
3573 transactions within a dialog, and to provide a means to uniquely identify transactions, and to differentiate
3574 between new requests and request retransmissions.

3575 Example:

3576 CSeq: 4711 INVITE

3577 22.17 Date

3578 The Date header field contains an RFC 1123 date (see [H14.18]). Note that unlike HTTP/1.1, SIP only
3579 supports the most recent RFC 1123 [36] formatting for dates. As in [H3.3], SIP restricts the timezone in
3580 SIP-date to "GMT", while RFC 1123 allows any timezone.

3581 The consistent use of GMT between Date, Expires and Retry-After headers allows implementation of simple
3582 clients that do not have a notion of absolute time.

3583 Note that rfc1123-date is case-sensitive.

3584 The Date header field reflects the time when the request or response is first sent.

3585 The Date header field can be used by simple end systems without a battery-backed clock to acquire a notion of
3586 current time. However, in its GMT-form, it requires clients to know their offset from GMT.

3587 Example:

3588 Date: Sat, 13 Nov 2001 23:29:00 GMT

3589 22.18 Error-Info

3590 The Error-Info header field provides a pointer to additional information about the error status response.

3591 SIP UACs have user interface capabilities ranging from pop up windows and audio on PC softclients to audio-
3592 only on "black" phones or endpoints connected via gateways. Rather than forcing a server generating an error to
3593 choose between sending an error status code with a detailed reason phrase and playing an audio recording, the
3594 Error-Info header field allows both to be sent. The UAC then has the choice of which error indicator to render to the
3595 caller.

3596 A UAC MAY treat a SIP URL in an Error-Info header field as if it were a Contact in a redirect and
3597 generate a new INVITE, resulting an a recorded announcement session being established. A non-SIP URL
3598 MAY be rendered to the user.

3599 Examples:

3600 SIP/2.0 404 The number you have dialed is not in service
3601 Error-Info: <sip:not-in-service-recording@atlanta.com>

3602 22.19 Expires

3603 The Expires header field gives the date and time after which the message (or content) expires. The precise
3604 meaning of this is method dependent.

3605 Note that the expiration time in an INVITE does *not* affect the duration of the actual session that may
3606 result from the invitation. Session description protocols may offer the ability to express time limits on the
3607 session duration, however.

3608 The value of this field can be either a date (see the Date header field) or an integer number of seconds
3609 (in decimal), measured from the receipt of the request. The latter approach is preferable for short durations,
3610 as it does not depend on clients and servers sharing a synchronized clock.

3611 Examples:

3612 Expires: Thu, 01 Dec 1994 16:00:00 GMT

3613 Expires: 5

3614 22.20 From

3615 The From header field indicates the initiator of the request. (Note that this may be different from the initiator
3616 of the dialog. Requests sent by the callee to the caller use the callee's address in the From header field.)

3617 The optional "display-name" is meant to be rendered by a human user interface. A system SHOULD
3618 use the display name "Anonymous" if the identity of the client is to remain hidden.

3619 Even if the "display-name" is empty, the "name-addr" form MUST be used if the "addr-spec" con-
3620 tains a comma, question mark, or semicolon. Syntax issues are discussed in Section 7.3.1.

3621 The short form of the header is f.

3622 Examples:

3623 From: "A. G. Bell" <sip:agb@bell-telephone.com> ;tag=a48s

3624 From: sip:+12125551212@server.phone2net.com;tag=887s

3625 f: Anonymous <sip:c8oqz84zk7z@privacy.org>;tag=hyh8

3626 22.21 In-Reply-To

3627 The In-Reply-To header field enumerates the Call-IDs that this call references or returns. These Call-IDs
3628 may have been cached by the client then included in this header in a return call.

3629 This allows automatic call distribution systems to route return calls to the originator of the first call and allows
3630 callees to filter calls, so that only calls that return calls they have originated will be accepted. This field is not a
3631 substitute for request authentication.

3632 Example:

3633 In-Reply-To: 70710@saturn.bell-tel.com, 17320@saturn.bell-tel.com

3634 22.22 Max-Forwards

3635 The Max-Forwards header field may be used with any SIP method to limit the number of proxies or gate-
3636 ways that can forward the request to the next downstream server. This can also be useful when the client is
3637 attempting to trace a request chain which appears to be failing or looping in mid-chain.

3638 The Max-Forwards value is a decimal integer indicating the remaining number of times this request
3639 message is allowed to be forwarded. This count is decremented by each server that forwards the request.

3640 Example:

3641 Max-Forwards: 6

3642 22.23 MIME-Version

3643 See [H19.4.1].

3644 Example:

3645 MIME-Version: 1.0

3646 22.24 Organization

3647 The Organization header field conveys the name of the organization to which the entity issuing the request
3648 or response belongs.

3649 The field MAY be used by client software to filter calls.

3650 Example:

3651 Organization: Boxes by Bob

3652 22.25 Priority

3653 The Priority header field indicates the urgency of the request as perceived by the client. Defined values
3654 include “non-urgent”, “normal”, “urgent”, and “emergency”.

3655 It is RECOMMENDED that the value of “emergency” only be used when life, limb or property are in
3656 imminent danger. Otherwise, there are no semantics defined for this header field.

3657 These are the values of RFC 2076 [37], with the addition of “emergency”.

3658 Examples:

3659 Subject: A tornado is heading our way!

3660 Priority: emergency

3661 OR

3662 Subject: Weekend plans

3663 Priority: non-urgent

3664 22.26 Proxy-Authenticate

3665 The Proxy-Authenticate header field consists of a challenge that indicates the authentication scheme and
3666 parameters applicable to the proxy for this Request-URI.

3667 The syntax for this header and use is defined in [H14.33]. See 20.2.3 for further details on its usage.

3668 Example:

3669 Proxy-Authenticate: Digest realm="Carrier SIP",
3670 domain="sip:ssl.carrier.com",
3671 nonce="f84f1cec41e6cbe5aea9c8e88d359",
3672 opaque="", stale=FALSE, algorithm=MD5

3673 22.27 Proxy-Authorization

3674 The Proxy-Authorization header field allows the client to identify itself (or its user) to a proxy which
3675 requires authentication. The Proxy-Authorization field value consists of credentials containing the authen-
3676 tication information of the user agent for the proxy and/or realm of the resource being requested.

3677 See [H14.34] for a definition of the syntax, and section 20.2.3 for a discussion of its usage.

3678 Note that this header field, along with Authorization breaks the general rules about multiple header
3679 fields. Although not a comma-separated list, this header field may be present multiple times, and MUST NOT
3680 be combined into a single header using the usual rules described in Section 7.3.1.

3681 Example:

3682 Proxy-Authorization: Digest username="Alice", realm="Atlanta ISP",
3683 nonce="c60f3082ee1212b402a21831ae",
3684 response="245f23415f11432b3434341c022"

3685 22.28 Proxy-Require

3686 The Proxy-Require header field is used to indicate proxy-sensitive features that must be supported by the
3687 proxy. See Section 22.30 for more details on the mechanics of this message and a usage example.

3688 Example:

3689 Proxy-Require: foo

3690 22.29 Record-Route

3691 The Record-Route is inserted by proxies in a request to force future requests in the session to route through
3692 the proxy.

3693 Details of its use with the Route header field are described in Section 16.4.

3694 Example:

3695 Record-Route: <sip:bob@biloxi.com;maddr=10.1.1.1>,
3696 <sip:bob@biloxi.com;maddr=10.2.1.1>

3697 22.30 Require

3698 The Require header field is used by clients to tell user agent servers about options that the client expects the
3699 server to support in order to properly process the request. Although an optional header, the Require MUST
3700 NOT be ignored if it is present.

3701 This is to make sure that the client-server interaction will proceed without delay when all options are understood
3702 by both sides, and only slow down if options are not understood (as in the example above). For a well-matched
3703 client-server pair, the interaction proceeds quickly, saving a round-trip often required by negotiation mechanisms.

3704 In addition, it also removes ambiguity when the client requires features that the server does not understand. Some
3705 features, such as call handling fields, are only of interest to end systems.

3706 Example:

3707 Require: com.example.billing

3708 22.31 Retry-After

3709 The Retry-After header field can be used with a 503 (Service Unavailable) response to indicate how long
3710 the service is expected to be unavailable to the requesting client and with a 404 (Not Found), 600 (Busy), or
3711 603 (Decline) response to indicate when the called party anticipates being available again. The value of this
3712 field can be either an SIP-date or an integer number of seconds (in decimal) after the time of the response.

3713 An optional comment can be used to indicate additional information about the time of callback. An
3714 optional "duration" parameter indicates how long the called party will be reachable starting at the initial
3715 time of availability. If no duration parameter is given, the service is assumed to be available indefinitely.

3716 Examples:

3717 Retry-After: Mon, 21 Jul 1997 18:48:34 GMT (I'm in a meeting)

3718 Retry-After: Mon, 01 Jan 9999 00:00:00 GMT

3719 (Dear John: Don't call me back, ever)

3720 Retry-After: Fri, 26 Sep 1997 21:00:00 GMT;duration=3600

3721 Retry-After: 120

3722 In the third example, the callee is reachable for one hour starting at 21:00 GMT. In the last example, the
3723 delay is 2 minutes.

3724 22.32 Route

3725 The Route is used to force routing for a request through the listed set of proxies. Details of its use with the
3726 Record-Route header field are described in Section 13.

3727 Example:

3728 Route: <sip:bob@biloxi.com;maddr=10.1.1.1>, <sip:bob@10.4.1.4>

3729 22.33 Server

3730 The Server header field contains information about the software used by the user agent server to handle the
3731 request. The syntax for this field is defined in [H14.38].

3732 Example:

3733 Server: HomeProxy v2

3734 22.34 Subject

3735 This header field provides a summary or indicates the nature of the call, allowing call filtering without having
3736 to parse the session description. (Note that the session description does not have to use the same subject
3737 indication as the invitation.)

3738 The short form of the header is s.

3739 Example:

3740 Subject: Need more boxes

3741 s: Tech Support

3742 22.35 Supported

3743 The Supported header field enumerates all the extensions supported by the client or server. If empty, it
3744 means that no extensions are supported.

3745 Example:

3746 Supported: foo, bar

3747 22.36 Timestamp

3748 The Timestamp header field describes when the client sent the request to the server. The use of the Times-
3749 tamp is covered in Section 13.

3750 Example:

3751 Timestamp: 54

3752 22.37 To

3753 The To header field specifies the logical recipient of the request.

3754 The optional “display-name” is meant to be rendered by a human-user interface. The “tag” parameter
3755 serves as a general mechanism to distinguish multiple instances of a user identified by a single SIP URL.

3756 See Section 13 for details of the “tag” parameter.

3757 Section 22.20 describes how To and From header fields are compared for the purpose of matching
3758 requests to dialogs. Even if the “display-name” is empty, the “name-addr” form MUST be used if the
3759 “addr-spec” contains a comma, question mark, or semicolon. Note that LWS is common, but **not** manda-
3760 tory between the display-name and the “<”.

3761 The short form of the header is t.

3762 The following are examples of valid To headers:

3763 To: The Operator <sip:operator@cs.columbia.edu>;tag=287447

3764 t: sip:+12125551212@server.phone2net.com

3765 22.38 Unsupported

3766 The Unsupported header field lists the features not supported by the server. See Section 22.30 for a usage
3767 example and motivation.

3768 Example:

3769 Unsupported: foo

3770 22.39 User-Agent

3771 The **User-Agent** header field contains information about the client user agent originating the request. The
3772 syntax and semantics are defined in [H14.43].

3773 Example:

```
3774 User-Agent: Softphone Beta1.5
```

3775 22.40 Via

3776 The **Via** field indicates the path taken by the request so far and indicate the path that should be followed in
3777 routing responses.

3778 The **Via** header field contains the transport protocol used to send the message, the client's host name or
3779 network address and, if not the default port number, the port number at which it wishes to receive responses.
3780 The **Via** header field can also contains parameters such as "maddr", "ttl", "received", and "branch" whose
3781 meaning and use are described in other sections.

3782 The short form of the header is **v**.

3783 Example:

```
3784 Via: SIP/2.0/UDP erlang.bell-telephone.com:5060
```

```
3785 Via: SIP/2.0/UDP 128.59.16.1:5060 ;received=128.59.19.3
```

3786 In this example, the message originated from a multi-homed host with two addresses, 128.59.16.1
3787 and 128.59.19.3. The sender guessed wrong as to which network interface would be used. Erlang.bell-
3788 telephone.com noticed the mismatch, and added a parameter to the previous hop's **Via** header field, contain-
3789 ing the address that the packet actually came from.

3790 Another example:

```
3791 Via: SIP/2.0/UDP first.example.com:4000;ttl=16
```

```
3792 ;maddr=224.2.0.1 ;branch=a7c6a8dlze.1
```

3793 22.41 Warning

3794 The **Warning** header field is used to carry additional information about the status of a response. **Warning**
3795 headers are sent with responses and contain a three digit warning code, host name, and warning text.

3796 The "warn-text" should be in a natural language that is most likely to be intelligible to the human user
3797 receiving the response. This decision can be based on any available knowledge, such as the location of the
3798 cache or user, the **Accept-Language** field in a request, or the **Content-Language** field in a response. The
3799 default language is i-default [38].

3800 The first digit of warning codes beginning with "3" indicates warnings specific to SIP.

3801 This is a list of the currently-defined "warn-code"s, each with a recommended warn-text in English, and
3802 a description of its meaning. Note that these warnings describe failures induced by the session description.

3803 Warnings 300 through 329 are reserved for indicating problems with keywords in the session description,
3804 330 through 339 are warnings related to basic network services requested in the session description, 370
3805 through 379 are warnings related to quantitative QoS parameters requested in the session description, and
3806 390 through 399 are miscellaneous warnings that do not fall into one of the above categories.

3807 **300 Incompatible network protocol:** One or more network protocols contained in the session description
3808 are not available.

3809 **301 Incompatible network address formats:** One or more network address formats contained in the ses-
3810 sion description are not available.

3811 **302 Incompatible transport protocol:** One or more transport protocols described in the session descrip-
3812 tion are not available.

3813 **303 Incompatible bandwidth units:** One or more bandwidth measurement units contained in the session
3814 description were not understood.

3815 **304 Media type not available:** One or more media types contained in the session description are not avail-
3816 able.

3817 **305 Incompatible media format:** One or more media formats contained in the session description are not
3818 available.

3819 **306 Attribute not understood:** One or more of the media attributes in the session description are not sup-
3820 ported.

3821 **307 Session description parameter not understood:** A parameter other than those listed above was not
3822 understood.

3823 **330 Multicast not available:** The site where the user is located does not support multicast.

3824 **331 Unicast not available:** The site where the user is located does not support unicast communication (usu-
3825 ally due to the presence of a firewall).

3826 **370 Insufficient bandwidth:** The bandwidth specified in the session description or defined by the media
3827 exceeds that known to be available.

3828 **399 Miscellaneous warning:** The warning text can include arbitrary information to be presented to a hu-
3829 man user, or logged. A system receiving this warning **MUST NOT** take any automated action.

3830 1xx and 2xx have been taken by HTTP/1.1.

3831 If the warning is caused by the session description, the status response **SHOULD** include a session de-
3832 scription similar to that included in **OPTIONS** responses indicating the capabilities of the UAS. Additional
3833 "warn-code"s, as in the example below, can be defined through IANA.

3834 Examples:

3835 Warning: 307 isi.edu "Session parameter 'foo' not understood"
3836 Warning: 301 isi.edu "Incompatible network address type 'E.164'"

3837 22.42 WWW-Authenticate

3838 The WWW-Authenticate header field consists of a challenge that indicates the authentication scheme and
3839 parameters applicable for this Request-URI.

3840 The syntax for this header and use is defined in [H14.47]. See 20.2.2 for further details on its usage.

3841 Example:

3842 WWW-Authenticate: Digest realm="Bob's Friends",
3843 domain="sip:boxesbybob.com",
3844 nonce="f84f1cec41e6cbe5aea9c8e88d359",
3845 opaque="", stale=FALSE, algorithm=MD5

3846 **23 Response Codes**

3847 The response codes are consistent with, and extend, HTTP/1.1 response codes. Not all HTTP/1.1 response
3848 codes are appropriate, and only those that are appropriate are given here. Other HTTP/1.1 response codes
3849 SHOULD NOT be used. Response codes not defined by HTTP/1.1 have codes x80 upwards to avoid clashes
3850 with future HTTP response codes. Also, SIP defines a new class, 6xx. The default behavior for unknown
3851 response codes is given for each category of codes.

3852 **23.1 Provisional 1xx**

3853 Provisional responses indicate that the server or proxy contacted is performing some further action and does
3854 not yet have a definitive response. A server typically sends a 1xx response if it expects to take more than
3855 200 ms to obtain a final response. Note that 1xx responses are not transmitted reliably, that is, they do not
3856 cause the client to send an ACK.

3857 Provisional (1xx) responses MAY contain message bodies, including session descriptions.

3858 Provisional responses are also known as informational responses.

3859 **23.1.1 100 Trying**

3860 This response indicates that the request has been received by the next hop server and that some unspeci-
3861 fied action is being taken on behalf of this call (e.g., a database is being consulted). This response stops
3862 retransmissions of an INVITE by a UAC.

3863 **23.1.2 180 Ringing**

3864 The user agent receiving the INVITE is trying to alert the user. This response MAY be used to initiate local
3865 ringback.

3866 **23.1.3 181 Call Is Being Forwarded**

3867 A proxy server MAY use this status code to indicate that the call is being forwarded to a different set of
3868 destinations.

3869 **23.1.4 182 Queued**

3870 The called party is temporarily unavailable, but the callee has decided to queue the call rather than reject it.
3871 When the callee becomes available, it will return the appropriate final status response. The reason phrase
3872 MAY give further details about the status of the call, e.g., "5 calls queued; expected waiting time is 15
3873 minutes". The server MAY issue several 182 responses to update the caller about the status of the queued
3874 call.

3875 **23.1.5 183 Session Progress**

3876 The 183 (Session Progress) response is used to convey information about the progress of the call which is
3877 not otherwise classified. The Reason-Phrase, header fields, or message body MAY be used to convey more
3878 details about the call progress.

3879 **23.2 Successful 2xx**

3880 The request was successful.

3881 **23.2.1 200 OK**

3882 The request has succeeded. The information returned with the response depends on the method used in the
3883 request.

3884 **23.3 Redirection 3xx**

3885 3xx responses give information about the user's new location, or about alternative services that might be
3886 able to satisfy the call.

3887 **23.3.1 300 Multiple Choices**

3888 The address in the request resolved to several choices, each with its own specific location, and the user (or
3889 user agent) can select a preferred communication end point and redirect its request to that location.

3890 The response MAY include a message body containing a list of resource characteristics and location(s)
3891 from which the user or user agent can choose the one most appropriate, if allowed by the Accept request
3892 header.

3893 The choices SHOULD also be listed as Contact fields (Section 22.10). Unlike HTTP, the SIP response
3894 MAY contain several Contact fields or a list of addresses in a Contact field. User agents MAY use the
3895 Contact header field value for automatic redirection or MAY ask the user to confirm a choice. However, this
3896 specification does not define any standard for such automatic selection.

3897 This status response is appropriate if the callee can be reached at several different locations and the server cannot
3898 or prefers not to proxy the request.

3899 **23.3.2 301 Moved Permanently**

3900 The user can no longer be found at the address in the Request-URI and the requesting client SHOULD retry
3901 at the new address given by the Contact header field (Section 22.10). The caller SHOULD update any local
3902 directories, address books and user location caches with this new value and redirect future requests to the
3903 address(es) listed.

3904 **23.3.3 302 Moved Temporarily**

3905 The requesting client SHOULD retry the request at the new address(es) given by the Contact header field
3906 (Section 22.10). The Request-URI of the new request uses the value of the Contact header in the response.
3907 The new request can take two different forms. In the first approach, the To, From, Call-ID, and CSeq
3908 header fields in the new request are the same as in the original request, with a new branch identifier in the

3909 Via header field. Proxies MUST follow this behavior and UACs MAY. In the second approach, UAs MAY
3910 also use the Contact information for the To header field, as well as a new Call-ID value.

3911 The duration of the redirection can be indicated through an Expires (Section 22.19) header. If there is
3912 no explicit expiration time, the address is only valid for this call and MUST NOT be cached for future calls.

3913 **23.3.4 305 Use Proxy**

3914 The requested resource MUST be accessed through the proxy given by the Contact field. The Contact
3915 field gives the URI of the proxy. The recipient is expected to repeat this single request via the proxy. 305
3916 responses MUST only be generated by user agent servers.

3917 **23.3.5 380 Alternative Service**

3918 The call was not successful, but alternative services are possible. The alternative services are described in
3919 the message body of the response. Formats for such bodies are not defined here, and may be the subject of
3920 future standardization.

3921 **23.4 Request Failure 4xx**

3922 4xx responses are definite failure responses from a particular server. The client SHOULD NOT retry the
3923 same request without modification (e.g., adding appropriate authorization). However, the same request to a
3924 different server might be successful.

3925 **23.4.1 400 Bad Request**

3926 The request could not be understood due to malformed syntax. The Reason-Phrase SHOULD identify the
3927 syntax problem in more detail, e.g., "Missing Call-ID header".

3928 **23.4.2 401 Unauthorized**

3929 The request requires user authentication. This response is issued by user agent servers and registrars, while
3930 407 (Proxy Authentication Required) is used by proxy servers.

3931 **23.4.3 402 Payment Required**

3932 Reserved for future use.

3933 **23.4.4 403 Forbidden**

3934 The server understood the request, but is refusing to fulfill it. Authorization will not help, and the request
3935 SHOULD NOT be repeated.

3936 **23.4.5 404 Not Found**

3937 The server has definitive information that the user does not exist at the domain specified in the Request-
3938 URI. This status is also returned if the domain in the Request-URI does not match any of the domains
3939 handled by the recipient of the request.

3940 **23.4.6 405 Method Not Allowed**

3941 The method specified in the Request-Line is not allowed for the address identified by the Request-URI.
3942 The response MUST include an Allow header field containing a list of valid methods for the indicated address.

3943 **23.4.7 406 Not Acceptable**

3944 The resource identified by the request is only capable of generating response entities which have content
3945 characteristics not acceptable according to the accept headers sent in the request.

3946 **23.4.8 407 Proxy Authentication Required**

3947 This code is similar to 401 (Unauthorized), but indicates that the client MUST first authenticate itself with
3948 the proxy. SIP access authentication is explained in section 20 and 20.2.3.

3949 This status code can be used for applications where access to the communication channel (e.g., a tele-
3950 phony gateway) rather than the callee requires authentication.

3951 **23.4.9 408 Request Timeout**

3952 The server could not produce a response within a suitable amount of time, for example, if it could not
3953 determine the location of the user in time. The client MAY repeat the request without modifications at any
3954 later time.

3955 **23.4.10 410 Gone**

3956 The requested resource is no longer available at the server and no forwarding address is known. This
3957 condition is expected to be considered permanent. If the server does not know, or has no facility to determine,
3958 whether or not the condition is permanent, the status code 404 (Not Found) SHOULD be used instead.

3959 **23.4.11 413 Request Entity Too Large**

3960 The server is refusing to process a request because the request entity is larger than the server is willing or
3961 able to process. The server MAY close the connection to prevent the client from continuing the request.

3962 If the condition is temporary, the server SHOULD include a Retry-After header field to indicate that it is
3963 temporary and after what time the client MAY try again.

3964 **23.4.12 414 Request-URI Too Long**

3965 The server is refusing to service the request because the Request-URI is longer than the server is willing to
3966 interpret.

3967 **23.4.13 415 Unsupported Media Type**

3968 The server is refusing to service the request because the message body of the request is in a format not sup-
3969 ported by the server for the requested method. The server SHOULD return a list of acceptable formats using
3970 the Accept, Accept-Encoding and Accept-Language header fields. UAC processing of this response is
3971 described in Section 8.1.3.4.

3972 **23.4.14 420 Bad Extension**

3973 The server did not understand the protocol extension specified in a **Proxy-Require** (Section 22.28) or **Re-**
3974 **quire** (Section 22.30) header field. The server **SHOULD** include a list of the unsupported extensions in an
3975 **Unsupported** header in the response. UAC processing of this response is described in Section 8.1.3.4.

3976 **23.4.15 421 Extension Required**

3977 The UAS needs a particular extension to process the request, but this extension is not listed in a **Supported**
3978 header in the request. Responses with this status code **MUST** contain a **Require** header listing the required
3979 extensions.

3980 In general, a UAS **SHOULD NOT** use this response when it wishes to apply an extension to a request. The
3981 end result will often be no service at all, and a break in interoperability. Rather, servers **SHOULD** process the
3982 request using baseline SIP capabilities and any extensions supported by the client.

3983 **23.4.16 480 Temporarily Unavailable**

3984 The callee's end system was contacted successfully but the callee is currently unavailable (e.g., not logged
3985 in, logged in in such a manner as to preclude communication with the callee or activated the "do not disturb"
3986 feature). The response **MAY** indicate a better time to call in the **Retry-After** header. The user could also be
3987 available elsewhere (unbeknownst to this host). The reason phrase **SHOULD** indicate a more precise cause
3988 as to why the callee is unavailable. This value **SHOULD** be setable by the user agent. Status 486 (Busy Here)
3989 **MAY** be used to more precisely indicate a particular reason for the call failure.

3990 This status is also returned by a redirect server that recognizes the user identified by the **Request-URI**,
3991 but does not currently have a valid forwarding location for that user.

3992 **23.4.17 481 Call/Transaction Does Not Exist**

3993 This status indicates that the UAS received a request that does not match any existing dialog or transaction.

3994 **23.4.18 482 Loop Detected**

3995 The server has detected a loop (Section 3).

3996 **23.4.19 483 Too Many Hops**

3997 The server received a request that contains a **Max-Forwards** (Section 22.22) header with the value zero.

3998 **23.4.20 484 Address Incomplete**

3999 The server received a request with a **Request-URI** that was incomplete. Additional information **SHOULD**
4000 be provided.

4001 This status code allows overlapped dialing. With overlapped dialing, the client does not know the length of the
4002 dialing string. It sends strings of increasing lengths, prompting the user for more input, until it no longer receives a
4003 484 status response.

4004 23.4.21 485 Ambiguous

4005 The callee address provided in the request was ambiguous. The response MAY contain a listing of possible
4006 unambiguous addresses in **Contact** headers.

4007 Revealing alternatives can infringe on privacy concerns of the user or the organization. It MUST be
4008 possible to configure a server to respond with status 404 (Not Found) or to suppress the listing of possible
4009 choices if the request address was ambiguous.

4010 Example response to a request with the URL `lee@example.com`:

4011 485 Ambiguous SIP/2.0

4012 Contact: Carol Lee <sip:carol.lee@example.com>

4013 Contact: Ping Lee <sip:p.lee@example.com>

4014 Contact: Lee M. Foote <sip:lee.foote@example.com>

4015 Some email and voice mail systems provide this functionality. A status code separate from 3xx is used since
4016 the semantics are different: for 300, it is assumed that the same person or service will be reached by the choices
4017 provided. While an automated choice or sequential search makes sense for a 3xx response, user intervention is
4018 required for a 485 response.

4019 23.4.22 486 Busy Here

4020 The callee's end system was contacted successfully but the callee is currently not willing or able to take
4021 additional calls at this end system. The response MAY indicate a better time to call in the **Retry-After**
4022 header. The user could also be available elsewhere, such as through a voice mail service. Status 600 (Busy
4023 Everywhere) SHOULD be used if the client knows that no other end system will be able to accept this call.

4024 23.4.23 487 Request Terminated

4025 The request was terminated by a BYE or CANCEL request. This response is never returned for a CANCEL
4026 request itself.

4027 23.4.24 488 Not Acceptable Here

4028 The response has the same meaning as 606 (Not Acceptable), but only applies to the specific entity addressed
4029 by the Request-URI and the request may succeed elsewhere.

4030 23.5 Server Failure 5xx

4031 5xx responses are failure responses given when a server itself has erred.

4032 23.5.1 500 Server Internal Error

4033 The server encountered an unexpected condition that prevented it from fulfilling the request. The client MAY
4034 display the specific error condition, and MAY retry the request after several seconds.

4035 If the condition is temporary, the server MAY indicate when the client may retry the request using the
4036 **Retry-After** header.

4037 **23.5.2 501 Not Implemented**

4038 The server does not support the functionality required to fulfill the request. This is the appropriate response
4039 when a UAS does not recognize the request method and is not capable of supporting it for any user. (Proxies
4040 forward all requests regardless of method.)

4041 **23.5.3 502 Bad Gateway**

4042 The server, while acting as a gateway or proxy, received an invalid response from the downstream server it
4043 accessed in attempting to fulfill the request.

4044 **23.5.4 503 Service Unavailable**

4045 The server is currently unable to handle the request due to a temporary overloading (i.e., congestion) or
4046 maintenance of the server. The implication is that this is a temporary condition which will be alleviated
4047 after some delay. If known, the length of the delay MAY be indicated in a **Retry-After** header. If no **Retry-
4048 After** is given, the client MUST handle the response as it would for a 500 response.

4049 A client (proxy or UAC) receiving a 503 SHOULD attempt to forward the request to an alternate server. It
4050 SHOULD NOT forward any other requests to that server for the duration specified in the **Retry-After** header,
4051 if present.

4052 Note: The existence of the 503 status code does not imply that a server has to use it when becoming
4053 overloaded. Some servers MAY wish to simply refuse the connection.

4054 **23.5.5 504 Server Time-out**

4055 The server did not receive a timely response from the server (e.g., a location server) it accessed in attempting
4056 to process the request. Note that 408 (Request Timeout) should be used if there was no response within the
4057 period specified in the **Expires** header field from the upstream server.

4058 **23.5.6 505 Version Not Supported**

4059 The server does not support, or refuses to support, the SIP protocol version that was used in the request
4060 message. The server is indicating that it is unable or unwilling to complete the request using the same major
4061 version as the client, other than with this error message. The response MAY contain an entity describing why
4062 that version is not supported and what other protocols are supported by that server. The format for such an
4063 entity is not defined here and may be the subject of future standardization.

4064 **23.5.7 513 Message Too Large**

4065 The server was unable to process the request since the message length exceeded its capabilities.

4066 **23.6 Global Failures 6xx**

4067 6xx responses indicate that a server has definitive information about a particular user, not just the particular
4068 instance indicated in the **Request-URI**.

4069 **23.6.1 600 Busy Everywhere**

4070 The callee's end system was contacted successfully but the callee is busy and does not wish to take the call
4071 at this time. The response MAY indicate a better time to call in the **Retry-After** header. If the callee does
4072 not wish to reveal the reason for declining the call, the callee uses status code 603 (Decline) instead. This
4073 status response is returned only if the client knows that no other end point (such as a voice mail system) will
4074 answer the request. Otherwise, 486 (Busy Here) should be returned.

4075 **23.6.2 603 Decline**

4076 The callee's machine was successfully contacted but the user explicitly does not wish to or cannot partici-
4077 pate. The response MAY indicate a better time to call in the **Retry-After** header.

4078 **23.6.3 604 Does Not Exist Anywhere**

4079 The server has authoritative information that the user indicated in the **Request-URI** does not exist anywhere.

4080 **23.6.4 606 Not Acceptable**

4081 The user's agent was contacted successfully but some aspects of the session description such as the requested
4082 media, bandwidth, or addressing style were not acceptable.

4083 A 606 (Not Acceptable) response means that the user wishes to communicate, but cannot adequately sup-
4084 port the session described. The 606 (Not Acceptable) response MAY contain a list of reasons in a **Warning**
4085 header field describing why the session described cannot be supported. Reasons are listed in Section 22.41.
4086 It is hoped that negotiation will not frequently be needed, and when a new user is being invited to join an
4087 already existing conference, negotiation may not be possible. It is up to the invitation initiator to decide
4088 whether or not to act on a 606 (Not Acceptable) response.

4089 **24 Locating a SIP Server**

4090 NOTE: Usage of SRV records is still under discussion with IESG, and therefore this section is likely to change
4091 in subsequent versions of bis.

4092 The SIP URI provides a way to identify a communications resource. For this URI to be useful in a SIP
4093 element, a mechanism is necessary to take this URI and determine the IP address, port, and transport of one
4094 or more servers that message destined for this URI should be sent to. We refer to the combination of an
4095 IP address, port, and transport as a *next hop*. There are two ways to determine the next hop. The next hop
4096 can be configured to be the same for all URIs. In this case, the next hop is referred to as a *outbound proxy*.
4097 This is commonly used in a user agent which is required to send all requests to a specific server for policy
4098 processing or firewall traversal, for example. The outbound proxy can be configured by any mechanism,
4099 including DHCP [39].

4100 When the next hop is not configured, a mechanism is needed to determine one or more next hops from
4101 the URI. Section 24.1 provides an algorithm which can be used to determine an ordered list of next hops.
4102 Typically, the URI that is used is from the **Request-URI** of a request, in order to determine where to send
4103 that request. However, in certain circumstances (which are documented in Section 19.2.2), a URI may have
4104 been extracted from a response in order to determine where to send the response.

4105 Once the ordered list of next hops is computed, they are used according to the procedures of Section
4106 24.2.

4107 **24.1 Computing the List of Next Hops**

4108 The algorithm for computing the list of next hops begins by setting three variables. The first variable is
4109 called the *target address*. The target address **MUST** be set to the contents of the `maddr` parameter of the
4110 URI, if present. If not present, it **MUST** be set to the `host` element of the URI. The next variable is called the
4111 *target port*. The target port **MUST** be set to the `port` element of the URI if present, else the target port **MUST**
4112 remain empty. The target transport **MUST** be set to the `headertransport` element of the URI if present, else
4113 the target transport **MUST** remain empty.

4114 The algorithm begins by examining the target address. If it contains a numeric IP address, the procedures
4115 of Section 24.1.1 **MUST** be followed. Otherwise, the target transport is examined. If it is empty, and the
4116 target port is either empty or contains a value of 5060, the procedures of Section 24.1.2 **MUST** be followed.
4117 If the target transport is not empty, and the target port is empty, the procedures of Section 24.1.2 **MUST** be
4118 followed if the target transport is UDP. If the target transport and target port are not empty, but the target
4119 port contains the default port for the target transport (5060 for UDP, TCP, and SCTP, 5061 for TLS), the
4120 procedures of Section 24.1.2 **MUST** also be followed. Otherwise, the procedures of Section 24.1.3 **MUST** be
4121 followed. Effectively, this case occurs when the target port and target transport don't "match", taking into
4122 account their defaults if empty.

4123 **24.1.1 Numeric Destination Address**

4124 The addresses of the next hops are all the same, and **MUST** be equal to the value of the target address.

4125 If the target transport is specified, and the element supports that transport, there is only a single next
4126 hop, using the target transport. If the target transport is not specified, the number of next hops is equal to
4127 the number of transports the element supports. The first next hop **MUST** be UDP, and the ordering of the
4128 remaining transports is at the discretion of the element.

4129 For each next hop, the port number is equal to the target port, if specified, otherwise the default port for
4130 that transport of that next hop.

4131 For example, consider the SIP URI `sip:joe@1.2.3.4` present in the `Request-URI` of a request. A
4132 UAC wishes to use this URI to determine the set of next hops. The UAC supports UDP and TLS. It applies
4133 the algorithm in this section, and ends up with the following ordered list of IP address, port, transport:

4134 {1.2.3.4, 5060, UDP}

4135 {1.2.3.4, 5061, TLS}

4136 **24.1.2 SRV Resolution of Host Name**

4137 DNS SRV records are retrieved according to RFC 2782 [40]. The service identifier for DNS SRV records is
4138 `._sip`. If the target transport is not empty, only records for that transport are retrieved. (If the element does
4139 not support the transport specified, the lookup fails.) If the target transport is empty, the element retrieves
4140 records for all transport protocols it supports. The results of all queries are merged and then sorted according
4141 to priority, independent of the transport protocol. If this list is empty, follow the procedure in Section 24.1.3.

4142 Note that the behavior above differs slightly from that described in RFC 2782. There, A records are
4143 consulted if the query for one transport protocol fails; here, we only abandon the SRV lookup if none of the

4144 transport protocols supported by the client yield an answer.

4145 Clients MUST NOT cache query results except according to the rules in RFC 1035 [41].

4146 **24.1.3 Address Record Resolution of Host Name**

4147 When the target address is not a numeric IP, and there is a target port which does not match the default port
4148 for the target transport, SRV records are not used. This is because SRV will normally provide ports, so if
4149 one is provided that is not a default, this would seem to imply the the URL is trying to explicitly identify the
4150 destination, rather than using SRV.

4151 In this case, the client queries the DNS server for address records for the destination address. Address
4152 records include A RR's, AAAA RR's, or other similar records, chosen according to the client's network
4153 protocol capabilities.

4154 The DNS address records are kept sorted in the order returned by the DNS server. For each address, the
4155 port is set to the target port. For each address, the transport is set to the target transport if not empty, other-
4156 wise, the target transport MUST be UDP for the first address, and is at the discretion of the implementation
4157 for the others.

4158 OPEN ISSUE #221: Selection of transports for the case when multiple A records are returned requires more
4159 work.

4160 Clients MUST NOT cache query results except according to the rules in RFC 1035 [41].

4161 **24.2 Contacting the Next Hops**

4162 The algorithms of the previous section will result in an ordered list of next hops. This section describes how
4163 that list is used.

4164 If the ordered list was obtained through SRV, servers are contacted as specified in the "Usage rules"
4165 section of RFC 2782 [40], which describes procedures for using the weight field to randomly select servers
4166 amongst those of equal priority.

4167 The SIP element takes the ordered list, and it tries to contact each next hop in turn, until a server
4168 responds. If contacting a next hop results in a failure, as defined in the next paragraph, the element moves
4169 to the next next hop in the list, until the list is exhausted. If the list is exhausted, then the element gives up.

4170 Failures SHOULD be detected through network failure indications or timeouts. If the element sending the
4171 message is a client sending a request using a client transaction, the client transaction will report any transport
4172 layer failures. If the element sending the message is a client sending a request directly to the transport layer,
4173 the transport layer will report any failures (See Section 19.4). In either case, the client SHOULD try the
4174 next address. This will involve creating a new client transaction for it in the former case. The new request
4175 MUST have a new branch ID in the Via header. Note also that the new destination might be with a different
4176 transport, which might require a change in other parts of the Via header.

4177 Response failures are handled by the transport layer itself, which may retry the response to the next next
4178 hop. See Section 19.2.2.

4179 Failures can be detected through timeouts only if the element is a client sending a request through the
4180 client transaction. In that case, if a timeout is reported by the client transaction, the client SHOULD try the
4181 next next hop in the list.

4182 OPEN ISSUE #219: It might be easier to encapsulate the SRV processing in one place, at the transport layer,
4183 rather than the behavior being dependent on client v. server. This can only be done if merging of srv records across
4184 transports is deprecated, along with failures based on timeouts.

4185 Once a next hop is successfully contacted, that same next hop address **MUST** be used for all subsequent
 4186 messages that share the same **Call-ID**. More specifically, once a request is delivered successfully to a par-
 4187 ticular next hop, all subsequent requests with the same **Call-ID** **MUST** be delivered to that next hop. Once a
 4188 response is delivered successfully to a particular next hop, all subsequent responses with the same **Call-ID**
 4189 **MUST** be delivered to that next hop. However, if that next hop fails, the selection algorithms **MUST** be re-run
 4190 for the top.

4191 This is a change from RFC2543, which only used the same address for requests within a transaction. Broadening
 4192 the scope to **Call-ID** helps, for example, ensure that requests with credentials after a challenge are delivered to the
 4193 same server that issued the challenge.

4194 A stateless proxy can accomplish this, for example, by using the modulo N of a hash of the **Call-ID**
 4195 value as the uniform random number described in the weighting algorithm of RFC 2782 [40]. Here, N is
 4196 the sum of weights within the priority class.

4197 **OPEN ISSUE #220:** This stateless selection algorithm doesn't work if there are failures.

4198 25 Examples

4199 In the following examples, we often omit the message body and the corresponding **Content-Length** and
 4200 **Content-Type** headers for brevity.

4201 25.1 Registration

4202 Bob registers on start-up. The message flow is shown in Figure 9.

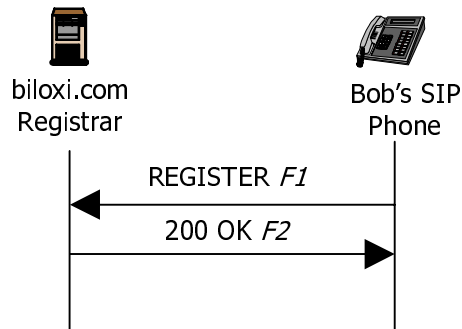


Figure 9: SIP Registration Example

4203
 4204 F1 REGISTER Bob -> Registrar
 4205
 4206 REGISTER sip:registrar.biloxi.com
 4207 Via: SIP/2.0/UDP 10.4.1.4:5060
 4208 To: Bob <sip:bob@biloxi.com>
 4209 From: Bob <sip:bob@biloxi.com>;tag=456248

4210 Call-ID: 843817637684230@phone21.bboxesbybob.com
4211 CSeq: 1826 REGISTER
4212 Contact: <sip:bob@10.4.1.4>
4213 Expires: 7200
4214 Contact-Length: 0

4215 The registration expires after two hours. The registrar responds with a 200 OK:

4216
4217 F2 200 OK Registrar -> Bob
4218
4219 SIP/2.0 200 OK
4220 Via: SIP/2.0/UDP 10.4.1.4:5060
4221 To: Bob <sip:bob@biloxi.com>
4222 From: Bob <sip:bob@biloxi.com>;tag=456248
4223 Call-ID: 843817637684230@phone21.bboxesbybob.com
4224 CSeq: 1826 REGISTER
4225 Contact: <sip:bob@10.4.1.4>
4226 Expires: 7200
4227 Contact-Length: 0
4228

4229 25.2 Session Setup

4230 This example contains the full details of the example session setup in Section 4. The message flow is shown
4231 in Figure 1.

4232
4233 F1 INVITE Alice -> atlanta.com proxy
4234
4235 INVITE sip:bob@biloxi.com SIP/2.0
4236 Via: SIP/2.0/UDP 10.1.3.3:5060
4237 To: Bob <sip:bob@biloxi.com>
4238 From: Alice <sip:alice@atlanta.com>;tag=1928301774
4239 Call-ID: a84b4c76e66710@10.1.3.3
4240 CSeq: 314159 INVITE
4241 Contact: <sip:alice@10.1.3.3>
4242 Content-Type: application/sdp
4243 Contact-Length: 142
4244
4245 (Alice's SDP not shown)

4246
4247 F2 100 Trying atlanta.com proxy -> Alice
4248

4249 SIP/2.0 100 Trying
4250 Via: SIP/2.0/UDP 10.1.3.3:5060
4251 To: Bob <sip:bob@biloxi.com>
4252 From: Alice <sip:alice@atlanta.com>;tag=1928301774
4253 Call-ID: a84b4c76e66710@10.1.3.3
4254 CSeq: 314159 INVITE
4255 Contact-Length: 0

4256
4257 F3 INVITE atlanta.com proxy -> biloxi.com proxy
4258
4259 INVITE sip:bob@biloxi.com SIP/2.0
4260 Via: SIP/2.0/UDP 10.1.1.1:5060;branch=77ef4c2312983.1
4261 Via: SIP/2.0/UDP 10.1.3.3:5060
4262 To: Bob <sip:bob@biloxi.com>
4263 From: Alice <sip:alice@atlanta.com>;tag=1928301774
4264 Call-ID: a84b4c76e66710@10.1.3.3
4265 CSeq: 314159 INVITE
4266 Contact: <sip:alice@10.1.3.3>
4267 Content-Type: application/sdp
4268 Contact-Length: 142
4269
4270 (Alice's SDP not shown)

4271
4272 F4 100 Trying biloxi.com proxy -> atlanta.com proxy
4273
4274 SIP/2.0 100 Trying
4275 Via: SIP/2.0/UDP 10.1.1.1:5060;branch=77ef4c2312983.1
4276 Via: SIP/2.0/UDP 10.1.3.3:5060
4277 To: Bob <sip:bob@biloxi.com>
4278 From: Alice <sip:alice@atlanta.com>;tag=1928301774
4279 Call-ID: a84b4c76e66710@10.1.3.3
4280 CSeq: 314159 INVITE
4281 Contact-Length: 0

4282
4283 F5 INVITE biloxi.com proxy -> Bob
4284
4285 INVITE sip:bob@10.4.1.4 SIP/2.0
4286 Via: SIP/2.0/UDP 10.2.1.1:5060;branch=4b43c2ff8.1
4287 Via: SIP/2.0/UDP 10.1.1.1:5060;branch=77ef4c2312983.1
4288 Via: SIP/2.0/UDP 10.1.3.3:5060
4289 To: Bob <sip:bob@biloxi.com>
4290 From: Alice <sip:alice@atlanta.com>;tag=1928301774

4291 Call-ID: a84b4c76e66710@10.1.3.3
4292 CSeq: 314159 INVITE
4293 Contact: <sip:alice@10.1.3.3>
4294 Content-Type: application/sdp
4295 Contact-Length: 142
4296
4297 (Alice's SDP not shown)

4298
4299 F6 180 Ringing Bob -> biloxi.com proxy
4300
4301 SIP/2.0 180 Ringing
4302 Via: SIP/2.0/UDP 10.2.1.1:5060;branch=4b43c2ff8.1
4303 Via: SIP/2.0/UDP 10.1.1.1:5060;branch=77ef4c2312983.1
4304 Via: SIP/2.0/UDP 10.1.3.3:5060
4305 To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
4306 From: Alice <sip:alice@atlanta.com>;tag=1928301774
4307 Call-ID: a84b4c76e66710@10.1.3.3
4308 CSeq: 314159 INVITE
4309 Contact-Length: 0

4310
4311 F7 180 Ringing biloxi.com proxy -> atlanta.com proxy
4312
4313 SIP/2.0 180 Ringing
4314 Via: SIP/2.0/UDP 10.1.1.1:5060;branch=77ef4c2312983.1
4315 Via: SIP/2.0/UDP 10.1.3.3:5060
4316 To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
4317 From: Alice <sip:alice@atlanta.com>;tag=1928301774
4318 Call-ID: a84b4c76e66710@10.1.3.3
4319 CSeq: 314159 INVITE
4320 Contact-Length: 0

4321
4322 F8 180 Ringing atlanta.com proxy -> Alice
4323
4324 SIP/2.0 180 Ringing
4325 Via: SIP/2.0/UDP 10.1.3.3:5060
4326 To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
4327 From: Alice <sip:alice@atlanta.com>;tag=1928301774
4328 Call-ID: a84b4c76e66710@10.1.3.3
4329 CSeq: 314159 INVITE
4330 Contact-Length: 0

4331

4332 F9 200 OK Bob -> biloxi.com proxy
4333
4334 SIP/2.0 200 OK
4335 Via: SIP/2.0/UDP 10.2.1.1:5060;branch=4b43c2ff8.1
4336 Via: SIP/2.0/UDP 10.1.1.1:5060;branch=77ef4c2312983.1
4337 Via: SIP/2.0/UDP 10.1.3.3:5060
4338 To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
4339 From: Alice <sip:alice@atlanta.com>;tag=1928301774
4340 Call-ID: a84b4c76e66710@10.1.3.3
4341 CSeq: 314159 INVITE
4342 Contact: <sip:bob@10.4.1.4>
4343 Content-Type: application/sdp
4344 Contact-Length: 131
4345
4346 (Bob's SDP not shown)

4347
4348 F10 200 OK biloxi.com proxy -> atlanta.com proxy
4349
4350 SIP/2.0 200 OK
4351 Via: SIP/2.0/UDP 10.1.1.1:5060;branch=77ef4c2312983.1
4352 Via: SIP/2.0/UDP 10.1.3.3:5060
4353 To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
4354 From: Alice <sip:alice@atlanta.com>;tag=1928301774
4355 Call-ID: a84b4c76e66710@10.1.3.3
4356 CSeq: 314159 INVITE
4357 Contact: <sip:bob@10.4.1.4>
4358 Content-Type: application/sdp
4359 Contact-Length: 131
4360
4361 (Bob's SDP not shown)

4362
4363 F11 200 OK atlanta.com proxy -> Alice
4364
4365 SIP/2.0 200 OK
4366 Via: SIP/2.0/UDP 10.1.3.3:5060
4367 To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
4368 From: Alice <sip:alice@atlanta.com>;tag=1928301774
4369 Call-ID: a84b4c76e66710@10.1.3.3
4370 CSeq: 314159 INVITE
4371 Contact: <sip:bob@10.4.1.4>
4372 Content-Type: application/sdp
4373 Contact-Length: 131
4374

4375 (Bob's SDP not shown)

4376

4377 F12 ACK Alice -> Bob

4378

4379 ACK sip:bob@10.4.1.4 SIP/2.0

4380 Via: SIP/2.0/UDP 10.1.3.3:5060

4381 To: Bob <sip:bob@biloxi.com>;tag=a6c85cf

4382 From: Alice <sip:alice@atlanta.com>;tag=1928301774

4383 Call-ID: a84b4c76e66710@10.1.3.3

4384 CSeq: 314159 ACK

4385 Contact-Length: 0

4386 The media session between Alice and Bob is now established.

4387 Bob hangs up first. Note that Bob's SIP phone maintains its own CSeq numbering space, which, in this
4388 example, begins with 231. Also note that since Bob is making the request, the To and From URLs and tags
4389 have been swapped.

4390

4391 F13 BYE Bob -> Alice

4392

4393 BYE sip:alice@10.1.3.3 SIP/2.0

4394 Via: SIP/2.0/UDP 10.4.1.4:5060

4395 From: Bob <sip:bob@biloxi.com>;tag=a6c85cf

4396 To: Alice <sip:alice@atlanta.com>;tag=1928301774

4397 Call-ID: a84b4c76e66710@10.1.3.3

4398 CSeq: 231 BYE

4399 Contact-Length: 0

4400

4401 F14 200 OK Alice -> Bob

4402

4403 SIP/2.0 200 OK

4404 Via: SIP/2.0/UDP 10.4.1.4:5060

4405 From: Bob <sip:bob@biloxi.com>;tag=a6c85cf

4406 To: Alice <sip:alice@atlanta.com>;tag=1928301774

4407 Call-ID: a84b4c76e66710@10.1.3.3

4408 CSeq: 231 BYE

4409 Contact-Length: 0

4410 The SIP Call Flows document [42] contains further examples of SIP messages.

4411 ;; This buffer is for notes you don't want to save, and for Lisp evaluation. ;; If you want to create a file,
4412 first visit that file with C-x C-f, ;; then enter the text in that file's own buffer.

4413 **26 Augmented BNF for the SIP Protocol**

4414 All of the mechanisms specified in this document are described in both prose and an augmented Backus-
4415 Naur Form (BNF) similar to that used by RFC 822 [12] and RFC 2234 [43]. Implementors will need to
4416 be familiar with the notation in order to understand this specification. The augmented BNF includes the
4417 following constructs:

4418 name = definition

4419 The name of a rule is simply the name itself (without any enclosing “<” and “>”) and is separated from
4420 its definition by the equal “=” character. White space is only significant in that indentation of continuation
4421 lines is used to indicate a rule definition that spans more than one line. Certain basic rules are in uppercase,
4422 such as SP, LWS, HT, CRLF, DIGIT, ALPHA, etc. Angle brackets are used within definitions whenever
4423 their presence will facilitate discerning the use of rule names.

4424 "literal"

4425 Quotation marks surround literal text. Unless stated otherwise, the text is case-insensitive.

4426 rule1 | rule2

4427 Elements separated by a bar (“|”) are alternatives, e.g., “yes | no” will accept yes or no.

4428 (rule1 rule2)

4429 Elements enclosed in parentheses are treated as a single element. Thus, “(elem (foo | bar) elem)” allows the
4430 token sequences “elem foo elem” and “elem bar elem”.

4431 *rule

4432 The character “*” preceding an element indicates repetition. The full form is “< n >* < m >element”
4433 indicating at least < n > and at most < m > occurrences of element. Default values are 0 and infinity so
4434 that “*(element)” allows any number, including zero; “1*element” requires at least one; and “1*2element”
4435 allows one or two.

4436 [rule]

4437 Square brackets enclose optional elements; “[foo bar]” is equivalent to “*1(foo bar)”.

4438 N rule

4439 Specific repetition: “<n>(element)” is equivalent to “<n>*<n>(element)”; that is, exactly <n> occur-
4440 rences of (element). Thus 2DIGIT is a 2-digit number, and 3ALPHA is a string of three alphabetic charac-
4441 ters.

4442 #rule

4443 A construct “#” is defined, similar to “*”, for defining lists of elements. The full form is “< n >#< m >
4444 element” indicating at least < n > and at most < m > elements, each separated by one or more commas
4445 (“,”) and OPTIONAL linear white space (LWS). This makes the usual form of lists very easy; a rule such as

4446 (*LWS element *(*LWS ”,“ *LWS element))

4447 can be shown as 1# element. Wherever this construct is used, null elements are allowed, but do not
4448 contribute to the count of elements present. That is, “(element), , (element)” is permitted, but counts
4449 as only two elements. Therefore, where at least one element is required, at least one non-null element
4450 MUST be present. Default values are 0 and infinity so that “#element” allows any number, including zero;
4451 “1#element” requires at least one; and “1#2element” allows one or two.

4452 ; comment

4453 A semi-colon, set off some distance to the right of rule text, starts a comment that continues to the end of
4454 line. This is a simple way of including useful notes in parallel with the specifications.

4455 26.1 Basic Rules

4456 The following rules are used throughout this specification to describe basic parsing constructs. The US-
4457 ASCII coded character set is defined by ANSI X3.4-1986.

OCTET	=	%x00-ff ; any 8-bit sequence of data
CHAR	=	%x00-7f ; any US-ASCII character (octets 0 - 127)
upalpha	=	"A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
lowalpha	=	"a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
alpha	=	lowalpha upalpha
DIGIT	=	"0" "1" "2" "3" "4" "5" "6" "7" "8" "9"
alphanumeric	=	alpha DIGIT
CTL	=	%x00-1f %x7f ; (octets 0 – 31) and DEL (127)
CR	=	%d13 ; US-ASCII CR, carriage return character
LF	=	%d10 ; US-ASCII LF, line feed character
SP	=	%d32 ; US-ASCII SP, space character
HT	=	%d09 ; US-ASCII HT, horizontal tab character
4458 CRLF	=	CR LF ; typically the end of a line

4459 The following are defined in RFC 2396 [9] for the SIP URI:

```

unreserved = alphanum | mark
mark       = "-" | "_" | "." | "!" | "'" | "*" | ""
           | "(" | ")"
4460 escaped = "%" hex hex

```

4461 SIP header field values can be folded onto multiple lines if the continuation line begins with a space or
4462 horizontal tab. All linear white space, including folding, has the same semantics as SP. A recipient MAY
4463 replace any linear white space with a single SP before interpreting the field value or forwarding the message
4464 downstream. This is intended to behave exactly as HTTP 1.1 as described in RFC2615 [8].

```

4465 LWS = *( SP | HT ) [CRLF] 1*( SP | HT ) ; linear whitespace

```

4466 To separate the header name from the rest of value, a colon is used, which, by the above rule allows
4467 whitespace before, but no line break, and whitespace after, including a linebreak. The HCOLON defines
4468 this construct.

```

4469 HCOLON = *( SP | HT ) ":" LWS

```

4470 The TEXT-UTF8 rule is only used for descriptive field contents and values that are not intended to be
4471 interpreted by the message parser. Words of *TEXT-UTF8 contain characters from the UTF-8 character
4472 set (RFC 2279 [11]). The TEXT-UTF8-TRIM rule is used for descriptive field contents that are *not* quoted
4473 strings, where leading and trailing LWS is not meaningful. In this regard, SIP differs from HTTP, which
4474 uses the ISO 8859-1 character set.

```

TEXT-UTF8      = *(TEXT-UTF8char | LWS)
TEXT-UTF8-TRIM = *TEXT-UTF8char *(LWS TEXT-UTF8char)
TEXT-UTF8char  = %x21-7e
               | UTF8-NONASCII
UTF8-NONASCII = %xc0-df 1UTF8-CONT
               | %xe0-ef 2UTF8-CONT
               | %xf0-f7 3UTF8-CONT
               | %xf8-fb 4UTF8-CONT
               | %xfc-fd 5UTF8-CONT
4475 UTF8-CONT   = %x80-bf

```

4476 A CRLF is allowed in the definition of TEXT-UTF8 only as part of a header field continuation. It is
4477 expected that the folding LWS will be replaced with a single SP before interpretation of the TEXT-UTF8
4478 value.

4479 Hexadecimal numeric characters are used in several protocol elements. Some elements (authentication)
4480 force hex alphas to be lower case.

```

4481 LHEX = digit | "a" | "b" | "c" | "d" | "e" | "f"

```

4482 Others allow mixed upped and lower case

```

4483 hex = LHEX | "A" | "B" | "C" | "D" | "E" | "F"

```

4484 Many SIP header field values consist of words separated by LWS or special characters. Unless otherwise
4485 stated, tokens are case-insensitive. These special characters **MUST** be in a quoted string to be used within a
4486 parameter value.

token = 1*(alphanumeric | "-" | "." | "!" | "%" | "*" | "_" | "+" | "'" | "''" | "'''")
 separators = "(" | ")" | "<" | ">" | "@" |
 "," | ";" | ":" | "\" | "<"> |
 "/" | "[" | "]" | "?" | "=" |
 "{" | "}" | SP | HT

4487

4488 When tokens are used or separators are used between elements, whitespace is often allowed before or
 4489 after these characters:

MINUS = LWS "-" LWS ; minus
 DOT = LWS "." LWS ; period
 PERCENT = LWS "%" LWS ; percent
 BANG = LWS "!" LWS ; exclamation
 PLUS = LWS "+" LWS ; plus
 STAR = LWS "*" LWS ; asterisk
 TILDE = LWS "~" LWS ; tilde
 EQUAL = LWS "=" LWS ; equal
 LPAREN = LWS "(" LWS ; left parenthesis
 RPAREN = LWS ")" LWS ; right parenthesis
 LANGLE = LWS "<" LWS ; left angle bracket
 RAQUOT = ">" LWS ; right angle quote
 LAQUOT = LWS "<"; left angle quote
 RANGLE = LWS ">" LWS ; right angle bracket
 BAR = LWS "—" LWS ; vertical bar
 ATSIGN = LWS "@" LWS ; atsign
 COMMA = LWS "," LWS ; comma
 SEMI = LWS ";" LWS ; semicolon
 COLON = LWS ":" LWS ; colon
 DQUOT = LWS "<"> LWS ; double quotation mark
 LDQUOT = LWS "<">; open double quotation mark
 RDQUOT = "<"> LWS ; close double quotation mark
 LBRACK = LWS "{" LWS ; left square bracket
 4490 RBRACK = LWS "}" LWS ; right square bracket

4491 Comments can be included in some SIP header fields by surrounding the comment text with parentheses.
 4492 Comments are only allowed in fields containing "comment" as part of their field value definition. In all other
 4493 fields, parentheses are considered part of the field value.

comment = LPAREN *(ctext | quoted-pair | comment) RPAREN
 4494 ctext = <any TEXT-UTF8 excluding "(" and ">

4495 A string of text is parsed as a single word if it is quoted using double-quote marks. In quoted strings,
 4496 quotation marks (") and backslashes (\) need to be escaped.

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

4497

UTF8-NONASCII

4498 The backslash character ("") MAY be used as a single-character quoting mechanism only within quoted-
 4499 string and comment constructs. Unlike HTTP/1.1, the characters CR and LF cannot be escaped by this
 4500 mechanism to avoid conflict with line folding and header separation.

```

4501 quoted-pair = "\" (%x00 - %x09 | %x0b | %x0c | %x0e - %x7f)

SIP-URL      = "sip:" [ userinfo "@" ] hostport
              url-parameters [ headers ]
userinfo     = [ user | telephone-subscriber [ ":" password ] ]
user        = *( unreserved | escaped | user-unreserved )
user-unreserved = "&" | "=" | "+" | "$" | "," | ";" | "?" | "/"
password    = *( unreserved | escaped |
              "&" | "=" | "+" | "$" | "," )

hostport    = host [ ":" port ]
host        = hostname | IPv4address | IPv6reference
hostname    = *( domainlabel "." ) toplabel [ "." ]
domainlabel = alphanum
              | alphanum *( alphanum | "-" ) alphanum
4502 toplabel = alpha | alpha *( alphanum | "-" ) alphanum

IPv4address = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
IPv6reference = "[" IPv6address "]"
IPv6address = hexpart [ ":" IPv4address ]
hexpart     = hexseq | hexseq ":" [ hexseq ] | ":" [ hexseq ]
hexseq     = hex4 *( ":" hex4)
hex4       = 1*4HEX
4503 port     = 1*DIGIT

url-parameters = *( ";" url-parameter)
url-parameter  = transport-param | user-param | method-param
                | ttl-param | maddr-param | other-param
transport-param = "transport="
                ( "udp" | "tcp" | "sctp" | "tls"
                  | other-transport)
other-transport = token
user-param     = "user=" ( "phone" | "ip" | other-user)
other-user    = token
method-param   = "method=" Method
ttl-param     = "ttl=" ttl
maddr-param   = "maddr=" host
other-param    = pname [ "=" pvalue ]
pname         = 1*paramchar
pvalue        = 1*paramchar
paramchar     = param-unreserved | unreserved | escaped
4504 param-unreserved = "[" | "]" | "/" | "." | "&" | "+" | "$"

```


headers = "?" header *("&" header)
header = hname "=" hvalue
hname = 1*(hnv-unreserved | unreserved | escaped)
hvalue = *(hnv-unreserved | unreserved | escaped)
4505 hnv-unreserved = "[" | "]" | "/" | "?" | ":" | "+" | "\$"

SIP-message = Request | Response
Request = Request-Line
 *(message-header)
 CRLF
 [message-body]
Request-Line = Method SP Request-URI SP SIP-Version CRLF
Request-URI = SIP-URL | absoluteURI
4506 SIP-Version = "SIP/2.0"

message-header

```
= Accept
| Accept-Encoding
| Accept-Language
| Alert-Info
| Allow
| Authentication-Info
| Authorization
| Call-ID
| Call-Info
| Contact
| Content-Disposition
| Content-Encoding
| Content-Language
| Content-Length
| Content-Type
| CSeq
| Date
| Error-Info
| Expires
| From
| In-Reply-To
| Max-Forwards
| MIME-Version
| Organization
| Priority
| Proxy-Authenticate
| Proxy-Authorization
| Proxy-Require
| Record-Route
| Require
| Retry-After
| Route
| Server
| Subject
| Supported
| Timestamp
| To
| Unsupported
| User-Agent
| Via
| Warning
| WWW-Authenticate
```

Method = "INVITE" | "ACK" | "OPTIONS" | "BYE"
 | "CANCEL" | "REGISTER" | extension-method
 extension-method = token
 option-tag = token
 Response = Status-Line
 *(message-header)
 CRLF
 [message-body]

4508

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

4509

Reason-Phrase = * <TEXT-UTF8, excluding CR, LF >
 Informational = "100" ; Trying
 | "180" ; Ringing
 | "181" ; Call Is Being Forwarded
 | "182" ; Queued
 | "183" ; Session Progress

4510

Success = "200" ; OK

4511

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

4512

Client-Error = "400" ; Bad Request
 | "401" ; Unauthorized
 | "402" ; Payment Required
 | "403" ; Forbidden
 | "404" ; Not Found
 | "405" ; Method Not Allowed
 | "406" ; Not Acceptable
 | "407" ; Proxy Authentication Required
 | "408" ; Request Timeout
 | "409" ; Conflict
 | "410" ; Gone
 | "413" ; Request Entity Too Large
 | "414" ; Request-URI Too Large
 | "415" ; Unsupported Media Type
 | "420" ; Bad Extension
 | "480" ; Temporarily not available
 | "481" ; Call Leg/Transaction Does Not Exist
 | "482" ; Loop Detected
 | "483" ; Too Many Hops
 | "484" ; Address Incomplete
 | "485" ; Ambiguous
 | "486" ; Busy Here
 | "487" ; Request Terminated
 4513 | "488" ; Not Acceptable Here

Server-Error = "500" ; Internal Server Error
 | "501" ; Not Implemented
 | "502" ; Bad Gateway
 | "503" ; Service Unavailable
 | "504" ; Server Time-out
 4514 | "505" ; SIP Version not supported

Global-Failure = "600" ; Busy Everywhere
 | "603" ; Decline
 | "604" ; Does not exist anywhere
 4515 | "606" ; Not Acceptable

Accept = "Accept" HCOLON
 #(media-range [accept-params])
 media-range = ("*"/*"
 | (type LWS "/" "*" LWS)
 | (type SLASH subtype)
) *(SEMI parameter)
 accept-params = SEMI "q" EQUAL qvalue *(accept-extension)
 4516 accept-extension = SEMI token [EQUAL (token | quoted-string)]

Accept-Encoding = "Accept-Encoding" HCOLON
 1#(codings [SEMI "q" EQUAL qvalue] LWS)
 codings = (content-coding | "*")
 content-coding = token
 qvalue = ("0" ["." 0*3DIGIT])
 4517 — ("1" ["." 0*3("0")])

Accept-Language = "Accept-Language" HCOLON
 1#(language-range [SEMI "q" EQUAL qvalue])
 4518 language-range = ((1*8ALPHA *(MINUS 1*8ALPHA)) — "*")

Alert-Info = "Alert-Info" HCOLON #
 (LAQUOT URI RAQUOT *(COLON generic-param))
 generic-param = token [EQUAL (token | host |
 4519 quoted-string)]

4520 Allow = "Allow" HCOLON 1#Method

Authorization = "Authorization" HCOLON credentials
 credentials = LWS "Digest" digest-response
 digest-response = 1#(username | realm | nonce | digest-uri
 | dresponse | [algorithm] | [cnonce]
 | [opaque] | [message-qop]
 | [nonce-count] | [auth-param])
 username = "username" EQUAL username-value
 username-value = quoted-string
 digest-uri = "uri" EQUAL digest-uri-value
 digest-uri-value = request-uri ; As specified by HTTP/1.1
 message-qop = "qop" EQUAL qop-value
 cnonce = "cnonce" EQUAL cnonce-value
 cnonce-value = nonce-value
 nonce-count = "nc" EQUAL nc-value
 dresponse = "response" EQUAL request-digest
 4521 request-digest = LDQUOT 32LHEX RDQUOT

AuthenticationInfo = "Authentication-info" HCOLON 1#(digest — nextnonce)
 nextnonce = "nextnonce" EQUAL nonce-value

4522 callid = token [ATSIGN token]

4523 Call-ID = ("Call-ID" | "i") HCOLON callid

Call-Info = "Call-Info" HCOLON # (LAQUOT URI RAQUOT
 *(SEMI info-param))
 info-param = "purpose" EQUAL ("icon" | "info"
 4524 | "card" | token) | generic-param

Contact = ("Contact" | "m") HCOLON
 (STAR | (1# ((name-addr | addr-spec)
 *(SEMI contact-params))))

name-addr = [display-name] LAQUOT addr-spec RAQUOT
 addr-spec = SIP-URL | URI
 4525 display-name = LWS (*token | quoted-string)

contact-params = "q" EQUAL qvalue
 | "action" EQUAL "proxy" | "redirect"
 | "expires" EQUAL delta-seconds |
 LDQUOT SIP-date RDQUOT
 | contact-extension

contact-extension = generic-param
 qvalue = ("0" ["." 0*3DIGIT])
 4526 | ("1" ["." 0*3("0")])

4527 delta-seconds = 1*DIGIT

Content-Disposition = "Content-Disposition" HCOLON
 disposition-type *(SEMI disposition-param)

disposition-type = "render" | "session" | "icon" | "alert"
 | disp-extension-token

disposition-param = "handling" EQUAL
 ("optional" | "required" |
 other-handling) | generic-param

other-handling = token
 4528 disp-extension-token = token

Content-Encoding = ("Content-Encoding" | "e") HCOLON
 4529 1#content-coding

Content-Language = "Content-Language" HCOLON 1#language-tag
 language-tag = primary-tag *(MINUS subtag)
 primary-tag = 1*8ALPHA
 4530 subtag = 1*8ALPHA

4531 Content-Length = ("Content-Length" | "l") HCOLON 1*DIGIT

4532 Content-Type = ("Content-Type" | "c") HCOLON media-type

4533 CSeq = "CSeq" HCOLON 1*DIGIT Method

Date = "Date" HCOLON SIP-date
 SIP-date = rfc1123-date
 rfc1123-date = wkday COMMA SP date1 SP time SP "GMT"
 date1 = 2DIGIT SP month SP 4DIGIT
 ; day month year (e.g., 02 Jun 1982)
 time = 2DIGIT ":" 2DIGIT ":" 2DIGIT
 ; 00:00:00 - 23:59:59
 wkday = "Mon" | "Tue" | "Wed"
 | "Thu" | "Fri" | "Sat" | "Sun"
 month = "Jan" | "Feb" | "Mar" | "Apr"
 | "May" | "Jun" | "Jul" | "Aug"
 | "Sep" | "Oct" | "Nov" | "Dec"

Error-Info = "Error-Info" HCOLON #
 (LAQUOT URI RAQUOT
 *(SEMI generic-param)

Expires = "Expires" HCOLON (SIP-date | delta-seconds)
 From = ("From" | "f") HCOLON
 (name-addr | addr-spec)
 *(SEMI from-param)

from-param = tag-param | generic-param
 tag-param = "tag" EQUAL token

In-Reply-To = "In-Reply-To" HCOLON 1# callid

Max-Forwards = "Max-Forwards" HCOLON 1*DIGIT

MIME-Version = "MIME-Version" HCOLON 1*DIGIT ":" 1*DIGIT

Organization = "Organization" HCOLON TEXT-UTF8-TRIM

Priority = "Priority" HCOLON priority-value
 priority-value = "emergency" | "urgent" | "normal"
 | "non-urgent" | other-priority
 other-priority = token

Proxy-Authenticate = "Proxy-Authenticate" HCOLON 1#challenge
 challenge = LWS "Digest" digest-challenge
 digest-challenge = 1#(realm | [domain] | nonce |
 [opaque] | [stale] | [algorithm] |
 [qop-options] | [auth-param])
 realm = "realm" EQUALS realm-value
 realm-value = quoted-string
 domain = "domain" EQUAL LDQUOT URI
 (1*SP URI) RDQUOT
 URI = absoluteURI | abs_path
 nonce = "nonce" EQUAL nonce-value
 nonce-value = quoted-string
 opaque = "opaque" EQUAL quoted-string
 stale = "stale" EQUAL ("true" | "false")
 algorithm = "algorithm" EQUAL ("MD5" | "MD5-sess" |
 token)
 qop-options = "qop" EQUAL LDQUOT 1#qop-value RDQUOT
 qop-value = "auth" | "auth-int" | token
 4542

Proxy-Authorization = "Proxy-Authorization" HCOLON credentials
 4543

Proxy-Require = "Proxy-Require" HCOLON 1#option-tag
 4544

Record-Route = "Record-Route" HCOLON 1#
 (name-addr *(SEMI rr-param))
 rr-param = generic-param
 4545

Require = "Require" HCOLON 1#option-tag
 4546

Retry-After = "Retry-After" HCOLON
 (SIP-date | delta-seconds)
 [comment] *(SEMI retry-param)
 retry-param = "duration" EQUAL delta-seconds |
 generic-param
 4547

Route = "Route" HCOLON 1# (name-addr
 *(SEMI rr-param))
 4548

Server = "Server" HCOLON 1*(product — comment)
 product = token [SLASH product-version]
 product-version = token
 4549

Subject = ("Subject" | "s") HCOLON TEXT-UTF8-TRIM
 4550

Supported = ("Supported" | "k") HCOLON 0#option-tag

Timestamp = "Timestamp" HCOLON *(DIGIT)
 ["." *(DIGIT)] [delay]
 4551 delay = *(DIGIT) ["." *(DIGIT)]

To = ("To" | "t") HCOLON (name-addr |
 addr-spec) *(SEMI to-param)
 4552 to-param = tag-param | generic-param

4553 Unsupported = "Unsupported" HCOLON 1#option-tag

4554 User-Agent = "User-Agent" HCOLON 1*(product — comment)

Via = ("Via" | "v") HCOLON
 1#(sent-protocol sent-by
 *(SEMI via-params) [comment])
 via-params = via-hidden | via-ttl | via-maddr
 | via-received | via-branch
 | via-extension
 via-hidden = "hidden"
 via-ttl = "ttl" EQUAL ttl
 via-maddr = "maddr" EQUAL host
 via-received = "received" EQUAL host
 via-branch = "branch" EQUAL token
 via-extension = generic-param
 sent-protocol = protocol-name SLASH protocol-version
 SLASH transport
 protocol-name = "SIP" | token
 protocol-version = token
 transport = "UDP" | "TCP" | "TLS" | "SCTP"
 | other-transport
 sent-by = host [COLON port]
 4555 ttl = 1*3DIGIT ; 0 to 255

Warning = "Warning" HCOLON 1#warning-value
 warning-value = warn-code SP warn-agent SP warn-text
 warn-code = 3DIGIT
 warn-agent = (host [COLON port]) | pseudonym
 ; the name or pseudonym of the server adding
 ; the Warning header, for use in debugging
 warn-text = quoted-string
 4556 pseudonym = token

4557 WWW-Authenticate = "WWW-Authenticate" HCOLON challenge

4558 27 IANA Considerations

4559 All new or experimental method names, header field names, and status codes used in SIP applications
4560 SHOULD be registered with IANA in order to prevent potential naming conflicts. It is RECOMMENDED that
4561 new “option- tag”s and “warn-code”s also be registered. Before IANA registration, new protocol elements
4562 SHOULD be characterized in an Internet- Draft or, preferably, an RFC.

4563 For Internet-Drafts, IANA is requested to make the draft available as part of the registration database.

4564 By the time an RFC is published, colliding names may have already been implemented.

4565 When a registration for either a new header field, new method or new status code is created based on
4566 an Internet-Draft, and that Internet-Draft becomes an RFC, the person that performed the registration MUST
4567 notify IANA to change the registration to point to the RFC instead of the Internet-Draft.

4568 Registrations should be sent to `iana@iana.org`.

4569 27.1 Option Tags

4570 Option tags are used in headers such as Require, Supported, Proxy-Require and Unsupported in support
4571 of SIP compatibility mechanisms for extensions. For more on the use of option tags in these headers see
4572 Section 21.2. The option tag itself is a string that is associated with a particular SIP option (e.g. an extension)
4573 in order to identify the option in signaling between SIP endpoints.

4574 When registering a new SIP option with IANA, the following information MUST be provided:

- 4575 ● Name and description of option. The name MAY be of any length, but SHOULD be no more than
4576 twenty characters long. The name MUST consist of alphanum (See Section 26) characters only
- 4577 ● A listing of any new SIP header fields, header parameter fields or parameter values defined by this
4578 option. A SIP option MUST NOT redefine header fields or parameters defined in either RFC 2543, any
4579 standards-track extensions to RFC 2543, or other extensions registered through IANA
- 4580 ● Indication of who has change control over the option (for example, IETF, ISO, ITU-T, other interna-
4581 tional standardization bodies, a consortium or a particular company or group of companies)
- 4582 ● A reference to a further description, if available, for example (in order of preference) an RFC, a
4583 published paper, a patent filing, a technical report, documented source code or a computer manual
- 4584 ● Contact information (postal and email address)

4585 This procedure has been borrowed from RTSP [4] and the RTP AVP [44].

4586 27.2 Warn-Codes

4587 Warning codes provide information supplemental to the status code in SIP response messages when the
4588 failure of the transaction results from a Session Description Protocol (SDP, [6]). New “warn-code” values
4589 can be registered with IANA as they arise.

4590 The “warn-code” consists of three digits. A first digit of “3” indicates warnings specific to SIP.

4591 Warnings 300 through 329 are reserved for indicating problems with keywords in the session description,
4592 330 through 339 are warnings related to basic network services requested in the session description, 370
4593 through 379 are warnings related to quantitative QoS parameters requested in the session description, and
4594 390 through 399 are miscellaneous warnings that do not fall into one of the above categories.

4595 1xx and 2xx have been taken by HTTP/1.1.

4596 **27.3 Header Field Names**

4597 Header field names do not require working group or working group chair review prior to IANA registration,
4598 but SHOULD be documented in an RFC or Internet- Draft before IANA is consulted.

4599 The following information needs to be provided to IANA in order to register a new header field name:

- 4600 • The name and email address of the individual performing the registration.
- 4601 • The name of the header field being registered.
- 4602 • A compact form version for that header field, if one is defined.
- 4603 • The name of the draft or RFC where the header field is defined.
- 4604 • A copy of the draft or RFC where the header field is defined.

4605 Header fields SHOULD NOT use the X- prefix notation and MUST NOT duplicate the names of header
4606 fields used by SMTP or HTTP unless the syntax is a compatible superset and the semantics are similar.
4607 Some common and widely used header fields MAY be assigned one-letter compact forms (Section 7.3.3).
4608 Compact forms can only be assigned after SIP working group review. In the absence of this working group,
4609 a designated expert reviews the request.

4610 **27.4 Method and Response Codes**

4611 Because the status code space is limited, they do require working group or working group chair review, and
4612 MUST be documented in an RFC or Internet draft. The same procedures apply to new method names.

4613 The following information needs to be provided to IANA in order to register a new response code or
4614 method:

- 4615 • The name and email address of the individual performing the registration.
- 4616 • The number of the response code or name of the method being registered.
- 4617 • The default reason phrase for that status code, if applicable.
- 4618 • The name of the draft or RFC where the method or status code is defined.
- 4619 • A copy of the draft or RFC where the method or status code is defined.

4620 **28 Changes Made in Version 00**

- 4621 • Indicated that UAC should send both CANCEL and BYE after a retransmission fails.
- 4622 • Added semicolon and question mark to the list of unreserved characters for the user part of SIP URLs
4623 to handle tel: URLs properly.
- 4624 • Uniform handling of if hop count Max-Forwards: return 483. Note that this differs from HTTP/1.1
4625 behavior, where only OPTIONS and TRACE allow this header, but respond as the final recipient when
4626 the value reaches zero.

- 4627 • Clarified that a forking proxy sends ACKs only for INVITE requests.
- 4628 • Clarified wording of DNS caching. Added paragraph on “negative caching”, i.e., what to do if one
4629 of the hosts failed. It is probably not a good idea to simply drop this host from the list if the DNS ttl
4630 value is more than a few minutes, since that would mean that load balancing may not work for quite a
4631 while after a server is brought back on line. This will be true in particular if a server group receives a
4632 large number of requests from a small number of upstream servers, as is likely to be the case for calls
4633 between major consumer ISPs. However, without getting into arbitrary and complicated retry rules, it
4634 seems hard to specify any general algorithm. Might it be worthwhile to simply limit the “black list”
4635 interval to a few minutes?
- 4636 • Added optional Call-Info and Alert-Info header fields that describe the caller and information to be
4637 used in alerting. (Currently, avoided use of “purpose” qualification since it is not yet clear whether
4638 rendering content without understanding its meaning is always appropriate. For example, if a UAS
4639 does not understand that this header is to replace ringing, it would mix both local ring tone and the
4640 indicated sound URL.) TBD!
- 4641 • SDP “s=” lines can’t be empty, unfortunately.
- 4642 • Noted that maddr could also contain a unicast address, but SHOULD contain the multicast address if
4643 the request is sent via multicast (Section 22.40).
- 4644 • Clarified that responses are sent to port in Via sent-by value.
- 4645 • Added “other-*” to the user URL parameter and the Hide and Content-Disposition headers.
- 4646 • Clarified generation of timeout (408) responses in forking proxies and mention the Expires header.
- 4647 • Clarified that CANCEL and INVITE are separate transactions (Fig. 7). Thus, the INVITE request
4648 generates a 487 (Request Terminated) if a CANCEL or BYE arrives.
- 4649 • Clarified that Record-Route SHOULD be inserted in every request, but that the route, once estab-
4650 lished, persists. This provides robustness if the called UAS crashes.
- 4651 • Emphasized that proxy, redirect, registrar and location servers are logical, not physical entities and
4652 that UAC and UAS roles are defined on a request-by-request basis. (Section 6)
- 4653 • In Section 22.40, noted that the maddr and received parameters also need to be encrypted when
4654 doing Via hiding.
- 4655 • Simplified Fig. 7 to only show INVITE transaction.
- 4656 • Added definition of the use of Contact (Section 22.10) for OPTIONS.
- 4657 • Added HTTP/RFC822 headers Content-Language and MIME-Version.
- 4658 • Added note in minimal section indicating that UAs need to support UDP.
- 4659 • Added explanation explaining what a UA should do when receiving an initial INVITE with a tag.
- 4660 • Clarified UA and proxy behavior for 302 responses.

- 4661 ● Added details on what a UAS should do when receiving a tagged INVITE request for an unknown call
4662 leg. This could occur if the UAS had crashed and the UAC sends a re-INVITE or if the BYE got lost
4663 and the UAC still believes to be in the call.
- 4664 ● Added definition of Contact in 4xx, 5xx and 6xx to “redirect” to more error details.
- 4665 ● Added note to forking proxy description to gather *-Authenticate from responses. This allows several
4666 branches to be authenticated simultaneously.
- 4667 ● Changed URI syntax to use URL escaping instead of quotation marks.
- 4668 ● Changed SIP URL definition to reference RFC 2806 for telephone-subscriber part.
- 4669 ● Clarified that the To URI should basically be ignored by the receiving UAS except for matching
4670 requests to call legs. In particular, To headers with a scheme or name unknown to the callee should
4671 be accepted.
- 4672 ● Clarified that maddr is to be added by any client, either proxy or UAC.
- 4673 ● Added response code 488 to indicate that there was no common media at the particular destination.
4674 (606 indicates such failure globally.)
- 4675 ● In Section 22.19, noted that registration updates can shorten the validity period.
- 4676 ● Added note to enclose the URI for digest in quotation marks. The BNF in RFC 2617 is in error.
- 4677 ● Clarified that registrars use Authorization and WWW-Authenticate, not proxy authentication.
- 4678 ● Added note in Section 22.10 that “headers” are copied from Contact into the new request.
- 4679 ● Changed URL syntax so that port specifications have to have at least one digit, in line with other URL
4680 formats such as “http”. Previously, an empty port number was permissible.
- 4681 ● In SDP section, added a section on how to add and delete streams in re-INVITEs.
- 4682 ● IETF-blessed extensions now have short names, without org.ietf. prefix.
- 4683 ● Cseq is unique within a call leg, not just within a call (Section 22.16).
- 4684 ● Added IPv6 literal addresses to the SIP URL definition, according to RFC 2732 [45]. Modified the
4685 IPv4 address to limit segments to at most three digits.
- 4686 ● modify registration procedure so that it explicitly references the URL comparison. Updates with
4687 shorter expiration time are now allowed.
- 4688 ● For send-only media, SDP still must indicate the address and port, since these are needed as destina-
4689 tions for RTCP messages.
- 4690 ● Changed references regarding DNS SRV records from RFC 2052 to RFC 2782, which is now a Pro-
4691 posed Standard. Integrated SRV into the search procedure and removed the SRV appendix. The only
4692 visible change is that protocol and service names are now prefixed by an underscore. Added wording
4693 that incorporates the precedence of maddr.

- 4694 • Allow parameters in Record-Route and Route headers.
- 4695 • In Table 1, list `udp` as the default value for the transport parameter in SIP URI.
- 4696 • Removed sentence that `From` can be encrypted. It cannot, since the header is needed for call-leg
4697 identification.
- 4698 • Added note that a UAC only copies a `To` tag into subsequent transactions if it arrives in a 200 OK to
4699 an INVITE. This avoids the problem that occurs when requests get resubmitted after receiving, say,
4700 a 407 (or possibly 500, 503, 504, 305, 400, 411, 413, maybe even 408). Under the old rules, these
4701 requests would have a tag, which would force the called UAS to reject the request, since it doesn't
4702 have an entry for this tag.
- 4703 • Loop detection has been modified to take the `request-URI` into account. This allows the same request
4704 to visit the server twice, but with different request URIs ("spiral").
- 4705 • Elaborated on URL comparison and comparison of `From/To` fields.
- 4706 • Added `np-queried` user parameter.
- 4707 • Changed `tag` syntax from `UUID` to `token`, since there's no reason to restrict it to hex.
- 4708 • Added `Content-Disposition` header based on earlier discussions about labeling what to do with a
4709 message body (part).
- 4710 • Clarification: proxies must insert `To` tags for locally generated responses.
- 4711 • Clarification: multicast may be used for subsequent registrations.
- 4712 • Feature: Added `Supported` header. Needed if client wants to indicate things the server can usefully
4713 return in the response.
- 4714 • Bug: The `From`, `To`, and `Via` headers were missing extension parameters. The `Encryption` and
4715 `Response-Key` header fields now "officially" allow parameters consisting only of a token, rather
4716 than just "token = value".
- 4717 • Bug: `Allow` was listed as optional in 405 responses in Table 2. It is mandatory.
- 4718 • Added: "A `BYE` request from either called or calling party terminates any pending INVITE, but the
4719 INVITE request transaction **MUST** be completed with a final response."
- 4720 • Clarified: "If an INVITE request for an existing session fails, the session description agreed upon in
4721 the last successful INVITE transaction remains in force."
- 4722 • Clarified what happens if two INVITE requests meet each other on the wire, either traveling the same
4723 or in opposite directions:

4724 A UAC **MUST NOT** issue another INVITE request for the same call leg before the pre-
4725 vious transaction has completed. A UAS that receives an INVITE before it sent the final
4726 response to an INVITE with a lower `CSeq` number **MUST** return a 400 (Bad Request)
4727 response and **MUST** include a `Retry-After` header field with a randomly chosen value of

4728 between 0 and 10 seconds. A UA that receives an INVITE while it has an INVITE transac-
4729 tion pending, returns a 500 (Internal Server Error) and also includes a **Retry-After** header
4730 field.

- 4731 • **Expires** header clarified: limits only duration of INVITE transaction, not the actual session. SDP
4732 does the latter.
- 4733 • The **In-Reply-To** header was added.
- 4734 • There were two incompatible BNFs for **WWW-Authenticate**. One defined for PGP, and the other
4735 borrowed from HTTP. For basic or digest:

4736 WWW-Authenticate: basic realm="Wallyworld"

4737 and for pgp:

4738 WWW-Authenticate: pgp; realm="Wallyworld"

4739 The latter is incorrect and the semicolon has been removed.

- 4740 • Added rules for **Route** construction from called to calling UA.
- 4741 • We now allow **Accept** and **Accept-Encoding** in **BYE** and **CANCEL** requests. There is no particular
4742 reason not to allow them, as both requests could theoretically return responses, particularly when
4743 interworking with other signaling systems.
- 4744 • PGP "pgp-pubalgorithm" allows server to request the desired public-key algorithm.
- 4745 • ABNF rules now describe tokens explicitly rather than by subtraction; explicit character enumeration
4746 for CTL, etc.
- 4747 • Registrars should be careful to check the **Date** header as the expiration time may well be in the past,
4748 as seen by the client.
- 4749 • **Content-Length** is mandatory; Table 2 erroneously marked it as optional.
- 4750 • **User-Agent** was classified in a syntax definition as a request header rather than a general header.
- 4751 • Clarified ordering of items to be signed and include realm in list.
- 4752 • Allow **Record-Route** in 401 and 484 responses.
- 4753 • Hop-by-hop headers need to precede end-to-end headers only if authentication is used.
- 4754 • 1xx message bodies MAY now contain session descriptions.
- 4755 • Changed references to HTTP/1.1 and authentication to point to the latest RFCs.
- 4756 • Added 487 (Request terminated) status response. It is issued if the original request was terminated
4757 via **CANCEL** or **BYE**.

- 4758 • The spec was not clear on the identification of a call leg. Section 1.3 says it's the combination of **To**,
 4759 **From**, and **Call-ID**. However, requests from the callee to the caller have the **To** and **From** reversed, so
 4760 this definition is not quite accurate. Additionally, the "tag" field should be included in the definition
 4761 of call leg. The spec now says that a call leg is defined as the combination of local-address, remote-
 4762 address, and call-id, where these addresses include tags.

4763 Text was added to Section 6.21 to emphasize that the **From** and **To** headers designate the originator
 4764 of the request, not that of the call leg.

- 4765 • All URI parameters, except **method**, are allowed in a **Request-URI**. Consequently, also updated the
 4766 description of which parameters are copied from 3xx responses in Sec. 22.10.

- 4767 • The use of CRLF, CR, or LF to terminate lines was confusing. Basically, each header line can be
 4768 terminated by a CR, LF, or CRLF. Furthermore, the end of the headers is signified by a "double
 4769 return". Simplified to require sending of CRLF, but require senders to receive CR and LF as well and
 4770 only allow CR CR, LF LF in addition to double CRLF as a header-body separator.

- 4771 • Round brackets in **Contact** header were part of the HTTP legacy, and very hard to implement. They
 4772 are also not that useful and were removed.

- 4773 • The spec said that a proxy is a back-to-back UAS/UAC. This is almost, but not quite, true. For
 4774 example, a UAS should insert a tag into a provisional response, but a proxy should not. This was
 4775 clarified.

- 4776 • Section 6.13 in the RFC begins mid-paragraph after the BNF. The following text was misplaced in the
 4777 conversion to ASCII:

4778 Even if the "display-name" is empty, the "name-addr" form **MUST** be used if the "addr-
 4779 spec" contains a comma, semicolon or question mark.

4780 **29 Changes Made in Version 01**

- 4781 • Uniform syntax specification for semicolon parameters:

```

4782   Foo          = "Foo" ":" something *( ";" foo-param )
   foo-param    = "bar" "=" token
                 | generic-param

```

- 4783 • Removed **np-queried user** parameter since this is now part of a tel URL extension parameter.
- 4784 • In SDP section, noted that if the capabilities intersection is empty, a dummy format list still has to be
 4785 returned due to SDP syntax constraints. Previously, the text had required that no formats be listed.
 4786 (Brian Rosen)
- 4787 • Reorganized tables 2 and 3 to show proxy interaction with headers rather than "end-to-end" or "hop-
 4788 by-hop".

4789 30 Changes Made in Version 02

- 4790 ● Added “or UAS” in description of received headers in Section 22.40. This makes the response
4791 algorithm work even if the last IP address in the Via is incorrect.
- 4792 ● Tentatively removed restriction that CANCEL requests cannot have Route headers. (Billy Biggs)
- 4793 ● Tentatively added Also header for BYE requests, as it is widely implemented and a simple means to
4794 implement unsupervised call transfer. Subject to removal if there is protest. (Billy Biggs)
- 4795 ● If a proxy sends a request by UDP (TCP), the spec did not disallow placing TCP (UDP) in the transport
4796 parameter of the Via field, which it should. Added a note that the transport protocol actually used is
4797 included.
- 4798 ● No default value for the q parameter in Contact is defined. This is not strictly needed, but is useful for
4799 consistent behaviors at recursive proxies and at UAC’s. Now 0.5.
- 4800 ● Clarified that To and From tag values should be different to simplify request matching when calling
4801 oneself.
- 4802 ● Removed ability to carry multiple requests in a single UDP packet (Section 22.14).
- 4803 ● Added note that Allow MAY be included in requests, to indicate requestor capabilities for the same
4804 call ID.
- 4805 ● Added note to Section 22.17 indicating that registrars MUST include the Date header to accomodate
4806 UAs that do not have a notion of absolute time.
- 4807 ● Added note emphasizing that non-SIP URIs are permissible in REGISTER.
- 4808 ● Rewrote the server lookup section to be more precise and more like pseudo-code, with nesting instead
4809 of “gotos”.
- 4810 ● Removed note

4811 Note that the two URLs example.com and example.com:5060, while considered equal,
4812 may not lead to the same server, as the former causes a DNS SRV lookup, while the latter
4813 only uses the A record.

4814 since that is no longer the case.

- 4815 ● Emphasized that proxies have to forward requests with unknown methods.
- 4816 ● Aligned definition of call leg with URI comparison rules.
- 4817 ● Required that second branch parameter be globally unique, so that a proxy can distinguish different
4818 branches in spiral scenarios similar to the following, with record-routing in place:

```
4819           B  ---> P1  -----> P2  -----> P1  -----> A
4820 BYE B    B/1      P1/2, B/1    P2/3, P1/2, B/1    P1/4, P2/3, P1/2, B/1
```

4821 Here, A/1 denotes the *Via* entry with host A and branch parameter 1. Also, this requires updating the
4822 definition of isomorphic requests, since the *Request-URI* is the same for all *BYE* that are record-
4823 routed.

4824 ● Removed *Via* hiding from spec, for the following reasons:

- 4825 – complexity, particularly hidden “gotchas” that surface at various points (as in this instance);
- 4826 – interference with loop detection and debugging;
- 4827 – Unlike HTTP, where *via*-hiding makes sense since all data is contained in the request or re-
4828 sponse, *Via*-hiding in SIP by itself does nothing to hide the caller or callee, as address informa-
4829 tion is revealed in a number of places:
 - 4830 * **Contact**;
 - 4831 * **Route/Record-Route**;
 - 4832 * **SDP**, including the *o=* and *c=* lines;
 - 4833 * possibly accidental leakage in **User-Agent** header and **Call-ID** headers.
- 4834 – Unless this is implemented everywhere, the feature is not likely to be very useful, without the
4835 sender having any recourse such as “don’t route this request unless you can hide”. It appears
4836 that almost all existing proxies simply ignore the *Hide* header.

4837 ● Added **Error-Info** header field.

4838 **31 Changes Made in Version 03**

- 4839 ● Description of **Route** and **Record-Route** moved to separate section, which is new. All UAs must
4840 now support this mechanism.
- 4841 ● Removed status code 411, since it cannot occur (Jonathan Rosenberg, James Jack).
- 4842 ● Rewrote **Record-Route** section to reflect new mechanism. In particular, requests from callee to caller
4843 now use the same path as in the opposite direction, without substituting the **From** header field values.
4844 The **maddr** parameter is now optional.
- 4845 ● Disallowed SIP URLs that only have a password, without a user name. The prototype from RFC 1738
4846 also doesn’t allow this.
- 4847 ● Allow registrar to set the expiration time.
- 4848 ● **CSeq** (Section 22.16) is counted within a call leg, not a call.
- 4849 ● Removed wording that connection closing is equivalent to **CANCEL** or 500. This does not work for
4850 connections that are used for multiple transactions and has other problems.
- 4851 ● Cleaned up **CSeq** section. Removed text about inserting **CSeq** method when it is absent. Clarified
4852 that **CSeq** increments for all requests, not just *invite*. Clarified that all out of order requests, not
4853 just out of order *INVITE*, are rejected with a 400 class response. Clarified the meaning of “initial”
4854 sequence number. Clarified that after a request forks, each 200 OK is a separate call leg, and thus,
4855 separate **CSeq** space. Clarified that **CSeq** numbers are independent for each direction of a call leg.

- 4856 • Massive reorganization and cleanup of the SDP section. Introduced the concept of the offer-answer
4857 model. Clarified that set of codecs in m line are usable all at the same time. Inserted size restriction
4858 on representation of values in o line. Explicitly describe forked media. New media lines for adding
4859 streams appear at the bottom of the SDP (used to say append).
- 4860 • Removed Also.
- 4861 • Added text to Require and Proxy-Require sections, making it a SHOULD to retry the request without
4862 the unsupported extension.
- 4863 • Added text to section on 415, saying that UAC SHOULD retry the request without the unsupported
4864 body.
- 4865 • Added text to section on CANCEL and ACK, clarifying much of the behavior.
- 4866 • Modified Content-Type to indicate that it can be present even if the body is empty.
- 4867 • From tags mandatory
- 4868 • Old text said that if you hang up before sending an ACK, you need not send the ACK. That is wrong.
4869 Text fixed so that an ACK is always sent.
- 4870 • Old text said that if you never got a response to an INVITE, the UAC should send both an INVITE and
4871 CANCEL. This doesn't make sense. Rahter, it should do nothing and consider the call terminated.
- 4872 • Added text that says pending requests are responded to with a 487 if a BYE is received.
- 4873 • Updated section 2.2, so that its clear that Contact is not used with BYE.
- 4874 • Clarified Via processing rules. Added text on handling loops when proxies route on headers besides
4875 the request URI. Added text on handling case when sent-by contains a domain name. Added text to
4876 6.47 on opening TCP connections to send responses upstream.
- 4877 • Clarified that a lxx with an unknown xx is not the same as the 100 response.
- 4878 • Removed usage of Retry-After in REGISTER.
- 4879 • Clarified usage of persistent connections.
- 4880 • Clarified that servers supporting HTTP basic or digest in rfc2617 MUST be backwards compatible
4881 with RFC 2069.
- 4882 • Clarified that ACK contains the same branch ID as the request its acknowledging.
- 4883 • Added definitions for spiral, B2BUA.
- 4884 • Rephrased definitions for UAC, UAS, Call, call-leg, caller, callee, making them more concrete.
- 4885 • URL comparison ignores parameters not present in both URLs only for unknown parameters.
- 4886 • Clarified that * in Contact is used only in REGISTER with Expires header zero. Mentioned * case
4887 in section on Contact syntax.

- 4888 • Removed text that says a UA can insert a **Contact** in 2xx that indicates the address of a proxy. Not
4889 likely to work in general.
- 4890 • Removed SDP text about aligning media streams within a media type to handle certain crash and
4891 restart cases.
- 4892 • Receiving a 481 to a mid-call request terminates that call leg. Agreed upon at IETF 49.
- 4893 • Introduced definition of regular transaction - non-INVITE excepting **ACK** and **CANCEL**.
- 4894 • Clarified rules for overlapping transactions.
- 4895 • Forking proxies **MUST** be stateful (used to say **SHOULD**). Proxies that send requests on multicast
4896 **MUST** be stateful (used to say nothing)
- 4897 • Text added recommending that registrars authorize that entity in **From** field can register address-of-
4898 record in the **To** field.
- 4899 • Forwarding of non-100 provisionals upstream in a proxy changed from **SHOULD** to **MUST**.
- 4900 • Removed PGP.

4901 **32 Changes Made in Version 04**

- 4902 • Removed **Unsupported** as a request header from Table 3.
- 4903 • Clarified SDP procedures for changing IP address and port. Specifically, spelled out the duration for
4904 which a UA needs to received media on the old port and address.
- 4905 • Added text in the SDP session which recommends that the answerer use the same ordering of codecs
4906 as used on the offer, in order to help ensure symmetric codec operation under normal conditions.
- 4907 • Fixed bug in the example in the SDP section, where the new media line was listed at the top. Should
4908 have been the bottom.
- 4909 • **Authorization** credentials are cached based on the URL of the **To** header, not the entire **To** header as
4910 10.48 implied.
- 4911 • Section 10.31, on **Proxy-Authenticate**, indicated that a server responds with a 401 if the client
4912 guessed wrong. This is incorrect. It should be 407.
- 4913 • Section 10.14, removed motivational text about **Contact** allowing an INVITE to be routed directly
4914 between end systems, since its confusing. Some have interpreted to mean that **Record-Route** is
4915 ignored when **Contact** is present.
- 4916 • Added reference to SCTP RFC.
- 4917 • Updated 2.2 to allow non-SIP URLs in **OPTIONS** and 2xx to **OPTIONS**.
- 4918 • Fixed example in 20.5. Added **ACK** for 487, and added **To** tag to 487 response.

- 4919 ● Clarified further URL comparisons. Its only URL parameters without defaults that are ignored if not
4920 present in both URLs.
- 4921 ● Section 1.5.2, UDP mandatory for all. TCP is a SHOULD for UA, MUST for proxy, registrar, redirect
4922 servers.
- 4923 ● Brought syntax for Contact, Via, and the SIP URL into alignment between the text and postscript
4924 versions.
- 4925 ● Updated the text in section 6 which said that the ordering of header fields follows HTTP, with the
4926 exception of Via, where order matters. However, the HTTP spec says that order matters, so this
4927 sentence is redundant and confusing. The sentence was removed.
- 4928 ● Added e lines to SDP examples in the Examples section.
- 4929 ● Rewrote Allow discussion, more formally defining its semantics and usage cases.
- 4930 ● Updated text on 604 status, to indicate that its based on the Request-URI, not the To.
- 4931 ● Added response registrations to IANA considerations. Provided more details on registration process.
- 4932 ● Clarified that only a UAS rejects a request because the To tag doesn't match a local value.
- 4933 ● Clarified that stateless proxies need to route based on static criteria only.
- 4934 ● Proxy and UAC CANCEL generation upon 2xx, 6xx if it forked is now a SHOULD; used to be a MAY.
- 4935 ● Added text saying that a UAS SHOULD send a BYE if it never gets an ACK for a 2xx establishing a
4936 call leg.
- 4937 ● Added text saying that a UAS SHOULD send a re-INVITE if it never gets an ACK for a 2xx to a
4938 re-INVITE.
- 4939 ● Added text on 503 processing, indicating that a client should try a different server when receiving a
4940 503, and that a proxy shouldn't forward a 503 upstream unless it can't service any other requests.
- 4941 ● Removed motivational text in Section 10.43 on Via headers since its not consistent with the text before
4942 it.
- 4943 ● Changed IPsec reference to RFC2401, from RFC1825.
- 4944 ● Updated retransmission definition in 17.3.4 to be consistent with the rest of the spec.
- 4945 ● Softened the language for insertion of the transport param in the record-route. Specifically, it can be
4946 inserted in private networks where it is known apriori that the specific transport is supported.
- 4947 ● Updated definition of B2BUA.
- 4948 ● Added text to section on 420 processing, which mandates that the client retry the request without
4949 extensions listed in the Unsupported header in the response.
- 4950 ● Allow Authentication-Info header to be used for HTTP digest.

33 Changes Made in Version 05

4951
4952
4953
4954
4955
4956
4957
4958
4959
4960
4961
4962
4963
4964
4965
4966
4967
4968
4969
4970
4971
4972
4973
4974
4975
4976
4977
4978
4979
4980
4981
4982
4983
4984

- Updated Table 2 to reflect that **Error-Info** is a response header in 3xx-6xx responses (it was previously listed as a request header).
- Removed **WWW-Authenticate** as a request header from Table 3. Authentication of responses is now done according to RFC2617.
- Updated the **Accept**, **Accept-Encoding** and **Accept-Language** sections. More details on precise semantics for the various requests and responses is now provided. Presence of these headers is now a **SHOULD** for **INVITE** and **2xx** to **INVITE** when a non-default value is present. Extra emphasis is placed on including the **Accept-Language** in **INVITE** and **2xx** in order to support internationalization. Usage of these three headers in **CANCEL** has been removed since it makes no sense.
- Generalized local outbound processing rules in Section 16.4.1 to cover the case where the UAS is using a local outbound proxy which was not in the initial call setup path.
- Updated record-routing section, so that a proxy can insert a transport param if it knows that the proxy on one side supports the specific transport (the previous text required the proxy to know whether the proxies on both sides supported the specific transport).
- Added **Authentication-Info** to Section 10.
- Clarified the meaning of Table 2 for responses.
- Updated Table 1 to reflect that **maddr** is no longer mandatory in **Record-Route**.
- Updated Table 3 so that header fields in responses to **ACK** are never listed as optional, mandatory, etc. - only not applicable. This is because responses to **ACK** are not allowed. Also improved wording in Section 5.1.1 to clarify that there **MUST NOT** be responses to **ACK**.
- Updated SRV procedures. Old text said to treat a failure to contact a server as a 4xx, which would stop the SRV processing. But, this is not so. Sentence was stricken.
- Updated 12.1 to clarify that **2xx INVITE** responses **MUST** contain session descriptions.
- Changed **User-Agent** to a request header in Table 3.
- Updated SDP section, so that a UA cannot change the SDP when it gets a re-**INVITE** with no SDP.
- Clarified Appendix B that a unicast offer **MUST** have a unicast response.
- Clarified that any request can be record-routed, but it may not be used by the UA, depending on the method.
- non-**2xx** responses to **INVITE** no longer retransmitted over TCP.
- Removed lower bound on **T1** and **T2** in private networks, which can use lower values. Furthermore, **T1** can be smaller on the public Internet if proper RTT estimation is used.
- UAS Cannot send a **BYE** for a call leg until it receives **ACK**, in order to eliminate a race condition between **BYE** and **200 OK**.

- 4985 • Support of CR or LF alone as line terminators, as opposed to CRLF, is no longer required.
- 4986 • Client behavior on receipt of a 3xx to re-INVITE is now specified, and it is no longer forbidden to
4987 generate a 3xx. This is needed to maintain the idempotency of INVITE, as a proxy might redirect
4988 without knowing its a 3xx.
- 4989 • CANCEL cannot be sent before a 1xx is received, in order to eliminate race condition between request
4990 and CANCEL.
- 4991 • Termination of the client and server transactions is now based entirely on timeouts, rather than re-
4992 transmission counters, in order to unify TCP and UDP behavior. Timeout values scale as a function
4993 of the RTT estimate, defined as T1. For reliable transports, many of these timers are now set to zero.
4994 Many timeouts differ than in bis-04.
- 4995 • Added a working RTT estimation algorithm using the Timestamp header, and specified it to be
4996 compliant to RFC 2988.
- 4997 • UAS accepting requests with unknown schemes in the URI in the To field is now a RECOMMENDED
4998 instead of SHOULD. This reflects the fact that processing a request when the To field doesn't match is
4999 a matter of policy.
- 5000 • Bodies are now allowed in any request and response, including CANCEL, although there may not be
5001 any semantics associated with that.
- 5002 • Supporting of INVITE without SDP is now a MUST (no strength was previously specified).
- 5003 • Registration procedures for visiting, which had a few sentences in bis-04, have been removed. Roam-
5004 ing is a complex issue, and should be treated elsewhere.
- 5005 • Bis-04 mandated that a 2xx response to REGISTER contain expires Contact parameters indicating
5006 the expiration time of a contact. This behavior has now been made consistent with requests, so that
5007 the expiration time of a contact is the same in either case: the expires param is used first if present,
5008 then the Expires header if present, else one hour for SIP URLs.
- 5009 • Action parameter in contact registrations is deprecated.
- 5010 • 2xx to REGISTER MUST contain current contacts. This was just a SHOULD in bis-04.
- 5011 • Multicast operation radically changed. Now, the treatment is no different than unicast. That is, only
5012 the first non-1xx response to a multicast request will be used. This is a natural consequence of the
5013 layering now applied to the protocol. This still enables anycast types of functions, mirroring the real
5014 usage of registrar discovery.
- 5015 • To completely separate transport rules from transaction rules, the rule in bis-04 that said a UAC
5016 SHOULD keep a connection opened until a response is received, has been turned into a timer recom-
5017 mendation. Specifically, the spec now says that it is RECOMMENDED that connections be kept opened
5018 for a minimum interval of sufficient duration to guarantee, with high probability, that responses are
5019 sent over the same connections as a request.

- 5020 • Re-use of existing connections for new requests to the same address and port is now RECOMMENDED,
5021 it was only a MAY in bis-04.
- 5022 • Modification of headers below the Authorization header by proxies is no longer disallowed, since the
5023 only mechanism that used Authorization in that way, PGP, has been deprecated previously.
- 5024 • Authentication of registrations now RECOMMENDED; no strength was defined previously.
- 5025 • Registering of new headers with IANA is now SHOULD; no strength was defined previously.
- 5026 • Proxy aggregation of challenges now a SHOULD; no strength was defined previously.
- 5027 • Server support of basic authentication downgraded from SHOULD to MAY.
- 5028 • UAC resubmitting requests with credentials after a challenge upgraded from MAY to SHOULD.
- 5029 • TLS is now RECOMMENDED as the transport layer security for SIP signaling.
- 5030 • UA recursion on a redirect is now SHOULD; no strength was assigned previously.
- 5031 • UA reuse of headers in a recursed request is now SHOULD; no strength was assigned previously.
- 5032 • Security considerations added for Call-Info and Alert-Info.
- 5033 • Proxies no longer forward a 6xx immediately on receiving it. Instead, they CANCEL pending
5034 branches immediately. This avoids a potential race condition that would result in a UAC getting a
5035 6xx followed by a 2xx. In all cases except this race condition, the result will be the same - the 6xx is
5036 forwarded upstream.
- 5037 • The term call-leg has been eliminated from the spec; a more generic term, dialog, is used in its place.
- 5038 • For SRV processing, subsequent requests with the same Call-ID (as opposed to the same transaction
5039 in bis-04) are sent to the same server.
- 5040 • SRV processing generalized to deal with the fact that the default port is transport dependent.
- 5041 • Per IESG request, draft-ietf-sip-serverfeatures has been integrated into bis.
- 5042 • Per IESG request, draft-ietf-sip-100rel will be integrated into bis. This is marked with a placeholder
5043 in this draft.
- 5044 • The BNF has been converted from implicit LWS to explicit LWS.
- 5045 • Caching of responses in a proxy to avoid redoing location server lookups used to be a SHOULD.
5046 Caching behavior for responses is now fully encapsulated in the transaction processing.
- 5047 • Proxy usage of SRV in processing Route headers upgraded from SHOULD to MUST.

5048 **34 Acknowledgments**

5049 We wish to thank the members of the IETF MMUSIC and SIP WGs for their comments and suggestions.
5050 Detailed comments were provided by Brian Bidulock, Jim Buller, Neil Deason, Dave Devanathan, Cdric
5051 Fluckiger, Yaron Goland, Bernie Hneisen, Phil Hoffer, Christian Huitema, Jean Jervis, Gadi Karmi, Peter
5052 Kjellerstedt, Anders Kristensen, Jonathan Lennox, Gethin Liddell, Keith Moore, Vern Paxson, Moshe J.
5053 Sambol, Chip Sharp, Igor Slepchin, Robert Sparks, Eric Tremblay., and Rick Workman.

5054 Brian Rosen provided the compiled BNF.

5055 This work is based, inter alia, on [46, 47].

5056 **35 Authors' Addresses**

5057 Authors addresses are listed alphabetically for the editors, the writers, and then the original authors of RFC
5058 2543.

5059 Jonathan Rosenberg
5060 dynamicsoft
5061 72 Eagle Rock Ave
5062 East Hanover, NJ 07936
5063 USA
5064 electronic mail: jdrosen@dynamicsoft.com

5065 Henning Schulzrinne
5066 Dept. of Computer Science
5067 Columbia University
5068 1214 Amsterdam Avenue
5069 New York, NY 10027
5070 USA
5071 electronic mail: schulzrinne@cs.columbia.edu

5072 Gonzalo Camarillo
5073 Ericsson
5074 Advanced Signalling Research Lab.
5075 FIN-02420 Jorvas
5076 Finland
5077 electronic mail: Gonzalo.Camarillo@ericsson.com

5078 Alan Johnston
5079 WorldCom
5080 100 South 4th Street
5081 St. Louis, MO 63102
5082 USA
5083 electronic mail: alan.johnston@wcom.com

5084 Jon Peterson
5085 NeuStar, Inc

5086 1800 Sutter Street, Suite 570
5087 Concord, CA 94520
5088 USA
5089 electronic mail: jon.peterson@neustar.com

5090 Robert Sparks
5091 dynamicsoft, Inc.
5092 5100 Tennyson Parkway
5093 Suite 1200
5094 Plano, Texas 75024
5095 USA
5096 electronic mail: rsparks@dynamicsoft.com

5097 Mark Handley
5098 ACIRI
5099 electronic mail: mjh@aciri.org

5100 Eve Schooler
5101 Computer Science Department 256-80
5102 California Institute of Technology
5103 Pasadena, CA 91125
5104 USA
5105 electronic mail: schooler@cs.caltech.edu

5106 **References**

- 5107 [1] R. Pandya, "Emerging mobile and personal communication systems," *IEEE Communications Maga-*
5108 *zine*, Vol. 33, pp. 44–52, June 1995.
- 5109 [2] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation protocol
5110 (RSVP) – version 1 functional specification," Request for Comments 2205, Internet Engineering Task
5111 Force, Sept. 1997.
- 5112 [3] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: a transport protocol for real-time
5113 applications," Request for Comments 1889, Internet Engineering Task Force, Jan. 1996.
- 5114 [4] H. Schulzrinne, A. Rao, and R. Lanphier, "Real time streaming protocol (RTSP)," Request for Com-
5115 ments 2326, Internet Engineering Task Force, Apr. 1998.
- 5116 [5] M. Handley, C. Perkins, and E. Whelan, "Session announcement protocol," Request for Comments
5117 2974, Internet Engineering Task Force, Oct. 2000.
- 5118 [6] M. Handley and V. Jacobson, "SDP: session description protocol," Request for Comments 2327, Inter-
5119 net Engineering Task Force, Apr. 1998.
- 5120 [7] S. Bradner, "Key words for use in RFCs to indicate requirement levels," Request for Comments 2119,
5121 Internet Engineering Task Force, Mar. 1997.

- 5122 [8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext
5123 transfer protocol – HTTP/1.1," Request for Comments 2616, Internet Engineering Task Force, June
5124 1999.
- 5125 [9] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform resource identifiers (URI): generic syntax,"
5126 Request for Comments 2396, Internet Engineering Task Force, Aug. 1998.
- 5127 [10] T. Berners-Lee, L. Masinter, and M. McCahill, "Uniform resource locators (URL)," Request for Com-
5128 ments 1738, Internet Engineering Task Force, Dec. 1994.
- 5129 [11] F. Yergeau, "UTF-8, a transformation format of ISO 10646," Request for Comments 2279, Internet
5130 Engineering Task Force, Jan. 1998.
- 5131 [12] D. Crocker, "Standard for the format of ARPA internet text messages," Request for Comments 822,
5132 Internet Engineering Task Force, Aug. 1982.
- 5133 [13] A. Vaha-Sipila, "URLs for telephone calls," Request for Comments 2806, Internet Engineering Task
5134 Force, Apr. 2000.
- 5135 [14] N. Freed and N. Borenstein, "Multipurpose internet mail extensions (MIME) part two: Media types,"
5136 Request for Comments 2046, Internet Engineering Task Force, Nov. 1996.
- 5137 [15] W. R. Stevens, *TCP/IP illustrated: the protocols*, Vol. 1. Reading, Massachusetts: Addison-Wesley,
5138 1994.
- 5139 [16] J. C. Mogul and S. E. Deering, "Path MTU discovery," Request for Comments 1191, Internet Engi-
5140 neering Task Force, Nov. 1990.
- 5141 [17] D. Eastlake, S. Crocker, and J. Schiller, "Randomness recommendations for security," Request for
5142 Comments 1750, Internet Engineering Task Force, Dec. 1994.
- 5143 [18] P. Hoffman, L. Masinter, and J. Zawinski, "The mailto URL scheme," Request for Comments 2368,
5144 Internet Engineering Task Force, July 1998.
- 5145 [19] D. Meyer, "Administratively scoped IP multicast," Request for Comments 2365, Internet Engineering
5146 Task Force, July 1998.
- 5147 [20] E. M. Schooler, "A multicast user directory service for synchronous rendezvous," Master's Thesis CS-
5148 TR-96-18, Department of Computer Science, California Institute of Technology, Pasadena, California,
5149 Aug. 1996.
- 5150 [21] S. Donovan, "The SIP INFO method," Request for Comments 2976, Internet Engineering Task Force,
5151 Oct. 2000.
- 5152 [22] J. Rosenberg and H. Schulzrinne, "An offer/answer model with sdp," Internet Draft, Internet Engineer-
5153 ing Task Force, Oct. 2001. Work in progress.
- 5154 [23] R. Rivest, "The MD5 message-digest algorithm," Request for Comments 1321, Internet Engineering
5155 Task Force, Apr. 1992.

- 5156 [24] V. Paxson and M. Allman, "Computing TCP's retransmission timer," Request for Comments 2988,
5157 Internet Engineering Task Force, Nov. 2000.
- 5158 [25] T. Dierks and C. Allen, "The TLS protocol version 1.0," Request for Comments 2246, Internet Engi-
5159 neering Task Force, Jan. 1999.
- 5160 [26] S. Kent and R. Atkinson, "Security architecture for the internet protocol," Request for Comments 2401,
5161 Internet Engineering Task Force, Nov. 1998.
- 5162 [27] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart, "HTTP
5163 authentication: Basic and digest access authentication," Request for Comments 2617, Internet Engi-
5164 neering Task Force, June 1999.
- 5165 [28] J. Franks, P. Hallam-Baker, J. Hostetler, P. Leach, A. Luotonen, E. Sink, and L. Stewart, "An exten-
5166 sion to HTTP : Digest access authentication," Request for Comments 2069, Internet Engineering Task
5167 Force, Jan. 1997.
- 5168 [29] J. Galvin, S. Murphy, S. Crocker, and N. Freed, "Security multipart for MIME: multipart/signed and
5169 multipart/encrypted," Request for Comments 1847, Internet Engineering Task Force, Oct. 1995.
- 5170 [30] J. Postel, "User datagram protocol," Request for Comments 768, Internet Engineering Task Force,
5171 Aug. 1980.
- 5172 [31] J. Postel, "DoD standard transmission control protocol," Request for Comments 761, Internet Engi-
5173 neering Task Force, Jan. 1980.
- 5174 [32] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang,
5175 and V. Paxson, "Stream control transmission protocol," Request for Comments 2960, Internet Engi-
5176 neering Task Force, Oct. 2000.
- 5177 [33] F. Dawson and T. Howes, "vcard MIME directory profile," Request for Comments 2426, Internet
5178 Engineering Task Force, Sept. 1998.
- 5179 [34] G. Good, "The LDAP data interchange format (LDIF) - technical specification," Request for Com-
5180 ments 2849, Internet Engineering Task Force, June 2000.
- 5181 [35] R. Troost and S. Dorner, "Communicating presentation information in internet messages: The content-
5182 disposition header," Request for Comments 1806, Internet Engineering Task Force, June 1995.
- 5183 [36] R. Braden and Ed, "Requirements for internet hosts - application and support," Request for Comments
5184 1123, Internet Engineering Task Force, Oct. 1989.
- 5185 [37] J. Palme, "Common internet message headers," Request for Comments 2076, Internet Engineering
5186 Task Force, Feb. 1997.
- 5187 [38] H. Alvestrand, "IETF policy on character sets and languages," Request for Comments 2277, Internet
5188 Engineering Task Force, Jan. 1998.
- 5189 [39] G. Nair and H. Schulzrinne, "DHCP option for SIP servers," Internet Draft, Internet Engineering Task
5190 Force, Mar. 2001. Work in progress.

- 5191 [40] A. Gulbrandsen, P. Vixie, and L. Esibov, "A DNS RR for specifying the location of services (DNS
5192 SRV)," Request for Comments 2782, Internet Engineering Task Force, Feb. 2000.
- 5193 [41] P. V. Mockapetris, "Domain names - implementation and specification," Request for Comments 1035,
5194 Internet Engineering Task Force, Nov. 1987.
- 5195 [42] A. Johnston, S. Donovan, R. Sparks, C. Cunningham, D. Willis, J. Rosenberg, K. Summers, and
5196 H. Schulzrinne, "SIP telephony call flow examples," Internet Draft, Internet Engineering Task Force,
5197 Apr. 2001. Work in progress.
- 5198 [43] D. Crocker, Ed., and P. Overell, "Augmented BNF for syntax specifications: ABNF," Request for
5199 Comments 2234, Internet Engineering Task Force, Nov. 1997.
- 5200 [44] H. Schulzrinne, "RTP profile for audio and video conferences with minimal control," Request for
5201 Comments 1890, Internet Engineering Task Force, Jan. 1996.
- 5202 [45] R. Hinden, B. Carpenter, and L. Masinter, "Format for literal IPv6 addresses in URL's," Request for
5203 Comments 2732, Internet Engineering Task Force, Dec. 1999.
- 5204 [46] E. M. Schooler, "Case study: multimedia conference control in a packet-switched teleconferencing
5205 system," *Journal of Internetworking: Research and Experience*, Vol. 4, pp. 99–120, June 1993. ISI
5206 reprint series ISI/RS-93-359.
- 5207 [47] H. Schulzrinne, "Personal mobility for multimedia services in the Internet," in *European Workshop on*
5208 *Interactive Distributed Multimedia Systems and Services (IDMS)*, (Berlin, Germany), Mar. 1996.

5209 Full Copyright Statement

5210 Copyright (c) The Internet Society (2001). All Rights Reserved.

5211 This document and translations of it may be copied and furnished to others, and derivative works that
5212 comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and
5213 distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and
5214 this paragraph are included on all such copies and derivative works. However, this document itself may not
5215 be modified in any way, such as by removing the copyright notice or references to the Internet Society or
5216 other Internet organizations, except as needed for the purpose of developing Internet standards in which case
5217 the procedures for copyrights defined in the Internet Standards process must be followed, or as required to
5218 translate it into languages other than English.

5219 The limited permissions granted above are perpetual and will not be revoked by the Internet Society or
5220 its successors or assigns.

5221 This document and the information contained herein is provided on an "AS IS" basis and THE IN-
5222 TERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WAR-
5223 RANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT
5224 THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED
5225 WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.