Network Working Group J. Rosenberg

Request for Comments: 3261 dynamicsoft

Obsoletes: 2543 H. Schulzrinne

Category: Standards Track Columbia U.

G. Camarillo

Ericsson

A. Johnston

WorldCom

J. Peterson

Neustar

R. Sparks

dynamicsoft

M. Handley

ICIR

E. Schooler

AT&T

June 2002

SIP: Session Initiation Protocol

Status of this Memo

This document specifies an Internet standards track protocol for the

Internet community, and requests discussion and suggestions for

improvements. Please refer to the current edition of the "Internet

Official Protocol Standards" (STD 1) for the standardization state

and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

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

Abstract

This document describes Session Initiation Protocol (SIP), an

application-layer control (signaling) protocol for creating,

modifying, and terminating sessions with one or more participants.

These sessions include Internet telephone calls, multimedia

distribution, and multimedia conferences.

SIP invitations used to create sessions carry session descriptions

that allow participants to agree on a set of compatible media types.

SIP makes use of elements called proxy servers to help route requests

to the user's current location, authenticate and authorize users for

services, implement provider call-routing policies, and provide

features to users. SIP also provides a registration function that

allows users to upload their current locations for use by proxy

servers. SIP runs on top of several different transport protocols.

Rosenberg, et. al. Standards Track [Page 1]

RFC 3261 SIP: Session Initiation Protocol June 2002

Table of Contents

1 Introduction ........................................ 8

2 Overview of SIP Functionality ....................... 9

3 Terminology ......................................... 10

4 Overview of Operation ............................... 10

5 Structure of the Protocol ........................... 18

6 Definitions ......................................... 20

7 SIP Messages ........................................ 26

7.1 Requests ............................................ 27

7.2 Responses ........................................... 28

7.3 Header Fields ....................................... 29

7.3.1 Header Field Format ................................. 30

7.3.2 Header Field Classification ......................... 32

7.3.3 Compact Form ........................................ 32

7.4 Bodies .............................................. 33

7.4.1 Message Body Type ................................... 33

7.4.2 Message Body Length ................................. 33

7.5 Framing SIP Messages ................................ 34

8 General User Agent Behavior ......................... 34

8.1 UAC Behavior ........................................ 35

8.1.1 Generating the Request .............................. 35

8.1.1.1 Request-URI ......................................... 35

8.1.1.2 To .................................................. 36

8.1.1.3 From ................................................ 37

8.1.1.4 Call-ID ............................................. 37

8.1.1.5 CSeq ................................................ 38

8.1.1.6 Max-Forwards ........................................ 38

8.1.1.7 Via ................................................. 39

8.1.1.8 Contact ............................................. 40

8.1.1.9 Supported and Require ............................... 40

8.1.1.10 Additional Message Components ....................... 41

8.1.2 Sending the Request ................................. 41

8.1.3 Processing Responses ................................ 42

8.1.3.1 Transaction Layer Errors ............................ 42

8.1.3.2 Unrecognized Responses .............................. 42

8.1.3.3 Vias ................................................ 43

8.1.3.4 Processing 3xx Responses ............................ 43

8.1.3.5 Processing 4xx Responses ............................ 45

8.2 UAS Behavior ........................................ 46

8.2.1 Method Inspection ................................... 46

8.2.2 Header Inspection ................................... 46

8.2.2.1 To and Request-URI .................................. 46

8.2.2.2 Merged Requests ..................................... 47

8.2.2.3 Require ............................................. 47

8.2.3 Content Processing .................................. 48

8.2.4 Applying Extensions ................................. 49

8.2.5 Processing the Request .............................. 49

Rosenberg, et. al. Standards Track [Page 2]

RFC 3261 SIP: Session Initiation Protocol June 2002

8.2.6 Generating the Response ............................. 49

8.2.6.1 Sending a Provisional Response ...................... 49

8.2.6.2 Headers and Tags .................................... 50

8.2.7 Stateless UAS Behavior .............................. 50

8.3 Redirect Servers .................................... 51

9 Canceling a Request ................................. 53

9.1 Client Behavior ..................................... 53

9.2 Server Behavior ..................................... 55

10 Registrations ....................................... 56

10.1 Overview ............................................ 56

10.2 Constructing the REGISTER Request ................... 57

10.2.1 Adding Bindings ..................................... 59

10.2.1.1 Setting the Expiration Interval of Contact Addresses 60

10.2.1.2 Preferences among Contact Addresses ................. 61

10.2.2 Removing Bindings ................................... 61

10.2.3 Fetching Bindings ................................... 61

10.2.4 Refreshing Bindings ................................. 61

10.2.5 Setting the Internal Clock .......................... 62

10.2.6 Discovering a Registrar ............................. 62

10.2.7 Transmitting a Request .............................. 62

10.2.8 Error Responses ..................................... 63

10.3 Processing REGISTER Requests ........................ 63

11 Querying for Capabilities ........................... 66

11.1 Construction of OPTIONS Request ..................... 67

11.2 Processing of OPTIONS Request ....................... 68

12 Dialogs ............................................. 69

12.1 Creation of a Dialog ................................ 70

12.1.1 UAS behavior ........................................ 70

12.1.2 UAC Behavior ........................................ 71

12.2 Requests within a Dialog ............................ 72

12.2.1 UAC Behavior ........................................ 73

12.2.1.1 Generating the Request .............................. 73

12.2.1.2 Processing the Responses ............................ 75

12.2.2 UAS Behavior ........................................ 76

12.3 Termination of a Dialog ............................. 77

13 Initiating a Session ................................ 77

13.1 Overview ............................................ 77

13.2 UAC Processing ...................................... 78

13.2.1 Creating the Initial INVITE ......................... 78

13.2.2 Processing INVITE Responses ......................... 81

13.2.2.1 1xx Responses ....................................... 81

13.2.2.2 3xx Responses ....................................... 81

13.2.2.3 4xx, 5xx and 6xx Responses .......................... 81

13.2.2.4 2xx Responses ....................................... 82

13.3 UAS Processing ...................................... 83

13.3.1 Processing of the INVITE ............................ 83

13.3.1.1 Progress ............................................ 84

13.3.1.2 The INVITE is Redirected ............................ 84

Rosenberg, et. al. Standards Track [Page 3]

RFC 3261 SIP: Session Initiation Protocol June 2002

13.3.1.3 The INVITE is Rejected .............................. 85

13.3.1.4 The INVITE is Accepted .............................. 85

14 Modifying an Existing Session ....................... 86

14.1 UAC Behavior ........................................ 86

14.2 UAS Behavior ........................................ 88

15 Terminating a Session ............................... 89

15.1 Terminating a Session with a BYE Request ............ 90

15.1.1 UAC Behavior ........................................ 90

15.1.2 UAS Behavior ........................................ 91

16 Proxy Behavior ...................................... 91

16.1 Overview ............................................ 91

16.2 Stateful Proxy ...................................... 92

16.3 Request Validation .................................. 94

16.4 Route Information Preprocessing ..................... 96

16.5 Determining Request Targets ......................... 97

16.6 Request Forwarding .................................. 99

16.7 Response Processing ................................. 107

16.8 Processing Timer C .................................. 114

16.9 Handling Transport Errors ........................... 115

16.10 CANCEL Processing ................................... 115

16.11 Stateless Proxy ..................................... 116

16.12 Summary of Proxy Route Processing ................... 118

16.12.1 Examples ............................................ 118

16.12.1.1 Basic SIP Trapezoid ................................. 118

16.12.1.2 Traversing a Strict-Routing Proxy ................... 120

16.12.1.3 Rewriting Record-Route Header Field Values .......... 121

17 Transactions ........................................ 122

17.1 Client Transaction .................................. 124

17.1.1 INVITE Client Transaction ........................... 125

17.1.1.1 Overview of INVITE Transaction ...................... 125

17.1.1.2 Formal Description .................................. 125

17.1.1.3 Construction of the ACK Request ..................... 129

17.1.2 Non-INVITE Client Transaction ....................... 130

17.1.2.1 Overview of the non-INVITE Transaction .............. 130

17.1.2.2 Formal Description .................................. 131

17.1.3 Matching Responses to Client Transactions ........... 132

17.1.4 Handling Transport Errors ........................... 133

17.2 Server Transaction .................................. 134

17.2.1 INVITE Server Transaction ........................... 134

17.2.2 Non-INVITE Server Transaction ....................... 137

17.2.3 Matching Requests to Server Transactions ............ 138

17.2.4 Handling Transport Errors ........................... 141

18 Transport ........................................... 141

18.1 Clients ............................................. 142

18.1.1 Sending Requests .................................... 142

18.1.2 Receiving Responses ................................. 144

18.2 Servers ............................................. 145

18.2.1 Receiving Requests .................................. 145

Rosenberg, et. al. Standards Track [Page 4]

RFC 3261 SIP: Session Initiation Protocol June 2002

18.2.2 Sending Responses ................................... 146

18.3 Framing ............................................. 147

18.4 Error Handling ...................................... 147

19 Common Message Components ........................... 147

19.1 SIP and SIPS Uniform Resource Indicators ............ 148

19.1.1 SIP and SIPS URI Components ......................... 148

19.1.2 Character Escaping Requirements ..................... 152

19.1.3 Example SIP and SIPS URIs ........................... 153

19.1.4 URI Comparison ...................................... 153

19.1.5 Forming Requests from a URI ......................... 156

19.1.6 Relating SIP URIs and tel URLs ...................... 157

19.2 Option Tags ......................................... 158

19.3 Tags ................................................ 159

20 Header Fields ....................................... 159

20.1 Accept .............................................. 161

20.2 Accept-Encoding ..................................... 163

20.3 Accept-Language ..................................... 164

20.4 Alert-Info .......................................... 164

20.5 Allow ............................................... 165

20.6 Authentication-Info ................................. 165

20.7 Authorization ....................................... 165

20.8 Call-ID ............................................. 166

20.9 Call-Info ........................................... 166

20.10 Contact ............................................. 167

20.11 Content-Disposition ................................. 168

20.12 Content-Encoding .................................... 169

20.13 Content-Language .................................... 169

20.14 Content-Length ...................................... 169

20.15 Content-Type ........................................ 170

20.16 CSeq ................................................ 170

20.17 Date ................................................ 170

20.18 Error-Info .......................................... 171

20.19 Expires ............................................. 171

20.20 From ................................................ 172

20.21 In-Reply-To ......................................... 172

20.22 Max-Forwards ........................................ 173

20.23 Min-Expires ......................................... 173

20.24 MIME-Version ........................................ 173

20.25 Organization ........................................ 174

20.26 Priority ............................................ 174

20.27 Proxy-Authenticate .................................. 174

20.28 Proxy-Authorization ................................. 175

20.29 Proxy-Require ....................................... 175

20.30 Record-Route ........................................ 175

20.31 Reply-To ............................................ 176

20.32 Require ............................................. 176

20.33 Retry-After ......................................... 176

20.34 Route ............................................... 177

Rosenberg, et. al. Standards Track [Page 5]

RFC 3261 SIP: Session Initiation Protocol June 2002

20.35 Server .............................................. 177

20.36 Subject ............................................. 177

20.37 Supported ........................................... 178

20.38 Timestamp ........................................... 178

20.39 To .................................................. 178

20.40 Unsupported ......................................... 179

20.41 User-Agent .......................................... 179

20.42 Via ................................................. 179

20.43 Warning ............................................. 180

20.44 WWW-Authenticate .................................... 182

21 Response Codes ...................................... 182

21.1 Provisional 1xx ..................................... 182

21.1.1 100 Trying .......................................... 183

21.1.2 180 Ringing ......................................... 183

21.1.3 181 Call Is Being Forwarded ......................... 183

21.1.4 182 Queued .......................................... 183

21.1.5 183 Session Progress ................................ 183

21.2 Successful 2xx ...................................... 183

21.2.1 200 OK .............................................. 183

21.3 Redirection 3xx ..................................... 184

21.3.1 300 Multiple Choices ................................ 184

21.3.2 301 Moved Permanently ............................... 184

21.3.3 302 Moved Temporarily ............................... 184

21.3.4 305 Use Proxy ....................................... 185

21.3.5 380 Alternative Service ............................. 185

21.4 Request Failure 4xx ................................. 185

21.4.1 400 Bad Request ..................................... 185

21.4.2 401 Unauthorized .................................... 185

21.4.3 402 Payment Required ................................ 186

21.4.4 403 Forbidden ....................................... 186

21.4.5 404 Not Found ....................................... 186

21.4.6 405 Method Not Allowed .............................. 186

21.4.7 406 Not Acceptable .................................. 186

21.4.8 407 Proxy Authentication Required ................... 186

21.4.9 408 Request Timeout ................................. 186

21.4.10 410 Gone ............................................ 187

21.4.11 413 Request Entity Too Large ........................ 187

21.4.12 414 Request-URI Too Long ............................ 187

21.4.13 415 Unsupported Media Type .......................... 187

21.4.14 416 Unsupported URI Scheme .......................... 187

21.4.15 420 Bad Extension ................................... 187

21.4.16 421 Extension Required .............................. 188

21.4.17 423 Interval Too Brief .............................. 188

21.4.18 480 Temporarily Unavailable ......................... 188

21.4.19 481 Call/Transaction Does Not Exist ................. 188

21.4.20 482 Loop Detected ................................... 188

21.4.21 483 Too Many Hops ................................... 189

21.4.22 484 Address Incomplete .............................. 189

Rosenberg, et. al. Standards Track [Page 6]

RFC 3261 SIP: Session Initiation Protocol June 2002

21.4.23 485 Ambiguous ....................................... 189

21.4.24 486 Busy Here ....................................... 189

21.4.25 487 Request Terminated .............................. 190

21.4.26 488 Not Acceptable Here ............................. 190

21.4.27 491 Request Pending ................................. 190

21.4.28 493 Undecipherable .................................. 190

21.5 Server Failure 5xx .................................. 190

21.5.1 500 Server Internal Error ........................... 190

21.5.2 501 Not Implemented ................................. 191

21.5.3 502 Bad Gateway ..................................... 191

21.5.4 503 Service Unavailable ............................. 191

21.5.5 504 Server Time-out ................................. 191

21.5.6 505 Version Not Supported ........................... 192

21.5.7 513 Message Too Large ............................... 192

21.6 Global Failures 6xx ................................. 192

21.6.1 600 Busy Everywhere ................................. 192

21.6.2 603 Decline ......................................... 192

21.6.3 604 Does Not Exist Anywhere ......................... 192

21.6.4 606 Not Acceptable .................................. 192

22 Usage of HTTP Authentication ........................ 193

22.1 Framework ........................................... 193

22.2 User-to-User Authentication ......................... 195

22.3 Proxy-to-User Authentication ........................ 197

22.4 The Digest Authentication Scheme .................... 199

23 S/MIME .............................................. 201

23.1 S/MIME Certificates ................................. 201

23.2 S/MIME Key Exchange ................................. 202

23.3 Securing MIME bodies ................................ 205

23.4 SIP Header Privacy and Integrity using S/MIME:

Tunneling SIP ....................................... 207

23.4.1 Integrity and Confidentiality Properties of SIP

Headers ............................................. 207

23.4.1.1 Integrity ........................................... 207

23.4.1.2 Confidentiality ..................................... 208

23.4.2 Tunneling Integrity and Authentication .............. 209

23.4.3 Tunneling Encryption ................................ 211

24 Examples ............................................ 213

24.1 Registration ........................................ 213

24.2 Session Setup ....................................... 214

25 Augmented BNF for the SIP Protocol .................. 219

25.1 Basic Rules ......................................... 219

26 Security Considerations: Threat Model and Security

Usage Recommendations ............................... 232

26.1 Attacks and Threat Models ........................... 233

26.1.1 Registration Hijacking .............................. 233

26.1.2 Impersonating a Server .............................. 234

26.1.3 Tampering with Message Bodies ....................... 235

26.1.4 Tearing Down Sessions ............................... 235

Rosenberg, et. al. Standards Track [Page 7]

RFC 3261 SIP: Session Initiation Protocol June 2002

26.1.5 Denial of Service and Amplification ................. 236

26.2 Security Mechanisms ................................. 237

26.2.1 Transport and Network Layer Security ................ 238

26.2.2 SIPS URI Scheme ..................................... 239

26.2.3 HTTP Authentication ................................. 240

26.2.4 S/MIME .............................................. 240

26.3 Implementing Security Mechanisms .................... 241

26.3.1 Requirements for Implementers of SIP ................ 241

26.3.2 Security Solutions .................................. 242

26.3.2.1 Registration ........................................ 242

26.3.2.2 Interdomain Requests ................................ 243

26.3.2.3 Peer-to-Peer Requests ............................... 245

26.3.2.4 DoS Protection ...................................... 246

26.4 Limitations ......................................... 247

26.4.1 HTTP Digest ......................................... 247

26.4.2 S/MIME .............................................. 248

26.4.3 TLS ................................................. 249

26.4.4 SIPS URIs ........................................... 249

26.5 Privacy ............................................. 251

27 IANA Considerations ................................. 252

27.1 Option Tags ......................................... 252

27.2 Warn-Codes .......................................... 252

27.3 Header Field Names .................................. 253

27.4 Method and Response Codes ........................... 253

27.5 The "message/sip" MIME type. ....................... 254

27.6 New Content-Disposition Parameter Registrations ..... 255

28 Changes From RFC 2543 ............................... 255

28.1 Major Functional Changes ............................ 255

28.2 Minor Functional Changes ............................ 260

29 Normative References ................................ 261

30 Informative References .............................. 262

A Table of Timer Values ............................... 265

Acknowledgments ................................................ 266

Authors' Addresses ............................................. 267

Full Copyright Statement ....................................... 269

1 Introduction

There are many applications of the Internet that require the creation

and management of a session, where a session is considered an

exchange of data between an association of participants. The

implementation of these applications is complicated by the practices

of participants: users may move between endpoints, they may be

addressable by multiple names, and they may communicate in several

different media - sometimes simultaneously. Numerous protocols have

been authored that carry various forms of real-time multimedia

session data such as voice, video, or text messages. The Session

Initiation Protocol (SIP) works in concert with these protocols by

Rosenberg, et. al. Standards Track [Page 8]

RFC 3261 SIP: Session Initiation Protocol June 2002

enabling Internet endpoints (called user agents) to discover one

another and to agree on a characterization of a session they would

like to share. For locating prospective session participants, and

for other functions, SIP enables the creation of an infrastructure of

network hosts (called proxy servers) to which user agents can send

registrations, invitations to sessions, and other requests. SIP is

an agile, general-purpose tool for creating, modifying, and

terminating sessions that works independently of underlying transport

protocols and without dependency on the type of session that is being

established.

2 Overview of SIP Functionality

SIP is an application-layer control protocol that can establish,

modify, and terminate multimedia sessions (conferences) such as

Internet telephony calls. SIP can also invite participants to

already existing sessions, such as multicast conferences. Media can

be added to (and removed from) an existing session. SIP

transparently supports name mapping and redirection services, which

supports personal mobility [27] - users can maintain a single

externally visible identifier regardless of their network location.

SIP supports five facets of establishing and terminating multimedia

communications:

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

communication;

User availability: determination of the willingness of the called

party to engage in communications;

User capabilities: determination of the media and media parameters

to be used;

Session setup: "ringing", establishment of session parameters at

both called and calling party;

Session management: including transfer and termination of

sessions, modifying session parameters, and invoking

services.

SIP is not a vertically integrated communications system. SIP is

rather a component that can be used with other IETF protocols to

build a complete multimedia architecture. Typically, these

architectures will include protocols such as the Real-time Transport

Protocol (RTP) (RFC 1889 [28]) for transporting real-time data and

providing QoS feedback, the Real-Time streaming protocol (RTSP) (RFC

2326 [29]) for controlling delivery of streaming media, the Media

Rosenberg, et. al. Standards Track [Page 9]

RFC 3261 SIP: Session Initiation Protocol June 2002

Gateway Control Protocol (MEGACO) (RFC 3015 [30]) for controlling

gateways to the Public Switched Telephone Network (PSTN), and the

Session Description Protocol (SDP) (RFC 2327 [1]) for describing

multimedia sessions. Therefore, SIP should be used in conjunction

with other protocols in order to provide complete services to the

users. However, the basic functionality and operation of SIP does

not depend on any of these protocols.

SIP does not provide services. Rather, SIP provides primitives that

can be used to implement different services. For example, SIP can

locate a user and deliver an opaque object to his current location.

If this primitive is used to deliver a session description written in

SDP, for instance, the endpoints can agree on the parameters of a

session. If the same primitive is used to deliver a photo of the

caller as well as the session description, a "caller ID" service can

be easily implemented. As this example shows, a single primitive is

typically used to provide several different services.

SIP does not offer conference control services such as floor control

or voting and does not prescribe how a conference is to be managed.

SIP can be used to initiate a session that uses some other conference

control protocol. Since SIP messages and the sessions they establish

can pass through entirely different networks, SIP cannot, and does

not, provide any kind of network resource reservation capabilities.

The nature of the services provided make security particularly

important. To that end, SIP provides a suite of security services,

which include denial-of-service prevention, authentication (both user

to user and proxy to user), integrity protection, and encryption and

privacy services.

SIP works with both IPv4 and IPv6.

3 Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED",

"SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT

RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as

described in BCP 14, RFC 2119 [2] and indicate requirement levels for

compliant SIP implementations.

4 Overview of Operation

This section introduces the basic operations of SIP using simple

examples. This section is tutorial in nature and does not contain

any normative statements.

Rosenberg, et. al. Standards Track [Page 10]

RFC 3261 SIP: Session Initiation Protocol June 2002

The first example shows the basic functions of SIP: location of an

end point, signal of a desire to communicate, negotiation of session

parameters to establish the session, and teardown of the session once

established.

Figure 1 shows a typical example of a SIP message exchange between

two users, Alice and Bob. (Each message is labeled with the letter

"F" and a number for reference by the text.) In this example, Alice

uses a SIP application on her PC (referred to as a softphone) to call

Bob on his SIP phone over the Internet. Also shown are two SIP proxy

servers that act on behalf of Alice and Bob to facilitate the session

establishment. This typical arrangement is often referred to as the

"SIP trapezoid" as shown by the geometric shape of the dotted lines

in Figure 1.

Alice "calls" Bob using his SIP identity, a type of Uniform Resource

Identifier (URI) called a SIP URI. SIP URIs are defined in Section

19.1. It has a similar form to an email address, typically

containing a username and a host name. In this case, it is

sip:bob@biloxi.com, where biloxi.com is the domain of Bob's SIP

service provider. Alice has a SIP URI of sip:alice@atlanta.com.

Alice might have typed in Bob's URI or perhaps clicked on a hyperlink

or an entry in an address book. SIP also provides a secure URI,

called a SIPS URI. An example would be sips:bob@biloxi.com. A call

made to a SIPS URI guarantees that secure, encrypted transport

(namely TLS) is used to carry all SIP messages from the caller to the

domain of the callee. From there, the request is sent securely to

the callee, but with security mechanisms that depend on the policy of

the domain of the callee.

SIP is based on an HTTP-like request/response transaction model.

Each transaction consists of a request that invokes a particular

method, or function, on the server and at least one response. In

this example, the transaction begins with Alice's softphone sending

an INVITE request addressed to Bob's SIP URI. INVITE is an example

of a SIP method that specifies the action that the requestor (Alice)

wants the server (Bob) to take. The INVITE request contains a number

of header fields. Header fields are named attributes that provide

additional information about a message. The ones present in an

INVITE include a unique identifier for the call, the destination

address, Alice's address, and information about the type of session

that Alice wishes to establish with Bob. The INVITE (message F1 in

Figure 1) might look like this:

Rosenberg, et. al. Standards Track [Page 11]

RFC 3261 SIP: Session Initiation Protocol June 2002

atlanta.com . . . biloxi.com

. proxy proxy .

. .

Alice's . . . . . . . . . . . . . . . . . . . . Bob's

softphone SIP Phone

| | | |

| INVITE F1 | | |

|--------------->| INVITE F2 | |

| 100 Trying F3 |--------------->| INVITE F4 |

|<---------------| 100 Trying F5 |--------------->|

| |<-------------- | 180 Ringing F6 |

| | 180 Ringing F7 |<---------------|

| 180 Ringing F8 |<---------------| 200 OK F9 |

|<---------------| 200 OK F10 |<---------------|

| 200 OK F11 |<---------------| |

|<---------------| | |

| ACK F12 |

|------------------------------------------------->|

| Media Session |

|<================================================>|

| BYE F13 |

|<-------------------------------------------------|

| 200 OK F14 |

|------------------------------------------------->|

| |

Figure 1: SIP session setup example with SIP trapezoid

INVITE sip:bob@biloxi.com SIP/2.0

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds

Max-Forwards: 70

To: Bob <sip:bob@biloxi.com>

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710@pc33.atlanta.com

CSeq: 314159 INVITE

Contact: <sip:alice@pc33.atlanta.com>

Content-Type: application/sdp

Content-Length: 142

(Alice's SDP not shown)

The first line of the text-encoded message contains the method name

(INVITE). The lines that follow are a list of header fields. This

example contains a minimum required set. The header fields are

briefly described below:

Rosenberg, et. al. Standards Track [Page 12]

RFC 3261 SIP: Session Initiation Protocol June 2002

Via contains the address (pc33.atlanta.com) at which Alice is

expecting to receive responses to this request. It also contains a

branch parameter that identifies this transaction.

To contains a display name (Bob) and a SIP or SIPS URI

(sip:bob@biloxi.com) towards which the request was originally

directed. Display names are described in RFC 2822 [3].

From also contains a display name (Alice) and a SIP or SIPS URI

(sip:alice@atlanta.com) that indicate the originator of the request.

This header field also has a tag parameter containing a random string

(1928301774) that was added to the URI by the softphone. It is used

for identification purposes.

Call-ID contains a globally unique identifier for this call,

generated by the combination of a random string and the softphone's

host name or IP address. The combination of the To tag, From tag,

and Call-ID completely defines a peer-to-peer SIP relationship

between Alice and Bob and is referred to as a dialog.

CSeq or Command Sequence contains an integer and a method name. The

CSeq number is incremented for each new request within a dialog and

is a traditional sequence number.

Contact contains a SIP or SIPS URI that represents a direct route to

contact Alice, usually composed of a username at a fully qualified

domain name (FQDN). While an FQDN is preferred, many end systems do

not have registered domain names, so IP addresses are permitted.

While the Via header field tells other elements where to send the

response, the Contact header field tells other elements where to send

future requests.

Max-Forwards serves to limit the number of hops a request can make on

the way to its destination. It consists of an integer that is

decremented by one at each hop.

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

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

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

The details of the session, such as the type of media, codec, or

sampling rate, are not described using SIP. Rather, the body of a

SIP message contains a description of the session, encoded in some

other protocol format. One such format is the Session Description

Protocol (SDP) (RFC 2327 [1]). This SDP message (not shown in the

Rosenberg, et. al. Standards Track [Page 13]

RFC 3261 SIP: Session Initiation Protocol June 2002

example) is carried by the SIP message in a way that is analogous to

a document attachment being carried by an email message, or a web

page being carried in an HTTP message.

Since the softphone does not know the location of Bob or the SIP

server in the biloxi.com domain, the softphone sends the INVITE to

the SIP server that serves Alice's domain, atlanta.com. The address

of the atlanta.com SIP server could have been configured in Alice's

softphone, or it could have been discovered by DHCP, for example.

The atlanta.com SIP server is a type of SIP server known as a proxy

server. A proxy server receives SIP requests and forwards them on

behalf of the requestor. In this example, the proxy server receives

the INVITE request and sends a 100 (Trying) response back to Alice's

softphone. The 100 (Trying) response indicates that the INVITE has

been received and that the proxy is working on her behalf to route

the INVITE to the destination. Responses in SIP use a three-digit

code followed by a descriptive phrase. This response contains the

same To, From, Call-ID, CSeq and branch parameter in the Via as the

INVITE, which allows Alice's softphone to correlate this response to

the sent INVITE. The atlanta.com proxy server locates the proxy

server at biloxi.com, possibly by performing a particular type of DNS

(Domain Name Service) lookup to find the SIP server that serves the

biloxi.com domain. This is described in [4]. As a result, it

obtains the IP address of the biloxi.com proxy server and forwards,

or proxies, the INVITE request there. Before forwarding the request,

the atlanta.com proxy server adds an additional Via header field

value that contains its own address (the INVITE already contains

Alice's address in the first Via). The biloxi.com proxy server

receives the INVITE and responds with a 100 (Trying) response back to

the atlanta.com proxy server to indicate that it has received the

INVITE and is processing the request. The proxy server consults a

database, generically called a location service, that contains the

current IP address of Bob. (We shall see in the next section how

this database can be populated.) The biloxi.com proxy server adds

another Via header field value with its own address to the INVITE and

proxies it to Bob's SIP phone.

Bob's SIP phone receives the INVITE and alerts Bob to the incoming

call from Alice so that Bob can decide whether to answer the call,

that is, Bob's phone rings. Bob's SIP phone indicates this in a 180

(Ringing) response, which is routed back through the two proxies in

the reverse direction. Each proxy uses the Via header field to

determine where to send the response and removes its own address from

the top. As a result, although DNS and location service lookups were

required to route the initial INVITE, the 180 (Ringing) response can

be returned to the caller without lookups or without state being

Rosenberg, et. al. Standards Track [Page 14]

RFC 3261 SIP: Session Initiation Protocol June 2002

maintained in the proxies. This also has the desirable property that

each proxy that sees the INVITE will also see all responses to the

INVITE.

When Alice's softphone receives the 180 (Ringing) response, it passes

this information to Alice, perhaps using an audio ringback tone or by

displaying a message on Alice's screen.

In this example, Bob decides to answer the call. When he picks up

the handset, his SIP phone sends a 200 (OK) response to indicate that

the call has been answered. The 200 (OK) contains a message body

with the SDP media description of the type of session that Bob is

willing to establish with Alice. As a result, there is a two-phase

exchange of SDP messages: Alice sent one to Bob, and Bob sent one

back to Alice. This two-phase exchange provides basic negotiation

capabilities and is based on a simple offer/answer model of SDP

exchange. If Bob did not wish to answer the call or was busy on

another call, an error response would have been sent instead of the

200 (OK), which would have resulted in no media session being

established. The complete list of SIP response codes is in Section

21. The 200 (OK) (message F9 in Figure 1) might look like this as

Bob sends it out:

SIP/2.0 200 OK

Via: SIP/2.0/UDP server10.biloxi.com

;branch=z9hG4bKnashds8;received=192.0.2.3

Via: SIP/2.0/UDP bigbox3.site3.atlanta.com

;branch=z9hG4bK77ef4c2312983.1;received=192.0.2.2

Via: SIP/2.0/UDP pc33.atlanta.com

;branch=z9hG4bK776asdhds ;received=192.0.2.1

To: Bob <sip:bob@biloxi.com>;tag=a6c85cf

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710@pc33.atlanta.com

CSeq: 314159 INVITE

Contact: <sip:bob@192.0.2.4>

Content-Type: application/sdp

Content-Length: 131

(Bob's SDP not shown)

The first line of the response contains the response code (200) and

the reason phrase (OK). The remaining lines contain header fields.

The Via, To, From, Call-ID, and CSeq header fields are copied from

the INVITE request. (There are three Via header field values - one

added by Alice's SIP phone, one added by the atlanta.com proxy, and

one added by the biloxi.com proxy.) Bob's SIP phone has added a tag

parameter to the To header field. This tag will be incorporated by

both endpoints into the dialog and will be included in all future

Rosenberg, et. al. Standards Track [Page 15]

RFC 3261 SIP: Session Initiation Protocol June 2002

requests and responses in this call. The Contact header field

contains a URI at which Bob can be directly reached at his SIP phone.

The Content-Type and Content-Length refer to the message body (not

shown) that contains Bob's SDP media information.

In addition to DNS and location service lookups shown in this

example, proxy servers can make flexible "routing decisions" to

decide where to send a request. For example, if Bob's SIP phone

returned a 486 (Busy Here) response, the biloxi.com proxy server

could proxy the INVITE to Bob's voicemail server. A proxy server can

also send an INVITE to a number of locations at the same time. This

type of parallel search is known as forking.

In this case, the 200 (OK) is routed back through the two proxies and

is received by Alice's softphone, which then stops the ringback tone

and indicates that the call has been answered. Finally, Alice's

softphone sends an acknowledgement message, ACK, to Bob's SIP phone

to confirm the reception of the final response (200 (OK)). In this

example, the ACK is sent directly from Alice's softphone to Bob's SIP

phone, bypassing the two proxies. This occurs because the endpoints

have learned each other's address from the Contact header fields

through the INVITE/200 (OK) exchange, which was not known when the

initial INVITE was sent. The lookups performed by the two proxies

are no longer needed, so the proxies drop out of the call flow. This

completes the INVITE/200/ACK three-way handshake used to establish

SIP sessions. Full details on session setup are in Section 13.

Alice and Bob's media session has now begun, and they send media

packets using the format to which they agreed in the exchange of SDP.

In general, the end-to-end media packets take a different path from

the SIP signaling messages.

During the session, either Alice or Bob may decide to change the

characteristics of the media session. This is accomplished by

sending a re-INVITE containing a new media description. This re-

INVITE references the existing dialog so that the other party knows

that it is to modify an existing session instead of establishing a

new session. The other party sends a 200 (OK) to accept the change.

The requestor responds to the 200 (OK) with an ACK. If the other

party does not accept the change, he sends an error response such as

488 (Not Acceptable Here), which also receives an ACK. However, the

failure of the re-INVITE does not cause the existing call to fail -

the session continues using the previously negotiated

characteristics. Full details on session modification are in Section

14.

Rosenberg, et. al. Standards Track [Page 16]

RFC 3261 SIP: Session Initiation Protocol June 2002

At the end of the call, Bob disconnects (hangs up) first and

generates a BYE message. This BYE is routed directly to Alice's

softphone, again bypassing the proxies. Alice confirms receipt of

the BYE with a 200 (OK) response, which terminates the session and

the BYE transaction. No ACK is sent - an ACK is only sent in

response to a response to an INVITE request. The reasons for this

special handling for INVITE will be discussed later, but relate to

the reliability mechanisms in SIP, the length of time it can take for

a ringing phone to be answered, and forking. For this reason,

request handling in SIP is often classified as either INVITE or non-

INVITE, referring to all other methods besides INVITE. Full details

on session termination are in Section 15.

Section 24.2 describes the messages shown in Figure 1 in full.

In some cases, it may be useful for proxies in the SIP signaling path

to see all the messaging between the endpoints for the duration of

the session. For example, if the biloxi.com proxy server wished to

remain in the SIP messaging path beyond the initial INVITE, it would

add to the INVITE a required routing header field known as Record-

Route that contained a URI resolving to the hostname or IP address of

the proxy. This information would be received by both Bob's SIP

phone and (due to the Record-Route header field being passed back in

the 200 (OK)) Alice's softphone and stored for the duration of the

dialog. The biloxi.com proxy server would then receive and proxy the

ACK, BYE, and 200 (OK) to the BYE. Each proxy can independently

decide to receive subsequent messages, and those messages will pass

through all proxies that elect to receive it. This capability is

frequently used for proxies that are providing mid-call features.

Registration is another common operation in SIP. Registration is one

way that the biloxi.com server can learn the current location of Bob.

Upon initialization, and at periodic intervals, Bob's SIP phone sends

REGISTER messages to a server in the biloxi.com domain known as a SIP

registrar. The REGISTER messages associate Bob's SIP or SIPS URI

(sip:bob@biloxi.com) with the machine into which he is currently

logged (conveyed as a SIP or SIPS URI in the Contact header field).

The registrar writes this association, also called a binding, to a

database, called the location service, where it can be used by the

proxy in the biloxi.com domain. Often, a registrar server for a

domain is co-located with the proxy for that domain. It is an

important concept that the distinction between types of SIP servers

is logical, not physical.

Bob is not limited to registering from a single device. For example,

both his SIP phone at home and the one in the office could send

registrations. This information is stored together in the location

Rosenberg, et. al. Standards Track [Page 17]

RFC 3261 SIP: Session Initiation Protocol June 2002

service and allows a proxy to perform various types of searches to

locate Bob. Similarly, more than one user can be registered on a

single device at the same time.

The location service is just an abstract concept. It generally

contains information that allows a proxy to input a URI and receive a

set of zero or more URIs that tell the proxy where to send the

request. Registrations are one way to create this information, but

not the only way. Arbitrary mapping functions can be configured at

the discretion of the administrator.

Finally, it is important to note that in SIP, registration is used

for routing incoming SIP requests and has no role in authorizing

outgoing requests. Authorization and authentication are handled in

SIP either on a request-by-request basis with a challenge/response

mechanism, or by using a lower layer scheme as discussed in Section

26.

The complete set of SIP message details for this registration example

is in Section 24.1.

Additional operations in SIP, such as querying for the capabilities

of a SIP server or client using OPTIONS, or canceling a pending

request using CANCEL, will be introduced in later sections.

5 Structure of the Protocol

SIP is structured as a layered protocol, which means that its

behavior is described in terms of a set of fairly independent

processing stages with only a loose coupling between each stage. The

protocol behavior is described as layers for the purpose of

presentation, allowing the description of functions common across

elements in a single section. It does not dictate an implementation

in any way. When we say that an element "contains" a layer, we mean

it is compliant to the set of rules defined by that layer.

Not every element specified by the protocol contains every layer.

Furthermore, the elements specified by SIP are logical elements, not

physical ones. A physical realization can choose to act as different

logical elements, perhaps even on a transaction-by-transaction basis.

The lowest layer of SIP is its syntax and encoding. Its encoding is

specified using an augmented Backus-Naur Form grammar (BNF). The

complete BNF is specified in Section 25; an overview of a SIP

message's structure can be found in Section 7.

Rosenberg, et. al. Standards Track [Page 18]

RFC 3261 SIP: Session Initiation Protocol June 2002

The second layer is the transport layer. It defines how a client

sends requests and receives responses and how a server receives

requests and sends responses over the network. All SIP elements

contain a transport layer. The transport layer is described in

Section 18.

The third layer is the transaction layer. Transactions are a

fundamental component of SIP. A transaction is a request sent by a

client transaction (using the transport layer) to a server

transaction, along with all responses to that request sent from the

server transaction back to the client. The transaction layer handles

application-layer retransmissions, matching of responses to requests,

and application-layer timeouts. Any task that a user agent client

(UAC) accomplishes takes place using a series of transactions.

Discussion of transactions can be found in Section 17. User agents

contain a transaction layer, as do stateful proxies. Stateless

proxies do not contain a transaction layer. The transaction layer

has a client component (referred to as a client transaction) and a

server component (referred to as a server transaction), each of which

are represented by a finite state machine that is constructed to

process a particular request.

The layer above the transaction layer is called the transaction user

(TU). Each of the SIP entities, except the stateless proxy, is a

transaction user. When a TU wishes to send a request, it creates a

client transaction instance and passes it the request along with the

destination IP address, port, and transport to which to send the

request. A TU that creates a client transaction can also cancel it.

When a client cancels a transaction, it requests that the server stop

further processing, revert to the state that existed before the

transaction was initiated, and generate a specific error response to

that transaction. This is done with a CANCEL request, which

constitutes its own transaction, but references the transaction to be

cancelled (Section 9).

The SIP elements, that is, user agent clients and servers, stateless

and stateful proxies and registrars, contain a core that

distinguishes them from each other. Cores, except for the stateless

proxy, are transaction users. While the behavior of the UAC and UAS

cores depends on the method, there are some common rules for all

methods (Section 8). For a UAC, these rules govern the construction

of a request; for a UAS, they govern the processing of a request and

generating a response. Since registrations play an important role in

SIP, a UAS that handles a REGISTER is given the special name

registrar. Section 10 describes UAC and UAS core behavior for the

REGISTER method. Section 11 describes UAC and UAS core behavior for

the OPTIONS method, used for determining the capabilities of a UA.

Rosenberg, et. al. Standards Track [Page 19]

RFC 3261 SIP: Session Initiation Protocol June 2002

Certain other requests are sent within a dialog. A dialog is a

peer-to-peer SIP relationship between two user agents that persists

for some time. The dialog facilitates sequencing of messages and

proper routing of requests between the user agents. The INVITE

method is the only way defined in this specification to establish a

dialog. When a UAC sends a request that is within the context of a

dialog, it follows the common UAC rules as discussed in Section 8 but

also the rules for mid-dialog requests. Section 12 discusses dialogs

and presents the procedures for their construction and maintenance,

in addition to construction of requests within a dialog.

The most important method in SIP is the INVITE method, which is used

to establish a session between participants. A session is a

collection of participants, and streams of media between them, for

the purposes of communication. Section 13 discusses how sessions are

initiated, resulting in one or more SIP dialogs. Section 14

discusses how characteristics of that session are modified through

the use of an INVITE request within a dialog. Finally, section 15

discusses how a session is terminated.

The procedures of Sections 8, 10, 11, 12, 13, 14, and 15 deal

entirely with the UA core (Section 9 describes cancellation, which

applies to both UA core and proxy core). Section 16 discusses the

proxy element, which facilitates routing of messages between user

agents.

6 Definitions

The following terms have special significance for SIP.

Address-of-Record: An address-of-record (AOR) is a SIP or SIPS URI

that points to a domain with a location service that can map

the URI to another URI where the user might be available.

Typically, the location service is populated through

registrations. An AOR is frequently thought of as the "public

address" of the user.

Back-to-Back User Agent: A back-to-back user agent (B2BUA) is a

logical entity that receives a request and processes it as a

user agent server (UAS). In order to determine how the request

should be answered, it acts as a user agent client (UAC) and

generates requests. Unlike a proxy server, it maintains dialog

state and must participate in all requests sent on the dialogs

it has established. Since it is a concatenation of a UAC and

UAS, no explicit definitions are needed for its behavior.

Rosenberg, et. al. Standards Track [Page 20]

RFC 3261 SIP: Session Initiation Protocol June 2002

Call: A call is an informal term that refers to some communication

between peers, generally set up for the purposes of a

multimedia conversation.

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

specification.

Call Stateful: A proxy is call stateful if it retains state for a

dialog from the initiating INVITE to the terminating BYE

request. A call stateful proxy is always transaction stateful,

but the converse is not necessarily true.

Client: A client is any network element that sends SIP requests

and receives SIP responses. Clients may or may not interact

directly with a human user. User agent clients and proxies are

clients.

Conference: A multimedia session (see below) that contains

multiple participants.

Core: Core designates the functions specific to a particular type

of SIP entity, i.e., specific to either a stateful or stateless

proxy, a user agent or registrar. All cores, except those for

the stateless proxy, are transaction users.

Dialog: A dialog is a peer-to-peer SIP relationship between two

UAs that persists for some time. A dialog is established by

SIP messages, such as a 2xx response to an INVITE request. A

dialog is identified by a call identifier, local tag, and a

remote tag. A dialog was formerly known as a call leg in RFC

2543.

Downstream: A direction of message forwarding within a transaction

that refers to the direction that requests flow from the user

agent client to user agent server.

Final Response: A response that terminates a SIP transaction, as

opposed to a provisional response that does not. All 2xx, 3xx,

4xx, 5xx and 6xx responses are final.

Header: A header is a component of a SIP message that conveys

information about the message. It is structured as a sequence

of header fields.

Header Field: A header field is a component of the SIP message

header. A header field can appear as one or more header field

rows. Header field rows consist of a header field name and zero

or more header field values. Multiple header field values on a

Rosenberg, et. al. Standards Track [Page 21]

RFC 3261 SIP: Session Initiation Protocol June 2002

given header field row are separated by commas. Some header

fields can only have a single header field value, and as a

result, always appear as a single header field row.

Header Field Value: A header field value is a single value; a

header field consists of zero or more header field values.

Home Domain: The domain providing service to a SIP user.

Typically, this is the domain present in the URI in the

address-of-record of a registration.

Informational Response: Same as a provisional response.

Initiator, Calling Party, Caller: The party initiating a session

(and dialog) with an INVITE request. A caller retains this

role from the time it sends the initial INVITE that established

a dialog until the termination of that dialog.

Invitation: An INVITE request.

Invitee, Invited User, Called Party, Callee: The party that

receives an INVITE request for the purpose of establishing a

new session. A callee retains this role from the time it

receives the INVITE until the termination of the dialog

established by that INVITE.

Location Service: A location service is used by a SIP redirect or

proxy server to obtain information about a callee's possible

location(s). It contains a list of bindings of address-of-

record keys to zero or more contact addresses. The bindings

can be created and removed in many ways; this specification

defines a REGISTER method that updates the bindings.

Loop: A request that arrives at a proxy, is forwarded, and later

arrives back at the same proxy. When it arrives the second

time, its Request-URI is identical to the first time, and other

header fields that affect proxy operation are unchanged, so

that the proxy would make the same processing decision on the

request it made the first time. Looped requests are errors,

and the procedures for detecting them and handling them are

described by the protocol.

Loose Routing: A proxy is said to be loose routing if it follows

the procedures defined in this specification for processing of

the Route header field. These procedures separate the

destination of the request (present in the Request-URI) from

Rosenberg, et. al. Standards Track [Page 22]

RFC 3261 SIP: Session Initiation Protocol June 2002

the set of proxies that need to be visited along the way

(present in the Route header field). A proxy compliant to

these mechanisms is also known as a loose router.

Message: Data sent between SIP elements as part of the protocol.

SIP messages are either requests or responses.

Method: The method is the primary function that a request is meant

to invoke on a server. The method is carried in the request

message itself. Example methods are INVITE and BYE.

Outbound Proxy: A proxy that receives requests from a client, even

though it may not be the server resolved by the Request-URI.

Typically, a UA is manually configured with an outbound proxy,

or can learn about one through auto-configuration protocols.

Parallel Search: In a parallel search, a proxy issues several

requests to possible user locations upon receiving an incoming

request. Rather than issuing one request and then waiting for

the final response before issuing the next request as in a

sequential search, a parallel search issues requests without

waiting for the result of previous requests.

Provisional Response: A response used by the server to indicate

progress, but that does not terminate a SIP transaction. 1xx

responses are provisional, other responses are considered

final.

Proxy, Proxy Server: An intermediary entity that acts as both a

server and a client for the purpose of making requests on

behalf of other clients. A proxy server primarily plays the

role of routing, which means its job is to ensure that a

request is sent to another entity "closer" to the targeted

user. Proxies are also useful for enforcing policy (for

example, making sure a user is allowed to make a call). A

proxy interprets, and, if necessary, rewrites specific parts of

a request message before forwarding it.

Recursion: A client recurses on a 3xx response when it generates a

new request to one or more of the URIs in the Contact header

field in the response.

Redirect Server: A redirect server is a user agent server that

generates 3xx responses to requests it receives, directing the

client to contact an alternate set of URIs.

Rosenberg, et. al. Standards Track [Page 23]

RFC 3261 SIP: Session Initiation Protocol June 2002

Registrar: A registrar is a server that accepts REGISTER requests

and places the information it receives in those requests into

the location service for the domain it handles.

Regular Transaction: A regular transaction is any transaction with

a method other than INVITE, ACK, or CANCEL.

Request: A SIP message sent from a client to a server, for the

purpose of invoking a particular operation.

Response: A SIP message sent from a server to a client, for

indicating the status of a request sent from the client to the

server.

Ringback: Ringback is the signaling tone produced by the calling

party's application indicating that a called party is being

alerted (ringing).

Route Set: A route set is a collection of ordered SIP or SIPS URI

which represent a list of proxies that must be traversed when

sending a particular request. A route set can be learned,

through headers like Record-Route, or it can be configured.

Server: A server is a network element that receives requests in

order to service them and sends back responses to those

requests. Examples of servers are proxies, user agent servers,

redirect servers, and registrars.

Sequential Search: In a sequential search, a proxy server attempts

each contact address in sequence, proceeding to the next one

only after the previous has generated a final response. A 2xx

or 6xx class final response always terminates a sequential

search.

Session: From the SDP specification: "A multimedia session is a

set of multimedia senders and receivers and the data streams

flowing from senders to receivers. A multimedia conference is

an example of a multimedia session." (RFC 2327 [1]) (A session

as defined for SDP can comprise one or more RTP sessions.) As

defined, a callee can be invited several times, by different

calls, to the same session. If SDP is used, a session is

defined by the concatenation of the SDP user name, session id,

network type, address type, and address elements in the origin

field.

SIP Transaction: A SIP transaction occurs between a client and a

server and comprises all messages from the first request sent

from the client to the server up to a final (non-1xx) response

Rosenberg, et. al. Standards Track [Page 24]

RFC 3261 SIP: Session Initiation Protocol June 2002

sent from the server to the client. If the request is INVITE

and the final response is a non-2xx, the transaction also

includes an ACK to the response. The ACK for a 2xx response to

an INVITE request is a separate transaction.

Spiral: A spiral is a SIP request that is routed to a proxy,

forwarded onwards, and arrives once again at that proxy, but

this time differs in a way that will result in a different

processing decision than the original request. Typically, this

means that the request's Request-URI differs from its previous

arrival. A spiral is not an error condition, unlike a loop. A

typical cause for this is call forwarding. A user calls

joe@example.com. The example.com proxy forwards it to Joe's

PC, which in turn, forwards it to bob@example.com. This

request is proxied back to the example.com proxy. However,

this is not a loop. Since the request is targeted at a

different user, it is considered a spiral, and is a valid

condition.

Stateful Proxy: A logical entity that maintains the client and

server transaction state machines defined by this specification

during the processing of a request, also known as a transaction

stateful proxy. The behavior of a stateful proxy is further

defined in Section 16. A (transaction) stateful proxy is not

the same as a call stateful proxy.

Stateless Proxy: A logical entity that does not maintain the

client or server transaction state machines defined in this

specification when it processes requests. A stateless proxy

forwards every request it receives downstream and every

response it receives upstream.

Strict Routing: A proxy is said to be strict routing if it follows

the Route processing rules of RFC 2543 and many prior work in

progress versions of this RFC. That rule caused proxies to

destroy the contents of the Request-URI when a Route header

field was present. Strict routing behavior is not used in this

specification, in favor of a loose routing behavior. Proxies

that perform strict routing are also known as strict routers.

Target Refresh Request: A target refresh request sent within a

dialog is defined as a request that can modify the remote

target of the dialog.

Transaction User (TU): The layer of protocol processing that

resides above the transaction layer. Transaction users include

the UAC core, UAS core, and proxy core.

Rosenberg, et. al. Standards Track [Page 25]

RFC 3261 SIP: Session Initiation Protocol June 2002

Upstream: A direction of message forwarding within a transaction

that refers to the direction that responses flow from the user

agent server back to the user agent client.

URL-encoded: A character string encoded according to RFC 2396,

Section 2.4 [5].

User Agent Client (UAC): A user agent client is a logical entity

that creates a new request, and then uses the client

transaction state machinery to send it. The role of UAC lasts

only for the duration of that transaction. In other words, if

a piece of software initiates a request, it acts as a UAC for

the duration of that transaction. If it receives a request

later, it assumes the role of a user agent server for the

processing of that transaction.

UAC Core: The set of processing functions required of a UAC that

reside above the transaction and transport layers.

User Agent Server (UAS): A user agent server is a logical entity

that generates a response to a SIP request. The response

accepts, rejects, or redirects the request. This role lasts

only for the duration of that transaction. In other words, if

a piece of software responds to a request, it acts as a UAS for

the duration of that transaction. If it generates a request

later, it assumes the role of a user agent client for the

processing of that transaction.

UAS Core: The set of processing functions required at a UAS that

resides above the transaction and transport layers.

User Agent (UA): A logical entity that can act as both a user

agent client and user agent server.

The role of UAC and UAS, as well as proxy and redirect servers, are

defined on a transaction-by-transaction basis. For example, the user

agent initiating a call acts as a UAC when sending the initial INVITE

request and as a UAS when receiving a BYE request from the callee.

Similarly, the same software can act as a proxy server for one

request and as a redirect server for the next request.

Proxy, location, and registrar servers defined above are logical

entities; implementations MAY combine them into a single application.

7 SIP Messages

SIP is a text-based protocol and uses the UTF-8 charset (RFC 2279

[7]).

Rosenberg, et. al. Standards Track [Page 26]

RFC 3261 SIP: Session Initiation Protocol June 2002

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

response from a server to a client.

Both Request (section 7.1) and Response (section 7.2) messages use

the basic format of RFC 2822 [3], even though the syntax differs in

character set and syntax specifics. (SIP allows header fields that

would not be valid RFC 2822 header fields, for example.) Both types

of messages consist of a start-line, one or more header fields, an

empty line indicating the end of the header fields, and an optional

message-body.

generic-message = start-line

\*message-header

CRLF

[ message-body ]

start-line = Request-Line / Status-Line

The start-line, each message-header line, and the empty line MUST be

terminated by a carriage-return line-feed sequence (CRLF). Note that

the empty line MUST be present even if the message-body is not.

Except for the above difference in character sets, much of SIP's

message and header field syntax is identical to HTTP/1.1. Rather

than repeating the syntax and semantics here, we use [HX.Y] to refer

to Section X.Y of the current HTTP/1.1 specification (RFC 2616 [8]).

However, SIP is not an extension of HTTP.

7.1 Requests

SIP requests are distinguished by having a Request-Line for a start-

line. A Request-Line contains a method name, a Request-URI, and the

protocol version separated by a single space (SP) character.

The Request-Line ends with CRLF. No CR or LF are allowed except in

the end-of-line CRLF sequence. No linear whitespace (LWS) is allowed

in any of the elements.

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

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, documented in standards track RFCs, may define

additional methods.

Rosenberg, et. al. Standards Track [Page 27]

RFC 3261 SIP: Session Initiation Protocol June 2002

Request-URI: The Request-URI is a SIP or SIPS URI as described in

Section 19.1 or a general URI (RFC 2396 [5]). 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 elements MAY support Request-URIs with schemes other than

"sip" and "sips", for example the "tel" URI scheme of RFC

2806 [9]. SIP elements MAY translate non-SIP URIs using any

mechanism at their disposal, resulting in SIP URI, SIPS 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 SIP-Version 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 from requests by having a Status-Line

as their 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 a single SP character.

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

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

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.

While this specification suggests specific wording for the reason

phrase, implementations MAY choose other text, for example, in the

language indicated in the Accept-Language header field of the

request.

Rosenberg, et. al. Standards Track [Page 28]

RFC 3261 SIP: Session Initiation Protocol June 2002

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

six values for the first digit:

1xx: Provisional -- request received, continuing to process the

request;

2xx: Success -- the action was successfully received, understood,

and accepted;

3xx: Redirection -- further action needs to be taken in order to

complete the request;

4xx: Client Error -- the request contains bad syntax or cannot be

fulfilled at this server;

5xx: Server Error -- the server failed to fulfill an apparently

valid request;

6xx: Global Failure -- the request cannot be fulfilled at any

server.

Section 21 defines these classes and describes the individual codes.

7.3 Header Fields

SIP header fields are similar to HTTP header fields in both syntax

and semantics. In particular, SIP header fields follow the [H4.2]

definitions of syntax for the message-header and the rules for

extending header fields over multiple lines. However, the latter is

specified in HTTP with implicit whitespace and folding. This

specification conforms to RFC 2234 [10] and uses only explicit

whitespace and folding as an integral part of the grammar.

[H4.2] also specifies that multiple header fields of the same field

name whose value is a comma-separated list can be combined into one

header field. That applies to SIP as well, but the specific rule is

different because of the different grammars. Specifically, any SIP

header whose grammar is of the form

header = "header-name" HCOLON header-value \*(COMMA header-value)

allows for combining header fields of the same name into a comma-

separated list. The Contact header field allows a comma-separated

list unless the header field value is "\*".

Rosenberg, et. al. Standards Track [Page 29]

RFC 3261 SIP: Session Initiation Protocol June 2002

7.3.1 Header Field Format

Header fields follow the same generic header format as that given in

Section 2.2 of RFC 2822 [3]. Each header field consists of a field

name followed by a colon (":") and the field value.

field-name: field-value

The formal grammar for a message-header specified in Section 25

allows for an arbitrary amount of whitespace on either side of the

colon; however, implementations should avoid spaces between the field

name and the colon and use a single space (SP) between the colon and

the field-value.

Subject: lunch

Subject : lunch

Subject :lunch

Subject: lunch

Thus, the above are all valid and equivalent, but the last is the

preferred form.

Header fields can be extended over multiple lines by preceding each

extra line with at least one SP or horizontal tab (HT). The line

break and the whitespace at the beginning of the next line are

treated as a single SP character. Thus, the following are

equivalent:

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

Subject: I know you're there,

pick up the phone

and talk to me!

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

significant. However, it is RECOMMENDED that header fields which are

needed for proxy processing (Via, Route, Record-Route, Proxy-Require,

Max-Forwards, and Proxy-Authorization, for example) appear towards

the top of the message to facilitate rapid parsing. The relative

order of header field rows with the same field name is important.

Multiple header field rows with the same field-name MAY be present in

a message if and only if the entire field-value for that header field

is defined as a comma-separated list (that is, if follows the grammar

defined in Section 7.3). It MUST be possible to combine the multiple

header field rows into one "field-name: field-value" pair, without

changing the semantics of the message, by appending each subsequent

field-value to the first, each separated by a comma. The exceptions

to this rule are the WWW-Authenticate, Authorization, Proxy-

Authenticate, and Proxy-Authorization header fields. Multiple header

Rosenberg, et. al. Standards Track [Page 30]

RFC 3261 SIP: Session Initiation Protocol June 2002

field rows with these names MAY be present in a message, but since

their grammar does not follow the general form listed in Section 7.3,

they MUST NOT be combined into a single header field row.

Implementations MUST be able to process multiple header field rows

with the same name in any combination of the single-value-per-line or

comma-separated value forms.

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

Route: <sip:alice@atlanta.com>

Subject: Lunch

Route: <sip:bob@biloxi.com>

Route: <sip:carol@chicago.com>

Route: <sip:alice@atlanta.com>, <sip:bob@biloxi.com>

Route: <sip:carol@chicago.com>

Subject: Lunch

Subject: Lunch

Route: <sip:alice@atlanta.com>, <sip:bob@biloxi.com>,

<sip:carol@chicago.com>

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

others:

Route: <sip:alice@atlanta.com>

Route: <sip:bob@biloxi.com>

Route: <sip:carol@chicago.com>

Route: <sip:bob@biloxi.com>

Route: <sip:alice@atlanta.com>

Route: <sip:carol@chicago.com>

Route: <sip:alice@atlanta.com>,<sip:carol@chicago.com>,

<sip:bob@biloxi.com>

The format of a header field-value is defined per header-name. It

will always be either an opaque sequence of TEXT-UTF8 octets, or a

combination of whitespace, tokens, separators, and quoted strings.

Many existing header fields will adhere to the general form of a

value followed by a semi-colon separated sequence of parameter-name,

parameter-value pairs:

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

Rosenberg, et. al. Standards Track [Page 31]

RFC 3261 SIP: Session Initiation Protocol June 2002

Even though an arbitrary number of parameter pairs may be attached to

a header field value, any given parameter-name MUST NOT appear more

than once.

When comparing header fields, field names are always case-

insensitive. Unless otherwise stated in the definition of a

particular header field, field values, parameter names, and parameter

values are case-insensitive. Tokens are always case-insensitive.

Unless specified otherwise, values expressed as quoted strings are

case-sensitive. For example,

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

is equivalent to

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

and

Content-Disposition: session;handling=optional

is equivalent to

content-disposition: Session;HANDLING=OPTIONAL

The following two header fields are not equivalent:

Warning: 370 devnull "Choose a bigger pipe"

Warning: 370 devnull "CHOOSE A BIGGER PIPE"

7.3.2 Header Field Classification

Some header fields only make sense in requests or responses. These

are called request header fields and response header fields,

respectively. If a header field appears in a message not matching

its category (such as a request header field in a response), it MUST

be ignored. Section 20 defines the classification of each header

field.

7.3.3 Compact Form

SIP provides a mechanism to represent common header field names in an

abbreviated form. This may be useful when messages would otherwise

become too large to be carried on the transport available to it

(exceeding the maximum transmission unit (MTU) when using UDP, for

example). These compact forms are defined in Section 20. A compact

form MAY be substituted for the longer form of a header field name at

any time without changing the semantics of the message. A header

Rosenberg, et. al. Standards Track [Page 32]

RFC 3261 SIP: Session Initiation Protocol June 2002

field name MAY appear in both long and short forms within the same

message. Implementations MUST accept both the long and short forms

of each header name.

7.4 Bodies

Requests, including new requests defined in extensions to this

specification, MAY contain message bodies unless otherwise noted.

The interpretation of the body depends on the request method.

For response messages, the request method and the response status

code determine the type and interpretation of any message body. All

responses MAY include a body.

7.4.1 Message Body Type

The Internet media type of the message body MUST be given by the

Content-Type header field. If the body has undergone any encoding

such as compression, then this MUST be indicated by the Content-

Encoding header field; otherwise, Content-Encoding MUST be omitted.

If applicable, the character set of the message body is indicated as

part of the Content-Type header-field value.

The "multipart" MIME type defined in RFC 2046 [11] MAY be used within

the body of the message. Implementations that send requests

containing multipart message bodies MUST send a session description

as a non-multipart message body if the remote implementation requests

this through an Accept header field that does not contain multipart.

SIP messages MAY contain binary bodies or body parts. When no

explicit charset parameter is provided by the sender, media subtypes

of the "text" type are defined to have a default charset value of

"UTF-8".

7.4.2 Message Body Length

The body length in bytes is provided by the Content-Length header

field. Section 20.14 describes the necessary contents of this header

field in detail.

The "chunked" transfer encoding of HTTP/1.1 MUST NOT be used for SIP.

(Note: The chunked encoding modifies the body of a message in order

to transfer it as a series of chunks, each with its own size

indicator.)

Rosenberg, et. al. Standards Track [Page 33]

RFC 3261 SIP: Session Initiation Protocol June 2002

7.5 Framing SIP Messages

Unlike HTTP, SIP implementations can use UDP or other unreliable

datagram protocols. Each such datagram carries one request or

response. See Section 18 on constraints on usage of unreliable

transports.

Implementations processing SIP messages over stream-oriented

transports MUST ignore any CRLF appearing before the start-line

[H4.1].

The Content-Length header field value is used to locate the end of

each SIP message in a stream. It will always be present when SIP

messages are sent over stream-oriented transports.

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 a response based on

user input, external stimulus, the result of a program execution, or

some other mechanism.

When a UAC sends a request, the request passes 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, based

on 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 requests that are outside of a dialog.

This includes, of course, the requests which themselves establish a

dialog.

Security procedures for requests and responses outside of a dialog

are described in Section 26. Specifically, mechanisms exist for the

UAS and UAC to mutually authenticate. A limited set of privacy

features are also supported through encryption of bodies using

S/MIME.

Rosenberg, et. al. Standards Track [Page 34]

RFC 3261 SIP: Session Initiation Protocol June 2002

8.1 UAC Behavior

This section covers UAC behavior outside of a dialog.

8.1.1 Generating the Request

A valid SIP request formulated by a UAC MUST, at a minimum, contain

the following header fields: To, From, CSeq, Call-ID, Max-Forwards,

and Via; all of these header fields are mandatory in all SIP

requests. These six header fields 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, limiting message propagation,

ordering of messages, and the unique identification of transactions.

These header fields are in addition to the mandatory request line,

which contains the method, Request-URI, and SIP version.

Examples of requests sent outside of a dialog include an INVITE to

establish a session (Section 13) and an OPTIONS to query for

capabilities (Section 11).

8.1.1.1 Request-URI

The initial Request-URI of the message SHOULD be set to the value of

the URI in the To field. One notable exception is the REGISTER

method; behavior for setting the Request-URI of REGISTER is given in

Section 10. It may also be undesirable for privacy reasons or

convenience to set these fields to the same value (especially if the

originating UA expects that the Request-URI will be changed during

transit).

In some special circumstances, the presence of a pre-existing route

set can affect the Request-URI of the message. A pre-existing route

set is an ordered set of URIs that identify a chain of servers, to

which a UAC will send outgoing requests that are outside of a dialog.

Commonly, they are configured on the UA by a user or service provider

manually, or through some other non-SIP mechanism. When a provider

wishes to configure a UA with an outbound proxy, it is RECOMMENDED

that this be done by providing it with a pre-existing route set with

a single URI, that of the outbound proxy.

When a pre-existing route set is present, the procedures for

populating the Request-URI and Route header field detailed in Section

12.2.1.1 MUST be followed (even though there is no dialog), using the

desired Request-URI as the remote target URI.

Rosenberg, et. al. Standards Track [Page 35]

RFC 3261 SIP: Session Initiation Protocol June 2002

8.1.1.2 To

The To header field first and foremost specifies the desired

"logical" recipient of the request, or the address-of-record of the

user or resource that is the target of this request. This may or may

not be the ultimate recipient of the request. The To header field

MAY contain a SIP or SIPS URI, but it may also make use of other URI

schemes (the tel URL (RFC 2806 [9]), for example) when appropriate.

All SIP implementations MUST support the SIP URI scheme. Any

implementation that supports TLS MUST support the SIPS URI scheme.

The To header field allows for a display name.

A UAC may learn how to populate the To header field for a particular

request in a number of ways. Usually the user will suggest the To

header field through a human interface, perhaps inputting the URI

manually or selecting it from some sort of address book. Frequently,

the user will not enter a complete URI, but rather a string of digits

or letters (for example, "bob"). It is at the discretion of the UA

to choose how to interpret this input. Using the string to form the

user part of a SIP URI implies that the UA wishes the name to be

resolved in the domain to the right-hand side (RHS) of the at-sign in

the SIP URI (for instance, sip:bob@example.com). Using the string to

form the user part of a SIPS URI implies that the UA wishes to

communicate securely, and that the name is to be resolved in the

domain to the RHS of the at-sign. The RHS will frequently be the

home domain of the requestor, which allows for the home domain to

process the outgoing request. This is useful for features like

"speed dial" that require interpretation of the user part in the home

domain. The tel URL may be used when the UA does not wish to specify

the domain that should interpret a telephone number that has been

input by the user. Rather, each domain through which the request

passes would be given that opportunity. As an example, a user in an

airport might log in and send requests through an outbound proxy in

the airport. If they enter "411" (this is the phone number for local

directory assistance in the United States), that needs to be

interpreted and processed by the outbound proxy in the airport, not

the user's home domain. In this case, tel:411 would be the right

choice.

A request outside of a dialog MUST NOT contain a To tag; the tag in

the To field of a request identifies the peer of the dialog. Since

no dialog is established, no tag is present.

For further information on the To header field, see Section 20.39.

The following is an example of a valid To header field:

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

Rosenberg, et. al. Standards Track [Page 36]

RFC 3261 SIP: Session Initiation Protocol June 2002

8.1.1.3 From

The From header field indicates the logical identity of the initiator

of the request, possibly the user's address-of-record. Like the To

header field, it contains a URI and optionally a display name. It is

used by SIP elements to determine which processing rules to apply to

a request (for example, automatic call rejection). As such, it is

very important that the From URI not contain IP addresses or the FQDN

of the host on which the UA is running, since these are not logical

names.

The From header field allows for a display name. A UAC SHOULD use

the display name "Anonymous", along with a syntactically correct, but

otherwise meaningless URI (like sip:thisis@anonymous.invalid), if the

identity of the client is to remain hidden.

Usually, the value that populates the From header field in requests

generated by a particular UA is pre-provisioned by the user or by the

administrators of the user's local domain. If a particular UA is

used by multiple users, it might have switchable profiles that

include a URI corresponding to the identity of the profiled user.

Recipients of requests can authenticate the originator of a request

in order to ascertain that they are who their From header field

claims they are (see Section 22 for more on authentication).

The From field MUST contain a new "tag" parameter, chosen by the UAC.

See Section 19.3 for details on choosing a tag.

For further information on the From header field, see Section 20.20.

Examples:

From: "Bob" <sips:bob@biloxi.com> ;tag=a48s

From: sip:+12125551212@phone2net.com;tag=887s

From: Anonymous <sip:c8oqz84zk7z@privacy.org>;tag=hyh8

8.1.1.4 Call-ID

The Call-ID header field acts as a unique identifier to group

together a series of messages. It MUST be the same for all requests

and responses sent by either UA in a dialog. It SHOULD be the same

in each registration from a UA.

In a new request created by a UAC outside of any dialog, the Call-ID

header field MUST be selected by the UAC as a globally unique

identifier over space and time unless overridden by method-specific

behavior. All SIP UAs must have a means to guarantee that the Call-

ID header fields they produce will not be inadvertently generated by

any other UA. Note that when requests are retried after certain

Rosenberg, et. al. Standards Track [Page 37]

RFC 3261 SIP: Session Initiation Protocol June 2002

failure responses that solicit an amendment to a request (for

example, a challenge for authentication), these retried requests are

not considered new requests, and therefore do not need new Call-ID

header fields; see Section 8.1.3.5.

Use of cryptographically random identifiers (RFC 1750 [12]) in the

generation of Call-IDs is RECOMMENDED. Implementations MAY use the

form "localid@host". Call-IDs are case-sensitive and are simply

compared byte-by-byte.

Using cryptographically random identifiers provides some

protection against session hijacking and reduces the likelihood of

unintentional Call-ID collisions.

No provisioning or human interface is required for the selection of

the Call-ID header field value for a request.

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

20.8.

Example:

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

8.1.1.5 CSeq

The CSeq header field serves as a way to identify and order

transactions. It consists of a sequence number and a method. The

method MUST match that of the request. For non-REGISTER requests

outside of a dialog, the sequence number value is arbitrary. The

sequence number value MUST be expressible as a 32-bit unsigned

integer and MUST be less than 2\*\*31. As long as it follows the above

guidelines, a client may use any mechanism it would like to select

CSeq header field values.

Section 12.2.1.1 discusses construction of the CSeq for requests

within a dialog.

Example:

CSeq: 4711 INVITE

Rosenberg, et. al. Standards Track [Page 38]

RFC 3261 SIP: Session Initiation Protocol June 2002

8.1.1.6 Max-Forwards

The Max-Forwards header field serves to limit the number of hops a

request can transit on the way to its destination. It consists of an

integer that is decremented by one at each hop. If the Max-Forwards

value reaches 0 before the request reaches its destination, it will

be rejected with a 483(Too Many Hops) error response.

A UAC MUST insert a Max-Forwards header field into each request it

originates with a value that SHOULD be 70. This number was chosen to

be sufficiently large to guarantee that a request would not be

dropped in any SIP network when there were no loops, but not so large

as to consume proxy resources when a loop does occur. Lower values

should be used with caution and only in networks where topologies are

known by the UA.

8.1.1.7 Via

The Via header field indicates the transport used for the transaction

and identifies the location where the response is to be sent. A Via

header field value is added only after the transport that will be

used to reach the next hop has been selected (which may involve the

usage of the procedures in [4]).

When the UAC creates a request, it MUST insert a Via into that

request. The protocol name and protocol version in the header field

MUST be SIP and 2.0, respectively. The Via header field value MUST

contain a branch parameter. This parameter is used to identify the

transaction created by that request. This parameter is used by both

the client and the server.

The branch parameter value MUST be unique across space and time for

all requests sent by the UA. The exceptions to this rule are CANCEL

and ACK for non-2xx responses. As discussed below, a CANCEL request

will have the same value of the branch parameter as the request it

cancels. As discussed in Section 17.1.1.3, an ACK for a non-2xx

response will also have the same branch ID as the INVITE whose

response it acknowledges.

The uniqueness property of the branch ID parameter, to facilitate

its use as a transaction ID, was not part of RFC 2543.

The branch ID inserted by an element compliant with this

specification MUST always begin with the characters "z9hG4bK". These

7 characters are used as a magic cookie (7 is deemed sufficient to

ensure that an older RFC 2543 implementation would not pick such a

value), so that servers receiving the request can determine that the

branch ID was constructed in the fashion described by this

Rosenberg, et. al. Standards Track [Page 39]

RFC 3261 SIP: Session Initiation Protocol June 2002

specification (that is, globally unique). Beyond this requirement,

the precise format of the branch token is implementation-defined.

The Via header maddr, ttl, and sent-by components will be set when

the request is processed by the transport layer (Section 18).

Via processing for proxies is described in Section 16.6 Item 8 and

Section 16.7 Item 3.

8.1.1.8 Contact

The Contact header field provides a SIP or SIPS URI that can be used

to contact that specific instance of the UA for subsequent requests.

The Contact header field MUST be present and contain exactly one SIP

or SIPS URI in any request that can result in the establishment of a

dialog. For the methods defined in this specification, that includes

only the INVITE request. For these requests, the scope of the

Contact is global. That is, the Contact header field value contains

the URI at which the UA would like to receive requests, and this URI

MUST be valid even if used in subsequent requests outside of any

dialogs.

If the Request-URI or top Route header field value contains a SIPS

URI, the Contact header field MUST contain a SIPS URI as well.

For further information on the Contact header field, see Section

20.10.

8.1.1.9 Supported and Require

If the UAC supports extensions to SIP that can be applied by the

server to the response, the UAC SHOULD include a Supported header

field in the request listing the option tags (Section 19.2) for those

extensions.

The option tags listed MUST only refer to extensions defined in

standards-track RFCs. This is to prevent servers from insisting that

clients implement non-standard, vendor-defined features in order to

receive service. Extensions defined by experimental and

informational RFCs are explicitly excluded from usage with the

Supported header field in a request, since they too are often used to

document vendor-defined extensions.

If the UAC wishes to insist that a UAS understand an extension that

the UAC will apply to the request in order to process the request, it

MUST insert a Require header field into the request listing the

option tag for that extension. If the UAC wishes to apply an

extension to the request and insist that any proxies that are

Rosenberg, et. al. Standards Track [Page 40]

RFC 3261 SIP: Session Initiation Protocol June 2002

traversed understand that extension, it MUST insert a Proxy-Require

header field into the request listing the option tag for that

extension.

As with the Supported header field, the option tags in the Require

and Proxy-Require header fields MUST only refer to extensions defined

in standards-track RFCs.

8.1.1.10 Additional Message Components

After a new request has been created, and the header fields described

above have been properly constructed, any additional optional header

fields are added, as are any header fields specific to the method.

SIP requests MAY contain a MIME-encoded message-body. Regardless of

the type of body that a request contains, certain header fields must

be formulated to characterize the contents of the body. For further

information on these header fields, see Sections 20.11 through 20.15.

8.1.2 Sending the Request

The destination for the request is then computed. Unless there is

local policy specifying otherwise, the destination MUST be determined

by applying the DNS procedures described in [4] as follows. If the

first element in the route set indicated a strict router (resulting

in forming the request as described in Section 12.2.1.1), the

procedures MUST be applied to the Request-URI of the request.

Otherwise, the procedures are applied to the first Route header field

value in the request (if one exists), or to the request's Request-URI

if there is no Route header field present. These procedures yield an

ordered set of address, port, and transports to attempt. Independent

of which URI is used as input to the procedures of [4], if the

Request-URI specifies a SIPS resource, the UAC MUST follow the

procedures of [4] as if the input URI were a SIPS URI.

Local policy MAY specify an alternate set of destinations to attempt.

If the Request-URI contains a SIPS URI, any alternate destinations

MUST be contacted with TLS. Beyond that, there are no restrictions

on the alternate destinations if the request contains no Route header

field. This provides a simple alternative to a pre-existing route

set as a way to specify an outbound proxy. However, that approach

for configuring an outbound proxy is NOT RECOMMENDED; a pre-existing

route set with a single URI SHOULD be used instead. If the request

contains a Route header field, the request SHOULD be sent to the

locations derived from its topmost value, but MAY be sent to any

server that the UA is certain will honor the Route and Request-URI

policies specified in this document (as opposed to those in RFC

2543). In particular, a UAC configured with an outbound proxy SHOULD

Rosenberg, et. al. Standards Track [Page 41]

RFC 3261 SIP: Session Initiation Protocol June 2002

attempt to send the request to the location indicated in the first

Route header field value instead of adopting the policy of sending

all messages to the outbound proxy.

This ensures that outbound proxies that do not add Record-Route

header field values will drop out of the path of subsequent

requests. It allows endpoints that cannot resolve the first Route

URI to delegate that task to an outbound proxy.

The UAC SHOULD follow the procedures defined in [4] for stateful

elements, trying each address until a server is contacted. Each try

constitutes a new transaction, and therefore each carries a different

topmost Via header field value with a new branch parameter.

Furthermore, the transport value in the Via header field is set to

whatever transport was determined for the target server.

8.1.3 Processing Responses

Responses are first processed by the transport layer and then passed

up to the transaction layer. The transaction layer performs its

processing and then passes the response up to the TU. The majority

of response processing in the TU is method specific. However, there

are some general behaviors independent of the method.

8.1.3.1 Transaction Layer Errors

In some cases, the response returned by the transaction layer will

not be a SIP message, but rather a transaction layer error. When a

timeout error is received from the transaction layer, it MUST be

treated as if a 408 (Request Timeout) status code has been received.

If a fatal transport error is reported by the transport layer

(generally, due to fatal ICMP errors in UDP or connection failures in

TCP), the condition MUST be treated as a 503 (Service Unavailable)

status code.

8.1.3.2 Unrecognized Responses

A UAC MUST treat any final response it does not recognize as being

equivalent to the x00 response code of that class, and MUST be able

to process the x00 response code for all classes. For example, if a

UAC receives an unrecognized response code of 431, it can safely

assume that there was something wrong with its request and treat the

response as if it had received a 400 (Bad Request) response code. A

UAC MUST treat any provisional response different than 100 that it

does not recognize as 183 (Session Progress). A UAC MUST be able to

process 100 and 183 responses.

Rosenberg, et. al. Standards Track [Page 42]

RFC 3261 SIP: Session Initiation Protocol June 2002

8.1.3.3 Vias

If more than one Via header field value is present in a response, the

UAC SHOULD discard the message.

The presence of additional Via header field values that precede

the originator of the request suggests that the message was

misrouted or possibly corrupted.

8.1.3.4 Processing 3xx Responses

Upon receipt of a redirection response (for example, a 301 response

status code), clients SHOULD use the URI(s) in the Contact header

field to formulate one or more new requests based on the redirected

request. This process is similar to that of a proxy recursing on a

3xx class response as detailed in Sections 16.5 and 16.6. A client

starts with an initial target set containing exactly one URI, the

Request-URI of the original request. If a client wishes to formulate

new requests based on a 3xx class response to that request, it places

the URIs to try into the target set. Subject to the restrictions in

this specification, a client can choose which Contact URIs it places

into the target set. As with proxy recursion, a client processing

3xx class responses MUST NOT add any given URI to the target set more

than once. If the original request had a SIPS URI in the Request-

URI, the client MAY choose to recurse to a non-SIPS URI, but SHOULD

inform the user of the redirection to an insecure URI.

Any new request may receive 3xx responses themselves containing

the original URI as a contact. Two locations can be configured to

redirect to each other. Placing any given URI in the target set

only once prevents infinite redirection loops.

As the target set grows, the client MAY generate new requests to the

URIs in any order. A common mechanism is to order the set by the "q"

parameter value from the Contact header field value. Requests to the

URIs MAY be generated serially or in parallel. One approach is to

process groups of decreasing q-values serially and process the URIs

in each q-value group in parallel. Another is to perform only serial

processing in decreasing q-value order, arbitrarily choosing between

contacts of equal q-value.

If contacting an address in the list results in a failure, as defined

in the next paragraph, the element moves to the next address in the

list, until the list is exhausted. If the list is exhausted, then

the request has failed.

Rosenberg, et. al. Standards Track [Page 43]

RFC 3261 SIP: Session Initiation Protocol June 2002

Failures SHOULD be detected through failure response codes (codes

greater than 399); for network errors the client transaction will

report any transport layer failures to the transaction user. Note

that some response codes (detailed in 8.1.3.5) indicate that the

request can be retried; requests that are reattempted should not be

considered failures.

When a failure for a particular contact address is received, the

client SHOULD try the next contact address. This will involve

creating a new client transaction to deliver a new request.

In order to create a request based on a contact address in a 3xx

response, a UAC MUST copy the entire URI from the target set into the

Request-URI, except for the "method-param" and "header" URI

parameters (see Section 19.1.1 for a definition of these parameters).

It uses the "header" parameters to create header field values for the

new request, overwriting header field values associated with the

redirected request in accordance with the guidelines in Section

19.1.5.

Note that in some instances, header fields that have been

communicated in the contact address may instead append to existing

request header fields in the original redirected request. As a

general rule, if the header field can accept a comma-separated list

of values, then the new header field value MAY be appended to any

existing values in the original redirected request. If the header

field does not accept multiple values, the value in the original

redirected request MAY be overwritten by the header field value

communicated in the contact address. For example, if a contact

address is returned with the following value:

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

Then any Subject header field in the original redirected request is

overwritten, but the HTTP URL is merely appended to any existing

Call-Info header field values.

It is RECOMMENDED that the UAC reuse the same To, From, and Call-ID

used in the original redirected request, but the UAC MAY also choose

to update the Call-ID header field value for new requests, for

example.

Finally, once the new request has been constructed, it is sent using

a new client transaction, and therefore MUST have a new branch ID in

the top Via field as discussed in Section 8.1.1.7.

Rosenberg, et. al. Standards Track [Page 44]

RFC 3261 SIP: Session Initiation Protocol June 2002

In all other respects, requests sent upon receipt of a redirect

response SHOULD re-use the header fields and bodies of the original

request.

In some instances, Contact header field values may be cached at UAC

temporarily or permanently depending on the status code received and

the presence of an expiration interval; see Sections 21.3.2 and

21.3.3.

8.1.3.5 Processing 4xx Responses

Certain 4xx response codes require specific UA processing,

independent of the method.

If a 401 (Unauthorized) or 407 (Proxy Authentication Required)

response is received, the UAC SHOULD follow the authorization

procedures of Section 22.2 and Section 22.3 to retry the request with

credentials.

If a 413 (Request Entity Too Large) response is received (Section

21.4.11), the request contained a body that was longer than the UAS

was willing to accept. If possible, the UAC SHOULD retry the

request, either omitting the body or using one of a smaller length.

If a 415 (Unsupported Media Type) response is received (Section

21.4.13), the request contained media types not supported by the UAS.

The UAC SHOULD retry sending the request, this time only using

content with types listed in the Accept header field in the response,

with encodings listed in the Accept-Encoding header field in the

response, and with languages listed in the Accept-Language in the

response.

If a 416 (Unsupported URI Scheme) response is received (Section

21.4.14), the Request-URI used a URI scheme not supported by the

server. The client SHOULD retry the request, this time, using a SIP

URI.

If a 420 (Bad Extension) response is received (Section 21.4.15), the

request contained a Require or Proxy-Require header field listing an

option-tag for a feature not supported by a proxy or UAS. The UAC

SHOULD retry the request, this time omitting any extensions listed in

the Unsupported header field in the response.

In all of the above cases, the request is retried by creating a new

request with the appropriate modifications. This new request

constitutes a new transaction and SHOULD have the same value of the

Call-ID, To, and From of the previous request, but the CSeq should

contain a new sequence number that is one higher than the previous.

Rosenberg, et. al. Standards Track [Page 45]

RFC 3261 SIP: Session Initiation Protocol June 2002

With other 4xx responses, including those yet to be defined, a retry

may or may not be possible depending on the method and the use case.

8.2 UAS Behavior

When a request outside of a dialog is processed by a UAS, there is a

set of processing rules that are followed, independent of the method.

Section 12 gives guidance on how a UAS can tell whether a request is

inside or outside of a dialog.

Note that request processing is atomic. If a request is accepted,

all state changes associated with it MUST be performed. If it is

rejected, all state changes MUST NOT be performed.

UASs SHOULD process the requests in the order of the steps that

follow in this section (that is, starting with authentication, then

inspecting the method, the header fields, and so on throughout the

remainder of this section).

8.2.1 Method Inspection

Once a request is authenticated (or authentication is skipped), the

UAS MUST inspect the method of the request. If the UAS recognizes

but does not support the method of a request, it MUST generate a 405

(Method Not Allowed) response. Procedures for generating responses

are described in Section 8.2.6. The UAS MUST also add an Allow

header field to the 405 (Method Not Allowed) response. The Allow

header field MUST list the set of methods supported by the UAS

generating the message. The Allow header field is presented in

Section 20.5.

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

8.2.2 Header Inspection

If a UAS does not understand a header field in a request (that is,

the header field is not defined in this specification or in any

supported extension), the server MUST ignore that header field and

continue processing the message. A UAS SHOULD ignore any malformed

header fields that are not necessary for processing requests.

8.2.2.1 To and Request-URI

The To header field identifies the original recipient of the request

designated by the user identified in the From field. The original

recipient may or may not be the UAS processing the request, due to

call forwarding or other proxy operations. A UAS MAY apply any

policy it wishes to determine whether to accept requests when the To

Rosenberg, et. al. Standards Track [Page 46]

RFC 3261 SIP: Session Initiation Protocol June 2002

header field is not the identity of the UAS. However, it is

RECOMMENDED that a UAS accept requests even if they do not recognize

the URI scheme (for example, a tel: URI) in the To header field, or

if the To header field does not address a known or current user of

this UAS. If, on the other hand, the UAS decides to reject the

request, it SHOULD generate a response with a 403 (Forbidden) status

code and pass it to the server transaction for transmission.

However, the Request-URI identifies the UAS that is to process the

request. If the Request-URI uses a scheme not supported by the UAS,

it SHOULD reject the request with a 416 (Unsupported URI Scheme)

response. If the Request-URI does not identify an address that the

UAS is willing to accept requests for, it SHOULD reject the request

with a 404 (Not Found) response. Typically, a UA that uses the

REGISTER method to bind its address-of-record to a specific contact

address will see requests whose Request-URI equals that contact

address. Other potential sources of received Request-URIs include

the Contact header fields of requests and responses sent by the UA

that establish or refresh dialogs.

8.2.2.2 Merged Requests

If the request has no tag in the To header field, the UAS core MUST

check the request against ongoing transactions. If the From tag,

Call-ID, and CSeq exactly match those associated with an ongoing

transaction, but the request does not match that transaction (based

on the matching rules in Section 17.2.3), the UAS core SHOULD

generate a 482 (Loop Detected) response and pass it to the server

transaction.

The same request has arrived at the UAS more than once, following

different paths, most likely due to forking. The UAS processes

the first such request received and responds with a 482 (Loop

Detected) to the rest of them.

8.2.2.3 Require

Assuming the UAS decides that it is the proper element to process the

request, it examines the Require header field, if present.

The Require header field is used by a UAC to tell a UAS about SIP

extensions that the UAC expects the UAS to support in order to

process the request properly. Its format is described in Section

20.32. If a UAS does not understand an option-tag listed in a

Require header field, it MUST respond by generating a response with

status code 420 (Bad Extension). The UAS MUST add an Unsupported

header field, and list in it those options it does not understand

amongst those in the Require header field of the request.

Rosenberg, et. al. Standards Track [Page 47]

RFC 3261 SIP: Session Initiation Protocol June 2002

Note that Require and Proxy-Require MUST NOT be used in a SIP CANCEL

request, or in an ACK request sent for a non-2xx response. These

header fields MUST be ignored if they are present in these requests.

An ACK request for a 2xx response MUST contain only those Require and

Proxy-Require values that were present in the initial request.

Example:

UAC->UAS: INVITE sip:watson@bell-telephone.com SIP/2.0

Require: 100rel

UAS->UAC: SIP/2.0 420 Bad Extension

Unsupported: 100rel

This behavior ensures that the client-server interaction will

proceed without delay when all options are understood by both

sides, and only slow down if options are not understood (as in the

example above). For a well-matched client-server pair, the

interaction proceeds quickly, saving a round-trip often required

by negotiation mechanisms. In addition, it also removes ambiguity

when the client requires features that the server does not

understand. Some features, such as call handling fields, are only

of interest to end systems.

8.2.3 Content Processing

Assuming the UAS understands any extensions required by the client,

the UAS examines the body of the message, and the header fields that

describe it. If there are any bodies whose type (indicated by the

Content-Type), language (indicated by the Content-Language) or

encoding (indicated by the Content-Encoding) are not understood, and

that body part is not optional (as indicated by the Content-

Disposition header field), the UAS MUST reject the request with a 415

(Unsupported Media Type) response. The response MUST contain an

Accept header field listing the types of all bodies it understands,

in the event the request contained bodies of types not supported by

the UAS. If the request contained content encodings not understood

by the UAS, the response MUST contain an Accept-Encoding header field

listing the encodings understood by the UAS. If the request

contained content with languages not understood by the UAS, the

response MUST contain an Accept-Language header field indicating the

languages understood by the UAS. Beyond these checks, body handling

depends on the method and type. For further information on the

processing of content-specific header fields, see Section 7.4 as well

as Section 20.11 through 20.15.

Rosenberg, et. al. Standards Track [Page 48]

RFC 3261 SIP: Session Initiation Protocol June 2002

8.2.4 Applying Extensions

A UAS that wishes to apply some extension when generating the

response MUST NOT do so unless support for that extension is

indicated in the Supported header field in the request. If the

desired extension is not supported, the server SHOULD rely only on

baseline SIP and any other extensions supported by the client. In

rare circumstances, where the server cannot process the request

without the extension, the server MAY send a 421 (Extension Required)

response. This response indicates that the proper response cannot be

generated without support of a specific extension. The needed

extension(s) MUST be included in a Require header field in the

response. This behavior is NOT RECOMMENDED, as it will generally

break interoperability.

Any extensions applied to a non-421 response MUST be listed in a

Require header field included in the response. Of course, the server

MUST NOT apply extensions not listed in the Supported header field in

the request. As a result of this, the Require header field in a

response will only ever contain option tags defined in standards-

track RFCs.

8.2.5 Processing the Request

Assuming all of the checks in the previous subsections are passed,

the UAS processing becomes method-specific. Section 10 covers the

REGISTER request, Section 11 covers the OPTIONS request, Section 13

covers the INVITE request, and Section 15 covers the BYE request.

8.2.6 Generating the Response

When a UAS wishes to construct a response to a request, it follows

the general procedures detailed in the following subsections.

Additional behaviors specific to the response code in question, which

are not detailed in this section, may also be required.

Once all procedures associated with the creation of a response have

been completed, the UAS hands the response back to the server

transaction from which it received the request.

8.2.6.1 Sending a Provisional Response

One largely non-method-specific guideline for the generation of

responses is that UASs SHOULD NOT issue a provisional response for a

non-INVITE request. Rather, UASs SHOULD generate a final response to

a non-INVITE request as soon as possible.

Rosenberg, et. al. Standards Track [Page 49]

RFC 3261 SIP: Session Initiation Protocol June 2002

When a 100 (Trying) response is generated, any Timestamp header field

present in the request MUST be copied into this 100 (Trying)

response. If there is a delay in generating the response, the UAS

SHOULD add a delay value into the Timestamp value in the response.

This value MUST contain the difference between the time of sending of

the response and receipt of the request, measured in seconds.

8.2.6.2 Headers and Tags

The From field of the response MUST equal the From header field of

the request. The Call-ID header field of the response MUST equal the

Call-ID header field of the request. The CSeq header field of the

response MUST equal the CSeq field of the request. The Via header

field values in the response MUST equal the Via header field values

in the request and MUST maintain the same ordering.

If a request contained a To tag in the request, the To header field

in the response MUST equal that of the request. However, if the To

header field in the request did not contain a tag, the URI in the To

header field in the response MUST equal the URI in the To header

field; additionally, the UAS MUST add a tag to the To header field in

the response (with the exception of the 100 (Trying) response, in

which a tag MAY be present). This serves to identify the UAS that is

responding, possibly resulting in a component of a dialog ID. The

same tag MUST be used for all responses to that request, both final

and provisional (again excepting the 100 (Trying)). Procedures for

the generation of tags are defined in Section 19.3.

8.2.7 Stateless UAS Behavior

A stateless UAS is a UAS that does not maintain transaction state.

It replies to requests normally, but discards any state that would

ordinarily be retained by a UAS after a response has been sent. If a

stateless UAS receives a retransmission of a request, it regenerates

the response and resends it, just as if it were replying to the first

instance of the request. A UAS cannot be stateless unless the request

processing for that method would always result in the same response

if the requests are identical. This rules out stateless registrars,

for example. Stateless UASs do not use a transaction layer; they

receive requests directly from the transport layer and send responses

directly to the transport layer.

The stateless UAS role is needed primarily to handle unauthenticated

requests for which a challenge response is issued. If

unauthenticated requests were handled statefully, then malicious

floods of unauthenticated requests could create massive amounts of

Rosenberg, et. al. Standards Track [Page 50]

RFC 3261 SIP: Session Initiation Protocol June 2002

transaction state that might slow or completely halt call processing

in a UAS, effectively creating a denial of service condition; for

more information see Section 26.1.5.

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

o A stateless UAS MUST NOT send provisional (1xx) responses.

o A stateless UAS MUST NOT retransmit responses.

o A stateless UAS MUST ignore ACK requests.

o A stateless UAS MUST ignore CANCEL requests.

o To header tags MUST be generated for responses in a stateless

manner - in a manner that will generate the same tag for the

same request consistently. For information on tag construction

see Section 19.3.

In all other respects, a stateless UAS behaves in the same manner as

a stateful UAS. A UAS can operate in either a stateful or stateless

mode for each new request.

8.3 Redirect Servers

In some architectures it may be desirable to reduce the processing

load on proxy servers that are responsible for routing requests, and

improve signaling path robustness, by relying on redirection.

Redirection allows servers to push routing information for a request

back in a response to the client, thereby taking themselves out of

the loop of further messaging for this transaction while still aiding

in locating the target of the request. When the originator of the

request receives the redirection, it will send a new request based on

the URI(s) it has received. By propagating URIs from the core of the

network to its edges, redirection allows for considerable network

scalability.

A redirect server is logically constituted of a server transaction

layer and a transaction user that has access to a location service of

some kind (see Section 10 for more on registrars and location

services). This location service is effectively a database

containing mappings between a single URI and a set of one or more

alternative locations at which the target of that URI can be found.

A redirect server does not issue any SIP requests of its own. After

receiving a request other than CANCEL, the server either refuses the

request or gathers the list of alternative locations from the

Rosenberg, et. al. Standards Track [Page 51]

RFC 3261 SIP: Session Initiation Protocol June 2002

location service and returns a final response of class 3xx. For

well-formed CANCEL requests, it SHOULD return a 2xx response. This

response ends the SIP transaction. The redirect server maintains

transaction state for an entire SIP transaction. It is the

responsibility of clients to detect forwarding loops between redirect

servers.

When a redirect server returns a 3xx response to a request, it

populates the list of (one or more) alternative locations into the

Contact header field. An "expires" parameter to the Contact header

field values may also be supplied to indicate the lifetime of the

Contact data.

The Contact header field contains URIs giving the new locations or

user names to try, or may simply specify additional transport

parameters. A 301 (Moved Permanently) or 302 (Moved Temporarily)

response may also give the same location and username that was

targeted by the initial request but specify additional transport

parameters such as a different server or multicast address to try, or

a change of SIP transport from UDP to TCP or vice versa.

However, redirect servers MUST NOT redirect a request to a URI equal

to the one in the Request-URI; instead, provided that the URI does

not point to itself, the server MAY proxy the request to the

destination URI, or MAY reject it with a 404.

If a client is using an outbound proxy, and that proxy actually

redirects requests, a potential arises for infinite redirection

loops.

Note that a Contact header field value MAY also refer to a different

resource than the one originally called. For example, a SIP call

connected to PSTN gateway may need to deliver a special informational

announcement such as "The number you have dialed has been changed."

A Contact response header field can contain any suitable URI

indicating where the called party can be reached, not limited to SIP

URIs. For example, it could contain URIs for phones, fax, or irc (if

they were defined) or a mailto: (RFC 2368 [32]) URL. Section 26.4.4

discusses implications and limitations of redirecting a SIPS URI to a

non-SIPS URI.

The "expires" parameter of a Contact header field value indicates how

long the URI is valid. The value of the parameter is a number

indicating seconds. If this parameter is not provided, the value of

the Expires header field determines how long the URI is valid.

Malformed values SHOULD be treated as equivalent to 3600.

Rosenberg, et. al. Standards Track [Page 52]

RFC 3261 SIP: Session Initiation Protocol June 2002

This provides a modest level of backwards compatibility with RFC

2543, which allowed absolute times in this header field. If an

absolute time is received, it will be treated as malformed, and

then default to 3600.

Redirect servers MUST ignore features that are not understood

(including unrecognized header fields, any unknown option tags in

Require, or even method names) and proceed with the redirection of

the request in question.

9 Canceling a Request

The previous section has discussed general UA behavior for generating

requests and processing responses for requests of all methods. In

this section, we discuss a general purpose method, called CANCEL.

The CANCEL request, as the name implies, is used to cancel a previous

request sent by a client. Specifically, it asks the UAS to cease

processing the request and to generate an error response to that

request. CANCEL has no effect on a request to which a UAS has

already given a final response. Because of this, it is most useful

to CANCEL requests to which it can take a server long time to

respond. For this reason, CANCEL is best for INVITE requests, which

can take a long time to generate a response. In that usage, a UAS

that receives a CANCEL request for an INVITE, but has not yet sent a

final response, would "stop ringing", and then respond to the INVITE

with a specific error response (a 487).

CANCEL requests can be constructed and sent by both proxies and user

agent clients. Section 15 discusses under what conditions a UAC

would CANCEL an INVITE request, and Section 16.10 discusses proxy

usage of CANCEL.

A stateful proxy responds to a CANCEL, rather than simply forwarding

a response it would receive from a downstream element. For that

reason, CANCEL is referred to as a "hop-by-hop" request, since it is

responded to at each stateful proxy hop.

9.1 Client Behavior

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

INVITE.

Since requests other than INVITE are responded to immediately,

sending a CANCEL for a non-INVITE request would always create a

race condition.

Rosenberg, et. al. Standards Track [Page 53]

RFC 3261 SIP: Session Initiation Protocol June 2002

The following procedures are used to construct a CANCEL request. The

Request-URI, Call-ID, To, the numeric part of CSeq, and From header

fields in the CANCEL request MUST be identical to those in the

request being cancelled, including tags. A CANCEL constructed by a

client MUST have only a single Via header field value matching the

top Via value in the request being cancelled. Using the same values

for these header fields allows the CANCEL to be matched with the

request it cancels (Section 9.2 indicates how such matching occurs).

However, the method part of the CSeq header field MUST have a value

of CANCEL. This allows it to be identified and processed as a

transaction in its own right (See Section 17).

If the request being cancelled contains a Route header field, the

CANCEL request MUST include that Route header field's values.

This is needed so that stateless proxies are able to route CANCEL

requests properly.

The CANCEL request MUST NOT contain any Require or Proxy-Require

header fields.

Once the CANCEL is constructed, the client SHOULD check whether it

has received any response (provisional or final) for the request

being cancelled (herein referred to as the "original request").

If no provisional response has been received, the CANCEL request MUST

NOT be sent; rather, the client MUST wait for the arrival of a

provisional response before sending the request. If the original

request has generated a final response, the CANCEL SHOULD NOT be

sent, as it is an effective no-op, since CANCEL has no effect on

requests that have already generated a final response. When the

client decides to send the CANCEL, it creates a client transaction

for the CANCEL and passes it the CANCEL request along with the

destination address, port, and transport. The destination address,

port, and transport for the CANCEL MUST be identical to those used to

send the original request.

If it was allowed to send the CANCEL before receiving a response

for the previous request, the server could receive the CANCEL

before the original request.

Note that both the transaction corresponding to the original request

and the CANCEL transaction will complete independently. However, a

UAC canceling a request cannot rely on receiving a 487 (Request

Terminated) response for the original request, as an RFC 2543-

compliant UAS will not generate such a response. If there is no

final response for the original request in 64\*T1 seconds (T1 is

Rosenberg, et. al. Standards Track [Page 54]

RFC 3261 SIP: Session Initiation Protocol June 2002

defined in Section 17.1.1.1), the client SHOULD then consider the

original transaction cancelled and SHOULD destroy the client

transaction handling the original request.

9.2 Server Behavior

The CANCEL method requests that the TU at the server side cancel a

pending transaction. The TU determines the transaction to be

cancelled by taking the CANCEL request, and then assuming that the

request method is anything but CANCEL or ACK and applying the

transaction matching procedures of Section 17.2.3. The matching

transaction is the one to be cancelled.

The processing of a CANCEL request at a server depends on the type of

server. A stateless proxy will forward it, a stateful proxy might

respond to it and generate some CANCEL requests of its own, and a UAS

will respond to it. See Section 16.10 for proxy treatment of CANCEL.

A UAS first processes the CANCEL request according to the general UAS

processing described in Section 8.2. However, since CANCEL requests

are hop-by-hop and cannot be resubmitted, they cannot be challenged

by the server in order to get proper credentials in an Authorization

header field. Note also that CANCEL requests do not contain a

Require header field.

If the UAS did not find a matching transaction for the CANCEL

according to the procedure above, it SHOULD respond to the CANCEL

with a 481 (Call Leg/Transaction Does Not Exist). If the transaction

for the original request still exists, the behavior of the UAS on

receiving a CANCEL request depends on whether it has already sent a

final response for the original request. If it has, the CANCEL

request has no effect on the processing of the original request, no

effect on any session state, and no effect on the responses generated

for the original request. If the UAS has not issued a final response

for the original request, its behavior depends on the method of the

original request. If the original request was an INVITE, the UAS

SHOULD immediately respond to the INVITE with a 487 (Request

Terminated). A CANCEL request has no impact on the processing of

transactions with any other method defined in this specification.

Regardless of the method of the original request, as long as the

CANCEL matched an existing transaction, the UAS answers the CANCEL

request itself with a 200 (OK) response. This response is

constructed following the procedures described in Section 8.2.6

noting that the To tag of the response to the CANCEL and the To tag

in the response to the original request SHOULD be the same. The

response to CANCEL is passed to the server transaction for

transmission.

Rosenberg, et. al. Standards Track [Page 55]

RFC 3261 SIP: Session Initiation Protocol June 2002

10 Registrations

10.1 Overview

SIP offers a discovery capability. If a user wants to initiate a

session with another user, SIP must discover the current host(s) at

which the destination user is reachable. This discovery process is

frequently accomplished by SIP network elements such as proxy servers

and redirect servers which are responsible for receiving a request,

determining where to send it based on knowledge of the location of

the user, and then sending it there. To do this, SIP network

elements consult an abstract service known as a location service,

which provides address bindings for a particular domain. These

address bindings map an incoming SIP or SIPS URI, sip:bob@biloxi.com,

for example, to one or more URIs that are somehow "closer" to the

desired user, sip:bob@engineering.biloxi.com, for example.

Ultimately, a proxy will consult a location service that maps a

received URI to the user agent(s) at which the desired recipient is

currently residing.

Registration creates bindings in a location service for a particular

domain that associates an address-of-record URI with one or more

contact addresses. Thus, when a proxy for that domain receives a

request whose Request-URI matches the address-of-record, the proxy

will forward the request to the contact addresses registered to that

address-of-record. Generally, it only makes sense to register an

address-of-record at a domain's location service when requests for

that address-of-record would be routed to that domain. In most

cases, this means that the domain of the registration will need to

match the domain in the URI of the address-of-record.

There are many ways by which the contents of the location service can

be established. One way is administratively. In the above example,

Bob is known to be a member of the engineering department through

access to a corporate database. However, SIP provides a mechanism

for a UA to create a binding explicitly. This mechanism is known as

registration.

Registration entails sending a REGISTER request to a special type of

UAS known as a registrar. A registrar acts as the front end to the

location service for a domain, reading and writing mappings based on

the contents of REGISTER requests. This location service is then

typically consulted by a proxy server that is responsible for routing

requests for that domain.

An illustration of the overall registration process is given in

Figure 2. Note that the registrar and proxy server are logical roles

that can be played by a single device in a network; for purposes of

Rosenberg, et. al. Standards Track [Page 56]

RFC 3261 SIP: Session Initiation Protocol June 2002

clarity the two are separated in this illustration. Also note that

UAs may send requests through a proxy server in order to reach a

registrar if the two are separate elements.

SIP does not mandate a particular mechanism for implementing the

location service. The only requirement is that a registrar for some

domain MUST be able to read and write data to the location service,

and a proxy or a redirect server for that domain MUST be capable of

reading that same data. A registrar MAY be co-located with a

particular SIP proxy server for the same domain.

10.2 Constructing the REGISTER Request

REGISTER requests add, remove, and query bindings. A REGISTER

request can add a new binding between an address-of-record and one or

more contact addresses. Registration on behalf of a particular

address-of-record can be performed by a suitably authorized third

party. A client can also remove previous bindings or query to

determine which bindings are currently in place for an address-of-

record.

Except as noted, the construction of the REGISTER request and the

behavior of clients sending a REGISTER request is identical to the

general UAC behavior described in Section 8.1 and Section 17.1.

A REGISTER request does not establish a dialog. A UAC MAY include a

Route header field in a REGISTER request based on a pre-existing

route set as described in Section 8.1. The Record-Route header field

has no meaning in REGISTER requests or responses, and MUST be ignored

if present. In particular, the UAC MUST NOT create a new route set

based on the presence or absence of a Record-Route header field in

any response to a REGISTER request.

The following header fields, except Contact, MUST be included in a

REGISTER request. A Contact header field MAY be included:

Request-URI: The Request-URI names the domain of the location

service for which the registration is meant (for example,

"sip:chicago.com"). The "userinfo" and "@" components of the

SIP URI MUST NOT be present.

To: The To header field contains the address of record whose

registration is to be created, queried, or modified. The To

header field and the Request-URI field typically differ, as

the former contains a user name. This address-of-record MUST

be a SIP URI or SIPS URI.

Rosenberg, et. al. Standards Track [Page 57]

RFC 3261 SIP: Session Initiation Protocol June 2002

From: The From header field contains the address-of-record of the

person responsible for the registration. The value is the

same as the To header field unless the request is a third-

party registration.

Call-ID: All registrations from a UAC SHOULD use the same Call-ID

header field value for registrations sent to a particular

registrar.

If the same client were to use different Call-ID values, a

registrar could not detect whether a delayed REGISTER request

might have arrived out of order.

CSeq: The CSeq value guarantees proper ordering of REGISTER

requests. A UA MUST increment the CSeq value by one for each

REGISTER request with the same Call-ID.

Contact: REGISTER requests MAY contain a Contact header field with

zero or more values containing address bindings.

UAs MUST NOT send a new registration (that is, containing new Contact

header field values, as opposed to a retransmission) until they have

received a final response from the registrar for the previous one or

the previous REGISTER request has timed out.

Rosenberg, et. al. Standards Track [Page 58]

RFC 3261 SIP: Session Initiation Protocol June 2002

bob

+----+

| UA |

| |

+----+

|

|3)INVITE

| carol@chicago.com

chicago.com +--------+ V

+---------+ 2)Store|Location|4)Query +-----+

|Registrar|=======>| Service|<=======|Proxy|sip.chicago.com

+---------+ +--------+=======>+-----+

A 5)Resp |

| |

| |

1)REGISTER| |

| |

+----+ |

| UA |<-------------------------------+

cube2214a| | 6)INVITE

+----+ carol@cube2214a.chicago.com

carol

Figure 2: REGISTER example

The following Contact header parameters have a special meaning in

REGISTER requests:

action: The "action" parameter from RFC 2543 has been deprecated.

UACs SHOULD NOT use the "action" parameter.

expires: The "expires" parameter indicates how long the UA would

like the binding to be valid. The value is a number

indicating seconds. If this parameter is not provided, the

value of the Expires header field is used instead.

Implementations MAY treat values larger than 2\*\*32-1

(4294967295 seconds or 136 years) as equivalent to 2\*\*32-1.

Malformed values SHOULD be treated as equivalent to 3600.

10.2.1 Adding Bindings

The REGISTER request sent to a registrar includes the contact

address(es) to which SIP requests for the address-of-record should be

forwarded. The address-of-record is included in the To header field

of the REGISTER request.

Rosenberg, et. al. Standards Track [Page 59]

RFC 3261 SIP: Session Initiation Protocol June 2002

The Contact header field values of the request typically consist of

SIP or SIPS URIs that identify particular SIP endpoints (for example,

"sip:carol@cube2214a.chicago.com"), but they MAY use any URI scheme.

A SIP UA can choose to register telephone numbers (with the tel URL,

RFC 2806 [9]) or email addresses (with a mailto URL, RFC 2368 [32])

as Contacts for an address-of-record, for example.

For example, Carol, with address-of-record "sip:carol@chicago.com",

would register with the SIP registrar of the domain chicago.com. Her

registrations would then be used by a proxy server in the chicago.com

domain to route requests for Carol's address-of-record to her SIP

endpoint.

Once a client has established bindings at a registrar, it MAY send

subsequent registrations containing new bindings or modifications to

existing bindings as necessary. The 2xx response to the REGISTER

request will contain, in a Contact header field, a complete list of

bindings that have been registered for this address-of-record at this

registrar.

If the address-of-record in the To header field of a REGISTER request

is a SIPS URI, then any Contact header field values in the request

SHOULD also be SIPS URIs. Clients should only register non-SIPS URIs

under a SIPS address-of-record when the security of the resource

represented by the contact address is guaranteed by other means.

This may be applicable to URIs that invoke protocols other than SIP,

or SIP devices secured by protocols other than TLS.

Registrations do not need to update all bindings. Typically, a UA

only updates its own contact addresses.

10.2.1.1 Setting the Expiration Interval of Contact Addresses

When a client sends a REGISTER request, it MAY suggest an expiration

interval that indicates how long the client would like the

registration to be valid. (As described in Section 10.3, the

registrar selects the actual time interval based on its local

policy.)

There are two ways in which a client can suggest an expiration

interval for a binding: through an Expires header field or an

"expires" Contact header parameter. The latter allows expiration

intervals to be suggested on a per-binding basis when more than one

binding is given in a single REGISTER request, whereas the former

suggests an expiration interval for all Contact header field values

that do not contain the "expires" parameter.

Rosenberg, et. al. Standards Track [Page 60]

RFC 3261 SIP: Session Initiation Protocol June 2002

If neither mechanism for expressing a suggested expiration time is

present in a REGISTER, the client is indicating its desire for the

server to choose.

10.2.1.2 Preferences among Contact Addresses

If more than one Contact is sent in a REGISTER request, the

registering UA intends to associate all of the URIs in these Contact

header field values with the address-of-record present in the To

field. This list can be prioritized with the "q" parameter in the

Contact header field. The "q" parameter indicates a relative

preference for the particular Contact header field value compared to

other bindings for this address-of-record. Section 16.6 describes

how a proxy server uses this preference indication.

10.2.2 Removing Bindings

Registrations are soft state and expire unless refreshed, but can

also be explicitly removed. A client can attempt to influence the

expiration interval selected by the registrar as described in Section

10.2.1. A UA requests the immediate removal of a binding by

specifying an expiration interval of "0" for that contact address in

a REGISTER request. UAs SHOULD support this mechanism so that

bindings can be removed before their expiration interval has passed.

The REGISTER-specific Contact header field value of "\*" applies to

all registrations, but it MUST NOT be used unless the Expires header

field is present with a value of "0".

Use of the "\*" Contact header field value allows a registering UA

to remove all bindings associated with an address-of-record

without knowing their precise values.

10.2.3 Fetching Bindings

A success response to any REGISTER request contains the complete list

of existing bindings, regardless of whether the request contained a

Contact header field. If no Contact header field is present in a

REGISTER request, the list of bindings is left unchanged.

10.2.4 Refreshing Bindings

Each UA is responsible for refreshing the bindings that it has

previously established. A UA SHOULD NOT refresh bindings set up by

other UAs.

Rosenberg, et. al. Standards Track [Page 61]

RFC 3261 SIP: Session Initiation Protocol June 2002

The 200 (OK) response from the registrar contains a list of Contact

fields enumerating all current bindings. The UA compares each

contact address to see if it created the contact address, using

comparison rules in Section 19.1.4. If so, it updates the expiration

time interval according to the expires parameter or, if absent, the

Expires field value. The UA then issues a REGISTER request for each

of its bindings before the expiration interval has elapsed. It MAY

combine several updates into one REGISTER request.

A UA SHOULD use the same Call-ID for all registrations during a

single boot cycle. Registration refreshes SHOULD be sent to the same

network address as the original registration, unless redirected.

10.2.5 Setting the Internal Clock

If the response for a REGISTER request contains a Date header field,

the client MAY use this header field to learn the current time in

order to set any internal clocks.

10.2.6 Discovering a Registrar

UAs can use three ways to determine the address to which to send

registrations: by configuration, using the address-of-record, and

multicast. A UA can be configured, in ways beyond the scope of this

specification, with a registrar address. If there is no configured

registrar address, the UA SHOULD use the host part of the address-

of-record as the Request-URI and address the request there, using the

normal SIP server location mechanisms [4]. For example, the UA for

the user "sip:carol@chicago.com" addresses the REGISTER request to

"sip:chicago.com".

Finally, a UA can be configured to use multicast. Multicast

registrations are addressed to the well-known "all SIP servers"

multicast address "sip.mcast.net" (224.0.1.75 for IPv4). No well-

known IPv6 multicast address has been allocated; such an allocation

will be documented separately when needed. SIP UAs MAY listen to

that address and use it to become aware of the location of other

local users (see [33]); however, they do not respond to the request.

Multicast registration may be inappropriate in some environments,

for example, if multiple businesses share the same local area

network.

10.2.7 Transmitting a Request

Once the REGISTER method has been constructed, and the destination of

the message identified, UACs follow the procedures described in

Section 8.1.2 to hand off the REGISTER to the transaction layer.

Rosenberg, et. al. Standards Track [Page 62]

RFC 3261 SIP: Session Initiation Protocol June 2002

If the transaction layer returns a timeout error because the REGISTER

yielded no response, the UAC SHOULD NOT immediately re-attempt a

registration to the same registrar.

An immediate re-attempt is likely to also timeout. Waiting some

reasonable time interval for the conditions causing the timeout to

be corrected reduces unnecessary load on the network. No specific

interval is mandated.

10.2.8 Error Responses

If a UA receives a 423 (Interval Too Brief) response, it MAY retry

the registration after making the expiration interval of all contact

addresses in the REGISTER request equal to or greater than the

expiration interval within the Min-Expires header field of the 423

(Interval Too Brief) response.

10.3 Processing REGISTER Requests

A registrar is a UAS that responds to REGISTER requests and maintains

a list of bindings that are accessible to proxy servers and redirect

servers within its administrative domain. A registrar handles

requests according to Section 8.2 and Section 17.2, but it accepts

only REGISTER requests. A registrar MUST not generate 6xx responses.

A registrar MAY redirect REGISTER requests as appropriate. One

common usage would be for a registrar listening on a multicast

interface to redirect multicast REGISTER requests to its own unicast

interface with a 302 (Moved Temporarily) response.

Registrars MUST ignore the Record-Route header field if it is

included in a REGISTER request. Registrars MUST NOT include a

Record-Route header field in any response to a REGISTER request.

A registrar might receive a request that traversed a proxy which

treats REGISTER as an unknown request and which added a Record-

Route header field value.

A registrar has to know (for example, through configuration) the set

of domain(s) for which it maintains bindings. REGISTER requests MUST

be processed by a registrar in the order that they are received.

REGISTER requests MUST also be processed atomically, meaning that a

particular REGISTER request is either processed completely or not at

all. Each REGISTER message MUST be processed independently of any

other registration or binding changes.

Rosenberg, et. al. Standards Track [Page 63]

RFC 3261 SIP: Session Initiation Protocol June 2002

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

1. The registrar inspects the Request-URI to determine whether it

has access to bindings for the domain identified in the

Request-URI. If not, and if the server also acts as a proxy

server, the server SHOULD forward the request to the addressed

domain, following the general behavior for proxying messages

described in Section 16.

2. To guarantee that the registrar supports any necessary

extensions, the registrar MUST process the Require header field

values as described for UASs in Section 8.2.2.

3. A registrar SHOULD authenticate the UAC. Mechanisms for the

authentication of SIP user agents are described in Section 22.

Registration behavior in no way overrides the generic

authentication framework for SIP. If no authentication

mechanism is available, the registrar MAY take the From address

as the asserted identity of the originator of the request.

4. The registrar SHOULD determine if the authenticated user is

authorized to modify registrations for this address-of-record.

For example, a registrar might consult an authorization

database that maps user names to a list of addresses-of-record

for which that user has authorization to modify bindings. If

the authenticated user is not authorized to modify bindings,

the registrar MUST return a 403 (Forbidden) and skip the

remaining steps.

In architectures that support third-party registration, one

entity may be responsible for updating the registrations

associated with multiple addresses-of-record.

5. The registrar extracts the address-of-record from the To header

field of the request. If the address-of-record is not valid

for the domain in the Request-URI, the registrar MUST send a

404 (Not Found) response and skip the remaining steps. The URI

MUST then be converted to a canonical form. To do that, all

URI parameters MUST be removed (including the user-param), and

any escaped characters MUST be converted to their unescaped

form. The result serves as an index into the list of bindings.

Rosenberg, et. al. Standards Track [Page 64]

RFC 3261 SIP: Session Initiation Protocol June 2002

6. The registrar checks whether the request contains the Contact

header field. If not, it skips to the last step. If the

Contact header field is present, the registrar checks if there

is one Contact field value that contains the special value "\*"

and an Expires field. If the request has additional Contact

fields or an expiration time other than zero, the request is

invalid, and the server MUST return a 400 (Invalid Request) and

skip the remaining steps. If not, the registrar checks whether

the Call-ID agrees with the value stored for each binding. If

not, it MUST remove the binding. If it does agree, it MUST

remove the binding only if the CSeq in the request is higher

than the value stored for that binding. Otherwise, the update

MUST be aborted and the request fails.

7. The registrar now processes each contact address in the Contact

header field in turn. For each address, it determines the

expiration interval as follows:

- If the field value has an "expires" parameter, that value

MUST be taken as the requested expiration.

- If there is no such parameter, but the request has an

Expires header field, that value MUST be taken as the

requested expiration.

- If there is neither, a locally-configured default value MUST

be taken as the requested expiration.

The registrar MAY choose an expiration less than the requested

expiration interval. If and only if the requested expiration

interval is greater than zero AND smaller than one hour AND

less than a registrar-configured minimum, the registrar MAY

reject the registration with a response of 423 (Interval Too

Brief). This response MUST contain a Min-Expires header field

that states the minimum expiration interval the registrar is

willing to honor. It then skips the remaining steps.

Allowing the registrar to set the registration interval

protects it against excessively frequent registration refreshes

while limiting the state that it needs to maintain and

decreasing the likelihood of registrations going stale. The

expiration interval of a registration is frequently used in the

creation of services. An example is a follow-me service, where

the user may only be available at a terminal for a brief

period. Therefore, registrars should accept brief

registrations; a request should only be rejected if the

interval is so short that the refreshes would degrade registrar

performance.

Rosenberg, et. al. Standards Track [Page 65]

RFC 3261 SIP: Session Initiation Protocol June 2002

For each address, the registrar then searches the list of

current bindings using the URI comparison rules. If the

binding does not exist, it is tentatively added. If the

binding does exist, the registrar checks the Call-ID value. If

the Call-ID value in the existing binding differs from the

Call-ID value in the request, the binding MUST be removed if

the expiration time is zero and updated otherwise. If they are

the same, the registrar compares the CSeq value. If the value

is higher than that of the existing binding, it MUST update or

remove the binding as above. If not, the update MUST be

aborted and the request fails.

This algorithm ensures that out-of-order requests from the same

UA are ignored.

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

the request.

The binding updates MUST be committed (that is, made visible to

the proxy or redirect server) if and only if all binding

updates and additions succeed. If any one of them fails (for

example, because the back-end database commit failed), the

request MUST fail with a 500 (Server Error) response and all

tentative binding updates MUST be removed.

8. The registrar returns a 200 (OK) response. The response MUST

contain Contact header field values enumerating all current

bindings. Each Contact value MUST feature an "expires"

parameter indicating its expiration interval chosen by the

registrar. The response SHOULD include a Date header field.

11 Querying for Capabilities

The SIP method OPTIONS allows a UA to query another UA or a proxy

server as to its capabilities. This allows a client to discover

information about the supported methods, content types, extensions,

codecs, etc. without "ringing" the other party. For example, before

a client inserts a Require header field into an INVITE listing an

option that it is not certain the destination UAS supports, the

client can query the destination UAS with an OPTIONS to see if this

option is returned in a Supported header field. All UAs MUST support

the OPTIONS method.

The target of the OPTIONS request is identified by the Request-URI,

which could identify another UA or a SIP server. If the OPTIONS is

addressed to a proxy server, the Request-URI is set without a user

part, similar to the way a Request-URI is set for a REGISTER request.

Rosenberg, et. al. Standards Track [Page 66]

RFC 3261 SIP: Session Initiation Protocol June 2002

Alternatively, a server receiving an OPTIONS request with a Max-

Forwards header field value of 0 MAY respond to the request

regardless of the Request-URI.

This behavior is common with HTTP/1.1. This behavior can be used

as a "traceroute" functionality to check the capabilities of

individual hop servers by sending a series of OPTIONS requests

with incremented Max-Forwards values.

As is the case for general UA behavior, the transaction layer can

return a timeout error if the OPTIONS yields no response. This may

indicate that the target is unreachable and hence unavailable.

An OPTIONS request MAY be sent as part of an established dialog to

query the peer on capabilities that may be utilized later in the

dialog.

11.1 Construction of OPTIONS Request

An OPTIONS request is constructed using the standard rules for a SIP

request as discussed in Section 8.1.1.

A Contact header field MAY be present in an OPTIONS.

An Accept header field SHOULD be included to indicate the type of

message body the UAC wishes to receive in the response. Typically,

this is set to a format that is used to describe the media

capabilities of a UA, such as SDP (application/sdp).

The response to an OPTIONS request is assumed to be scoped to the

Request-URI in the original request. However, only when an OPTIONS

is sent as part of an established dialog is it guaranteed that future

requests will be received by the server that generated the OPTIONS

response.

Example OPTIONS request:

OPTIONS sip:carol@chicago.com SIP/2.0

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKhjhs8ass877

Max-Forwards: 70

To: <sip:carol@chicago.com>

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 63104 OPTIONS

Contact: <sip:alice@pc33.atlanta.com>

Accept: application/sdp

Content-Length: 0

Rosenberg, et. al. Standards Track [Page 67]

RFC 3261 SIP: Session Initiation Protocol June 2002

11.2 Processing of OPTIONS Request

The response to an OPTIONS is constructed using the standard rules

for a SIP response as discussed in Section 8.2.6. The response code

chosen MUST be the same that would have been chosen had the request

been an INVITE. That is, a 200 (OK) would be returned if the UAS is

ready to accept a call, a 486 (Busy Here) would be returned if the

UAS is busy, etc. This allows an OPTIONS request to be used to

determine the basic state of a UAS, which can be an indication of

whether the UAS will accept an INVITE request.

An OPTIONS request received within a dialog generates a 200 (OK)

response that is identical to one constructed outside a dialog and

does not have any impact on the dialog.

This use of OPTIONS has limitations due to the differences in proxy

handling of OPTIONS and INVITE requests. While a forked INVITE can

result in multiple 200 (OK) responses being returned, a forked

OPTIONS will only result in a single 200 (OK) response, since it is

treated by proxies using the non-INVITE handling. See Section 16.7

for the normative details.

If the response to an OPTIONS is generated by a proxy server, the

proxy returns a 200 (OK), listing the capabilities of the server.

The response does not contain a message body.

Allow, Accept, Accept-Encoding, Accept-Language, and Supported header

fields SHOULD be present in a 200 (OK) response to an OPTIONS

request. If the response is generated by a proxy, the Allow header

field SHOULD be omitted as it is ambiguous since a proxy is method

agnostic. Contact header fields MAY be present in a 200 (OK)

response and have the same semantics as in a 3xx response. That is,

they may list a set of alternative names and methods of reaching the

user. A Warning header field MAY be present.

A message body MAY be sent, the type of which is determined by the

Accept header field in the OPTIONS request (application/sdp is the

default if the Accept header field is not present). If the types

include one that can describe media capabilities, the UAS SHOULD

include a body in the response for that purpose. Details on the

construction of such a body in the case of application/sdp are

described in [13].

Rosenberg, et. al. Standards Track [Page 68]

RFC 3261 SIP: Session Initiation Protocol June 2002

Example OPTIONS response generated by a UAS (corresponding to the

request in Section 11.1):

SIP/2.0 200 OK

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKhjhs8ass877

;received=192.0.2.4

To: <sip:carol@chicago.com>;tag=93810874

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 63104 OPTIONS

Contact: <sip:carol@chicago.com>

Contact: <mailto:carol@chicago.com>

Allow: INVITE, ACK, CANCEL, OPTIONS, BYE

Accept: application/sdp

Accept-Encoding: gzip

Accept-Language: en

Supported: foo

Content-Type: application/sdp

Content-Length: 274

(SDP not shown)

12 Dialogs

A key concept for a user agent is that of a dialog. A dialog

represents a peer-to-peer SIP relationship between two user agents

that persists for some time. The dialog facilitates sequencing of

messages between the user agents and proper routing of requests

between both of them. The dialog represents a context in which to

interpret SIP messages. Section 8 discussed method independent UA

processing for requests and responses outside of a dialog. This

section discusses how those requests and responses are used to

construct a dialog, and then how subsequent requests and responses

are sent within a dialog.

A dialog is identified at each UA with a dialog ID, which consists of

a Call-ID value, a local tag and a remote tag. The dialog ID at each

UA involved in the dialog is not the same. Specifically, the local

tag at one UA is identical to the remote tag at the peer UA. The

tags are opaque tokens that facilitate the generation of unique

dialog IDs.

A dialog ID is also associated with all responses and with any

request that contains a tag in the To field. The rules for computing

the dialog ID of a message depend on whether the SIP element is a UAC

or UAS. For a UAC, the Call-ID value of the dialog ID is set to the

Call-ID of the message, the remote tag is set to the tag in the To

field of the message, and the local tag is set to the tag in the From

Rosenberg, et. al. Standards Track [Page 69]

RFC 3261 SIP: Session Initiation Protocol June 2002

field of the message (these rules apply to both requests and

responses). As one would expect for a UAS, the Call-ID value of the

dialog ID is set to the Call-ID of the message, the remote tag is set

to the tag in the From field of the message, and the local tag is set

to the tag in the To field of the message.

A dialog contains certain pieces of state needed for further message

transmissions within the dialog. This state consists of the dialog

ID, a local sequence number (used to order requests from the UA to

its peer), a remote sequence number (used to order requests from its

peer to the UA), a local URI, a remote URI, remote target, a boolean

flag called "secure", and a route set, which is an ordered list of

URIs. The route set is the list of servers that need to be traversed

to send a request to the peer. A dialog can also be in the "early"

state, which occurs when it is created with a provisional response,

and then transition to the "confirmed" state when a 2xx final

response arrives. For other responses, or if no response arrives at

all on that dialog, the early dialog terminates.

12.1 Creation of a Dialog

Dialogs are created through the generation of non-failure responses

to requests with specific methods. Within this specification, only

2xx and 101-199 responses with a To tag, where the request was

INVITE, will establish a dialog. A dialog established by a non-final

response to a request is in the "early" state and it is called an

early dialog. Extensions MAY define other means for creating

dialogs. Section 13 gives more details that are specific to the

INVITE method. Here, we describe the process for creation of dialog

state that is not dependent on the method.

UAs MUST assign values to the dialog ID components as described

below.

12.1.1 UAS behavior

When a UAS responds to a request with a response that establishes a

dialog (such as a 2xx to INVITE), the UAS MUST copy all Record-Route

header field values from the request into the response (including the

URIs, URI parameters, and any Record-Route header field parameters,

whether they are known or unknown to the UAS) and MUST maintain the

order of those values. The UAS MUST add a Contact header field to

the response. The Contact header field contains an address where the

UAS would like to be contacted for subsequent requests in the dialog

(which includes the ACK for a 2xx response in the case of an INVITE).

Generally, the host portion of this URI is the IP address or FQDN of

the host. The URI provided in the Contact header field MUST be a SIP

or SIPS URI. If the request that initiated the dialog contained a

Rosenberg, et. al. Standards Track [Page 70]

RFC 3261 SIP: Session Initiation Protocol June 2002

SIPS URI in the Request-URI or in the top Record-Route header field

value, if there was any, or the Contact header field if there was no

Record-Route header field, the Contact header field in the response

MUST be a SIPS URI. The URI SHOULD have global scope (that is, the

same URI can be used in messages outside this dialog). The same way,

the scope of the URI in the Contact header field of the INVITE is not

limited to this dialog either. It can therefore be used in messages

to the UAC even outside this dialog.

The UAS then constructs the state of the dialog. This state MUST be

maintained for the duration of the dialog.

If the request arrived over TLS, and the Request-URI contained a SIPS

URI, the "secure" flag is set to TRUE.

The route set MUST be set to the list of URIs in the Record-Route

header field from the request, taken in order and preserving all URI

parameters. If no Record-Route header field is present in the

request, the route set MUST be set to the empty set. This route set,

even if empty, overrides any pre-existing route set for future

requests in this dialog. The remote target MUST be set to the URI

from the Contact header field of the request.

The remote sequence number MUST be set to the value of the sequence

number in the CSeq header field of the request. The local sequence

number MUST be empty. The call identifier component of the dialog ID

MUST be set to the value of the Call-ID in the request. The local

tag component of the dialog ID MUST be set to the tag in the To field

in the response to the request (which always includes a tag), and the

remote tag component of the dialog ID MUST be set to the tag from the

From field in the request. A UAS MUST be prepared to receive a

request without a tag in the From field, in which case the tag is

considered to have a value of null.

This is to maintain backwards compatibility with RFC 2543, which

did not mandate From tags.

The remote URI MUST be set to the URI in the From field, and the

local URI MUST be set to the URI in the To field.

12.1.2 UAC Behavior

When a UAC sends a request that can establish a dialog (such as an

INVITE) it MUST provide a SIP or SIPS URI with global scope (i.e.,

the same SIP URI can be used in messages outside this dialog) in the

Contact header field of the request. If the request has a Request-

URI or a topmost Route header field value with a SIPS URI, the

Contact header field MUST contain a SIPS URI.

Rosenberg, et. al. Standards Track [Page 71]

RFC 3261 SIP: Session Initiation Protocol June 2002

When a UAC receives a response that establishes a dialog, it

constructs the state of the dialog. This state MUST be maintained

for the duration of the dialog.

If the request was sent over TLS, and the Request-URI contained a

SIPS URI, the "secure" flag is set to TRUE.

The route set MUST be set to the list of URIs in the Record-Route

header field from the response, taken in reverse order and preserving

all URI parameters. If no Record-Route header field is present in

the response, the route set MUST be set to the empty set. This route

set, even if empty, overrides any pre-existing route set for future

requests in this dialog. The remote target MUST be set to the URI

from the Contact header field of the response.

The local sequence number MUST be set to the value of the sequence

number in the CSeq header field of the request. The remote sequence

number MUST be empty (it is established when the remote UA sends a

request within the dialog). The call identifier component of the

dialog ID MUST be set to the value of the Call-ID in the request.

The local tag component of the dialog ID MUST be set to the tag in

the From field in the request, and the remote tag component of the

dialog ID MUST be set to the tag in the To field of the response. A

UAC MUST be prepared to receive a response without a tag in the To

field, in which case the tag is considered to have a value of null.

This is to maintain backwards compatibility with RFC 2543, which

did not mandate To tags.

The remote URI MUST be set to the URI in the To field, and the local

URI MUST be set to the URI in the From field.

12.2 Requests within a Dialog

Once a dialog has been established between two UAs, either of them

MAY initiate new transactions as needed within the dialog. The UA

sending the request will take the UAC role for the transaction. The

UA receiving the request will take the UAS role. Note that these may

be different roles than the UAs held during the transaction that

established the dialog.

Requests within a dialog MAY contain Record-Route and Contact header

fields. However, these requests do not cause the dialog's route set

to be modified, although they may modify the remote target URI.

Specifically, requests that are not target refresh requests do not

modify the dialog's remote target URI, and requests that are target

refresh requests do. For dialogs that have been established with an

Rosenberg, et. al. Standards Track [Page 72]

RFC 3261 SIP: Session Initiation Protocol June 2002

INVITE, the only target refresh request defined is re-INVITE (see

Section 14). Other extensions may define different target refresh

requests for dialogs established in other ways.

Note that an ACK is NOT a target refresh request.

Target refresh requests only update the dialog's remote target URI,

and not the route set formed from the Record-Route. Updating the

latter would introduce severe backwards compatibility problems with

RFC 2543-compliant systems.

12.2.1 UAC Behavior

12.2.1.1 Generating the Request

A request within a dialog is constructed by using many of the

components of the state stored as part of the dialog.

The URI in the To field of the request MUST be set to the remote URI

from the dialog state. The tag in the To header field of the request

MUST be set to the remote tag of the dialog ID. The From URI of the

request MUST be set to the local URI from the dialog state. The tag

in the From header field of the request MUST be set to the local tag

of the dialog ID. If the value of the remote or local tags is null,

the tag parameter MUST be omitted from the To or From header fields,

respectively.

Usage of the URI from the To and From fields in the original

request within subsequent requests is done for backwards

compatibility with RFC 2543, which used the URI for dialog

identification. In this specification, only the tags are used for

dialog identification. It is expected that mandatory reflection

of the original To and From URI in mid-dialog requests will be

deprecated in a subsequent revision of this specification.

The Call-ID of the request MUST be set to the Call-ID of the dialog.

Requests within a dialog MUST contain strictly monotonically

increasing and contiguous CSeq sequence numbers (increasing-by-one)

in each direction (excepting ACK and CANCEL of course, whose numbers

equal the requests being acknowledged or cancelled). Therefore, if

the local sequence number is not empty, the value of the local

sequence number MUST be incremented by one, and this value MUST be

placed into the CSeq header field. If the local sequence number is

empty, an initial value MUST be chosen using the guidelines of

Section 8.1.1.5. The method field in the CSeq header field value

MUST match the method of the request.

Rosenberg, et. al. Standards Track [Page 73]

RFC 3261 SIP: Session Initiation Protocol June 2002

With a length of 32 bits, a client could generate, within a single

call, one request a second for about 136 years before needing to

wrap around. The initial value of the sequence number is chosen

so that subsequent requests within the same call will not wrap

around. A non-zero initial value allows clients to use a time-

based initial sequence number. A client could, for example,

choose the 31 most significant bits of a 32-bit second clock as an

initial sequence number.

The UAC uses the remote target and route set to build the Request-URI

and Route header field of the request.

If the route set is empty, the UAC MUST place the remote target URI

into the Request-URI. The UAC MUST NOT add a Route header field to

the request.

If the route set is not empty, and the first URI in the route set

contains the lr parameter (see Section 19.1.1), the UAC MUST place

the remote target URI into the Request-URI and MUST include a Route

header field containing the route set values in order, including all

parameters.

If the route set is not empty, and its first URI does not contain the

lr parameter, the UAC MUST place the first URI from the route set

into the Request-URI, stripping any parameters that are not allowed

in a Request-URI. The UAC MUST add a Route header field containing

the remainder of the route set values in order, including all

parameters. The UAC MUST then place the remote target URI into the

Route header field as the last value.

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

set contains:

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

The request will be formed with the following Request-URI and Route

header field:

METHOD sip:proxy1

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

If the first URI of the route set does not contain the lr

parameter, the proxy indicated does not understand the routing

mechanisms described in this document and will act as specified in

RFC 2543, replacing the Request-URI with the first Route header

field value it receives while forwarding the message. Placing the

Request-URI at the end of the Route header field preserves the

Rosenberg, et. al. Standards Track [Page 74]

RFC 3261 SIP: Session Initiation Protocol June 2002

information in that Request-URI across the strict router (it will

be returned to the Request-URI when the request reaches a loose-

router).

A UAC SHOULD include a Contact header field in any target refresh

requests within a dialog, and unless there is a need to change it,

the URI SHOULD be the same as used in previous requests within the

dialog. If the "secure" flag is true, that URI MUST be a SIPS URI.

As discussed in Section 12.2.2, a Contact header field in a target

refresh request updates the remote target URI. This allows a UA to

provide a new contact address, should its address change during the

duration of the dialog.

However, requests that are not target refresh requests do not affect

the remote target URI for the dialog.

The rest of the request is formed as described in Section 8.1.1.

Once the request has been constructed, the address of the server is

computed and the request is sent, using the same procedures for

requests outside of a dialog (Section 8.1.2).

The procedures in Section 8.1.2 will normally result in the

request being sent to the address indicated by the topmost Route

header field value or the Request-URI if no Route header field is

present. Subject to certain restrictions, they allow the request

to be sent to an alternate address (such as a default outbound

proxy not represented in the route set).

12.2.1.2 Processing the Responses

The UAC will receive responses to the request from the transaction

layer. If the client transaction returns a timeout, this is treated

as a 408 (Request Timeout) response.

The behavior of a UAC that receives a 3xx response for a request sent

within a dialog is the same as if the request had been sent outside a

dialog. This behavior is described in Section 8.1.3.4.

Note, however, that when the UAC tries alternative locations, it

still uses the route set for the dialog to build the Route header

of the request.

When a UAC receives a 2xx response to a target refresh request, it

MUST replace the dialog's remote target URI with the URI from the

Contact header field in that response, if present.

Rosenberg, et. al. Standards Track [Page 75]

RFC 3261 SIP: Session Initiation Protocol June 2002

If the response for a request within a dialog is a 481

(Call/Transaction Does Not Exist) or a 408 (Request Timeout), the UAC

SHOULD terminate the dialog. A UAC SHOULD also terminate a dialog if

no response at all is received for the request (the client

transaction would inform the TU about the timeout.)

For INVITE initiated dialogs, terminating the dialog consists of

sending a BYE.

12.2.2 UAS Behavior

Requests sent within a dialog, as any other requests, are atomic. If

a particular request is accepted by the UAS, all the state changes

associated with it are performed. If the request is rejected, none

of the state changes are performed.

Note that some requests, such as INVITEs, affect several pieces of

state.

The UAS will receive the request from the transaction layer. If the

request has a tag in the To header field, the UAS core computes the

dialog identifier corresponding to the request and compares it with

existing dialogs. If there is a match, this is a mid-dialog request.

In that case, the UAS first applies the same processing rules for

requests outside of a dialog, discussed in Section 8.2.

If the request has a tag in the To header field, but the dialog

identifier does not match any existing dialogs, the UAS may have

crashed and restarted, or it may have received a request for a

different (possibly failed) UAS (the UASs can construct the To tags

so that a UAS can identify that the tag was for a UAS for which it is

providing recovery). Another possibility is that the incoming

request has been simply misrouted. Based on the To tag, the UAS MAY

either accept or reject the request. Accepting the request for

acceptable To tags provides robustness, so that dialogs can persist

even through crashes. UAs wishing to support this capability must

take into consideration some issues such as choosing monotonically

increasing CSeq sequence numbers even across reboots, reconstructing

the route set, and accepting out-of-range RTP timestamps and sequence

numbers.

If the UAS wishes to reject the request because it does not wish to

recreate the dialog, it MUST respond to the request with a 481

(Call/Transaction Does Not Exist) status code and pass that to the

server transaction.

Rosenberg, et. al. Standards Track [Page 76]

RFC 3261 SIP: Session Initiation Protocol June 2002

Requests that do not change in any way the state of a dialog may be

received within a dialog (for example, an OPTIONS request). They are

processed as if they had been received outside the dialog.

If the remote sequence number is empty, it MUST be set to the value

of the sequence number in the CSeq header field value in the request.

If the remote sequence number was not empty, but the sequence number

of the request is lower than the remote sequence number, the request

is out of order and MUST be rejected with a 500 (Server Internal

Error) response. If the remote sequence number was not empty, and

the sequence number of the request is greater than the remote

sequence number, the request is in order. It is possible for the

CSeq sequence number to be higher than the remote sequence number by

more than one. This is not an error condition, and a UAS SHOULD be

prepared to receive and process requests with CSeq values more than

one higher than the previous received request. The UAS MUST then set

the remote sequence number to the value of the sequence number in the

CSeq header field value in the request.

If a proxy challenges a request generated by the UAC, the UAC has

to resubmit the request with credentials. The resubmitted request

will have a new CSeq number. The UAS will never see the first

request, and thus, it will notice a gap in the CSeq number space.

Such a gap does not represent any error condition.

When a UAS receives a target refresh request, it MUST replace the

dialog's remote target URI with the URI from the Contact header field

in that request, if present.

12.3 Termination of a Dialog

Independent of the method, if a request outside of a dialog generates

a non-2xx final response, any early dialogs created through

provisional responses to that request are terminated. The mechanism

for terminating confirmed dialogs is method specific. In this

specification, the BYE method terminates a session and the dialog

associated with it. See Section 15 for details.

13 Initiating a Session

13.1 Overview

When a user agent client desires to initiate a session (for example,

audio, video, or a game), it formulates an INVITE request. The

INVITE request asks a server to establish a session. This request

may be forwarded by proxies, eventually arriving at one or more UAS

that can potentially accept the invitation. These UASs will

frequently need to query the user about whether to accept the

Rosenberg, et. al. Standards Track [Page 77]

RFC 3261 SIP: Session Initiation Protocol June 2002

invitation. After some time, those UASs can accept the invitation

(meaning the session is to be established) by sending a 2xx response.

If the invitation is not accepted, a 3xx, 4xx, 5xx or 6xx response is

sent, depending on the reason for the rejection. Before sending a

final response, the UAS can also send provisional responses (1xx) to

advise the UAC of progress in contacting the called user.

After possibly receiving one or more provisional responses, the UAC

will get one or more 2xx responses or one non-2xx final response.

Because of the protracted amount of time it can take to receive final

responses to INVITE, the reliability mechanisms for INVITE

transactions differ from those of other requests (like OPTIONS).

Once it receives a final response, the UAC needs to send an ACK for

every final response it receives. The procedure for sending this ACK

depends on the type of response. For final responses between 300 and

699, the ACK processing is done in the transaction layer and follows

one set of rules (See Section 17). For 2xx responses, the ACK is

generated by the UAC core.

A 2xx response to an INVITE establishes a session, and it also

creates a dialog between the UA that issued the INVITE and the UA

that generated the 2xx response. Therefore, when multiple 2xx

responses are received from different remote UAs (because the INVITE

forked), each 2xx establishes a different dialog. All these dialogs

are part of the same call.

This section provides details on the establishment of a session using

INVITE. A UA that supports INVITE MUST also support ACK, CANCEL and

BYE.

13.2 UAC Processing

13.2.1 Creating the Initial INVITE

Since the initial INVITE represents a request outside of a dialog,

its construction follows the procedures of Section 8.1.1. Additional

processing is required for the specific case of INVITE.

An Allow header field (Section 20.5) SHOULD be present in the INVITE.

It indicates what methods can be invoked within a dialog, on the UA

sending the INVITE, for the duration of the dialog. For example, a

UA capable of receiving INFO requests within a dialog [34] SHOULD

include an Allow header field listing the INFO method.

A Supported header field (Section 20.37) SHOULD be present in the

INVITE. It enumerates all the extensions understood by the UAC.

Rosenberg, et. al. Standards Track [Page 78]

RFC 3261 SIP: Session Initiation Protocol June 2002

An Accept (Section 20.1) header field MAY be present in the INVITE.

It indicates which Content-Types are acceptable to the UA, in both

the response received by it, and in any subsequent requests sent to

it within dialogs established by the INVITE. The Accept header field

is especially useful for indicating support of various session

description formats.

The UAC MAY add an Expires header field (Section 20.19) to limit the

validity of the invitation. If the time indicated in the Expires

header field is reached and no final answer for the INVITE has been

received, the UAC core SHOULD generate a CANCEL request for the

INVITE, as per Section 9.

A UAC MAY also find it useful to add, among others, Subject (Section

20.36), Organization (Section 20.25) and User-Agent (Section 20.41)

header fields. They all contain information related to the INVITE.

The UAC MAY choose to add a message body to the INVITE. Section

8.1.1.10 deals with how to construct the header fields -- Content-

Type among others -- needed to describe the message body.

There are special rules for message bodies that contain a session

description - their corresponding Content-Disposition is "session".

SIP uses an offer/answer model where one UA sends a session

description, called the offer, which contains a proposed description

of the session. The offer indicates the desired communications means

(audio, video, games), parameters of those means (such as codec

types) and addresses for receiving media from the answerer. The

other UA responds with another session description, called the

answer, which indicates which communications means are accepted, the

parameters that apply to those means, and addresses for receiving

media from the offerer. An offer/answer exchange is within the

context of a dialog, so that if a SIP INVITE results in multiple

dialogs, each is a separate offer/answer exchange. The offer/answer

model defines restrictions on when offers and answers can be made

(for example, you cannot make a new offer while one is in progress).

This results in restrictions on where the offers and answers can

appear in SIP messages. In this specification, offers and answers

can only appear in INVITE requests and responses, and ACK. The usage

of offers and answers is further restricted. For the initial INVITE

transaction, the rules are:

o The initial offer MUST be in either an INVITE or, if not there,

in the first reliable non-failure message from the UAS back to

the UAC. In this specification, that is the final 2xx

response.

Rosenberg, et. al. Standards Track [Page 79]

RFC 3261 SIP: Session Initiation Protocol June 2002

o If the initial offer is in an INVITE, the answer MUST be in a

reliable non-failure message from UAS back to UAC which is

correlated to that INVITE. For this specification, that is

only the final 2xx response to that INVITE. That same exact

answer MAY also be placed in any provisional responses sent

prior to the answer. The UAC MUST treat the first session

description it receives as the answer, and MUST ignore any

session descriptions in subsequent responses to the initial

INVITE.

o If the initial offer is in the first reliable non-failure

message from the UAS back to UAC, the answer MUST be in the

acknowledgement for that message (in this specification, ACK

for a 2xx response).

o After having sent or received an answer to the first offer, the

UAC MAY generate subsequent offers in requests based on rules

specified for that method, but only if it has received answers

to any previous offers, and has not sent any offers to which it

hasn't gotten an answer.

o Once the UAS has sent or received an answer to the initial

offer, it MUST NOT generate subsequent offers in any responses

to the initial INVITE. This means that a UAS based on this

specification alone can never generate subsequent offers until

completion of the initial transaction.

Concretely, the above rules specify two exchanges for UAs compliant

to this specification alone - the offer is in the INVITE, and the

answer in the 2xx (and possibly in a 1xx as well, with the same

value), or the offer is in the 2xx, and the answer is in the ACK.

All user agents that support INVITE MUST support these two exchanges.

The Session Description Protocol (SDP) (RFC 2327 [1]) MUST be

supported by all user agents as a means to describe sessions, and its

usage for constructing offers and answers MUST follow the procedures

defined in [13].

The restrictions of the offer-answer model just described only apply

to bodies whose Content-Disposition header field value is "session".

Therefore, it is possible that both the INVITE and the ACK contain a

body message (for example, the INVITE carries a photo (Content-

Disposition: render) and the ACK a session description (Content-

Disposition: session)).

If the Content-Disposition header field is missing, bodies of

Content-Type application/sdp imply the disposition "session", while

other content types imply "render".

Rosenberg, et. al. Standards Track [Page 80]

RFC 3261 SIP: Session Initiation Protocol June 2002

Once the INVITE has been created, the UAC follows the procedures

defined for sending requests outside of a dialog (Section 8). This

results in the construction of a client transaction that will

ultimately send the request and deliver responses to the UAC.

13.2.2 Processing INVITE Responses

Once the INVITE has been passed to the INVITE client transaction, the

UAC waits for responses for the INVITE. If the INVITE client

transaction returns a timeout rather than a response the TU acts as

if a 408 (Request Timeout) response had been received, as described

in Section 8.1.3.

13.2.2.1 1xx Responses

Zero, one or multiple provisional responses may arrive before one or

more final responses are received. Provisional responses for an

INVITE request can create "early dialogs". If a provisional response

has a tag in the To field, and if the dialog ID of the response does

not match an existing dialog, one is constructed using the procedures

defined in Section 12.1.2.

The early dialog will only be needed if the UAC needs to send a

request to its peer within the dialog before the initial INVITE

transaction completes. Header fields present in a provisional

response are applicable as long as the dialog is in the early state

(for example, an Allow header field in a provisional response

contains the methods that can be used in the dialog while this is in

the early state).

13.2.2.2 3xx Responses

A 3xx response may contain one or more Contact header field values

providing new addresses where the callee might be reachable.

Depending on the status code of the 3xx response (see Section 21.3),

the UAC MAY choose to try those new addresses.

13.2.2.3 4xx, 5xx and 6xx Responses

A single non-2xx final response may be received for the INVITE. 4xx,

5xx and 6xx responses may contain a Contact header field value

indicating the location where additional information about the error

can be found. Subsequent final responses (which would only arrive

under error conditions) MUST be ignored.

All early dialogs are considered terminated upon reception of the

non-2xx final response.

Rosenberg, et. al. Standards Track [Page 81]

RFC 3261 SIP: Session Initiation Protocol June 2002

After having received the non-2xx final response the UAC core

considers the INVITE transaction completed. The INVITE client

transaction handles the generation of ACKs for the response (see

Section 17).

13.2.2.4 2xx Responses

Multiple 2xx responses may arrive at the UAC for a single INVITE

request due to a forking proxy. Each response is distinguished by

the tag parameter in the To header field, and each represents a

distinct dialog, with a distinct dialog identifier.

If the dialog identifier in the 2xx response matches the dialog

identifier of an existing dialog, the dialog MUST be transitioned to

the "confirmed" state, and the route set for the dialog MUST be

recomputed based on the 2xx response using the procedures of Section

12.2.1.2. Otherwise, a new dialog in the "confirmed" state MUST be

constructed using the procedures of Section 12.1.2.

Note that the only piece of state that is recomputed is the route

set. Other pieces of state such as the highest sequence numbers

(remote and local) sent within the dialog are not recomputed. The

route set only is recomputed for backwards compatibility. RFC

2543 did not mandate mirroring of the Record-Route header field in

a 1xx, only 2xx. However, we cannot update the entire state of

the dialog, since mid-dialog requests may have been sent within

the early dialog, modifying the sequence numbers, for example.

The UAC core MUST generate an ACK request for each 2xx received from

the transaction layer. The header fields of the ACK are constructed

in the same way as for any request sent within a dialog (see Section

12) with the exception of the CSeq and the header fields related to

authentication. The sequence number of the CSeq header field MUST be

the same as the INVITE being acknowledged, but the CSeq method MUST

be ACK. The ACK MUST contain the same credentials as the INVITE. If

the 2xx contains an offer (based on the rules above), the ACK MUST

carry an answer in its body. If the offer in the 2xx response is not

acceptable, the UAC core MUST generate a valid answer in the ACK and

then send a BYE immediately.

Once the ACK has been constructed, the procedures of [4] are used to

determine the destination address, port and transport. However, the

request is passed to the transport layer directly for transmission,

rather than a client transaction. This is because the UAC core

handles retransmissions of the ACK, not the transaction layer. The

ACK MUST be passed to the client transport every time a

retransmission of the 2xx final response that triggered the ACK

arrives.

Rosenberg, et. al. Standards Track [Page 82]

RFC 3261 SIP: Session Initiation Protocol June 2002

The UAC core considers the INVITE transaction completed 64\*T1 seconds

after the reception of the first 2xx response. At this point all the

early dialogs that have not transitioned to established dialogs are

terminated. Once the INVITE transaction is considered completed by

the UAC core, no more new 2xx responses are expected to arrive.

If, after acknowledging any 2xx response to an INVITE, the UAC does

not want to continue with that dialog, then the UAC MUST terminate

the dialog by sending a BYE request as described in Section 15.

13.3 UAS Processing

13.3.1 Processing of the INVITE

The UAS core will receive INVITE requests from the transaction layer.

It first performs the request processing procedures of Section 8.2,

which are applied for both requests inside and outside of a dialog.

Assuming these processing states are completed without generating a

response, the UAS core performs the additional processing steps:

1. If the request is an INVITE that contains an Expires header

field, the UAS core sets a timer for the number of seconds

indicated in the header field value. When the timer fires, the

invitation is considered to be expired. If the invitation

expires before the UAS has generated a final response, a 487

(Request Terminated) response SHOULD be generated.

2. If the request is a mid-dialog request, the method-independent

processing described in Section 12.2.2 is first applied. It

might also modify the session; Section 14 provides details.

3. If the request has a tag in the To header field but the dialog

identifier does not match any of the existing dialogs, the UAS

may have crashed and restarted, or may have received a request

for a different (possibly failed) UAS. Section 12.2.2 provides

guidelines to achieve a robust behavior under such a situation.

Processing from here forward assumes that the INVITE is outside of a

dialog, and is thus for the purposes of establishing a new session.

The INVITE may contain a session description, in which case the UAS

is being presented with an offer for that session. It is possible

that the user is already a participant in that session, even though

the INVITE is outside of a dialog. This can happen when a user is

invited to the same multicast conference by multiple other

participants. If desired, the UAS MAY use identifiers within the

session description to detect this duplication. For example, SDP

Rosenberg, et. al. Standards Track [Page 83]

RFC 3261 SIP: Session Initiation Protocol June 2002

contains a session id and version number in the origin (o) field. If

the user is already a member of the session, and the session

parameters contained in the session description have not changed, the

UAS MAY silently accept the INVITE (that is, send a 2xx response

without prompting the user).

If the INVITE does not contain a session description, the UAS is

being asked to participate in a session, and the UAC has asked that

the UAS provide the offer of the session. It MUST provide the offer

in its first non-failure reliable message back to the UAC. In this

specification, that is a 2xx response to the INVITE.

The UAS can indicate progress, accept, redirect, or reject the

invitation. In all of these cases, it formulates a response using

the procedures described in Section 8.2.6.

13.3.1.1 Progress

If the UAS is not able to answer the invitation immediately, it can

choose to indicate some kind of progress to the UAC (for example, an

indication that a phone is ringing). This is accomplished with a

provisional response between 101 and 199. These provisional

responses establish early dialogs and therefore follow the procedures

of Section 12.1.1 in addition to those of Section 8.2.6. A UAS MAY

send as many provisional responses as it likes. Each of these MUST

indicate the same dialog ID. However, these will not be delivered

reliably.

If the UAS desires an extended period of time to answer the INVITE,

it will need to ask for an "extension" in order to prevent proxies

from canceling the transaction. A proxy has the option of canceling

a transaction when there is a gap of 3 minutes between responses in a

transaction. To prevent cancellation, the UAS MUST send a non-100

provisional response at every minute, to handle the possibility of

lost provisional responses.

An INVITE transaction can go on for extended durations when the

user is placed on hold, or when interworking with PSTN systems

which allow communications to take place without answering the

call. The latter is common in Interactive Voice Response (IVR)

systems.

13.3.1.2 The INVITE is Redirected

If the UAS decides to redirect the call, a 3xx response is sent. A

300 (Multiple Choices), 301 (Moved Permanently) or 302 (Moved

Temporarily) response SHOULD contain a Contact header field

Rosenberg, et. al. Standards Track [Page 84]

RFC 3261 SIP: Session Initiation Protocol June 2002

containing one or more URIs of new addresses to be tried. The

response is passed to the INVITE server transaction, which will deal

with its retransmissions.

13.3.1.3 The INVITE is Rejected

A common scenario occurs when the callee is currently not willing or

able to take additional calls at this end system. A 486 (Busy Here)

SHOULD be returned in such a scenario. If the UAS knows that no

other end system will be able to accept this call, a 600 (Busy

Everywhere) response SHOULD be sent instead. However, it is unlikely

that a UAS will be able to know this in general, and thus this

response will not usually be used. The response is passed to the

INVITE server transaction, which will deal with its retransmissions.

A UAS rejecting an offer contained in an INVITE SHOULD return a 488

(Not Acceptable Here) response. Such a response SHOULD include a

Warning header field value explaining why the offer was rejected.

13.3.1.4 The INVITE is Accepted

The UAS core generates a 2xx response. This response establishes a

dialog, and therefore follows the procedures of Section 12.1.1 in

addition to those of Section 8.2.6.

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

the Supported header field, and MAY contain the Accept header field.

Including these header fields allows the UAC to determine the

features and extensions supported by the UAS for the duration of the

call, without probing.

If the INVITE request contained an offer, and the UAS had not yet

sent an answer, the 2xx MUST contain an answer. If the INVITE did

not contain an offer, the 2xx MUST contain an offer if the UAS had

not yet sent an offer.

Once the response has been constructed, it is passed to the INVITE

server transaction. Note, however, that the INVITE server

transaction will be destroyed as soon as it receives this final

response and passes it to the transport. Therefore, it is necessary

to periodically pass the response directly to the transport until the

ACK arrives. The 2xx response is passed to the transport with an

interval that starts at T1 seconds and doubles for each

retransmission until it reaches T2 seconds (T1 and T2 are defined in

Section 17). Response retransmissions cease when an ACK request for

the response is received. This is independent of whatever transport

protocols are used to send the response.

Rosenberg, et. al. Standards Track [Page 85]

RFC 3261 SIP: Session Initiation Protocol June 2002

Since 2xx is retransmitted end-to-end, there may be hops between

UAS and UAC that are UDP. To ensure reliable delivery across

these hops, the response is retransmitted periodically even if the

transport at the UAS is reliable.

If the server retransmits the 2xx response for 64\*T1 seconds without

receiving an ACK, the dialog is confirmed, but the session SHOULD be

terminated. This is accomplished with a BYE, as described in Section

15.

14 Modifying an Existing Session

A successful INVITE request (see Section 13) establishes both a

dialog between two user agents and a session using the offer-answer

model. Section 12 explains how to modify an existing dialog using a

target refresh request (for example, changing the remote target URI

of the dialog). This section describes how to modify the actual

session. This modification can involve changing addresses or ports,

adding a media stream, deleting a media stream, and so on. This is

accomplished by sending a new INVITE request within the same dialog

that established the session. An INVITE request sent within an

existing dialog is known as a re-INVITE.

Note that a single re-INVITE can modify the dialog and the

parameters of the session at the same time.

Either the caller or callee can modify an existing session.

The behavior of a UA on detection of media failure is a matter of

local policy. However, automated generation of re-INVITE or BYE is

NOT RECOMMENDED to avoid flooding the network with traffic when there

is congestion. In any case, if these messages are sent

automatically, they SHOULD be sent after some randomized interval.

Note that the paragraph above refers to automatically generated

BYEs and re-INVITEs. If the user hangs up upon media failure, the

UA would send a BYE request as usual.

14.1 UAC Behavior

The same offer-answer model that applies to session descriptions in

INVITEs (Section 13.2.1) applies to re-INVITEs. As a result, a UAC

that wants to add a media stream, for example, will create a new

offer that contains this media stream, and send that in an INVITE

request to its peer. It is important to note that the full

description of the session, not just the change, is sent. This

supports stateless session processing in various elements, and

supports failover and recovery capabilities. Of course, a UAC MAY

Rosenberg, et. al. Standards Track [Page 86]

RFC 3261 SIP: Session Initiation Protocol June 2002

send a re-INVITE with no session description, in which case the first

reliable non-failure response to the re-INVITE will contain the offer

(in this specification, that is a 2xx response).

If the session description format has the capability for version

numbers, the offerer SHOULD indicate that the version of the session

description has changed.

The To, From, Call-ID, CSeq, and Request-URI of a re-INVITE are set

following the same rules as for regular requests within an existing

dialog, described in Section 12.

A UAC MAY choose not to add an Alert-Info header field or a body with

Content-Disposition "alert" to re-INVITEs because UASs do not

typically alert the user upon reception of a re-INVITE.

Unlike an INVITE, which can fork, a re-INVITE will never fork, and

therefore, only ever generate a single final response. The reason a

re-INVITE will never fork is that the Request-URI identifies the

target as the UA instance it established the dialog with, rather than

identifying an address-of-record for the user.

Note that a UAC MUST NOT initiate a new INVITE transaction within a

dialog while another INVITE transaction is in progress in either

direction.

1. If there is an ongoing INVITE client transaction, the TU MUST

wait until the transaction reaches the completed or terminated

state before initiating the new INVITE.

2. If there is an ongoing INVITE server transaction, the TU MUST

wait until the transaction reaches the confirmed or terminated

state before initiating the new INVITE.

However, a UA MAY initiate a regular transaction while an INVITE

transaction is in progress. A UA MAY also initiate an INVITE

transaction while a regular transaction is in progress.

If a UA receives a non-2xx final response to a re-INVITE, the session

parameters MUST remain unchanged, as if no re-INVITE had been issued.

Note that, as stated in Section 12.2.1.2, if the non-2xx final

response is a 481 (Call/Transaction Does Not Exist), or a 408

(Request Timeout), or no response at all is received for the re-

INVITE (that is, a timeout is returned by the INVITE client

transaction), the UAC will terminate the dialog.

Rosenberg, et. al. Standards Track [Page 87]

RFC 3261 SIP: Session Initiation Protocol June 2002

If a UAC receives a 491 response to a re-INVITE, it SHOULD start a

timer with a value T chosen as follows:

1. If the UAC is the owner of the Call-ID of the dialog ID

(meaning it generated the value), T has a randomly chosen value

between 2.1 and 4 seconds in units of 10 ms.

2. If the UAC is not the owner of the Call-ID of the dialog ID, T

has a randomly chosen value of between 0 and 2 seconds in units

of 10 ms.

When the timer fires, the UAC SHOULD attempt the re-INVITE once more,

if it still desires for that session modification to take place. For

example, if the call was already hung up with a BYE, the re-INVITE

would not take place.

The rules for transmitting a re-INVITE and for generating an ACK for

a 2xx response to re-INVITE are the same as for the initial INVITE

(Section 13.2.1).

14.2 UAS Behavior

Section 13.3.1 describes the procedure for distinguishing incoming

re-INVITEs from incoming initial INVITEs and handling a re-INVITE for

an existing dialog.

A UAS that receives a second INVITE before it sends the final

response to a first INVITE with a lower CSeq sequence number on the

same dialog MUST return a 500 (Server Internal Error) response to the

second INVITE and MUST include a Retry-After header field with a

randomly chosen value of between 0 and 10 seconds.

A UAS that receives an INVITE on a dialog while an INVITE it had sent

on that dialog is in progress MUST return a 491 (Request Pending)

response to the received INVITE.

If a UA receives a re-INVITE for an existing dialog, it MUST check

any version identifiers in the session description or, if there are

no version identifiers, the content of the session description to see

if it has changed. If the session description has changed, the UAS

MUST adjust the session parameters accordingly, possibly after asking

the user for confirmation.

Versioning of the session description can be used to accommodate

the capabilities of new arrivals to a conference, add or delete

media, or change from a unicast to a multicast conference.

Rosenberg, et. al. Standards Track [Page 88]

RFC 3261 SIP: Session Initiation Protocol June 2002

If the new session description is not acceptable, the UAS can reject

it by returning a 488 (Not Acceptable Here) response for the re-

INVITE. This response SHOULD include a Warning header field.

If a UAS generates a 2xx response and never receives an ACK, it

SHOULD generate a BYE to terminate the dialog.

A UAS MAY choose not to generate 180 (Ringing) responses for a re-

INVITE because UACs do not typically render this information to the

user. For the same reason, UASs MAY choose not to use an Alert-Info

header field or a body with Content-Disposition "alert" in responses

to a re-INVITE.

A UAS providing an offer in a 2xx (because the INVITE did not contain

an offer) SHOULD construct the offer as if the UAS were making a

brand new call, subject to the constraints of sending an offer that

updates an existing session, as described in [13] in the case of SDP.

Specifically, this means that it SHOULD include as many media formats

and media types that the UA is willing to support. The UAS MUST

ensure that the session description overlaps with its previous

session description in media formats, transports, or other parameters

that require support from the peer. This is to avoid the need for

the peer to reject the session description. If, however, it is

unacceptable to the UAC, the UAC SHOULD generate an answer with a

valid session description, and then send a BYE to terminate the

session.

15 Terminating a Session

This section describes the procedures for terminating a session

established by SIP. The state of the session and the state of the

dialog are very closely related. When a session is initiated with an

INVITE, each 1xx or 2xx response from a distinct UAS creates a

dialog, and if that response completes the offer/answer exchange, it

also creates a session. As a result, each session is "associated"

with a single dialog - the one which resulted in its creation. If an

initial INVITE generates a non-2xx final response, that terminates

all sessions (if any) and all dialogs (if any) that were created

through responses to the request. By virtue of completing the

transaction, a non-2xx final response also prevents further sessions

from being created as a result of the INVITE. The BYE request is

used to terminate a specific session or attempted session. In this

case, the specific session is the one with the peer UA on the other

side of the dialog. When a BYE is received on a dialog, any session

associated with that dialog SHOULD terminate. A UA MUST NOT send a

BYE outside of a dialog. The caller's UA MAY send a BYE for either

confirmed or early dialogs, and the callee's UA MAY send a BYE on

confirmed dialogs, but MUST NOT send a BYE on early dialogs.

Rosenberg, et. al. Standards Track [Page 89]

RFC 3261 SIP: Session Initiation Protocol June 2002

However, the callee's UA MUST NOT send a BYE on a confirmed dialog

until it has received an ACK for its 2xx response or until the server

transaction times out. If no SIP extensions have defined other

application layer states associated with the dialog, the BYE also

terminates the dialog.

The impact of a non-2xx final response to INVITE on dialogs and

sessions makes the use of CANCEL attractive. The CANCEL attempts to

force a non-2xx response to the INVITE (in particular, a 487).

Therefore, if a UAC wishes to give up on its call attempt entirely,

it can send a CANCEL. If the INVITE results in 2xx final response(s)

to the INVITE, this means that a UAS accepted the invitation while

the CANCEL was in progress. The UAC MAY continue with the sessions

established by any 2xx responses, or MAY terminate them with BYE.

The notion of "hanging up" is not well defined within SIP. It is

specific to a particular, albeit common, user interface.

Typically, when the user hangs up, it indicates a desire to

terminate the attempt to establish a session, and to terminate any

sessions already created. For the caller's UA, this would imply a

CANCEL request if the initial INVITE has not generated a final

response, and a BYE to all confirmed dialogs after a final

response. For the callee's UA, it would typically imply a BYE;

presumably, when the user picked up the phone, a 2xx was

generated, and so hanging up would result in a BYE after the ACK

is received. This does not mean a user cannot hang up before

receipt of the ACK, it just means that the software in his phone

needs to maintain state for a short while in order to clean up

properly. If the particular UI allows for the user to reject a

call before its answered, a 403 (Forbidden) is a good way to

express that. As per the rules above, a BYE can't be sent.

15.1 Terminating a Session with a BYE Request

15.1.1 UAC Behavior

A BYE request is constructed as would any other request within a

dialog, as described in Section 12.

Once the BYE is constructed, the UAC core creates a new non-INVITE

client transaction, and passes it the BYE request. The UAC MUST

consider the session terminated (and therefore stop sending or

listening for media) as soon as the BYE request is passed to the

client transaction. If the response for the BYE is a 481

(Call/Transaction Does Not Exist) or a 408 (Request Timeout) or no

Rosenberg, et. al. Standards Track [Page 90]

RFC 3261 SIP: Session Initiation Protocol June 2002

response at all is received for the BYE (that is, a timeout is

returned by the client transaction), the UAC MUST consider the

session and the dialog terminated.

15.1.2 UAS Behavior

A UAS first processes the BYE request according to the general UAS

processing described in Section 8.2. A UAS core receiving a BYE

request checks if it matches an existing dialog. If the BYE does not

match an existing dialog, the UAS core SHOULD generate a 481

(Call/Transaction Does Not Exist) response and pass that to the

server transaction.

This rule means that a BYE sent without tags by a UAC will be

rejected. This is a change from RFC 2543, which allowed BYE

without tags.

A UAS core receiving a BYE request for an existing dialog MUST follow

the procedures of Section 12.2.2 to process the request. Once done,

the UAS SHOULD terminate the session (and therefore stop sending and

listening for media). The only case where it can elect not to are

multicast sessions, where participation is possible even if the other

participant in the dialog has terminated its involvement in the

session. Whether or not it ends its participation on the session,

the UAS core MUST generate a 2xx response to the BYE, and MUST pass

that to the server transaction for transmission.

The UAS MUST still respond to any pending requests received for that

dialog. It is RECOMMENDED that a 487 (Request Terminated) response

be generated to those pending requests.

16 Proxy Behavior

16.1 Overview

SIP proxies are elements that route SIP requests to user agent

servers and SIP responses to user agent clients. A request may

traverse several proxies on its way to a UAS. Each will make routing

decisions, modifying the request before forwarding it to the next

element. Responses will route through the same set of proxies

traversed by the request in the reverse order.

Being a proxy is a logical role for a SIP element. When a request

arrives, an element that can play the role of a proxy first decides

if it needs to respond to the request on its own. For instance, the

request may be malformed or the element may need credentials from the

client before acting as a proxy. The element MAY respond with any

Rosenberg, et. al. Standards Track [Page 91]

RFC 3261 SIP: Session Initiation Protocol June 2002

appropriate error code. When responding directly to a request, the

element is playing the role of a UAS and MUST behave as described in

Section 8.2.

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

new request. When stateless, a proxy acts as a simple forwarding

element. It forwards each request downstream to a single element

determined by making a targeting and routing decision based on the

request. It simply forwards every response it receives upstream. A

stateless proxy discards information about a message once the message

has been forwarded. A stateful proxy remembers information

(specifically, transaction state) about each incoming request and any

requests it sends as a result of processing the incoming request. It

uses this information to affect the processing of future messages

associated with that request. A stateful proxy MAY choose to "fork"

a request, routing it to multiple destinations. Any request that is

forwarded to more than one location MUST be handled statefully.

In some circumstances, a proxy MAY forward requests using stateful

transports (such as TCP) without being transaction-stateful. For

instance, a proxy MAY forward a request from one TCP connection to

another transaction statelessly as long as it places enough

information in the message to be able to forward the response down

the same connection the request arrived on. Requests forwarded

between different types of transports where the proxy's TU must take

an active role in ensuring reliable delivery on one of the transports

MUST be forwarded transaction statefully.

A stateful proxy MAY transition to stateless operation at any time

during the processing of a request, so long as it did not do anything

that would otherwise prevent it from being stateless initially

(forking, for example, or generation of a 100 response). When

performing such a transition, all state is simply discarded. The

proxy SHOULD NOT initiate a CANCEL request.

Much of the processing involved when acting statelessly or statefully

for a request is identical. The next several subsections are written

from the point of view of a stateful proxy. The last section calls

out those places where a stateless proxy behaves differently.

16.2 Stateful Proxy

When stateful, a proxy is purely a SIP transaction processing engine.

Its behavior is modeled here in terms of the server and client

transactions defined in Section 17. A stateful proxy has a server

transaction associated with one or more client transactions by a

higher layer proxy processing component (see figure 3), known as a

proxy core. An incoming request is processed by a server

Rosenberg, et. al. Standards Track [Page 92]

RFC 3261 SIP: Session Initiation Protocol June 2002

transaction. Requests from the server transaction are passed to a

proxy core. The proxy core determines where to route the request,

choosing one or more next-hop locations. An outgoing request for

each next-hop location is processed by its own associated client

transaction. The proxy core collects the responses from the client

transactions and uses them to send responses to the server

transaction.

A stateful proxy creates a new server transaction for each new

request received. Any retransmissions of the request will then be

handled by that server transaction per Section 17. The proxy core

MUST behave as a UAS with respect to sending an immediate provisional

on that server transaction (such as 100 Trying) as described in

Section 8.2.6. Thus, a stateful proxy SHOULD NOT generate 100

(Trying) responses to non-INVITE requests.

This is a model of proxy behavior, not of software. An

implementation is free to take any approach that replicates the

external behavior this model defines.

For all new requests, including any with unknown methods, an element

intending to proxy the request MUST:

1. Validate the request (Section 16.3)

2. Preprocess routing information (Section 16.4)

3. Determine target(s) for the request (Section 16.5)

+--------------------+

| | +---+

| | | C |

| | | T |

| | +---+

+---+ | Proxy | +---+ CT = Client Transaction

| S | | "Higher" Layer | | C |

| T | | | | T | ST = Server Transaction

+---+ | | +---+

| | +---+

| | | C |

| | | T |

| | +---+

+--------------------+

Figure 3: Stateful Proxy Model

Rosenberg, et. al. Standards Track [Page 93]

RFC 3261 SIP: Session Initiation Protocol June 2002

4. Forward the request to each target (Section 16.6)

5. Process all responses (Section 16.7)

16.3 Request Validation

Before an element can proxy a request, it MUST verify the message's

validity. A valid message must pass the following checks:

1. Reasonable Syntax

2. URI scheme

3. Max-Forwards

4. (Optional) Loop Detection

5. Proxy-Require

6. Proxy-Authorization

If any of these checks fail, the element MUST behave as a user agent

server (see Section 8.2) and respond with an error code.

Notice that a proxy is not required to detect merged requests and

MUST NOT treat merged requests as an error condition. The endpoints

receiving the requests will resolve the merge as described in Section

8.2.2.2.

1. Reasonable syntax check

The request MUST be well-formed enough to be handled with a server

transaction. Any components involved in the remainder of these

Request Validation steps or the Request Forwarding section MUST be

well-formed. Any other components, well-formed or not, SHOULD be

ignored and remain unchanged when the message is forwarded. For

instance, an element would not reject a request because of a

malformed Date header field. Likewise, a proxy would not remove a

malformed Date header field before forwarding a request.

This protocol is designed to be extended. Future extensions may

define new methods and header fields at any time. An element MUST

NOT refuse to proxy a request because it contains a method or

header field it does not know about.

Rosenberg, et. al. Standards Track [Page 94]

RFC 3261 SIP: Session Initiation Protocol June 2002

2. URI scheme check

If the Request-URI has a URI whose scheme is not understood by the

proxy, the proxy SHOULD reject the request with a 416 (Unsupported

URI Scheme) response.

3. Max-Forwards check

The Max-Forwards header field (Section 20.22) is used to limit the

number of elements a SIP request can traverse.

If the request does not contain a Max-Forwards header field, this

check is passed.

If the request contains a Max-Forwards header field with a field

value greater than zero, the check is passed.

If the request contains a Max-Forwards header field with a field

value of zero (0), the element MUST NOT forward the request. If

the request was for OPTIONS, the element MAY act as the final

recipient and respond per Section 11. Otherwise, the element MUST

return a 483 (Too many hops) response.

4. Optional Loop Detection check

An element MAY check for forwarding loops before forwarding a

request. If the request contains a Via header field with a sent-

by value that equals a value placed into previous requests by the

proxy, the request has been forwarded by this element before. The

request has either looped or is legitimately spiraling through the

element. To determine if the request has looped, the element MAY

perform the branch parameter calculation described in Step 8 of

Section 16.6 on this message and compare it to the parameter

received in that Via header field. If the parameters match, the

request has looped. If they differ, the request is spiraling, and

processing continues. If a loop is detected, the element MAY

return a 482 (Loop Detected) response.

5. Proxy-Require check

Future extensions to this protocol may introduce features that

require special handling by proxies. Endpoints will include a

Proxy-Require header field in requests that use these features,

telling the proxy not to process the request unless the feature is

understood.

Rosenberg, et. al. Standards Track [Page 95]

RFC 3261 SIP: Session Initiation Protocol June 2002

If the request contains a Proxy-Require header field (Section

20.29) with one or more option-tags this element does not

understand, the element MUST return a 420 (Bad Extension)

response. The response MUST include an Unsupported (Section

20.40) header field listing those option-tags the element did not

understand.

6. Proxy-Authorization check

If an element requires credentials before forwarding a request,

the request MUST be inspected as described in Section 22.3. That

section also defines what the element must do if the inspection

fails.

16.4 Route Information Preprocessing

The proxy MUST inspect the Request-URI of the request. If the

Request-URI of the request contains a value this proxy previously

placed into a Record-Route header field (see Section 16.6 item 4),

the proxy MUST replace the Request-URI in the request with the last

value from the Route header field, and remove that value from the

Route header field. The proxy MUST then proceed as if it received

this modified request.

This will only happen when the element sending the request to the

proxy (which may have been an endpoint) is a strict router. This

rewrite on receive is necessary to enable backwards compatibility

with those elements. It also allows elements following this

specification to preserve the Request-URI through strict-routing

proxies (see Section 12.2.1.1).

This requirement does not obligate a proxy to keep state in order

to detect URIs it previously placed in Record-Route header fields.

Instead, a proxy need only place enough information in those URIs

to recognize them as values it provided when they later appear.

If the Request-URI contains a maddr parameter, the proxy MUST check

to see if its value is in the set of addresses or domains the proxy

is configured to be responsible for. If the Request-URI has a maddr

parameter with a value the proxy is responsible for, and the request

was received using the port and transport indicated (explicitly or by

default) in the Request-URI, the proxy MUST strip the maddr and any

non-default port or transport parameter and continue processing as if

those values had not been present in the request.

Rosenberg, et. al. Standards Track [Page 96]

RFC 3261 SIP: Session Initiation Protocol June 2002

A request may arrive with a maddr matching the proxy, but on a

port or transport different from that indicated in the URI. Such

a request needs to be forwarded to the proxy using the indicated

port and transport.

If the first value in the Route header field indicates this proxy,

the proxy MUST remove that value from the request.

16.5 Determining Request Targets

Next, the proxy calculates the target(s) of the request. The set of

targets will either be predetermined by the contents of the request

or will be obtained from an abstract location service. Each target

in the set is represented as a URI.

If the Request-URI of the request contains an maddr parameter, the

Request-URI MUST be placed into the target set as the only target

URI, and the proxy MUST proceed to Section 16.6.

If the domain of the Request-URI indicates a domain this element is

not responsible for, the Request-URI MUST be placed into the target

set as the only target, and the element MUST proceed to the task of

Request Forwarding (Section 16.6).

There are many circumstances in which a proxy might receive a

request for a domain it is not responsible for. A firewall proxy

handling outgoing calls (the way HTTP proxies handle outgoing

requests) is an example of where this is likely to occur.

If the target set for the request has not been predetermined as

described above, this implies that the element is responsible for the

domain in the Request-URI, and the element MAY use whatever mechanism

it desires to determine where to send the request. Any of these

mechanisms can be modeled as accessing an abstract Location Service.

This may consist of obtaining information from a location service

created by a SIP Registrar, reading a database, consulting a presence

server, utilizing other protocols, or simply performing an

algorithmic substitution on the Request-URI. When accessing the

location service constructed by a registrar, the Request-URI MUST

first be canonicalized as described in Section 10.3 before being used

as an index. The output of these mechanisms is used to construct the

target set.

If the Request-URI does not provide sufficient information for the

proxy to determine the target set, it SHOULD return a 485 (Ambiguous)

response. This response SHOULD contain a Contact header field

containing URIs of new addresses to be tried. For example, an INVITE

Rosenberg, et. al. Standards Track [Page 97]

RFC 3261 SIP: Session Initiation Protocol June 2002

to sip:John.Smith@company.com may be ambiguous at a proxy whose

location service has multiple John Smiths listed. See Section

21.4.23 for details.

Any information in or about the request or the current environment of

the element MAY be used in the construction of the target set. For

instance, different sets may be constructed depending on contents or

the presence of header fields and bodies, the time of day of the

request's arrival, the interface on which the request arrived,

failure of previous requests, or even the element's current level of

utilization.

As potential targets are located through these services, their URIs

are added to the target set. Targets can only be placed in the

target set once. If a target URI is already present in the set

(based on the definition of equality for the URI type), it MUST NOT

be added again.

A proxy MUST NOT add additional targets to the target set if the

Request-URI of the original request does not indicate a resource this

proxy is responsible for.

A proxy can only change the Request-URI of a request during

forwarding if it is responsible for that URI. If the proxy is not

responsible for that URI, it will not recurse on 3xx or 416

responses as described below.

If the Request-URI of the original request indicates a resource this

proxy is responsible for, the proxy MAY continue to add targets to

the set after beginning Request Forwarding. It MAY use any

information obtained during that processing to determine new targets.

For instance, a proxy may choose to incorporate contacts obtained in

a redirect response (3xx) into the target set. If a proxy uses a

dynamic source of information while building the target set (for

instance, if it consults a SIP Registrar), it SHOULD monitor that

source for the duration of processing the request. New locations

SHOULD be added to the target set as they become available. As

above, any given URI MUST NOT be added to the set more than once.

Allowing a URI to be added to the set only once reduces

unnecessary network traffic, and in the case of incorporating

contacts from redirect requests prevents infinite recursion.

For example, a trivial location service is a "no-op", where the

target URI is equal to the incoming request URI. The request is sent

to a specific next hop proxy for further processing. During request

Rosenberg, et. al. Standards Track [Page 98]

RFC 3261 SIP: Session Initiation Protocol June 2002

forwarding of Section 16.6, Item 6, the identity of that next hop,

expressed as a SIP or SIPS URI, is inserted as the top-most Route

header field value into the request.

If the Request-URI indicates a resource at this proxy that does not

exist, the proxy MUST return a 404 (Not Found) response.

If the target set remains empty after applying all of the above, the

proxy MUST return an error response, which SHOULD be the 480

(Temporarily Unavailable) response.

16.6 Request Forwarding

As soon as the target set is non-empty, a proxy MAY begin forwarding

the request. A stateful proxy MAY process the set in any order. It

MAY process multiple targets serially, allowing each client

transaction to complete before starting the next. It MAY start

client transactions with every target in parallel. It also MAY

arbitrarily divide the set into groups, processing the groups

serially and processing the targets in each group in parallel.

A common ordering mechanism is to use the qvalue parameter of targets

obtained from Contact header fields (see Section 20.10). Targets are

processed from highest qvalue to lowest. Targets with equal qvalues

may be processed in parallel.

A stateful proxy must have a mechanism to maintain the target set as

responses are received and associate the responses to each forwarded

request with the original request. For the purposes of this model,

this mechanism is a "response context" created by the proxy layer

before forwarding the first request.

For each target, the proxy forwards the request following these

steps:

1. Make a copy of the received request

2. Update the Request-URI

3. Update the Max-Forwards header field

4. Optionally add a Record-route header field value

5. Optionally add additional header fields

6. Postprocess routing information

7. Determine the next-hop address, port, and transport

Rosenberg, et. al. Standards Track [Page 99]

RFC 3261 SIP: Session Initiation Protocol June 2002

8. Add a Via header field value

9. Add a Content-Length header field if necessary

10. Forward the new request

11. Set timer C

Each of these steps is detailed below:

1. Copy request

The proxy starts with a copy of the received request. The copy

MUST initially contain all of the header fields from the

received request. Fields not detailed in the processing

described below MUST NOT be removed. The copy SHOULD maintain

the ordering of the header fields as in the received request.

The proxy MUST NOT reorder field values with a common field

name (See Section 7.3.1). The proxy MUST NOT add to, modify,

or remove the message body.

An actual implementation need not perform a copy; the primary

requirement is that the processing for each next hop begin with

the same request.

2. Request-URI

The Request-URI in the copy's start line MUST be replaced with

the URI for this target. If the URI contains any parameters

not allowed in a Request-URI, they MUST be removed.

This is the essence of a proxy's role. This is the mechanism

through which a proxy routes a request toward its destination.

In some circumstances, the received Request-URI is placed into

the target set without being modified. For that target, the

replacement above is effectively a no-op.

3. Max-Forwards

If the copy contains a Max-Forwards header field, the proxy

MUST decrement its value by one (1).

If the copy does not contain a Max-Forwards header field, the

proxy MUST add one with a field value, which SHOULD be 70.

Some existing UAs will not provide a Max-Forwards header field

in a request.

Rosenberg, et. al. Standards Track [Page 100]

RFC 3261 SIP: Session Initiation Protocol June 2002

4. Record-Route

If this proxy wishes to remain on the path of future requests

in a dialog created by this request (assuming the request

creates a dialog), it MUST insert a Record-Route header field

value into the copy before any existing Record-Route header

field values, even if a Route header field is already present.

Requests establishing a dialog may contain a preloaded Route

header field.

If this request is already part of a dialog, the proxy SHOULD

insert a Record-Route header field value if it wishes to remain

on the path of future requests in the dialog. In normal

endpoint operation as described in Section 12, these Record-

Route header field values will not have any effect on the route

sets used by the endpoints.

The proxy will remain on the path if it chooses to not insert a

Record-Route header field value into requests that are already

part of a dialog. However, it would be removed from the path

when an endpoint that has failed reconstitutes the dialog.

A proxy MAY insert a Record-Route header field value into any

request. If the request does not initiate a dialog, the

endpoints will ignore the value. See Section 12 for details on

how endpoints use the Record-Route header field values to

construct Route header fields.

Each proxy in the path of a request chooses whether to add a

Record-Route header field value independently - the presence of

a Record-Route header field in a request does not obligate this

proxy to add a value.

The URI placed in the Record-Route header field value MUST be a

SIP or SIPS URI. This URI MUST contain an lr parameter (see

Section 19.1.1). This URI MAY be different for each

destination the request is forwarded to. The URI SHOULD NOT

contain the transport parameter unless the proxy has knowledge

(such as in a private network) that the next downstream element

that will be in the path of subsequent requests supports that

transport.

The URI this proxy provides will be used by some other element

to make a routing decision. This proxy, in general, has no way

of knowing the capabilities of that element, so it must

restrict itself to the mandatory elements of a SIP

implementation: SIP URIs and either the TCP or UDP transports.

Rosenberg, et. al. Standards Track [Page 101]

RFC 3261 SIP: Session Initiation Protocol June 2002

The URI placed in the Record-Route header field MUST resolve to

the element inserting it (or a suitable stand-in) when the

server location procedures of [4] are applied to it, so that

subsequent requests reach the same SIP element. If the

Request-URI contains a SIPS URI, or the topmost Route header

field value (after the post processing of bullet 6) contains a

SIPS URI, the URI placed into the Record-Route header field

MUST be a SIPS URI. Furthermore, if the request was not

received over TLS, the proxy MUST insert a Record-Route header

field. In a similar fashion, a proxy that receives a request

over TLS, but generates a request without a SIPS URI in the

Request-URI or topmost Route header field value (after the post

processing of bullet 6), MUST insert a Record-Route header

field that is not a SIPS URI.

A proxy at a security perimeter must remain on the perimeter

throughout the dialog.

If the URI placed in the Record-Route header field needs to be

rewritten when it passes back through in a response, the URI

MUST be distinct enough to locate at that time. (The request

may spiral through this proxy, resulting in more than one

Record-Route header field value being added). Item 8 of

Section 16.7 recommends a mechanism to make the URI

sufficiently distinct.

The proxy MAY include parameters in the Record-Route header

field value. These will be echoed in some responses to the

request such as the 200 (OK) responses to INVITE. Such

parameters may be useful for keeping state in the message

rather than the proxy.

If a proxy needs to be in the path of any type of dialog (such

as one straddling a firewall), it SHOULD add a Record-Route

header field value to every request with a method it does not

understand since that method may have dialog semantics.

The URI a proxy places into a Record-Route header field is only

valid for the lifetime of any dialog created by the transaction

in which it occurs. A dialog-stateful proxy, for example, MAY

refuse to accept future requests with that value in the

Request-URI after the dialog has terminated. Non-dialog-

stateful proxies, of course, have no concept of when the dialog

has terminated, but they MAY encode enough information in the

value to compare it against the dialog identifier of future

requests and MAY reject requests not matching that information.

Endpoints MUST NOT use a URI obtained from a Record-Route

header field outside the dialog in which it was provided. See

Rosenberg, et. al. Standards Track [Page 102]

RFC 3261 SIP: Session Initiation Protocol June 2002

Section 12 for more information on an endpoint's use of

Record-Route header fields.

Record-routing may be required by certain services where the

proxy needs to observe all messages in a dialog. However, it

slows down processing and impairs scalability and thus proxies

should only record-route if required for a particular service.

The Record-Route process is designed to work for any SIP

request that initiates a dialog. INVITE is the only such

request in this specification, but extensions to the protocol

MAY define others.

5. Add Additional Header Fields

The proxy MAY add any other appropriate header fields to the

copy at this point.

6. Postprocess routing information

A proxy MAY have a local policy that mandates that a request

visit a specific set of proxies before being delivered to the

destination. A proxy MUST ensure that all such proxies are

loose routers. Generally, this can only be known with

certainty if the proxies are within the same administrative

domain. This set of proxies is represented by a set of URIs

(each of which contains the lr parameter). This set MUST be

pushed into the Route header field of the copy ahead of any

existing values, if present. If the Route header field is

absent, it MUST be added, containing that list of URIs.

If the proxy has a local policy that mandates that the request

visit one specific proxy, an alternative to pushing a Route

value into the Route header field is to bypass the forwarding

logic of item 10 below, and instead just send the request to

the address, port, and transport for that specific proxy. If

the request has a Route header field, this alternative MUST NOT

be used unless it is known that next hop proxy is a loose

router. Otherwise, this approach MAY be used, but the Route

insertion mechanism above is preferred for its robustness,

flexibility, generality and consistency of operation.

Furthermore, if the Request-URI contains a SIPS URI, TLS MUST

be used to communicate with that proxy.

If the copy contains a Route header field, the proxy MUST

inspect the URI in its first value. If that URI does not

contain an lr parameter, the proxy MUST modify the copy as

follows:

Rosenberg, et. al. Standards Track [Page 103]

RFC 3261 SIP: Session Initiation Protocol June 2002

- The proxy MUST place the Request-URI into the Route header

field as the last value.

- The proxy MUST then place the first Route header field value

into the Request-URI and remove that value from the Route

header field.

Appending the Request-URI to the Route header field is part of

a mechanism used to pass the information in that Request-URI

through strict-routing elements. "Popping" the first Route

header field value into the Request-URI formats the message the

way a strict-routing element expects to receive it (with its

own URI in the Request-URI and the next location to visit in

the first Route header field value).

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

The proxy MAY have a local policy to send the request to a

specific IP address, port, and transport, independent of the

values of the Route and Request-URI. Such a policy MUST NOT be

used if the proxy is not certain that the IP address, port, and

transport correspond to a server that is a loose router.

However, this mechanism for sending the request through a

specific next hop is NOT RECOMMENDED; instead a Route header

field should be used for that purpose as described above.

In the absence of such an overriding mechanism, the proxy

applies the procedures listed in [4] as follows to determine

where to send the request. If the proxy has reformatted the

request to send to a strict-routing element as described in

step 6 above, the proxy MUST apply those procedures to the

Request-URI of the request. Otherwise, the proxy MUST apply

the procedures to the first value in the Route header field, if

present, else the Request-URI. The procedures will produce an

ordered set of (address, port, transport) tuples.

Independently of which URI is being used as input to the

procedures of [4], if the Request-URI specifies a SIPS

resource, the proxy MUST follow the procedures of [4] as if the

input URI were a SIPS URI.

As described in [4], the proxy MUST attempt to deliver the

message to the first tuple in that set, and proceed through the

set in order until the delivery attempt succeeds.

For each tuple attempted, the proxy MUST format the message as

appropriate for the tuple and send the request using a new

client transaction as detailed in steps 8 through 10.

Rosenberg, et. al. Standards Track [Page 104]

RFC 3261 SIP: Session Initiation Protocol June 2002

Since each attempt uses a new client transaction, it represents

a new branch. Thus, the branch parameter provided with the Via

header field inserted in step 8 MUST be different for each

attempt.

If the client transaction reports failure to send the request

or a timeout from its state machine, the proxy continues to the

next address in that ordered set. If the ordered set is

exhausted, the request cannot be forwarded to this element in

the target set. The proxy does not need to place anything in

the response context, but otherwise acts as if this element of

the target set returned a 408 (Request Timeout) final response.

8. Add a Via header field value

The proxy MUST insert a Via header field value into the copy

before the existing Via header field values. The construction

of this value follows the same guidelines of Section 8.1.1.7.

This implies that the proxy will compute its own branch

parameter, which will be globally unique for that branch, and

contain the requisite magic cookie. Note that this implies that

the branch parameter will be different for different instances

of a spiraled or looped request through a proxy.

Proxies choosing to detect loops have an additional constraint

in the value they use for construction of the branch parameter.

A proxy choosing to detect loops SHOULD create a branch

parameter separable into two parts by the implementation. The

first part MUST satisfy the constraints of Section 8.1.1.7 as

described above. The second is used to perform loop detection

and distinguish loops from spirals.

Loop detection is performed by verifying that, when a request

returns to a proxy, those fields having an impact on the

processing of the request have not changed. The value placed

in this part of the branch parameter SHOULD reflect all of

those fields (including any Route, Proxy-Require and Proxy-

Authorization header fields). This is to ensure that if the

request is routed back to the proxy and one of those fields

changes, it is treated as a spiral and not a loop (see Section

16.3). A common way to create this value is to compute a

cryptographic hash of the To tag, From tag, Call-ID header

field, the Request-URI of the request received (before

translation), the topmost Via header, and the sequence number

from the CSeq header field, in addition to any Proxy-Require

and Proxy-Authorization header fields that may be present. The

Rosenberg, et. al. Standards Track [Page 105]

RFC 3261 SIP: Session Initiation Protocol June 2002

algorithm used to compute the hash is implementation-dependent,

but MD5 (RFC 1321 [35]), expressed in hexadecimal, is a

reasonable choice. (Base64 is not permissible for a token.)

If a proxy wishes to detect loops, the "branch" parameter it

supplies MUST depend on all information affecting processing of

a request, including the incoming Request-URI and any header

fields affecting the request's admission or routing. This is

necessary to distinguish looped requests from requests whose

routing parameters have changed before returning to this

server.

The request method MUST NOT be included in the calculation of

the branch parameter. In particular, CANCEL and ACK requests

(for non-2xx responses) MUST have the same branch value as the

corresponding request they cancel or acknowledge. The branch

parameter is used in correlating those requests at the server

handling them (see Sections 17.2.3 and 9.2).

9. Add a Content-Length header field if necessary

If the request will be sent to the next hop using a stream-

based transport and the copy contains no Content-Length header

field, the proxy MUST insert one with the correct value for the

body of the request (see Section 20.14).

10. Forward Request

A stateful proxy MUST create a new client transaction for this

request as described in Section 17.1 and instructs the

transaction to send the request using the address, port and

transport determined in step 7.

11. Set timer C

In order to handle the case where an INVITE request never

generates a final response, the TU uses a timer which is called

timer C. Timer C MUST be set for each client transaction when

an INVITE request is proxied. The timer MUST be larger than 3

minutes. Section 16.7 bullet 2 discusses how this timer is

updated with provisional responses, and Section 16.8 discusses

processing when it fires.

Rosenberg, et. al. Standards Track [Page 106]

RFC 3261 SIP: Session Initiation Protocol June 2002

16.7 Response Processing

When a response is received by an element, it first tries to locate a

client transaction (Section 17.1.3) matching the response. If none

is found, the element MUST process the response (even if it is an

informational response) as a stateless proxy (described below). If a

match is found, the response is handed to the client transaction.

Forwarding responses for which a client transaction (or more

generally any knowledge of having sent an associated request) is

not found improves robustness. In particular, it ensures that

"late" 2xx responses to INVITE requests are forwarded properly.

As client transactions pass responses to the proxy layer, the

following processing MUST take place:

1. Find the appropriate response context

2. Update timer C for provisional responses

3. Remove the topmost Via

4. Add the response to the response context

5. Check to see if this response should be forwarded immediately

6. When necessary, choose the best final response from the

response context

If no final response has been forwarded after every client

transaction associated with the response context has been terminated,

the proxy must choose and forward the "best" response from those it

has seen so far.

The following processing MUST be performed on each response that is

forwarded. It is likely that more than one response to each request

will be forwarded: at least each provisional and one final response.

7. Aggregate authorization header field values if necessary

8. Optionally rewrite Record-Route header field values

9. Forward the response

10. Generate any necessary CANCEL requests

Rosenberg, et. al. Standards Track [Page 107]

RFC 3261 SIP: Session Initiation Protocol June 2002

Each of the above steps are detailed below:

1. Find Context

The proxy locates the "response context" it created before

forwarding the original request using the key described in

Section 16.6. The remaining processing steps take place in

this context.

2. Update timer C for provisional responses

For an INVITE transaction, if the response is a provisional

response with status codes 101 to 199 inclusive (i.e., anything

but 100), the proxy MUST reset timer C for that client

transaction. The timer MAY be reset to a different value, but

this value MUST be greater than 3 minutes.

3. Via

The proxy removes the topmost Via header field value from the

response.

If no Via header field values remain in the response, the

response was meant for this element and MUST NOT be forwarded.

The remainder of the processing described in this section is

not performed on this message, the UAC processing rules

described in Section 8.1.3 are followed instead (transport

layer processing has already occurred).

This will happen, for instance, when the element generates

CANCEL requests as described in Section 10.

4. Add response to context

Final responses received are stored in the response context

until a final response is generated on the server transaction

associated with this context. The response may be a candidate

for the best final response to be returned on that server

transaction. Information from this response may be needed in

forming the best response, even if this response is not chosen.

If the proxy chooses to recurse on any contacts in a 3xx

response by adding them to the target set, it MUST remove them

from the response before adding the response to the response

context. However, a proxy SHOULD NOT recurse to a non-SIPS URI

if the Request-URI of the original request was a SIPS URI. If

Rosenberg, et. al. Standards Track [Page 108]

RFC 3261 SIP: Session Initiation Protocol June 2002

the proxy recurses on all of the contacts in a 3xx response,

the proxy SHOULD NOT add the resulting contactless response to

the response context.

Removing the contact before adding the response to the response

context prevents the next element upstream from retrying a

location this proxy has already attempted.

3xx responses may contain a mixture of SIP, SIPS, and non-SIP

URIs. A proxy may choose to recurse on the SIP and SIPS URIs

and place the remainder into the response context to be

returned, potentially in the final response.

If a proxy receives a 416 (Unsupported URI Scheme) response to

a request whose Request-URI scheme was not SIP, but the scheme

in the original received request was SIP or SIPS (that is, the

proxy changed the scheme from SIP or SIPS to something else

when it proxied a request), the proxy SHOULD add a new URI to

the target set. This URI SHOULD be a SIP URI version of the

non-SIP URI that was just tried. In the case of the tel URL,

this is accomplished by placing the telephone-subscriber part

of the tel URL into the user part of the SIP URI, and setting

the hostpart to the domain where the prior request was sent.

See Section 19.1.6 for more detail on forming SIP URIs from tel

URLs.

As with a 3xx response, if a proxy "recurses" on the 416 by

trying a SIP or SIPS URI instead, the 416 response SHOULD NOT

be added to the response context.

5. Check response for forwarding

Until a final response has been sent on the server transaction,

the following responses MUST be forwarded immediately:

- Any provisional response other than 100 (Trying)

- Any 2xx response

If a 6xx response is received, it is not immediately forwarded,

but the stateful proxy SHOULD cancel all client pending

transactions as described in Section 10, and it MUST NOT create

any new branches in this context.

This is a change from RFC 2543, which mandated that the proxy

was to forward the 6xx response immediately. For an INVITE

transaction, this approach had the problem that a 2xx response

could arrive on another branch, in which case the proxy would

Rosenberg, et. al. Standards Track [Page 109]

RFC 3261 SIP: Session Initiation Protocol June 2002

have to forward the 2xx. The result was that the UAC could

receive a 6xx response followed by a 2xx response, which should

never be allowed to happen. Under the new rules, upon

receiving a 6xx, a proxy will issue a CANCEL request, which

will generally result in 487 responses from all outstanding

client transactions, and then at that point the 6xx is

forwarded upstream.

After a final response has been sent on the server transaction,

the following responses MUST be forwarded immediately:

- Any 2xx response to an INVITE request

A stateful proxy MUST NOT immediately forward any other

responses. In particular, a stateful proxy MUST NOT forward

any 100 (Trying) response. Those responses that are candidates

for forwarding later as the "best" response have been gathered

as described in step "Add Response to Context".

Any response chosen for immediate forwarding MUST be processed

as described in steps "Aggregate Authorization Header Field

Values" through "Record-Route".

This step, combined with the next, ensures that a stateful

proxy will forward exactly one final response to a non-INVITE

request, and either exactly one non-2xx response or one or more

2xx responses to an INVITE request.

6. Choosing the best response

A stateful proxy MUST send a final response to a response

context's server transaction if no final responses have been

immediately forwarded by the above rules and all client

transactions in this response context have been terminated.

The stateful proxy MUST choose the "best" final response among

those received and stored in the response context.

If there are no final responses in the context, the proxy MUST

send a 408 (Request Timeout) response to the server

transaction.

Otherwise, the proxy MUST forward a response from the responses

stored in the response context. It MUST choose from the 6xx

class responses if any exist in the context. If no 6xx class

responses are present, the proxy SHOULD choose from the lowest

response class stored in the response context. The proxy MAY

select any response within that chosen class. The proxy SHOULD

Rosenberg, et. al. Standards Track [Page 110]

RFC 3261 SIP: Session Initiation Protocol June 2002

give preference to responses that provide information affecting

resubmission of this request, such as 401, 407, 415, 420, and

484 if the 4xx class is chosen.

A proxy which receives a 503 (Service Unavailable) response

SHOULD NOT forward it upstream unless it can determine that any

subsequent requests it might proxy will also generate a 503.

In other words, forwarding a 503 means that the proxy knows it

cannot service any requests, not just the one for the Request-

URI in the request which generated the 503. If the only

response that was received is a 503, the proxy SHOULD generate

a 500 response and forward that upstream.

The forwarded response MUST be processed as described in steps

"Aggregate Authorization Header Field Values" through "Record-

Route".

For example, if a proxy forwarded a request to 4 locations, and

received 503, 407, 501, and 404 responses, it may choose to

forward the 407 (Proxy Authentication Required) response.

1xx and 2xx responses may be involved in the establishment of

dialogs. When a request does not contain a To tag, the To tag

in the response is used by the UAC to distinguish multiple

responses to a dialog creating request. A proxy MUST NOT

insert a tag into the To header field of a 1xx or 2xx response

if the request did not contain one. A proxy MUST NOT modify

the tag in the To header field of a 1xx or 2xx response.

Since a proxy may not insert a tag into the To header field of

a 1xx response to a request that did not contain one, it cannot

issue non-100 provisional responses on its own. However, it

can branch the request to a UAS sharing the same element as the

proxy. This UAS can return its own provisional responses,

entering into an early dialog with the initiator of the

request. The UAS does not have to be a discreet process from

the proxy. It could be a virtual UAS implemented in the same

code space as the proxy.

3-6xx responses are delivered hop-by-hop. When issuing a 3-6xx

response, the element is effectively acting as a UAS, issuing

its own response, usually based on the responses received from

downstream elements. An element SHOULD preserve the To tag

when simply forwarding a 3-6xx response to a request that did

not contain a To tag.

A proxy MUST NOT modify the To tag in any forwarded response to

a request that contains a To tag.

Rosenberg, et. al. Standards Track [Page 111]

RFC 3261 SIP: Session Initiation Protocol June 2002

While it makes no difference to the upstream elements if the

proxy replaced the To tag in a forwarded 3-6xx response,

preserving the original tag may assist with debugging.

When the proxy is aggregating information from several

responses, choosing a To tag from among them is arbitrary, and

generating a new To tag may make debugging easier. This

happens, for instance, when combining 401 (Unauthorized) and

407 (Proxy Authentication Required) challenges, or combining

Contact values from unencrypted and unauthenticated 3xx

responses.

7. Aggregate Authorization Header Field Values

If the selected response is a 401 (Unauthorized) or 407 (Proxy

Authentication Required), the proxy MUST collect any WWW-

Authenticate and Proxy-Authenticate header field values from

all other 401 (Unauthorized) and 407 (Proxy Authentication

Required) responses received so far in this response context

and add them to this response without modification before

forwarding. The resulting 401 (Unauthorized) or 407 (Proxy

Authentication Required) response could have several WWW-

Authenticate AND Proxy-Authenticate header field values.

This is necessary because any or all of the destinations the

request was forwarded to may have requested credentials. The

client needs to receive all of those challenges and supply

credentials for each of them when it retries the request.

Motivation for this behavior is provided in Section 26.

8. Record-Route

If the selected response contains a Record-Route header field

value originally provided by this proxy, the proxy MAY choose

to rewrite the value before forwarding the response. This

allows the proxy to provide different URIs for itself to the

next upstream and downstream elements. A proxy may choose to

use this mechanism for any reason. For instance, it is useful

for multi-homed hosts.

If the proxy received the request over TLS, and sent it out

over a non-TLS connection, the proxy MUST rewrite the URI in

the Record-Route header field to be a SIPS URI. If the proxy

received the request over a non-TLS connection, and sent it out

over TLS, the proxy MUST rewrite the URI in the Record-Route

header field to be a SIP URI.

Rosenberg, et. al. Standards Track [Page 112]

RFC 3261 SIP: Session Initiation Protocol June 2002

The new URI provided by the proxy MUST satisfy the same

constraints on URIs placed in Record-Route header fields in

requests (see Step 4 of Section 16.6) with the following

modifications:

The URI SHOULD NOT contain the transport parameter unless the

proxy has knowledge that the next upstream (as opposed to

downstream) element that will be in the path of subsequent

requests supports that transport.

When a proxy does decide to modify the Record-Route header

field in the response, one of the operations it performs is

locating the Record-Route value that it had inserted. If the

request spiraled, and the proxy inserted a Record-Route value

in each iteration of the spiral, locating the correct value in

the response (which must be the proper iteration in the reverse

direction) is tricky. The rules above recommend that a proxy

wishing to rewrite Record-Route header field values insert

sufficiently distinct URIs into the Record-Route header field

so that the right one may be selected for rewriting. A

RECOMMENDED mechanism to achieve this is for the proxy to

append a unique identifier for the proxy instance to the user

portion of the URI.

When the response arrives, the proxy modifies the first

Record-Route whose identifier matches the proxy instance. The

modification results in a URI without this piece of data

appended to the user portion of the URI. Upon the next

iteration, the same algorithm (find the topmost Record-Route

header field value with the parameter) will correctly extract

the next Record-Route header field value inserted by that

proxy.

Not every response to a request to which a proxy adds a

Record-Route header field value will contain a Record-Route

header field. If the response does contain a Record-Route

header field, it will contain the value the proxy added.

9. Forward response

After performing the processing described in steps "Aggregate

Authorization Header Field Values" through "Record-Route", the

proxy MAY perform any feature specific manipulations on the

selected response. The proxy MUST NOT add to, modify, or

remove the message body. Unless otherwise specified, the proxy

MUST NOT remove any header field values other than the Via

header field value discussed in Section 16.7 Item 3. In

particular, the proxy MUST NOT remove any "received" parameter

Rosenberg, et. al. Standards Track [Page 113]

RFC 3261 SIP: Session Initiation Protocol June 2002

it may have added to the next Via header field value while

processing the request associated with this response. The

proxy MUST pass the response to the server transaction

associated with the response context. This will result in the

response being sent to the location now indicated in the

topmost Via header field value. If the server transaction is

no longer available to handle the transmission, the element

MUST forward the response statelessly by sending it to the

server transport. The server transaction might indicate

failure to send the response or signal a timeout in its state

machine. These errors would be logged for diagnostic purposes

as appropriate, but the protocol requires no remedial action

from the proxy.

The proxy MUST maintain the response context until all of its

associated transactions have been terminated, even after

forwarding a final response.

10. Generate CANCELs

If the forwarded response was a final response, the proxy MUST

generate a CANCEL request for all pending client transactions

associated with this response context. A proxy SHOULD also

generate a CANCEL request for all pending client transactions

associated with this response context when it receives a 6xx

response. A pending client transaction is one that has

received a provisional response, but no final response (it is

in the proceeding state) and has not had an associated CANCEL

generated for it. Generating CANCEL requests is described in

Section 9.1.

The requirement to CANCEL pending client transactions upon

forwarding a final response does not guarantee that an endpoint

will not receive multiple 200 (OK) responses to an INVITE. 200

(OK) responses on more than one branch may be generated before

the CANCEL requests can be sent and processed. Further, it is

reasonable to expect that a future extension may override this

requirement to issue CANCEL requests.

16.8 Processing Timer C

If timer C should fire, the proxy MUST either reset the timer with

any value it chooses, or terminate the client transaction. If the

client transaction has received a provisional response, the proxy

MUST generate a CANCEL request matching that transaction. If the

client transaction has not received a provisional response, the proxy

MUST behave as if the transaction received a 408 (Request Timeout)

response.

Rosenberg, et. al. Standards Track [Page 114]

RFC 3261 SIP: Session Initiation Protocol June 2002

Allowing the proxy to reset the timer allows the proxy to dynamically

extend the transaction's lifetime based on current conditions (such

as utilization) when the timer fires.

16.9 Handling Transport Errors

If the transport layer notifies a proxy of an error when it tries to

forward a request (see Section 18.4), the proxy MUST behave as if the

forwarded request received a 503 (Service Unavailable) response.

If the proxy is notified of an error when forwarding a response, it

drops the response. The proxy SHOULD NOT cancel any outstanding

client transactions associated with this response context due to this

notification.

If a proxy cancels its outstanding client transactions, a single

malicious or misbehaving client can cause all transactions to fail

through its Via header field.

16.10 CANCEL Processing

A stateful proxy MAY generate a CANCEL to any other request it has

generated at any time (subject to receiving a provisional response to

that request as described in section 9.1). A proxy MUST cancel any

pending client transactions associated with a response context when

it receives a matching CANCEL request.

A stateful proxy MAY generate CANCEL requests for pending INVITE

client transactions based on the period specified in the INVITE's

Expires header field elapsing. However, this is generally

unnecessary since the endpoints involved will take care of signaling

the end of the transaction.

While a CANCEL request is handled in a stateful proxy by its own

server transaction, a new response context is not created for it.

Instead, the proxy layer searches its existing response contexts for

the server transaction handling the request associated with this

CANCEL. If a matching response context is found, the element MUST

immediately return a 200 (OK) response to the CANCEL request. In

this case, the element is acting as a user agent server as defined in

Section 8.2. Furthermore, the element MUST generate CANCEL requests

for all pending client transactions in the context as described in

Section 16.7 step 10.

If a response context is not found, the element does not have any

knowledge of the request to apply the CANCEL to. It MUST statelessly

forward the CANCEL request (it may have statelessly forwarded the

associated request previously).

Rosenberg, et. al. Standards Track [Page 115]

RFC 3261 SIP: Session Initiation Protocol June 2002

16.11 Stateless Proxy

When acting statelessly, a proxy is a simple message forwarder. Much

of the processing performed when acting statelessly is the same as

when behaving statefully. The differences are detailed here.

A stateless proxy does not have any notion of a transaction, or of

the response context used to describe stateful proxy behavior.

Instead, the stateless proxy takes messages, both requests and

responses, directly from the transport layer (See section 18). As a

result, stateless proxies do not retransmit messages on their own.

They do, however, forward all retransmissions they receive (they do

not have the ability to distinguish a retransmission from the

original message). Furthermore, when handling a request statelessly,

an element MUST NOT generate its own 100 (Trying) or any other

provisional response.

A stateless proxy MUST validate a request as described in Section

16.3

A stateless proxy MUST follow the request processing steps described

in Sections 16.4 through 16.5 with the following exception:

o A stateless proxy MUST choose one and only one target from the

target set. This choice MUST only rely on fields in the

message and time-invariant properties of the server. In

particular, a retransmitted request MUST be forwarded to the

same destination each time it is processed. Furthermore,

CANCEL and non-Routed ACK requests MUST generate the same

choice as their associated INVITE.

A stateless proxy MUST follow the request processing steps described

in Section 16.6 with the following exceptions:

o The requirement for unique branch IDs across space and time

applies to stateless proxies as well. However, a stateless

proxy cannot simply use a random number generator to compute

the first component of the branch ID, as described in Section

16.6 bullet 8. This is because retransmissions of a request

need to have the same value, and a stateless proxy cannot tell

a retransmission from the original request. Therefore, the

component of the branch parameter that makes it unique MUST be

the same each time a retransmitted request is forwarded. Thus

for a stateless proxy, the branch parameter MUST be computed as

a combinatoric function of message parameters which are

invariant on retransmission.

Rosenberg, et. al. Standards Track [Page 116]

RFC 3261 SIP: Session Initiation Protocol June 2002

The stateless proxy MAY use any technique it likes to guarantee

uniqueness of its branch IDs across transactions. However, the

following procedure is RECOMMENDED. The proxy examines the

branch ID in the topmost Via header field of the received

request. If it begins with the magic cookie, the first

component of the branch ID of the outgoing request is computed

as a hash of the received branch ID. Otherwise, the first

component of the branch ID is computed as a hash of the topmost

Via, the tag in the To header field, the tag in the From header

field, the Call-ID header field, the CSeq number (but not

method), and the Request-URI from the received request. One of

these fields will always vary across two different

transactions.

o All other message transformations specified in Section 16.6

MUST result in the same transformation of a retransmitted

request. In particular, if the proxy inserts a Record-Route

value or pushes URIs into the Route header field, it MUST place

the same values in retransmissions of the request. As for the

Via branch parameter, this implies that the transformations

MUST be based on time-invariant configuration or

retransmission-invariant properties of the request.

o A stateless proxy determines where to forward the request as

described for stateful proxies in Section 16.6 Item 10. The

request is sent directly to the transport layer instead of

through a client transaction.

Since a stateless proxy must forward retransmitted requests to

the same destination and add identical branch parameters to

each of them, it can only use information from the message

itself and time-invariant configuration data for those

calculations. If the configuration state is not time-invariant

(for example, if a routing table is updated) any requests that

could be affected by the change may not be forwarded

statelessly during an interval equal to the transaction timeout

window before or after the change. The method of processing

the affected requests in that interval is an implementation

decision. A common solution is to forward them transaction

statefully.

Stateless proxies MUST NOT perform special processing for CANCEL

requests. They are processed by the above rules as any other

requests. In particular, a stateless proxy applies the same Route

header field processing to CANCEL requests that it applies to any

other request.

Rosenberg, et. al. Standards Track [Page 117]

RFC 3261 SIP: Session Initiation Protocol June 2002

Response processing as described in Section 16.7 does not apply to a

proxy behaving statelessly. When a response arrives at a stateless

proxy, the proxy MUST inspect the sent-by value in the first

(topmost) Via header field value. If that address matches the proxy,

(it equals a value this proxy has inserted into previous requests)

the proxy MUST remove that header field value from the response and

forward the result to the location indicated in the next Via header

field value. The proxy MUST NOT add to, modify, or remove the

message body. Unless specified otherwise, the proxy MUST NOT remove

any other header field values. If the address does not match the

proxy, the message MUST be silently discarded.

16.12 Summary of Proxy Route Processing

In the absence of local policy to the contrary, the processing a

proxy performs on a request containing a Route header field can be

summarized in the following steps.

1. The proxy will inspect the Request-URI. If it indicates a

resource owned by this proxy, the proxy will replace it with

the results of running a location service. Otherwise, the

proxy will not change the Request-URI.

2. The proxy will inspect the URI in the topmost Route header

field value. If it indicates this proxy, the proxy removes it

from the Route header field (this route node has been

reached).

3. The proxy will forward the request to the resource indicated

by the URI in the topmost Route header field value or in the

Request-URI if no Route header field is present. The proxy

determines the address, port and transport to use when

forwarding the request by applying the procedures in [4] to

that URI.

If no strict-routing elements are encountered on the path of the

request, the Request-URI will always indicate the target of the

request.

16.12.1 Examples

16.12.1.1 Basic SIP Trapezoid

This scenario is the basic SIP trapezoid, U1 -> P1 -> P2 -> U2, with

both proxies record-routing. Here is the flow.

Rosenberg, et. al. Standards Track [Page 118]

RFC 3261 SIP: Session Initiation Protocol June 2002

U1 sends:

INVITE sip:callee@domain.com SIP/2.0

Contact: sip:caller@u1.example.com

to P1. P1 is an outbound proxy. P1 is not responsible for

domain.com, so it looks it up in DNS and sends it there. It also

adds a Record-Route header field value:

INVITE sip:callee@domain.com SIP/2.0

Contact: sip:caller@u1.example.com

Record-Route: <sip:p1.example.com;lr>

P2 gets this. It is responsible for domain.com so it runs a location

service and rewrites the Request-URI. It also adds a Record-Route

header field value. There is no Route header field, so it resolves

the new Request-URI to determine where to send the request:

INVITE sip:callee@u2.domain.com SIP/2.0

Contact: sip:caller@u1.example.com

Record-Route: <sip:p2.domain.com;lr>

Record-Route: <sip:p1.example.com;lr>

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

SIP/2.0 200 OK

Contact: sip:callee@u2.domain.com

Record-Route: <sip:p2.domain.com;lr>

Record-Route: <sip:p1.example.com;lr>

The callee at u2 also sets its dialog state's remote target URI to

sip:caller@u1.example.com and its route set to:

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

This is forwarded by P2 to P1 to U1 as normal. Now, U1 sets its

dialog state's remote target URI to sip:callee@u2.domain.com and its

route set to:

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

Since all the route set elements contain the lr parameter, U1

constructs the following BYE request:

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

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

Rosenberg, et. al. Standards Track [Page 119]

RFC 3261 SIP: Session Initiation Protocol June 2002

As any other element (including proxies) would do, it resolves the

URI in the topmost Route header field value using DNS to determine

where to send the request. This goes to P1. P1 notices that it is

not responsible for the resource indicated in the Request-URI so it

doesn't change it. It does see that it is the first value in the

Route header field, so it removes that value, and forwards the

request to P2:

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

Route: <sip:p2.domain.com;lr>

P2 also notices it is not responsible for the resource indicated by

the Request-URI (it is responsible for domain.com, not

u2.domain.com), so it doesn't change it. It does see itself in the

first Route header field value, so it removes it and forwards the

following to u2.domain.com based on a DNS lookup against the

Request-URI:

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

16.12.1.2 Traversing a Strict-Routing Proxy

In this scenario, a dialog is established across four proxies, each

of which adds Record-Route header field values. The third proxy

implements the strict-routing procedures specified in RFC 2543 and

many works in progress.

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

The INVITE arriving at U2 contains:

INVITE sip:callee@u2.domain.com SIP/2.0

Contact: sip:caller@u1.example.com

Record-Route: <sip:p4.domain.com;lr>

Record-Route: <sip:p3.middle.com>

Record-Route: <sip:p2.example.com;lr>

Record-Route: <sip:p1.example.com;lr>

Which U2 responds to with a 200 OK. Later, U2 sends the following

BYE request to P4 based on the first Route header field value.

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

Route: <sip:p4.domain.com;lr>

Route: <sip:p3.middle.com>

Route: <sip:p2.example.com;lr>

Route: <sip:p1.example.com;lr>

Rosenberg, et. al. Standards Track [Page 120]

RFC 3261 SIP: Session Initiation Protocol June 2002

P4 is not responsible for the resource indicated in the Request-URI

so it will leave it alone. It notices that it is the element in the

first Route header field value so it removes it. It then prepares to

send the request based on the now first Route header field value of

sip:p3.middle.com, but it notices that this URI does not contain the

lr parameter, so before sending, it reformats the request to be:

BYE sip:p3.middle.com SIP/2.0

Route: <sip:p2.example.com;lr>

Route: <sip:p1.example.com;lr>

Route: <sip:caller@u1.example.com>

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

BYE sip:p2.example.com;lr SIP/2.0

Route: <sip:p1.example.com;lr>

Route: <sip:caller@u1.example.com>

P2 sees the request-URI is a value it placed into a Record-Route

header field, so before further processing, it rewrites the request

to be:

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

Route: <sip:p1.example.com;lr>

P2 is not responsible for u1.example.com, so it sends the request to

P1 based on the resolution of the Route header field value.

P1 notices itself in the topmost Route header field value, so it

removes it, resulting in:

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

Since P1 is not responsible for u1.example.com and there is no Route

header field, P1 will forward the request to u1.example.com based on

the Request-URI.

16.12.1.3 Rewriting Record-Route Header Field Values

In this scenario, U1 and U2 are in different private namespaces and

they enter a dialog through a proxy P1, which acts as a gateway

between the namespaces.

U1->P1->U2

Rosenberg, et. al. Standards Track [Page 121]

RFC 3261 SIP: Session Initiation Protocol June 2002

U1 sends:

INVITE sip:callee@gateway.leftprivatespace.com SIP/2.0

Contact: <sip:caller@u1.leftprivatespace.com>

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

INVITE sip:callee@rightprivatespace.com SIP/2.0

Contact: <sip:caller@u1.leftprivatespace.com>

Record-Route: <sip:gateway.rightprivatespace.com;lr>

U2 sends this 200 (OK) back to P1:

SIP/2.0 200 OK

Contact: <sip:callee@u2.rightprivatespace.com>

Record-Route: <sip:gateway.rightprivatespace.com;lr>

P1 rewrites its Record-Route header parameter to provide a value that

U1 will find useful, and sends the following to U1:

SIP/2.0 200 OK

Contact: <sip:callee@u2.rightprivatespace.com>

Record-Route: <sip:gateway.leftprivatespace.com;lr>

Later, U1 sends the following BYE request to P1:

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

Route: <sip:gateway.leftprivatespace.com;lr>

which P1 forwards to U2 as:

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

17 Transactions

SIP is a transactional protocol: interactions between components take

place in a series of independent message exchanges. Specifically, a

SIP transaction consists of a single request and any responses to

that request, which include zero or more provisional responses and

one or more final responses. In the case of a transaction where the

request was an INVITE (known as an INVITE transaction), the

transaction also includes the ACK only if the final response was not

a 2xx response. If the response was a 2xx, the ACK is not considered

part of the transaction.

The reason for this separation is rooted in the importance of

delivering all 200 (OK) responses to an INVITE to the UAC. To

deliver them all to the UAC, the UAS alone takes responsibility

Rosenberg, et. al. Standards Track [Page 122]

RFC 3261 SIP: Session Initiation Protocol June 2002

for retransmitting them (see Section 13.3.1.4), and the UAC alone

takes responsibility for acknowledging them with ACK (see Section

13.2.2.4). Since this ACK is retransmitted only by the UAC, it is

effectively considered its own transaction.

Transactions have a client side and a server side. The client side

is known as a client transaction and the server side as a server

transaction. The client transaction sends the request, and the

server transaction sends the response. The client and server

transactions are logical functions that are embedded in any number of

elements. Specifically, they exist within user agents and stateful

proxy servers. Consider the example in Section 4. In this example,

the UAC executes the client transaction, and its outbound proxy

executes the server transaction. The outbound proxy also executes a

client transaction, which sends the request to a server transaction

in the inbound proxy. That proxy also executes a client transaction,

which in turn sends the request to a server transaction in the UAS.

This is shown in Figure 4.

+---------+ +---------+ +---------+ +---------+

| +-+|Request |+-+ +-+|Request |+-+ +-+|Request |+-+ |

| |C||------->||S| |C||------->||S| |C||------->||S| |

| |l|| ||e| |l|| ||e| |l|| ||e| |

| |i|| ||r| |i|| ||r| |i|| ||r| |

| |e|| ||v| |e|| ||v| |e|| ||v| |

| |n|| ||e| |n|| ||e| |n|| ||e| |

| |t|| ||r| |t|| ||r| |t|| ||r| |

| | || || | | || || | | || || | |

| |T|| ||T| |T|| ||T| |T|| ||T| |

| |r|| ||r| |r|| ||r| |r|| ||r| |

| |a|| ||a| |a|| ||a| |a|| ||a| |

| |n|| ||n| |n|| ||n| |n|| ||n| |

| |s||Response||s| |s||Response||s| |s||Response||s| |

| +-+|<-------|+-+ +-+|<-------|+-+ +-+|<-------|+-+ |

+---------+ +---------+ +---------+ +---------+

UAC Outbound Inbound UAS

Proxy Proxy

Figure 4: Transaction relationships

A stateless proxy does not contain a client or server transaction.

The transaction exists between the UA or stateful proxy on one side,

and the UA or stateful proxy on the other side. As far as SIP

transactions are concerned, stateless proxies are effectively

transparent. The purpose of the client transaction is to receive a

request from the element in which the client is embedded (call this

element the "Transaction User" or TU; it can be a UA or a stateful

proxy), and reliably deliver the request to a server transaction.

Rosenberg, et. al. Standards Track [Page 123]

RFC 3261 SIP: Session Initiation Protocol June 2002

The client transaction is also responsible for receiving responses

and delivering them to the TU, filtering out any response

retransmissions or disallowed responses (such as a response to ACK).

Additionally, in the case of an INVITE request, the client

transaction is responsible for generating the ACK request for any

final response accepting a 2xx response.

Similarly, the purpose of the server transaction is to receive

requests from the transport layer and deliver them to the TU. The

server transaction filters any request retransmissions from the

network. The server transaction accepts responses from the TU and

delivers them to the transport layer for transmission over the

network. In the case of an INVITE transaction, it absorbs the ACK

request for any final response excepting a 2xx response.

The 2xx response and its ACK receive special treatment. This

response is retransmitted only by a UAS, and its ACK generated only

by the UAC. This end-to-end treatment is needed so that a caller

knows the entire set of users that have accepted the call. Because

of this special handling, retransmissions of the 2xx response are

handled by the UA core, not the transaction layer. Similarly,

generation of the ACK for the 2xx is handled by the UA core. Each

proxy along the path merely forwards each 2xx response to INVITE and

its corresponding ACK.

17.1 Client Transaction

The client transaction provides its functionality through the

maintenance of a state machine.

The TU communicates with the client transaction through a simple

interface. When the TU wishes to initiate a new transaction, it

creates a client transaction and passes it the SIP request to send

and an IP address, port, and transport to which to send it. The

client transaction begins execution of its state machine. Valid

responses are passed up to the TU from the client transaction.

There are two types of client transaction state machines, depending

on the method of the request passed by the TU. One handles client

transactions for INVITE requests. This type of machine is referred

to as an INVITE client transaction. Another type handles client

transactions for all requests except INVITE and ACK. This is

referred to as a non-INVITE client transaction. There is no client

transaction for ACK. If the TU wishes to send an ACK, it passes one

directly to the transport layer for transmission.

Rosenberg, et. al. Standards Track [Page 124]

RFC 3261 SIP: Session Initiation Protocol June 2002

The INVITE transaction is different from those of other methods

because of its extended duration. Normally, human input is required

in order to respond to an INVITE. The long delays expected for

sending a response argue for a three-way handshake. On the other

hand, requests of other methods are expected to complete rapidly.

Because of the non-INVITE transaction's reliance on a two-way

handshake, TUs SHOULD respond immediately to non-INVITE requests.

17.1.1 INVITE Client Transaction

17.1.1.1 Overview of INVITE Transaction

The INVITE transaction consists of a three-way handshake. The client

transaction sends an INVITE, the server transaction sends responses,

and the client transaction sends an ACK. For unreliable transports

(such as UDP), the client transaction retransmits requests at an

interval that starts at T1 seconds and doubles after every

retransmission. T1 is an estimate of the round-trip time (RTT), and

it defaults to 500 ms. Nearly all of the transaction timers

described here scale with T1, and changing T1 adjusts their values.

The request is not retransmitted over reliable transports. After

receiving a 1xx response, any retransmissions cease altogether, and

the client waits for further responses. The server transaction can

send additional 1xx responses, which are not transmitted reliably by

the server transaction. Eventually, the server transaction decides

to send a final response. For unreliable transports, that response

is retransmitted periodically, and for reliable transports, it is

sent once. For each final response that is received at the client

transaction, the client transaction sends an ACK, the purpose of

which is to quench retransmissions of the response.

17.1.1.2 Formal Description

The state machine for the INVITE client transaction is shown in

Figure 5. The initial state, "calling", MUST be entered when the TU

initiates a new client transaction with an INVITE request. The

client transaction MUST pass the request to the transport layer for

transmission (see Section 18). If an unreliable transport is being

used, the client transaction MUST start timer A with a value of T1.

If a reliable transport is being used, the client transaction SHOULD

NOT start timer A (Timer A controls request retransmissions). For

any transport, the client transaction MUST start timer B with a value

of 64\*T1 seconds (Timer B controls transaction timeouts).

When timer A fires, the client transaction MUST retransmit the

request by passing it to the transport layer, and MUST reset the

timer with a value of 2\*T1. The formal definition of retransmit

Rosenberg, et. al. Standards Track [Page 125]

RFC 3261 SIP: Session Initiation Protocol June 2002

within the context of the transaction layer is to take the message

previously sent to the transport layer and pass it to the transport

layer once more.

When timer A fires 2\*T1 seconds later, the request MUST be

retransmitted again (assuming the client transaction is still in this

state). This process MUST continue so that the request is

retransmitted with intervals that double after each transmission.

These retransmissions SHOULD only be done while the client

transaction is in the "calling" state.

The default value for T1 is 500 ms. T1 is an estimate of the RTT

between the client and server transactions. Elements MAY (though it

is NOT RECOMMENDED) use smaller values of T1 within closed, private

networks that do not permit general Internet connection. T1 MAY be

chosen larger, and this is RECOMMENDED if it is known in advance

(such as on high latency access links) that the RTT is larger.

Whatever the value of T1, the exponential backoffs on retransmissions

described in this section MUST be used.

If the client transaction is still in the "Calling" state when timer

B fires, the client transaction SHOULD inform the TU that a timeout

has occurred. The client transaction MUST NOT generate an ACK. The

value of 64\*T1 is equal to the amount of time required to send seven

requests in the case of an unreliable transport.

If the client transaction receives a provisional response while in

the "Calling" state, it transitions to the "Proceeding" state. In the

"Proceeding" state, the client transaction SHOULD NOT retransmit the

request any longer. Furthermore, the provisional response MUST be

passed to the TU. Any further provisional responses MUST be passed

up to the TU while in the "Proceeding" state.

When in either the "Calling" or "Proceeding" states, reception of a

response with status code from 300-699 MUST cause the client

transaction to transition to "Completed". The client transaction

MUST pass the received response up to the TU, and the client

transaction MUST generate an ACK request, even if the transport is

reliable (guidelines for constructing the ACK from the response are

given in Section 17.1.1.3) and then pass the ACK to the transport

layer for transmission. The ACK MUST be sent to the same address,

port, and transport to which the original request was sent. The

client transaction SHOULD start timer D when it enters the

"Completed" state, with a value of at least 32 seconds for unreliable

transports, and a value of zero seconds for reliable transports.

Timer D reflects the amount of time that the server transaction can

remain in the "Completed" state when unreliable transports are used.

This is equal to Timer H in the INVITE server transaction, whose

Rosenberg, et. al. Standards Track [Page 126]

RFC 3261 SIP: Session Initiation Protocol June 2002

default is 64\*T1. However, the client transaction does not know the

value of T1 in use by the server transaction, so an absolute minimum

of 32s is used instead of basing Timer D on T1.

Any retransmissions of the final response that are received while in

the "Completed" state MUST cause the ACK to be re-passed to the

transport layer for retransmission, but the newly received response

MUST NOT be passed up to the TU. A retransmission of the response is

defined as any response which would match the same client transaction

based on the rules of Section 17.1.3.

Rosenberg, et. al. Standards Track [Page 127]

RFC 3261 SIP: Session Initiation Protocol June 2002

|INVITE from TU

Timer A fires |INVITE sent

Reset A, V Timer B fires

INVITE sent +-----------+ or Transport Err.

+---------| |---------------+inform TU

| | Calling | |

+-------->| |-------------->|

+-----------+ 2xx |

| | 2xx to TU |

| |1xx |

300-699 +---------------+ |1xx to TU |

ACK sent | | |

resp. to TU | 1xx V |

| 1xx to TU -----------+ |

| +---------| | |

| | |Proceeding |-------------->|

| +-------->| | 2xx |

| +-----------+ 2xx to TU |

| 300-699 | |

| ACK sent, | |

| resp. to TU| |

| | | NOTE:

| 300-699 V |

| ACK sent +-----------+Transport Err. | transitions

| +---------| |Inform TU | labeled with

| | | Completed |-------------->| the event

| +-------->| | | over the action

| +-----------+ | to take

| ^ | |

| | | Timer D fires |

+--------------+ | - |

| |

V |

+-----------+ |

| | |

| Terminated|<--------------+

| |

+-----------+

Figure 5: INVITE client transaction

If timer D fires while the client transaction is in the "Completed"

state, the client transaction MUST move to the terminated state.

When in either the "Calling" or "Proceeding" states, reception of a

2xx response MUST cause the client transaction to enter the

"Terminated" state, and the response MUST be passed up to the TU.

The handling of this response depends on whether the TU is a proxy

Rosenberg, et. al. Standards Track [Page 128]

RFC 3261 SIP: Session Initiation Protocol June 2002

core or a UAC core. A UAC core will handle generation of the ACK for

this response, while a proxy core will always forward the 200 (OK)

upstream. The differing treatment of 200 (OK) between proxy and UAC

is the reason that handling of it does not take place in the

transaction layer.

The client transaction MUST be destroyed the instant it enters the

"Terminated" state. This is actually necessary to guarantee correct

operation. The reason is that 2xx responses to an INVITE are treated

differently; each one is forwarded by proxies, and the ACK handling

in a UAC is different. Thus, each 2xx needs to be passed to a proxy

core (so that it can be forwarded) and to a UAC core (so it can be

acknowledged). No transaction layer processing takes place.

Whenever a response is received by the transport, if the transport

layer finds no matching client transaction (using the rules of

Section 17.1.3), the response is passed directly to the core. Since

the matching client transaction is destroyed by the first 2xx,

subsequent 2xx will find no match and therefore be passed to the

core.

17.1.1.3 Construction of the ACK Request

This section specifies the construction of ACK requests sent within

the client transaction. A UAC core that generates an ACK for 2xx

MUST instead follow the rules described in Section 13.

The ACK request constructed by the client transaction MUST contain

values for the Call-ID, From, and Request-URI that are equal to the

values of those header fields in the request passed to the transport

by the client transaction (call this the "original request"). The To

header field in the ACK MUST equal the To header field in the

response being acknowledged, and therefore will usually differ from

the To header field in the original request by the addition of the

tag parameter. The ACK MUST contain a single Via header field, and

this MUST be equal to the top Via header field of the original

request. The CSeq header field in the ACK MUST contain the same

value for the sequence number as was present in the original request,

but the method parameter MUST be equal to "ACK".

Rosenberg, et. al. Standards Track [Page 129]

RFC 3261 SIP: Session Initiation Protocol June 2002

If the INVITE request whose response is being acknowledged had Route

header fields, those header fields MUST appear in the ACK. This is

to ensure that the ACK can be routed properly through any downstream

stateless proxies.

Although any request MAY contain a body, a body in an ACK is special

since the request cannot be rejected if the body is not understood.

Therefore, placement of bodies in ACK for non-2xx is NOT RECOMMENDED,

but if done, the body types are restricted to any that appeared in

the INVITE, assuming that the response to the INVITE was not 415. If

it was, the body in the ACK MAY be any type listed in the Accept

header field in the 415.

For example, consider the following request:

INVITE sip:bob@biloxi.com SIP/2.0

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKkjshdyff

To: Bob <sip:bob@biloxi.com>

From: Alice <sip:alice@atlanta.com>;tag=88sja8x

Max-Forwards: 70

Call-ID: 987asjd97y7atg

CSeq: 986759 INVITE

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

look like this:

ACK sip:bob@biloxi.com SIP/2.0

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKkjshdyff

To: Bob <sip:bob@biloxi.com>;tag=99sa0xk

From: Alice <sip:alice@atlanta.com>;tag=88sja8x

Max-Forwards: 70

Call-ID: 987asjd97y7atg

CSeq: 986759 ACK

17.1.2 Non-INVITE Client Transaction

17.1.2.1 Overview of the non-INVITE Transaction

Non-INVITE transactions do not make use of ACK. They are simple

request-response interactions. For unreliable transports, requests

are retransmitted at an interval which starts at T1 and doubles until

it hits T2. If a provisional response is received, retransmissions

continue for unreliable transports, but at an interval of T2. The

server transaction retransmits the last response it sent, which can

be a provisional or final response, only when a retransmission of the

request is received. This is why request retransmissions need to

continue even after a provisional response; they are to ensure

reliable delivery of the final response.

Rosenberg, et. al. Standards Track [Page 130]

RFC 3261 SIP: Session Initiation Protocol June 2002

Unlike an INVITE transaction, a non-INVITE transaction has no special

handling for the 2xx response. The result is that only a single 2xx

response to a non-INVITE is ever delivered to a UAC.

17.1.2.2 Formal Description

The state machine for the non-INVITE client transaction is shown in

Figure 6. It is very similar to the state machine for INVITE.

The "Trying" state is entered when the TU initiates a new client

transaction with a request. When entering this state, the client

transaction SHOULD set timer F to fire in 64\*T1 seconds. The request

MUST be passed to the transport layer for transmission. If an

unreliable transport is in use, the client transaction MUST set timer

E to fire in T1 seconds. If timer E fires while still in this state,

the timer is reset, but this time with a value of MIN(2\*T1, T2).

When the timer fires again, it is reset to a MIN(4\*T1, T2). This

process continues so that retransmissions occur with an exponentially

increasing interval that caps at T2. The default value of T2 is 4s,

and it represents the amount of time a non-INVITE server transaction

will take to respond to a request, if it does not respond

immediately. For the default values of T1 and T2, this results in

intervals of 500 ms, 1 s, 2 s, 4 s, 4 s, 4 s, etc.

If Timer F fires while the client transaction is still in the

"Trying" state, the client transaction SHOULD inform the TU about the

timeout, and then it SHOULD enter the "Terminated" state. If a

provisional response is received while in the "Trying" state, the

response MUST be passed to the TU, and then the client transaction

SHOULD move to the "Proceeding" state. If a final response (status

codes 200-699) is received while in the "Trying" state, the response

MUST be passed to the TU, and the client transaction MUST transition

to the "Completed" state.

If Timer E fires while in the "Proceeding" state, the request MUST be

passed to the transport layer for retransmission, and Timer E MUST be

reset with a value of T2 seconds. If timer F fires while in the

"Proceeding" state, the TU MUST be informed of a timeout, and the

client transaction MUST transition to the terminated state. If a

final response (status codes 200-699) is received while in the

"Proceeding" state, the response MUST be passed to the TU, and the

client transaction MUST transition to the "Completed" state.

Once the client transaction enters the "Completed" state, it MUST set

Timer K to fire in T4 seconds for unreliable transports, and zero

seconds for reliable transports. The "Completed" state exists to

buffer any additional response retransmissions that may be received

(which is why the client transaction remains there only for

Rosenberg, et. al. Standards Track [Page 131]

RFC 3261 SIP: Session Initiation Protocol June 2002

unreliable transports). T4 represents the amount of time the network

will take to clear messages between client and server transactions.

The default value of T4 is 5s. A response is a retransmission when

it matches the same transaction, using the rules specified in Section

17.1.3. If Timer K fires while in this state, the client transaction

MUST transition to the "Terminated" state.

Once the transaction is in the terminated state, it MUST be destroyed

immediately.

17.1.3 Matching Responses to Client Transactions

When the transport layer in the client receives a response, it has to

determine which client transaction will handle the response, so that

the processing of Sections 17.1.1 and 17.1.2 can take place. The

branch parameter in the top Via header field is used for this

purpose. A response matches a client transaction under two

conditions:

1. If the response has the same value of the branch parameter in

the top Via header field as the branch parameter in the top

Via header field of the request that created the transaction.

2. If the method parameter in the CSeq header field matches the

method of the request that created the transaction. The

method is needed since a CANCEL request constitutes a

different transaction, but shares the same value of the branch

parameter.

If a request is sent via multicast, it is possible that it will

generate multiple responses from different servers. These responses

will all have the same branch parameter in the topmost Via, but vary

in the To tag. The first response received, based on the rules

above, will be used, and others will be viewed as retransmissions.

That is not an error; multicast SIP provides only a rudimentary

"single-hop-discovery-like" service that is limited to processing a

single response. See Section 18.1.1 for details.

Rosenberg, et. al. Standards Track [Page 132]

RFC 3261 SIP: Session Initiation Protocol June 2002

17.1.4 Handling Transport Errors

|Request from TU

|send request

Timer E V

send request +-----------+

+---------| |-------------------+

| | Trying | Timer F |

+-------->| | or Transport Err.|

+-----------+ inform TU |

200-699 | | |

resp. to TU | |1xx |

+---------------+ |resp. to TU |

| | |

| Timer E V Timer F |

| send req +-----------+ or Transport Err. |

| +---------| | inform TU |

| | |Proceeding |------------------>|

| +-------->| |-----+ |

| +-----------+ |1xx |

| | ^ |resp to TU |

| 200-699 | +--------+ |

| resp. to TU | |

| | |

| V |

| +-----------+ |

| | | |

| | Completed | |

| | | |

| +-----------+ |

| ^ | |

| | | Timer K |

+--------------+ | - |

| |

V |

NOTE: +-----------+ |

| | |

transitions | Terminated|<------------------+

labeled with | |

the event +-----------+

over the action

to take

Figure 6: non-INVITE client transaction

When the client transaction sends a request to the transport layer to

be sent, the following procedures are followed if the transport layer

indicates a failure.

Rosenberg, et. al. Standards Track [Page 133]

RFC 3261 SIP: Session Initiation Protocol June 2002

The client transaction SHOULD inform the TU that a transport failure

has occurred, and the client transaction SHOULD transition directly

to the "Terminated" state. The TU will handle the failover

mechanisms described in [4].

17.2 Server Transaction

The server transaction is responsible for the delivery of requests to

the TU and the reliable transmission of responses. It accomplishes

this through a state machine. Server transactions are created by the

core when a request is received, and transaction handling is desired

for that request (this is not always the case).

As with the client transactions, the state machine depends on whether

the received request is an INVITE request.

17.2.1 INVITE Server Transaction

The state diagram for the INVITE server transaction is shown in

Figure 7.

When a server transaction is constructed for a request, it enters the

"Proceeding" state. The server transaction MUST generate a 100

(Trying) response unless it knows that the TU will generate a

provisional or final response within 200 ms, in which case it MAY

generate a 100 (Trying) response. This provisional response is

needed to quench request retransmissions rapidly in order to avoid

network congestion. The 100 (Trying) response is constructed

according to the procedures in Section 8.2.6, except that the

insertion of tags in the To header field of the response (when none

was present in the request) is downgraded from MAY to SHOULD NOT.

The request MUST be passed to the TU.

The TU passes any number of provisional responses to the server

transaction. So long as the server transaction is in the

"Proceeding" state, each of these MUST be passed to the transport

layer for transmission. They are not sent reliably by the

transaction layer (they are not retransmitted by it) and do not cause

a change in the state of the server transaction. If a request

retransmission is received while in the "Proceeding" state, the most

recent provisional response that was received from the TU MUST be

passed to the transport layer for retransmission. A request is a

retransmission if it matches the same server transaction based on the

rules of Section 17.2.3.

If, while in the "Proceeding" state, the TU passes a 2xx response to

the server transaction, the server transaction MUST pass this

response to the transport layer for transmission. It is not

Rosenberg, et. al. Standards Track [Page 134]

RFC 3261 SIP: Session Initiation Protocol June 2002

retransmitted by the server transaction; retransmissions of 2xx

responses are handled by the TU. The server transaction MUST then

transition to the "Terminated" state.

While in the "Proceeding" state, if the TU passes a response with

status code from 300 to 699 to the server transaction, the response

MUST be passed to the transport layer for transmission, and the state

machine MUST enter the "Completed" state. For unreliable transports,

timer G is set to fire in T1 seconds, and is not set to fire for

reliable transports.

This is a change from RFC 2543, where responses were always

retransmitted, even over reliable transports.

When the "Completed" state is entered, timer H MUST be set to fire in

64\*T1 seconds for all transports. Timer H determines when the server

transaction abandons retransmitting the response. Its value is

chosen to equal Timer B, the amount of time a client transaction will

continue to retry sending a request. If timer G fires, the response

is passed to the transport layer once more for retransmission, and

timer G is set to fire in MIN(2\*T1, T2) seconds. From then on, when

timer G fires, the response is passed to the transport again for

transmission, and timer G is reset with a value that doubles, unless

that value exceeds T2, in which case it is reset with the value of

T2. This is identical to the retransmit behavior for requests in the

"Trying" state of the non-INVITE client transaction. Furthermore,

while in the "Completed" state, if a request retransmission is

received, the server SHOULD pass the response to the transport for

retransmission.

If an ACK is received while the server transaction is in the

"Completed" state, the server transaction MUST transition to the

"Confirmed" state. As Timer G is ignored in this state, any

retransmissions of the response will cease.

If timer H fires while in the "Completed" state, it implies that the

ACK was never received. In this case, the server transaction MUST

transition to the "Terminated" state, and MUST indicate to the TU

that a transaction failure has occurred.

Rosenberg, et. al. Standards Track [Page 135]

RFC 3261 SIP: Session Initiation Protocol June 2002

|INVITE

|pass INV to TU

INVITE V send 100 if TU won't in 200ms

send response+-----------+

+--------| |--------+101-199 from TU

| | Proceeding| |send response

+------->| |<-------+

| | Transport Err.

| | Inform TU

| |--------------->+

+-----------+ |

300-699 from TU | |2xx from TU |

send response | |send response |

| +------------------>+

| |

INVITE V Timer G fires |

send response+-----------+ send response |

+--------| |--------+ |

| | Completed | | |

+------->| |<-------+ |

+-----------+ |

| | |

ACK | | |

- | +------------------>+

| Timer H fires |

V or Transport Err.|

+-----------+ Inform TU |

| | |

| Confirmed | |

| | |

+-----------+ |

| |

|Timer I fires |

|- |

| |

V |

+-----------+ |

| | |

| Terminated|<---------------+

| |

+-----------+

Figure 7: INVITE server transaction

Rosenberg, et. al. Standards Track [Page 136]

RFC 3261 SIP: Session Initiation Protocol June 2002

The purpose of the "Confirmed" state is to absorb any additional ACK

messages that arrive, triggered from retransmissions of the final

response. When this state is entered, timer I is set to fire in T4

seconds for unreliable transports, and zero seconds for reliable

transports. Once timer I fires, the server MUST transition to the

"Terminated" state.

Once the transaction is in the "Terminated" state, it MUST be

destroyed immediately. As with client transactions, this is needed

to ensure reliability of the 2xx responses to INVITE.

17.2.2 Non-INVITE Server Transaction

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

Figure 8.

The state machine is initialized in the "Trying" state and is passed

a request other than INVITE or ACK when initialized. This request is

passed up to the TU. Once in the "Trying" state, any further request

retransmissions are discarded. A request is a retransmission if it

matches the same server transaction, using the rules specified in

Section 17.2.3.

While in the "Trying" state, if the TU passes a provisional response

to the server transaction, the server transaction MUST enter the

"Proceeding" state. The response MUST be passed to the transport

layer for transmission. Any further provisional responses that are

received from the TU while in the "Proceeding" state MUST be passed

to the transport layer for transmission. If a retransmission of the

request is received while in the "Proceeding" state, the most

recently sent provisional response MUST be passed to the transport

layer for retransmission. If the TU passes a final response (status

codes 200-699) to the server while in the "Proceeding" state, the

transaction MUST enter the "Completed" state, and the response MUST

be passed to the transport layer for transmission.

When the server transaction enters the "Completed" state, it MUST set

Timer J to fire in 64\*T1 seconds for unreliable transports, and zero

seconds for reliable transports. While in the "Completed" state, the

server transaction MUST pass the final response to the transport

layer for retransmission whenever a retransmission of the request is

received. Any other final responses passed by the TU to the server

transaction MUST be discarded while in the "Completed" state. The

server transaction remains in this state until Timer J fires, at

which point it MUST transition to the "Terminated" state.

The server transaction MUST be destroyed the instant it enters the

"Terminated" state.

Rosenberg, et. al. Standards Track [Page 137]

RFC 3261 SIP: Session Initiation Protocol June 2002

17.2.3 Matching Requests to Server Transactions

When a request is received from the network by the server, it has to

be matched to an existing transaction. This is accomplished in the

following manner.

The branch parameter in the topmost Via header field of the request

is examined. If it is present and begins with the magic cookie

"z9hG4bK", the request was generated by a client transaction

compliant to this specification. Therefore, the branch parameter

will be unique across all transactions sent by that client. The

request matches a transaction if:

1. the branch parameter in the request is equal to the one in the

top Via header field of the request that created the

transaction, and

2. the sent-by value in the top Via of the request is equal to the

one in the request that created the transaction, and

3. the method of the request matches the one that created the

transaction, except for ACK, where the method of the request

that created the transaction is INVITE.

This matching rule applies to both INVITE and non-INVITE transactions

alike.

The sent-by value is used as part of the matching process because

there could be accidental or malicious duplication of branch

parameters from different clients.

If the branch parameter in the top Via header field is not present,

or does not contain the magic cookie, the following procedures are

used. These exist to handle backwards compatibility with RFC 2543

compliant implementations.

The INVITE request matches a transaction if the Request-URI, To tag,

From tag, Call-ID, CSeq, and top Via header field match those of the

INVITE request which created the transaction. In this case, the

INVITE is a retransmission of the original one that created the

transaction. The ACK request matches a transaction if the Request-

URI, From tag, Call-ID, CSeq number (not the method), and top Via

header field match those of the INVITE request which created the

transaction, and the To tag of the ACK matches the To tag of the

response sent by the server transaction. Matching is done based on

the matching rules defined for each of those header fields.

Inclusion of the tag in the To header field in the ACK matching

process helps disambiguate ACK for 2xx from ACK for other responses

Rosenberg, et. al. Standards Track [Page 138]

RFC 3261 SIP: Session Initiation Protocol June 2002

at a proxy, which may have forwarded both responses (This can occur

in unusual conditions. Specifically, when a proxy forked a request,

and then crashes, the responses may be delivered to another proxy,

which might end up forwarding multiple responses upstream). An ACK

request that matches an INVITE transaction matched by a previous ACK

is considered a retransmission of that previous ACK.

Rosenberg, et. al. Standards Track [Page 139]

RFC 3261 SIP: Session Initiation Protocol June 2002

|Request received

|pass to TU

V

+-----------+

| |

| Trying |-------------+

| | |

+-----------+ |200-699 from TU

| |send response

|1xx from TU |

|send response |

| |

Request V 1xx from TU |

send response+-----------+send response|

+--------| |--------+ |

| | Proceeding| | |

+------->| |<-------+ |

+<--------------| | |

|Trnsprt Err +-----------+ |

|Inform TU | |

| | |

| |200-699 from TU |

| |send response |

| Request V |

| send response+-----------+ |

| +--------| | |

| | | Completed |<------------+

| +------->| |

+<--------------| |

|Trnsprt Err +-----------+

|Inform TU |

| |Timer J fires

| |-

| |

| V

| +-----------+

| | |

+-------------->| Terminated|

| |

+-----------+

Figure 8: non-INVITE server transaction

For all other request methods, a request is matched to a transaction

if the Request-URI, To tag, From tag, Call-ID, CSeq (including the

method), and top Via header field match those of the request that

created the transaction. Matching is done based on the matching

Rosenberg, et. al. Standards Track [Page 140]

RFC 3261 SIP: Session Initiation Protocol June 2002

rules defined for each of those header fields. When a non-INVITE

request matches an existing transaction, it is a retransmission of

the request that created that transaction.

Because the matching rules include the Request-URI, the server cannot

match a response to a transaction. When the TU passes a response to

the server transaction, it must pass it to the specific server

transaction for which the response is targeted.

17.2.4 Handling Transport Errors

When the server transaction sends a response to the transport layer

to be sent, the following procedures are followed if the transport

layer indicates a failure.

First, the procedures in [4] are followed, which attempt to deliver

the response to a backup. If those should all fail, based on the

definition of failure in [4], the server transaction SHOULD inform

the TU that a failure has occurred, and SHOULD transition to the

terminated state.

18 Transport

The transport layer is responsible for the actual transmission of

requests and responses over network transports. This includes

determination of the connection to use for a request or response in

the case of connection-oriented transports.

The transport layer is responsible for managing persistent

connections for transport protocols like TCP and SCTP, or TLS over

those, including ones opened to the transport layer. This includes

connections opened by the client or server transports, so that

connections are shared between client and server transport functions.

These connections are indexed by the tuple formed from the address,

port, and transport protocol at the far end of the connection. When

a connection is opened by the transport layer, this index is set to

the destination IP, port and transport. When the connection is

accepted by the transport layer, this index is set to the source IP

address, port number, and transport. Note that, because the source

port is often ephemeral, but it cannot be known whether it is

ephemeral or selected through procedures in [4], connections accepted

by the transport layer will frequently not be reused. The result is

that two proxies in a "peering" relationship using a connection-

oriented transport frequently will have two connections in use, one

for transactions initiated in each direction.

Rosenberg, et. al. Standards Track [Page 141]

RFC 3261 SIP: Session Initiation Protocol June 2002

It is RECOMMENDED that connections be kept open for some

implementation-defined duration after the last message was sent or

received over that connection. This duration SHOULD at least equal

the longest amount of time the element would need in order to bring a

transaction from instantiation to the terminated state. This is to

make it likely that transactions are completed over the same

connection on which they are initiated (for example, request,

response, and in the case of INVITE, ACK for non-2xx responses).

This usually means at least 64\*T1 (see Section 17.1.1.1 for a

definition of T1). However, it could be larger in an element that

has a TU using a large value for timer C (bullet 11 of Section 16.6),

for example.

All SIP elements MUST implement UDP and TCP. SIP elements MAY

implement other protocols.

Making TCP mandatory for the UA is a substantial change from RFC

2543. It has arisen out of the need to handle larger messages,

which MUST use TCP, as discussed below. Thus, even if an element

never sends large messages, it may receive one and needs to be

able to handle them.

18.1 Clients

18.1.1 Sending Requests

The client side of the transport layer is responsible for sending the

request and receiving responses. The user of the transport layer

passes the client transport the request, an IP address, port,

transport, and possibly TTL for multicast destinations.

If a request is within 200 bytes of the path MTU, or if it is larger

than 1300 bytes and the path MTU is unknown, the request MUST be sent

using an RFC 2914 [43] congestion controlled transport protocol, such

as TCP. If this causes a change in the transport protocol from the

one indicated in the top Via, the value in the top Via MUST be

changed. This prevents fragmentation of messages over UDP and

provides congestion control for larger messages. However,

implementations MUST be able to handle messages up to the maximum

datagram packet size. For UDP, this size is 65,535 bytes, including

IP and UDP headers.

The 200 byte "buffer" between the message size and the MTU

accommodates the fact that the response in SIP can be larger than

the request. This happens due to the addition of Record-Route

header field values to the responses to INVITE, for example. With

the extra buffer, the response can be about 170 bytes larger than

the request, and still not be fragmented on IPv4 (about 30 bytes

Rosenberg, et. al. Standards Track [Page 142]

RFC 3261 SIP: Session Initiation Protocol June 2002

is consumed by IP/UDP, assuming no IPSec). 1300 is chosen when

path MTU is not known, based on the assumption of a 1500 byte

Ethernet MTU.

If an element sends a request over TCP because of these message size

constraints, and that request would have otherwise been sent over

UDP, if the attempt to establish the connection generates either an

ICMP Protocol Not Supported, or results in a TCP reset, the element

SHOULD retry the request, using UDP. This is only to provide

backwards compatibility with RFC 2543 compliant implementations that

do not support TCP. It is anticipated that this behavior will be

deprecated in a future revision of this specification.

A client that sends a request to a multicast address MUST add the

"maddr" parameter to its Via header field value containing the

destination multicast address, and for IPv4, SHOULD add the "ttl"

parameter with a value of 1. Usage of IPv6 multicast is not defined

in this specification, and will be a subject of future

standardization when the need arises.

These rules result in a purposeful limitation of multicast in SIP.

Its primary function is to provide a "single-hop-discovery-like"

service, delivering a request to a group of homogeneous servers,

where it is only required to process the response from any one of

them. This functionality is most useful for registrations. In fact,

based on the transaction processing rules in Section 17.1.3, the

client transaction will accept the first response, and view any

others as retransmissions because they all contain the same Via

branch identifier.

Before a request is sent, the client transport MUST insert a value of

the "sent-by" field into the Via header field. This field contains

an IP address or host name, and port. The usage of an FQDN is

RECOMMENDED. This field is used for sending responses under certain

conditions, described below. If the port is absent, the default

value depends on the transport. It is 5060 for UDP, TCP and SCTP,

5061 for TLS.

For reliable transports, the response is normally sent on the

connection on which the request was received. Therefore, the client

transport MUST be prepared to receive the response on the same

connection used to send the request. Under error conditions, the

server may attempt to open a new connection to send the response. To

handle this case, the transport layer MUST also be prepared to

receive an incoming connection on the source IP address from which

the request was sent and port number in the "sent-by" field. It also

Rosenberg, et. al. Standards Track [Page 143]

RFC 3261 SIP: Session Initiation Protocol June 2002

MUST be prepared to receive incoming connections on any address and

port that would be selected by a server based on the procedures

described in Section 5 of [4].

For unreliable unicast transports, the client transport MUST be

prepared to receive responses on the source IP address from which the

request is sent (as responses are sent back to the source address)

and the port number in the "sent-by" field. Furthermore, as with

reliable transports, in certain cases the response will be sent

elsewhere. The client MUST be prepared to receive responses on any

address and port that would be selected by a server based on the

procedures described in Section 5 of [4].

For multicast, the client transport MUST be prepared to receive

responses on the same multicast group and port to which the request

is sent (that is, it needs to be a member of the multicast group it

sent the request to.)

If a request is destined to an IP address, port, and transport to

which an existing connection is open, it is RECOMMENDED that this

connection be used to send the request, but another connection MAY be

opened and used.

If a request is sent using multicast, it is sent to the group

address, port, and TTL provided by the transport user. If a request

is sent using unicast unreliable transports, it is sent to the IP

address and port provided by the transport user.

18.1.2 Receiving Responses

When a response is received, the client transport examines the top

Via header field value. If the value of the "sent-by" parameter in

that header field value does not correspond to a value that the

client transport is configured to insert into requests, the response

MUST be silently discarded.

If there are any client transactions in existence, the client

transport uses the matching procedures of Section 17.1.3 to attempt

to match the response to an existing transaction. If there is a

match, the response MUST be passed to that transaction. Otherwise,

the response MUST be passed to the core (whether it be stateless

proxy, stateful proxy, or UA) for further processing. Handling of

these "stray" responses is dependent on the core (a proxy will

forward them, while a UA will discard, for example).

Rosenberg, et. al. Standards Track [Page 144]

RFC 3261 SIP: Session Initiation Protocol June 2002

18.2 Servers

18.2.1 Receiving Requests

A server SHOULD be prepared to receive requests on any IP address,

port and transport combination that can be the result of a DNS lookup

on a SIP or SIPS URI [4] that is handed out for the purposes of

communicating with that server. In this context, "handing out"

includes placing a URI in a Contact header field in a REGISTER

request or a redirect response, or in a Record-Route header field in

a request or response. A URI can also be "handed out" by placing it

on a web page or business card. It is also RECOMMENDED that a server

listen for requests on the default SIP ports (5060 for TCP and UDP,

5061 for TLS over TCP) on all public interfaces. The typical

exception would be private networks, or when multiple server

instances are running on the same host. For any port and interface

that a server listens on for UDP, it MUST listen on that same port

and interface for TCP. This is because a message may need to be sent

using TCP, rather than UDP, if it is too large. As a result, the

converse is not true. A server need not listen for UDP on a

particular address and port just because it is listening on that same

address and port for TCP. There may, of course, be other reasons why

a server needs to listen for UDP on a particular address and port.

When the server transport receives a request over any transport, it

MUST examine the value of the "sent-by" parameter in the top Via

header field value. If the host portion of the "sent-by" parameter

contains a domain name, or if it contains an IP address that differs

from the packet source address, the server MUST add a "received"

parameter to that Via header field value. This parameter MUST

contain the source address from which the packet was received. This

is to assist the server transport layer in sending the response,

since it must be sent to the source IP address from which the request

came.

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

in part:

INVITE sip:bob@Biloxi.com SIP/2.0

Via: SIP/2.0/UDP bobspc.biloxi.com:5060

The request is received with a source IP address of 192.0.2.4.

Before passing the request up, the transport adds a "received"

parameter, so that the request would look like, in part:

INVITE sip:bob@Biloxi.com SIP/2.0

Via: SIP/2.0/UDP bobspc.biloxi.com:5060;received=192.0.2.4

Rosenberg, et. al. Standards Track [Page 145]

RFC 3261 SIP: Session Initiation Protocol June 2002

Next, the server transport attempts to match the request to a server

transaction. It does so using the matching rules described in

Section 17.2.3. If a matching server transaction is found, the

request is passed to that transaction for processing. If no match is

found, the request is passed to the core, which may decide to

construct a new server transaction for that request. Note that when

a UAS core sends a 2xx response to INVITE, the server transaction is

destroyed. This means that when the ACK arrives, there will be no

matching server transaction, and based on this rule, the ACK is

passed to the UAS core, where it is processed.

18.2.2 Sending Responses

The server transport uses the value of the top Via header field in

order to determine where to send a response. It MUST follow the

following process:

o If the "sent-protocol" is a reliable transport protocol such as

TCP or SCTP, or TLS over those, the response MUST be sent using

the existing connection to the source of the original request

that created the transaction, if that connection is still open.

This requires the server transport to maintain an association

between server transactions and transport connections. If that

connection is no longer open, the server SHOULD open a

connection to the IP address in the "received" parameter, if

present, using the port in the "sent-by" value, or the default

port for that transport, if no port is specified. If that

connection attempt fails, the server SHOULD use the procedures

in [4] for servers in order to determine the IP address and

port to open the connection and send the response to.

o Otherwise, if the Via header field value contains a "maddr"

parameter, the response MUST be forwarded to the address listed

there, using the port indicated in "sent-by", or port 5060 if

none is present. If the address is a multicast address, the

response SHOULD be sent using the TTL indicated in the "ttl"

parameter, or with a TTL of 1 if that parameter is not present.

o Otherwise (for unreliable unicast transports), if the top Via

has a "received" parameter, the response MUST be sent to the

address in the "received" parameter, using the port indicated

in the "sent-by" value, or using port 5060 if none is specified

explicitly. If this fails, for example, elicits an ICMP "port

unreachable" response, the procedures of Section 5 of [4]

SHOULD be used to determine where to send the response.

Rosenberg, et. al. Standards Track [Page 146]

RFC 3261 SIP: Session Initiation Protocol June 2002

o Otherwise, if it is not receiver-tagged, the response MUST be

sent to the address indicated by the "sent-by" value, using the

procedures in Section 5 of [4].

18.3 Framing

In the case of message-oriented transports (such as UDP), if the

message has a Content-Length header field, the message body is

assumed to contain that many bytes. If there are additional bytes in

the transport packet beyond the end of the body, they MUST be

discarded. If the transport packet ends before the end of the

message body, this is considered an error. If the message is a

response, it MUST be discarded. If the message is a request, the

element SHOULD generate a 400 (Bad Request) response. If the message

has no Content-Length header field, the message body is assumed to

end at the end of the transport packet.

In the case of stream-oriented transports such as TCP, the Content-

Length header field indicates the size of the body. The Content-

Length header field MUST be used with stream oriented transports.

18.4 Error Handling

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

response.

If the transport user asks for a message to be sent over an

unreliable transport, and the result is an ICMP error, the behavior

depends on the type of ICMP error. Host, network, port or protocol

unreachable errors, or parameter problem errors SHOULD cause the

transport layer to inform the transport user of a failure in sending.

Source quench and TTL exceeded ICMP errors SHOULD be ignored.

If the transport user asks for a request to be sent over a reliable

transport, and the result is a connection failure, the transport

layer SHOULD inform the transport user of a failure in sending.

19 Common Message Components

There are certain components of SIP messages that appear in various

places within SIP messages (and sometimes, outside of them) that

merit separate discussion.

Rosenberg, et. al. Standards Track [Page 147]

RFC 3261 SIP: Session Initiation Protocol June 2002

19.1 SIP and SIPS Uniform Resource Indicators

A SIP or SIPS URI identifies a communications resource. Like all

URIs, SIP and SIPS URIs may be placed in web pages, email messages,

or printed literature. They contain sufficient information to

initiate and maintain a communication session with the resource.

Examples of communications resources include the following:

o a user of an online service

o an appearance on a multi-line phone

o a mailbox on a messaging system

o a PSTN number at a gateway service

o a group (such as "sales" or "helpdesk") in an organization

A SIPS URI specifies that the resource be contacted securely. This

means, in particular, that TLS is to be used between the UAC and the

domain that owns the URI. From there, secure communications are used

to reach the user, where the specific security mechanism depends on

the policy of the domain. Any resource described by a SIP URI can be

"upgraded" to a SIPS URI by just changing the scheme, if it is

desired to communicate with that resource securely.

19.1.1 SIP and SIPS URI Components

The "sip:" and "sips:" schemes follow the guidelines in RFC 2396 [5].

They use a form similar to the mailto URL, allowing the specification

of SIP request-header fields and the SIP message-body. This makes it

possible to specify the subject, media type, or urgency of sessions

initiated by using a URI on a web page or in an email message. The

formal syntax for a SIP or SIPS URI is presented in Section 25. Its

general form, in the case of a SIP URI, is:

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

The format for a SIPS URI is the same, except that the scheme is

"sips" instead of sip. These tokens, and some of the tokens in their

expansions, have the following meanings:

user: The identifier of a particular resource at the host being

addressed. The term "host" in this context frequently refers

to a domain. The "userinfo" of a URI consists of this user

field, the password field, and the @ sign following them. The

userinfo part of a URI is optional and MAY be absent when the

Rosenberg, et. al. Standards Track [Page 148]

RFC 3261 SIP: Session Initiation Protocol June 2002

destination host does not have a notion of users or when the

host itself is the resource being identified. If the @ sign is

present in a SIP or SIPS URI, the user field MUST NOT be empty.

If the host being addressed can process telephone numbers, for

instance, an Internet telephony gateway, a telephone-

subscriber field defined in RFC 2806 [9] MAY be used to

populate the user field. There are special escaping rules for

encoding telephone-subscriber fields in SIP and SIPS URIs

described in Section 19.1.2.

password: A password associated with the user. While the SIP and

SIPS URI syntax allows this field to be present, its use is NOT

RECOMMENDED, because the passing of authentication information

in clear text (such as URIs) has proven to be a security risk

in almost every case where it has been used. For instance,

transporting a PIN number in this field exposes the PIN.

Note that the password field is just an extension of the user

portion. Implementations not wishing to give special

significance to the password portion of the field MAY simply

treat "user:password" as a single string.

host: The host providing the SIP resource. The host part contains

either a fully-qualified domain name or numeric IPv4 or IPv6

address. Using the fully-qualified domain name form is

RECOMMENDED whenever possible.

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

URI parameters: Parameters affecting a request constructed from

the URI.

URI parameters are added after the hostport component and are

separated by semi-colons.

URI parameters take the form:

parameter-name "=" parameter-value

Even though an arbitrary number of URI parameters may be

included in a URI, any given parameter-name MUST NOT appear

more than once.

This extensible mechanism includes the transport, maddr, ttl,

user, method and lr parameters.

Rosenberg, et. al. Standards Track [Page 149]

RFC 3261 SIP: Session Initiation Protocol June 2002

The transport parameter determines the transport mechanism to

be used for sending SIP messages, as specified in [4]. SIP can

use any network transport protocol. Parameter names are

defined for UDP (RFC 768 [14]), TCP (RFC 761 [15]), and SCTP

(RFC 2960 [16]). For a SIPS URI, the transport parameter MUST

indicate a reliable transport.

The maddr parameter indicates the server address to be

contacted for this user, overriding any address derived from

the host field. When an maddr parameter is present, the port

and transport components of the URI apply to the address

indicated in the maddr parameter value. [4] describes the

proper interpretation of the transport, maddr, and hostport in

order to obtain the destination address, port, and transport

for sending a request.

The maddr field has been used as a simple form of loose source

routing. It allows a URI to specify a proxy that must be

traversed en-route to the destination. Continuing to use the

maddr parameter this way is strongly discouraged (the

mechanisms that enable it are deprecated). Implementations

should instead use the Route mechanism described in this

document, establishing a pre-existing route set if necessary

(see Section 8.1.1.1). This provides a full URI to describe

the node to be traversed.

The ttl parameter determines the time-to-live value of the UDP

multicast packet and MUST only be used if maddr is a multicast

address and the transport protocol is UDP. For example, to

specify a call to alice@atlanta.com using multicast to

239.255.255.1 with a ttl of 15, the following URI would be

used:

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

The set of valid telephone-subscriber strings is a subset of

valid user strings. The user URI parameter exists to

distinguish telephone numbers from user names that happen to

look like telephone numbers. If the user string contains a

telephone number formatted as a telephone-subscriber, the user

parameter value "phone" SHOULD be present. Even without this

parameter, recipients of SIP and SIPS URIs MAY interpret the

pre-@ part as a telephone number if local restrictions on the

name space for user name allow it.

The method of the SIP request constructed from the URI can be

specified with the method parameter.

Rosenberg, et. al. Standards Track [Page 150]

RFC 3261 SIP: Session Initiation Protocol June 2002

The lr parameter, when present, indicates that the element

responsible for this resource implements the routing mechanisms

specified in this document. This parameter will be used in the

URIs proxies place into Record-Route header field values, and

may appear in the URIs in a pre-existing route set.

This parameter is used to achieve backwards compatibility with

systems implementing the strict-routing mechanisms of RFC 2543

and the rfc2543bis drafts up to bis-05. An element preparing

to send a request based on a URI not containing this parameter

can assume the receiving element implements strict-routing and

reformat the message to preserve the information in the

Request-URI.

Since the uri-parameter mechanism is extensible, SIP elements

MUST silently ignore any uri-parameters that they do not

understand.

Headers: Header fields to be included in a request constructed

from the URI.

Headers fields in the SIP request can be specified with the "?"

mechanism within a URI. The header names and values are

encoded in ampersand separated hname = hvalue pairs. The

special hname "body" indicates that the associated hvalue is

the message-body of the SIP request.

Table 1 summarizes the use of SIP and SIPS URI components based on

the context in which the URI appears. The external column describes

URIs appearing anywhere outside of a SIP message, for instance on a

web page or business card. Entries marked "m" are mandatory, those

marked "o" are optional, and those marked "-" are not allowed.

Elements processing URIs SHOULD ignore any disallowed components if

they are present. The second column indicates the default value of

an optional element if it is not present. "--" indicates that the

element is either not optional, or has no default value.

URIs in Contact header fields have different restrictions depending

on the context in which the header field appears. One set applies to

messages that establish and maintain dialogs (INVITE and its 200 (OK)

response). The other applies to registration and redirection

messages (REGISTER, its 200 (OK) response, and 3xx class responses to

any method).

Rosenberg, et. al. Standards Track [Page 151]

RFC 3261 SIP: Session Initiation Protocol June 2002

19.1.2 Character Escaping Requirements

dialog

reg./redir. Contact/

default Req.-URI To From Contact R-R/Route external

user -- o o o o o o

password -- o o o o o o

host -- m m m m m m

port (1) o - - o o o

user-param ip o o o o o o

method INVITE - - - - - o

maddr-param -- o - - o o o

ttl-param 1 o - - o - o

transp.-param (2) o - - o o o

lr-param -- o - - - o o

other-param -- o o o o o o

headers -- - - - o - o

(1): The default port value is transport and scheme dependent. The

default is 5060 for sip: using UDP, TCP, or SCTP. The default is

5061 for sip: using TLS over TCP and sips: over TCP.

(2): The default transport is scheme dependent. For sip:, it is UDP.

For sips:, it is TCP.

Table 1: Use and default values of URI components for SIP header

field values, Request-URI and references

SIP follows the requirements and guidelines of RFC 2396 [5] when

defining the set of characters that must be escaped in a SIP URI, and

uses its ""%" HEX HEX" mechanism for escaping. From RFC 2396 [5]:

The set of characters actually reserved within any given URI

component is defined by that component. In general, a character

is reserved if the semantics of the URI changes if the character

is replaced with its escaped US-ASCII encoding [5]. Excluded US-

ASCII characters (RFC 2396 [5]), such as space and control

characters and characters used as URI delimiters, also MUST be

escaped. URIs MUST NOT contain unescaped space and control

characters.

For each component, the set of valid BNF expansions defines exactly

which characters may appear unescaped. All other characters MUST be

escaped.

For example, "@" is not in the set of characters in the user

component, so the user "j@s0n" must have at least the @ sign encoded,

as in "j%40s0n".

Rosenberg, et. al. Standards Track [Page 152]

RFC 3261 SIP: Session Initiation Protocol June 2002

Expanding the hname and hvalue tokens in Section 25 show that all URI

reserved characters in header field names and values MUST be escaped.

The telephone-subscriber subset of the user component has special

escaping considerations. The set of characters not reserved in the

RFC 2806 [9] description of telephone-subscriber contains a number of

characters in various syntax elements that need to be escaped when

used in SIP URIs. Any characters occurring in a telephone-subscriber

that do not appear in an expansion of the BNF for the user rule MUST

be escaped.

Note that character escaping is not allowed in the host component of

a SIP or SIPS URI (the % character is not valid in its expansion).

This is likely to change in the future as requirements for

Internationalized Domain Names are finalized. Current

implementations MUST NOT attempt to improve robustness by treating

received escaped characters in the host component as literally

equivalent to their unescaped counterpart. The behavior required to

meet the requirements of IDN may be significantly different.

19.1.3 Example SIP and SIPS URIs

sip:alice@atlanta.com

sip:alice:secretword@atlanta.com;transport=tcp

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

sip:+1-212-555-1212:1234@gateway.com;user=phone

sips:1212@gateway.com

sip:alice@192.0.2.4

sip:atlanta.com;method=REGISTER?to=alice%40atlanta.com

sip:alice;day=tuesday@atlanta.com

The last sample URI above has a user field value of

"alice;day=tuesday". The escaping rules defined above allow a

semicolon to appear unescaped in this field. For the purposes of

this protocol, the field is opaque. The structure of that value is

only useful to the SIP element responsible for the resource.

19.1.4 URI Comparison

Some operations in this specification require determining whether two

SIP or SIPS URIs are equivalent. In this specification, registrars

need to compare bindings in Contact URIs in REGISTER requests (see

Section 10.3.). SIP and SIPS URIs are compared for equality

according to the following rules:

o A SIP and SIPS URI are never equivalent.

Rosenberg, et. al. Standards Track [Page 153]

RFC 3261 SIP: Session Initiation Protocol June 2002

o Comparison of the userinfo of SIP and SIPS URIs is case-

sensitive. This includes userinfo containing passwords or

formatted as telephone-subscribers. Comparison of all other

components of the URI is case-insensitive unless explicitly

defined otherwise.

o The ordering of parameters and header fields is not significant

in comparing SIP and SIPS URIs.

o Characters other than those in the "reserved" set (see RFC 2396

[5]) are equivalent to their ""%" HEX HEX" encoding.

o An IP address that is the result of a DNS lookup of a host name

does not match that host name.

o For two URIs to be equal, the user, password, host, and port

components must match.

A URI omitting the user component will not match a URI that

includes one. A URI omitting the password component will not

match a URI that includes one.

A URI omitting any component with a default value will not

match a URI explicitly containing that component with its

default value. For instance, a URI omitting the optional port

component will not match a URI explicitly declaring port 5060.

The same is true for the transport-parameter, ttl-parameter,

user-parameter, and method components.

Defining sip:user@host to not be equivalent to

sip:user@host:5060 is a change from RFC 2543. When deriving

addresses from URIs, equivalent addresses are expected from

equivalent URIs. The URI sip:user@host:5060 will always

resolve to port 5060. The URI sip:user@host may resolve to

other ports through the DNS SRV mechanisms detailed in [4].

o URI uri-parameter components are compared as follows:

- Any uri-parameter appearing in both URIs must match.

- A user, ttl, or method uri-parameter appearing in only one

URI never matches, even if it contains the default value.

- A URI that includes an maddr parameter will not match a URI

that contains no maddr parameter.

- All other uri-parameters appearing in only one URI are

ignored when comparing the URIs.

Rosenberg, et. al. Standards Track [Page 154]

RFC 3261 SIP: Session Initiation Protocol June 2002

o URI header components are never ignored. Any present header

component MUST be present in both URIs and match for the URIs

to match. The matching rules are defined for each header field

in Section 20.

The URIs within each of the following sets are equivalent:

sip:%61lice@atlanta.com;transport=TCP

sip:alice@AtLanTa.CoM;Transport=tcp

sip:carol@chicago.com

sip:carol@chicago.com;newparam=5

sip:carol@chicago.com;security=on

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

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

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

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

The URIs within each of the following sets are not equivalent:

SIP:ALICE@AtLanTa.CoM;Transport=udp (different usernames)

sip:alice@AtLanTa.CoM;Transport=UDP

sip:bob@biloxi.com (can resolve to different ports)

sip:bob@biloxi.com:5060

sip:bob@biloxi.com (can resolve to different transports)

sip:bob@biloxi.com;transport=udp

sip:bob@biloxi.com (can resolve to different port and transports)

sip:bob@biloxi.com:6000;transport=tcp

sip:carol@chicago.com (different header component)

sip:carol@chicago.com?Subject=next%20meeting

sip:bob@phone21.boxesbybob.com (even though that's what

sip:bob@192.0.2.4 phone21.boxesbybob.com resolves to)

Note that equality is not transitive:

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

equivalent

o sip:carol@chicago.com and sip:carol@chicago.com;security=off

are equivalent

Rosenberg, et. al. Standards Track [Page 155]

RFC 3261 SIP: Session Initiation Protocol June 2002

o sip:carol@chicago.com;security=on and

sip:carol@chicago.com;security=off are not equivalent

19.1.5 Forming Requests from a URI

An implementation needs to take care when forming requests directly

from a URI. URIs from business cards, web pages, and even from

sources inside the protocol such as registered contacts may contain

inappropriate header fields or body parts.

An implementation MUST include any provided transport, maddr, ttl, or

user parameter in the Request-URI of the formed request. If the URI

contains a method parameter, its value MUST be used as the method of

the request. The method parameter MUST NOT be placed in the

Request-URI. Unknown URI parameters MUST be placed in the message's

Request-URI.

An implementation SHOULD treat the presence of any headers or body

parts in the URI as a desire to include them in the message, and

choose to honor the request on a per-component basis.

An implementation SHOULD NOT honor these obviously dangerous header

fields: From, Call-ID, CSeq, Via, and Record-Route.

An implementation SHOULD NOT honor any requested Route header field

values in order to not be used as an unwitting agent in malicious

attacks.

An implementation SHOULD NOT honor requests to include header fields

that may cause it to falsely advertise its location or capabilities.

These include: Accept, Accept-Encoding, Accept-Language, Allow,

Contact (in its dialog usage), Organization, Supported, and User-

Agent.

An implementation SHOULD verify the accuracy of any requested

descriptive header fields, including: Content-Disposition, Content-

Encoding, Content-Language, Content-Length, Content-Type, Date,

Mime-Version, and Timestamp.

If the request formed from constructing a message from a given URI is

not a valid SIP request, the URI is invalid. An implementation MUST

NOT proceed with transmitting the request. It should instead pursue

the course of action due an invalid URI in the context it occurs.

The constructed request can be invalid in many ways. These

include, but are not limited to, syntax error in header fields,

invalid combinations of URI parameters, or an incorrect

description of the message body.

Rosenberg, et. al. Standards Track [Page 156]

RFC 3261 SIP: Session Initiation Protocol June 2002

Sending a request formed from a given URI may require capabilities

unavailable to the implementation. The URI might indicate use of an

unimplemented transport or extension, for example. An implementation

SHOULD refuse to send these requests rather than modifying them to

match their capabilities. An implementation MUST NOT send a request

requiring an extension that it does not support.

For example, such a request can be formed through the presence of

a Require header parameter or a method URI parameter with an

unknown or explicitly unsupported value.

19.1.6 Relating SIP URIs and tel URLs

When a tel URL (RFC 2806 [9]) is converted to a SIP or SIPS URI, the

entire telephone-subscriber portion of the tel URL, including any

parameters, is placed into the userinfo part of the SIP or SIPS URI.

Thus, tel:+358-555-1234567;postd=pp22 becomes

sip:+358-555-1234567;postd=pp22@foo.com;user=phone

or

sips:+358-555-1234567;postd=pp22@foo.com;user=phone

not

sip:+358-555-1234567@foo.com;postd=pp22;user=phone

or

sips:+358-555-1234567@foo.com;postd=pp22;user=phone

In general, equivalent "tel" URLs converted to SIP or SIPS URIs in

this fashion may not produce equivalent SIP or SIPS URIs. The

userinfo of SIP and SIPS URIs are compared as a case-sensitive

string. Variance in case-insensitive portions of tel URLs and

reordering of tel URL parameters does not affect tel URL equivalence,

but does affect the equivalence of SIP URIs formed from them.

For example,

tel:+358-555-1234567;postd=pp22

tel:+358-555-1234567;POSTD=PP22

are equivalent, while

sip:+358-555-1234567;postd=pp22@foo.com;user=phone

sip:+358-555-1234567;POSTD=PP22@foo.com;user=phone

Rosenberg, et. al. Standards Track [Page 157]

RFC 3261 SIP: Session Initiation Protocol June 2002

are not.

Likewise,

tel:+358-555-1234567;postd=pp22;isub=1411

tel:+358-555-1234567;isub=1411;postd=pp22

are equivalent, while

sip:+358-555-1234567;postd=pp22;isub=1411@foo.com;user=phone

sip:+358-555-1234567;isub=1411;postd=pp22@foo.com;user=phone

are not.

To mitigate this problem, elements constructing telephone-subscriber

fields to place in the userinfo part of a SIP or SIPS URI SHOULD fold

any case-insensitive portion of telephone-subscriber to lower case,

and order the telephone-subscriber parameters lexically by parameter

name, excepting isdn-subaddress and post-dial, which occur first and

in that order. (All components of a tel URL except for future-

extension parameters are defined to be compared case-insensitive.)

Following this suggestion, both

tel:+358-555-1234567;postd=pp22

tel:+358-555-1234567;POSTD=PP22

become

sip:+358-555-1234567;postd=pp22@foo.com;user=phone

and both

tel:+358-555-1234567;tsp=a.b;phone-context=5

tel:+358-555-1234567;phone-context=5;tsp=a.b

become

sip:+358-555-1234567;phone-context=5;tsp=a.b@foo.com;user=phone

19.2 Option Tags

Option tags are unique identifiers used to designate new options

(extensions) in SIP. These tags are used in Require (Section 20.32),

Proxy-Require (Section 20.29), Supported (Section 20.37) and

Unsupported (Section 20.40) header fields. Note that these options

appear as parameters in those header fields in an option-tag = token

form (see Section 25 for the definition of token).

Rosenberg, et. al. Standards Track [Page 158]

RFC 3261 SIP: Session Initiation Protocol June 2002

Option tags are defined in standards track RFCs. This is a change

from past practice, and is instituted to ensure continuing multi-

vendor interoperability (see discussion in Section 20.32 and Section

20.37). An IANA registry of option tags is used to ensure easy

reference.

19.3 Tags

The "tag" parameter is used in the To and From header fields of SIP

messages. It serves as a general mechanism to identify a dialog,

which is the combination of the Call-ID along with two tags, one from

each participant in the dialog. When a UA sends a request outside of

a dialog, it contains a From tag only, providing "half" of the dialog

ID. The dialog is completed from the response(s), each of which

contributes the second half in the To header field. The forking of

SIP requests means that multiple dialogs can be established from a

single request. This also explains the need for the two-sided dialog

identifier; without a contribution from the recipients, the

originator could not disambiguate the multiple dialogs established

from a single request.

When a tag is generated by a UA for insertion into a request or

response, it MUST be globally unique and cryptographically random

with at least 32 bits of randomness. A property of this selection

requirement is that a UA will place a different tag into the From

header of an INVITE than it would place into the To header of the

response to the same INVITE. This is needed in order for a UA to

invite itself to a session, a common case for "hairpinning" of calls

in PSTN gateways. Similarly, two INVITEs for different calls will

have different From tags, and two responses for different calls will

have different To tags.

Besides the requirement for global uniqueness, the algorithm for

generating a tag is implementation-specific. Tags are helpful in

fault tolerant systems, where a dialog is to be recovered on an

alternate server after a failure. A UAS can select the tag in such a

way that a backup can recognize a request as part of a dialog on the

failed server, and therefore determine that it should attempt to

recover the dialog and any other state associated with it.

20 Header Fields

The general syntax for header fields is covered in Section 7.3. This

section lists the full set of header fields along with notes on

syntax, meaning, and usage. Throughout this section, we use [HX.Y]

to refer to Section X.Y of the current HTTP/1.1 specification RFC

2616 [8]. Examples of each header field are given.

Rosenberg, et. al. Standards Track [Page 159]

RFC 3261 SIP: Session Initiation Protocol June 2002

Information about header fields in relation to methods and proxy

processing is summarized in Tables 2 and 3.

The "where" column describes the request and response types in which

the header field can be used. Values in this column are:

R: header field may only appear in requests;

r: header field may only appear in responses;

2xx, 4xx, etc.: A numerical value or range indicates response

codes with which the header field can be used;

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

An empty entry in the "where" column indicates that the header

field may be present in all requests and responses.

The "proxy" column describes the operations a proxy may perform on a

header field:

a: A proxy can add or concatenate the header field if not present.

m: A proxy can modify an existing header field value.

d: A proxy can delete a header field value.

r: A proxy must be able to read the header field, and thus this

header field cannot be encrypted.

The next six columns relate to the presence of a header field in a

method:

c: Conditional; requirements on the header field depend on the

context of the message.

m: The header field is mandatory.

m\*: The header field SHOULD be sent, but clients/servers need to

be prepared to receive messages without that header field.

o: The header field is optional.

t: The header field SHOULD be sent, but clients/servers need to be

prepared to receive messages without that header field.

If a stream-based protocol (such as TCP) is used as a

transport, then the header field MUST be sent.

Rosenberg, et. al. Standards Track [Page 160]

RFC 3261 SIP: Session Initiation Protocol June 2002

\*: The header field is required if the message body is not empty.

See Sections 20.14, 20.15 and 7.4 for details.

-: The header field is not applicable.

"Optional" means that an element MAY include the header field in a

request or response, and a UA MAY ignore the header field if present

in the request or response (The exception to this rule is the Require

header field discussed in 20.32). A "mandatory" header field MUST be

present in a request, and MUST be understood by the UAS receiving the

request. A mandatory response header field MUST be present in the

response, and the header field MUST be understood by the UAC

processing the response. "Not applicable" means that the header

field MUST NOT be present in a request. If one is placed in a

request by mistake, it MUST be ignored by the UAS receiving the

request. Similarly, a header field labeled "not applicable" for a

response means that the UAS MUST NOT place the header field in the

response, and the UAC MUST ignore the header field in the response.

A UA SHOULD ignore extension header parameters that are not

understood.

A compact form of some common header field names is also defined for

use when overall message size is an issue.

The Contact, From, and To header fields contain a URI. If the URI

contains a comma, question mark or semicolon, the URI MUST be

enclosed in angle brackets (< and >). Any URI parameters are

contained within these brackets. If the URI is not enclosed in angle

brackets, any semicolon-delimited parameters are header-parameters,

not URI parameters.

20.1 Accept

The Accept header field follows the syntax defined in [H14.1]. The

semantics are also identical, with the exception that if no Accept

header field is present, the server SHOULD assume a default value of

application/sdp.

An empty Accept header field means that no formats are acceptable.

Rosenberg, et. al. Standards Track [Page 161]

RFC 3261 SIP: Session Initiation Protocol June 2002

Example:

Header field where proxy ACK BYE CAN INV OPT REG

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Accept R - o - o m\* o

Accept 2xx - - - o m\* o

Accept 415 - c - c c c

Accept-Encoding R - o - o o o

Accept-Encoding 2xx - - - o m\* o

Accept-Encoding 415 - c - c c c

Accept-Language R - o - o o o

Accept-Language 2xx - - - o m\* o

Accept-Language 415 - c - c c c

Alert-Info R ar - - - o - -

Alert-Info 180 ar - - - o - -

Allow R - o - o o o

Allow 2xx - o - m\* m\* o

Allow r - o - o o o

Allow 405 - m - m m m

Authentication-Info 2xx - o - o o o

Authorization R o o o o o o

Call-ID c r m m m m m m

Call-Info ar - - - o o o

Contact R o - - m o o

Contact 1xx - - - o - -

Contact 2xx - - - m o o

Contact 3xx d - o - o o o

Contact 485 - o - o o o

Content-Disposition o o - o o o

Content-Encoding o o - o o o

Content-Language o o - o o o

Content-Length ar t t t t t t

Content-Type \* \* - \* \* \*

CSeq c r m m m m m m

Date a o o o o o o

Error-Info 300-699 a - o o o o o

Expires - - - o - o

From c r m m m m m m

In-Reply-To R - - - o - -

Max-Forwards R amr m m m m m m

Min-Expires 423 - - - - - m

MIME-Version o o - o o o

Organization ar - - - o o o

Table 2: Summary of header fields, A--O

Rosenberg, et. al. Standards Track [Page 162]

RFC 3261 SIP: Session Initiation Protocol June 2002

Header field where proxy ACK BYE CAN INV OPT REG

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Priority R ar - - - o - -

Proxy-Authenticate 407 ar - m - m m m

Proxy-Authenticate 401 ar - o o o o o

Proxy-Authorization R dr o o - o o o

Proxy-Require R ar - o - o o o

Record-Route R ar o o o o o -

Record-Route 2xx,18x mr - o o o o -

Reply-To - - - o - -

Require ar - c - c c c

Retry-After 404,413,480,486 - o o o o o

500,503 - o o o o o

600,603 - o o o o o

Route R adr c c c c c c

Server r - o o o o o

Subject R - - - o - -

Supported R - o o m\* o o

Supported 2xx - o o m\* m\* o

Timestamp o o o o o o

To c(1) r m m m m m m

Unsupported 420 - m - m m m

User-Agent o o o o o o

Via R amr m m m m m m

Via rc dr m m m m m m

Warning r - o o o o o

WWW-Authenticate 401 ar - m - m m m

WWW-Authenticate 407 ar - o - o o o

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

addition of tag

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

20.2 Accept-Encoding

The Accept-Encoding header field is similar to Accept, but restricts

the content-codings [H3.5] that are acceptable in the response. See

[H14.3]. The semantics in SIP are identical to those defined in

[H14.3].

An empty Accept-Encoding header field is permissible. It is

equivalent to Accept-Encoding: identity, that is, only the identity

encoding, meaning no encoding, is permissible.

If no Accept-Encoding header field is present, the server SHOULD

assume a default value of identity.

Rosenberg, et. al. Standards Track [Page 163]

RFC 3261 SIP: Session Initiation Protocol June 2002

This differs slightly from the HTTP definition, which indicates that

when not present, any encoding can be used, but the identity encoding

is preferred.

Example:

Accept-Encoding: gzip

20.3 Accept-Language

The Accept-Language header field is used in requests to indicate the

preferred languages for reason phrases, session descriptions, or

status responses carried as message bodies in the response. If no

Accept-Language header field is present, the server SHOULD assume all

languages are acceptable to the client.

The Accept-Language header field follows the syntax defined in

[H14.4]. The rules for ordering the languages based on the "q"

parameter apply to SIP as well.

Example:

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

20.4 Alert-Info

When present in an INVITE request, the Alert-Info header field

specifies an alternative ring tone to the UAS. When present in a 180

(Ringing) response, the Alert-Info header field specifies an

alternative ringback tone to the UAC. A typical usage is for a proxy

to insert this header field to provide a distinctive ring feature.

The Alert-Info header field can introduce security risks. These

risks and the ways to handle them are discussed in Section 20.9,

which discusses the Call-Info header field since the risks are

identical.

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

selectively.

This helps prevent disruptions that could result from the use of

this header field by untrusted elements.

Example:

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

Rosenberg, et. al. Standards Track [Page 164]

RFC 3261 SIP: Session Initiation Protocol June 2002

20.5 Allow

The Allow header field lists the set of methods supported by the UA

generating the message.

All methods, including ACK and CANCEL, understood by the UA MUST be

included in the list of methods in the Allow header field, when

present. The absence of an Allow header field MUST NOT be

interpreted to mean that the UA sending the message supports no

methods. Rather, it implies that the UA is not providing any

information on what methods it supports.

Supplying an Allow header field in responses to methods other than

OPTIONS reduces the number of messages needed.

Example:

Allow: INVITE, ACK, OPTIONS, CANCEL, BYE

20.6 Authentication-Info

The Authentication-Info header field provides for mutual

authentication with HTTP Digest. A UAS MAY include this header field

in a 2xx response to a request that was successfully authenticated

using digest based on the Authorization header field.

Syntax and semantics follow those specified in RFC 2617 [17].

Example:

Authentication-Info: nextnonce="47364c23432d2e131a5fb210812c"

20.7 Authorization

The Authorization header field contains authentication credentials of

a UA. Section 22.2 overviews the use of the Authorization header

field, and Section 22.4 describes the syntax and semantics when used

with HTTP authentication.

This header field, along with Proxy-Authorization, breaks the general

rules about multiple header field values. Although not a comma-

separated list, this header field name may be present multiple times,

and MUST NOT be combined into a single header line using the usual

rules described in Section 7.3.

Rosenberg, et. al. Standards Track [Page 165]

RFC 3261 SIP: Session Initiation Protocol June 2002

In the example below, there are no quotes around the Digest

parameter:

Authorization: Digest username="Alice", realm="atlanta.com",

nonce="84a4cc6f3082121f32b42a2187831a9e",

response="7587245234b3434cc3412213e5f113a5432"

20.8 Call-ID

The Call-ID header field uniquely identifies a particular invitation

or all registrations of a particular client. A single multimedia

conference can give rise to several calls with different Call-IDs,

for example, if a user invites a single individual several times to

the same (long-running) conference. Call-IDs are case-sensitive and

are simply compared byte-by-byte.

The compact form of the Call-ID header field is i.

Examples:

Call-ID: f81d4fae-7dec-11d0-a765-00a0c91e6bf6@biloxi.com

i:f81d4fae-7dec-11d0-a765-00a0c91e6bf6@192.0.2.4

20.9 Call-Info

The Call-Info header field provides additional information about the

caller or callee, depending on whether it is found in a request or

response. The purpose of the URI is described by the "purpose"

parameter. The "icon" parameter designates an image suitable as an

iconic representation of the caller or callee. The "info" parameter

describes the caller or callee in general, for example, through a web

page. The "card" parameter provides a business card, for example, in

vCard [36] or LDIF [37] formats. Additional tokens can be registered

using IANA and the procedures in Section 27.

Use of the Call-Info header field can pose a security risk. If a

callee fetches the URIs provided by a malicious caller, the callee

may be at risk for displaying inappropriate or offensive content,

dangerous or illegal content, and so on. Therefore, it is

RECOMMENDED that a UA only render the information in the Call-Info

header field if it can verify the authenticity of the element that

originated the header field and trusts that element. This need not

be the peer UA; a proxy can insert this header field into requests.

Example:

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

<http://www.example.com/alice/> ;purpose=info

Rosenberg, et. al. Standards Track [Page 166]

RFC 3261 SIP: Session Initiation Protocol June 2002

20.10 Contact

A Contact header field value provides a URI whose meaning depends on

the type of request or response it is in.

A Contact header field value can contain a display name, a URI with

URI parameters, and header parameters.

This document defines the Contact parameters "q" and "expires".

These parameters are only used when the Contact is present in a

REGISTER request or response, or in a 3xx response. Additional

parameters may be defined in other specifications.

When the header field value contains a display name, the URI

including all URI parameters is enclosed in "<" and ">". If no "<"

and ">" are present, all parameters after the URI are header

parameters, not URI parameters. The display name can be tokens, or a

quoted string, if a larger character set is desired.

Even if the "display-name" is empty, the "name-addr" form MUST be

used if the "addr-spec" contains a comma, semicolon, or question

mark. There may or may not be LWS between the display-name and the

"<".

These rules for parsing a display name, URI and URI parameters, and

header parameters also apply for the header fields To and From.

The Contact header field has a role similar to the Location header

field in HTTP. However, the HTTP header field only allows one

address, unquoted. Since URIs can contain commas and semicolons

as reserved characters, they can be mistaken for header or

parameter delimiters, respectively.

The compact form of the Contact header field is m (for "moved").

Examples:

Contact: "Mr. Watson" <sip:watson@worcester.bell-telephone.com>

;q=0.7; expires=3600,

"Mr. Watson" <mailto:watson@bell-telephone.com> ;q=0.1

m: <sips:bob@192.0.2.4>;expires=60

Rosenberg, et. al. Standards Track [Page 167]

RFC 3261 SIP: Session Initiation Protocol June 2002

20.11 Content-Disposition

The Content-Disposition header field describes how the message body

or, for multipart messages, a message body part is to be interpreted

by the UAC or UAS. This SIP header field extends the MIME Content-

Type (RFC 2183 [18]).

Several new "disposition-types" of the Content-Disposition header are

defined by SIP. The value "session" indicates that the body part

describes a session, for either calls or early (pre-call) media. The

value "render" indicates that the body part should be displayed or

otherwise rendered to the user. Note that the value "render" is used

rather than "inline" to avoid the connotation that the MIME body is

displayed as a part of the rendering of the entire message (since the

MIME bodies of SIP messages oftentimes are not displayed to users).

For backward-compatibility, if the Content-Disposition header field

is missing, the server SHOULD assume bodies of Content-Type

application/sdp are the disposition "session", while other content

types are "render".

The disposition type "icon" indicates that the body part contains an

image suitable as an iconic representation of the caller or callee

that could be rendered informationally by a user agent when a message

has been received, or persistently while a dialog takes place. The

value "alert" indicates that the body part contains information, such

as an audio clip, that should be rendered by the user agent in an

attempt to alert the user to the receipt of a request, generally a

request that initiates a dialog; this alerting body could for example

be rendered as a ring tone for a phone call after a 180 Ringing

provisional response has been sent.

Any MIME body with a "disposition-type" that renders content to the

user should only be processed when a message has been properly

authenticated.

The handling parameter, handling-param, describes how the UAS should

react if it receives a message body whose content type or disposition

type it does not understand. The parameter has defined values of

"optional" and "required". If the handling parameter is missing, the

value "required" SHOULD be assumed. The handling parameter is

described in RFC 3204 [19].

If this header field is missing, the MIME type determines the default

content disposition. If there is none, "render" is assumed.

Example:

Content-Disposition: session

Rosenberg, et. al. Standards Track [Page 168]

RFC 3261 SIP: Session Initiation Protocol June 2002

20.12 Content-Encoding

The Content-Encoding header field is used as a modifier to the

"media-type". When present, its value indicates what additional

content codings have been applied to the entity-body, and thus what

decoding mechanisms MUST be applied in order to obtain the media-type

referenced by the Content-Type header field. Content-Encoding is

primarily used to allow a body to be compressed without losing the

identity of its underlying media type.

If multiple encodings have been applied to an entity-body, the

content codings MUST be listed in the order in which they were

applied.

All content-coding values are case-insensitive. IANA acts as a

registry for content-coding value tokens. See [H3.5] for a

definition of the syntax for content-coding.

Clients MAY apply content encodings to the body in requests. A

server MAY apply content encodings to the bodies in responses. The

server MUST only use encodings listed in the Accept-Encoding header

field in the request.

The compact form of the Content-Encoding header field is e.

Examples:

Content-Encoding: gzip

e: tar

20.13 Content-Language

See [H14.12]. Example:

Content-Language: fr

20.14 Content-Length

The Content-Length header field indicates the size of the message-

body, in decimal number of octets, sent to the recipient.

Applications SHOULD use this field to indicate the size of the

message-body to be transferred, regardless of the media type of the

entity. If a stream-based protocol (such as TCP) is used as

transport, the header field MUST be used.

The size of the message-body does not include the CRLF separating

header fields and body. Any Content-Length greater than or equal to

zero is a valid value. If no body is present in a message, then the

Content-Length header field value MUST be set to zero.

Rosenberg, et. al. Standards Track [Page 169]

RFC 3261 SIP: Session Initiation Protocol June 2002

The ability to omit Content-Length simplifies the creation of

cgi-like scripts that dynamically generate responses.

The compact form of the header field is l.

Examples:

Content-Length: 349

l: 173

20.15 Content-Type

The Content-Type header field indicates the media type of the

message-body sent to the recipient. The "media-type" element is

defined in [H3.7]. The Content-Type header field MUST be present if

the body is not empty. If the body is empty, and a Content-Type

header field is present, it indicates that the body of the specific

type has zero length (for example, an empty audio file).

The compact form of the header field is c.

Examples:

Content-Type: application/sdp

c: text/html; charset=ISO-8859-4

20.16 CSeq

A CSeq header field in a request contains a single decimal sequence

number and the request method. The sequence number MUST be

expressible as a 32-bit unsigned integer. The method part of CSeq is

case-sensitive. The CSeq header field serves to order transactions

within a dialog, to provide a means to uniquely identify

transactions, and to differentiate between new requests and request

retransmissions. Two CSeq header fields are considered equal if the

sequence number and the request method are identical. Example:

CSeq: 4711 INVITE

20.17 Date

The Date header field contains the date and time. Unlike HTTP/1.1,

SIP only supports the most recent RFC 1123 [20] format for dates. As

in [H3.3], SIP restricts the time zone in SIP-date to "GMT", while

RFC 1123 allows any time zone. An RFC 1123 date is case-sensitive.

The Date header field reflects the time when the request or response

is first sent.

Rosenberg, et. al. Standards Track [Page 170]

RFC 3261 SIP: Session Initiation Protocol June 2002

The Date header field can be used by simple end systems without a

battery-backed clock to acquire a notion of current time.

However, in its GMT form, it requires clients to know their offset

from GMT.

Example:

Date: Sat, 13 Nov 2010 23:29:00 GMT

20.18 Error-Info

The Error-Info header field provides a pointer to additional

information about the error status response.

SIP UACs have user interface capabilities ranging from pop-up

windows and audio on PC softclients to audio-only on "black"

phones or endpoints connected via gateways. Rather than forcing a

server generating an error to choose between sending an error

status code with a detailed reason phrase and playing an audio

recording, the Error-Info header field allows both to be sent.

The UAC then has the choice of which error indicator to render to

the caller.

A UAC MAY treat a SIP or SIPS URI in an Error-Info header field as if

it were a Contact in a redirect and generate a new INVITE, resulting

in a recorded announcement session being established. A non-SIP URI

MAY be rendered to the user.

Examples:

SIP/2.0 404 The number you have dialed is not in service

Error-Info: <sip:not-in-service-recording@atlanta.com>

20.19 Expires

The Expires header field gives the relative time after which the

message (or content) expires.

The precise meaning of this is method dependent.

The expiration time in an INVITE does not affect the duration of the

actual session that may result from the invitation. Session

description protocols may offer the ability to express time limits on

the session duration, however.

The value of this field is an integral number of seconds (in decimal)

between 0 and (2\*\*32)-1, measured from the receipt of the request.

Rosenberg, et. al. Standards Track [Page 171]

RFC 3261 SIP: Session Initiation Protocol June 2002

Example:

Expires: 5

20.20 From

The From header field indicates the initiator of the request. This

may be different from the initiator of the dialog. Requests sent by

the callee to the caller use the callee's address in the From header

field.

The optional "display-name" is meant to be rendered by a human user

interface. A system SHOULD use the display name "Anonymous" if the

identity of the client is to remain hidden. Even if the "display-

name" is empty, the "name-addr" form MUST be used if the "addr-spec"

contains a comma, question mark, or semicolon. Syntax issues are

discussed in Section 7.3.1.

Two From header fields are equivalent if their URIs match, and their

parameters match. Extension parameters in one header field, not

present in the other are ignored for the purposes of comparison. This

means that the display name and presence or absence of angle brackets

do not affect matching.

See Section 20.10 for the rules for parsing a display name, URI and

URI parameters, and header field parameters.

The compact form of the From header field is f.

Examples:

From: "A. G. Bell" <sip:agb@bell-telephone.com> ;tag=a48s

From: sip:+12125551212@server.phone2net.com;tag=887s

f: Anonymous <sip:c8oqz84zk7z@privacy.org>;tag=hyh8

20.21 In-Reply-To

The In-Reply-To header field enumerates the Call-IDs that this call

references or returns. These Call-IDs may have been cached by the

client then included in this header field in a return call.

This allows automatic call distribution systems to route return

calls to the originator of the first call. This also allows

callees to filter calls, so that only return calls for calls they

originated will be accepted. This field is not a substitute for

request authentication.

Rosenberg, et. al. Standards Track [Page 172]

RFC 3261 SIP: Session Initiation Protocol June 2002

Example:

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

20.22 Max-Forwards

The Max-Forwards header field must be used with any SIP method to

limit the number of proxies or gateways that can forward the request

to the next downstream server. This can also be useful when the

client is attempting to trace a request chain that appears to be

failing or looping in mid-chain.

The Max-Forwards value is an integer in the range 0-255 indicating

the remaining number of times this request message is allowed to be

forwarded. This count is decremented by each server that forwards

the request. The recommended initial value is 70.

This header field should be inserted by elements that can not

otherwise guarantee loop detection. For example, a B2BUA should

insert a Max-Forwards header field.

Example:

Max-Forwards: 6

20.23 Min-Expires

The Min-Expires header field conveys the minimum refresh interval

supported for soft-state elements managed by that server. This

includes Contact header fields that are stored by a registrar. The

header field contains a decimal integer number of seconds from 0 to

(2\*\*32)-1. The use of the header field in a 423 (Interval Too Brief)

response is described in Sections 10.2.8, 10.3, and 21.4.17.

Example:

Min-Expires: 60

20.24 MIME-Version

See [H19.4.1].

Example:

MIME-Version: 1.0

Rosenberg, et. al. Standards Track [Page 173]

RFC 3261 SIP: Session Initiation Protocol June 2002

20.25 Organization

The Organization header field conveys the name of the organization to

which the SIP element issuing the request or response belongs.

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

Example:

Organization: Boxes by Bob

20.26 Priority

The Priority header field indicates the urgency of the request as

perceived by the client. The Priority header field describes the

priority that the SIP request should have to the receiving human or

its agent. For example, it may be factored into decisions about call

routing and acceptance. For these decisions, a message containing no

Priority header field SHOULD be treated as if it specified a Priority

of "normal". The Priority header field does not influence the use of

communications resources such as packet forwarding priority in

routers or access to circuits in PSTN gateways. The header field can

have the values "non-urgent", "normal", "urgent", and "emergency",

but additional values can be defined elsewhere. It is RECOMMENDED

that the value of "emergency" only be used when life, limb, or

property are in imminent danger. Otherwise, there are no semantics

defined for this header field.

These are the values of RFC 2076 [38], with the addition of

"emergency".

Examples:

Subject: A tornado is heading our way!

Priority: emergency

or

Subject: Weekend plans

Priority: non-urgent

20.27 Proxy-Authenticate

A Proxy-Authenticate header field value contains an authentication

challenge.

The use of this header field is defined in [H14.33]. See Section

22.3 for further details on its usage.

Rosenberg, et. al. Standards Track [Page 174]

RFC 3261 SIP: Session Initiation Protocol June 2002

Example:

Proxy-Authenticate: Digest realm="atlanta.com",

domain="sip:ss1.carrier.com", qop="auth",

nonce="f84f1cec41e6cbe5aea9c8e88d359",

opaque="", stale=FALSE, algorithm=MD5

20.28 Proxy-Authorization

The Proxy-Authorization header field allows the client to identify

itself (or its user) to a proxy that requires authentication. A

Proxy-Authorization field value consists of credentials containing

the authentication information of the user agent for the proxy and/or

realm of the resource being requested.

See Section 22.3 for a definition of the usage of this header field.

This header field, along with Authorization, breaks the general rules

about multiple header field names. Although not a comma-separated

list, this header field name may be present multiple times, and MUST

NOT be combined into a single header line using the usual rules

described in Section 7.3.1.

Example:

Proxy-Authorization: Digest username="Alice", realm="atlanta.com",

nonce="c60f3082ee1212b402a21831ae",

response="245f23415f11432b3434341c022"

20.29 Proxy-Require

The Proxy-Require header field is used to indicate proxy-sensitive

features that must be supported by the proxy. See Section 20.32 for

more details on the mechanics of this message and a usage example.

Example:

Proxy-Require: foo

20.30 Record-Route

The Record-Route header field is inserted by proxies in a request to

force future requests in the dialog to be routed through the proxy.

Examples of its use with the Route header field are described in

Sections 16.12.1.

Rosenberg, et. al. Standards Track [Page 175]

RFC 3261 SIP: Session Initiation Protocol June 2002

Example:

Record-Route: <sip:server10.biloxi.com;lr>,

<sip:bigbox3.site3.atlanta.com;lr>

20.31 Reply-To

The Reply-To header field contains a logical return URI that may be

different from the From header field. For example, the URI MAY be

used to return missed calls or unestablished sessions. If the user

wished to remain anonymous, the header field SHOULD either be omitted

from the request or populated in such a way that does not reveal any

private information.

Even if the "display-name" is empty, the "name-addr" form MUST be

used if the "addr-spec" contains a comma, question mark, or

semicolon. Syntax issues are discussed in Section 7.3.1.

Example:

Reply-To: Bob <sip:bob@biloxi.com>

20.32 Require

The Require header field is used by UACs to tell UASs about options

that the UAC expects the UAS to support in order to process the

request. Although an optional header field, the Require MUST NOT be

ignored if it is present.

The Require header field contains a list of option tags, described in

Section 19.2. Each option tag defines a SIP extension that MUST be

understood to process the request. Frequently, this is used to

indicate that a specific set of extension header fields need to be

understood. A UAC compliant to this specification MUST only include

option tags corresponding to standards-track RFCs.

Example:

Require: 100rel

20.33 Retry-After

The Retry-After header field can be used with a 500 (Server Internal

Error) or 503 (Service Unavailable) response to indicate how long the

service is expected to be unavailable to the requesting client and

with a 404 (Not Found), 413 (Request Entity Too Large), 480

(Temporarily Unavailable), 486 (Busy Here), 600 (Busy), or 603

Rosenberg, et. al. Standards Track [Page 176]

RFC 3261 SIP: Session Initiation Protocol June 2002

(Decline) response to indicate when the called party anticipates

being available again. The value of this field is a positive integer

number of seconds (in decimal) after the time of the response.

An optional comment can be used to indicate additional information

about the time of callback. An optional "duration" parameter

indicates how long the called party will be reachable starting at the

initial time of availability. If no duration parameter is given, the

service is assumed to be available indefinitely.

Examples:

Retry-After: 18000;duration=3600

Retry-After: 120 (I'm in a meeting)

20.34 Route

The Route header field is used to force routing for a request through

the listed set of proxies. Examples of the use of the Route header

field are in Section 16.12.1.

Example:

Route: <sip:bigbox3.site3.atlanta.com;lr>,

<sip:server10.biloxi.com;lr>

20.35 Server

The Server header field contains information about the software used

by the UAS to handle the request.

Revealing the specific software version of the server might allow the

server to become more vulnerable to attacks against software that is

known to contain security holes. Implementers SHOULD make the Server

header field a configurable option.

Example:

Server: HomeServer v2

20.36 Subject

The Subject header field provides a summary or indicates the nature

of the call, allowing call filtering without having to parse the

session description. The session description does not have to use

the same subject indication as the invitation.

The compact form of the Subject header field is s.

Rosenberg, et. al. Standards Track [Page 177]

RFC 3261 SIP: Session Initiation Protocol June 2002

Example:

Subject: Need more boxes

s: Tech Support

20.37 Supported

The Supported header field enumerates all the extensions supported by

the UAC or UAS.

The Supported header field contains a list of option tags, described

in Section 19.2, that are understood by the UAC or UAS. A UA

compliant to this specification MUST only include option tags

corresponding to standards-track RFCs. If empty, it means that no

extensions are supported.

The compact form of the Supported header field is k.

Example:

Supported: 100rel

20.38 Timestamp

The Timestamp header field describes when the UAC sent the request to

the UAS.

See Section 8.2.6 for details on how to generate a response to a

request that contains the header field. Although there is no

normative behavior defined here that makes use of the header, it

allows for extensions or SIP applications to obtain RTT estimates.

Example:

Timestamp: 54

20.39 To

The To header field specifies the logical recipient of the request.

The optional "display-name" is meant to be rendered by a human-user

interface. The "tag" parameter serves as a general mechanism for

dialog identification.

See Section 19.3 for details of the "tag" parameter.

Rosenberg, et. al. Standards Track [Page 178]

RFC 3261 SIP: Session Initiation Protocol June 2002

Comparison of To header fields for equality is identical to

comparison of From header fields. See Section 20.10 for the rules

for parsing a display name, URI and URI parameters, and header field

parameters.

The compact form of the To header field is t.

The following are examples of valid To header fields:

To: The Operator <sip:operator@cs.columbia.edu>;tag=287447

t: sip:+12125551212@server.phone2net.com

20.40 Unsupported

The Unsupported header field lists the features not supported by the

UAS. See Section 20.32 for motivation.

Example:

Unsupported: foo

20.41 User-Agent

The User-Agent header field contains information about the UAC

originating the request. The semantics of this header field are

defined in [H14.43].

Revealing the specific software version of the user agent might allow

the user agent to become more vulnerable to attacks against software

that is known to contain security holes. Implementers SHOULD make

the User-Agent header field a configurable option.

Example:

User-Agent: Softphone Beta1.5

20.42 Via

The Via header field indicates the path taken by the request so far

and indicates the path that should be followed in routing responses.

The branch ID parameter in the Via header field values serves as a

transaction identifier, and is used by proxies to detect loops.

A Via header field value contains the transport protocol used to send

the message, the client's host name or network address, and possibly

the port number at which it wishes to receive responses. A Via

header field value can also contain parameters such as "maddr",

"ttl", "received", and "branch", whose meaning and use are described

Rosenberg, et. al. Standards Track [Page 179]

RFC 3261 SIP: Session Initiation Protocol June 2002

in other sections. For implementations compliant to this

specification, the value of the branch parameter MUST start with the

magic cookie "z9hG4bK", as discussed in Section 8.1.1.7.

Transport protocols defined here are "UDP", "TCP", "TLS", and "SCTP".

"TLS" means TLS over TCP. When a request is sent to a SIPS URI, the

protocol still indicates "SIP", and the transport protocol is TLS.

Via: SIP/2.0/UDP erlang.bell-telephone.com:5060;branch=z9hG4bK87asdks7

Via: SIP/2.0/UDP 192.0.2.1:5060 ;received=192.0.2.207

;branch=z9hG4bK77asjd

The compact form of the Via header field is v.

In this example, the message originated from a multi-homed host with

two addresses, 192.0.2.1 and 192.0.2.207. The sender guessed wrong

as to which network interface would be used. Erlang.bell-

telephone.com noticed the mismatch and added a parameter to the

previous hop's Via header field value, containing the address that

the packet actually came from.

The host or network address and port number are not required to

follow the SIP URI syntax. Specifically, LWS on either side of the

":" or "/" is allowed, as shown here:

Via: SIP / 2.0 / UDP first.example.com: 4000;ttl=16

;maddr=224.2.0.1 ;branch=z9hG4bKa7c6a8dlze.1

Even though this specification mandates that the branch parameter be

present in all requests, the BNF for the header field indicates that

it is optional. This allows interoperation with RFC 2543 elements,

which did not have to insert the branch parameter.

Two Via header fields are equal if their sent-protocol and sent-by

fields are equal, both have the same set of parameters, and the

values of all parameters are equal.

20.43 Warning

The Warning header field is used to carry additional information

about the status of a response. Warning header field values are sent

with responses and contain a three-digit warning code, host name, and

warning text.

The "warn-text" should be in a natural language that is most likely

to be intelligible to the human user receiving the response. This

decision can be based on any available knowledge, such as the

location of the user, the Accept-Language field in a request, or the

Rosenberg, et. al. Standards Track [Page 180]

RFC 3261 SIP: Session Initiation Protocol June 2002

Content-Language field in a response. The default language is i-

default [21].

The currently-defined "warn-code"s are listed below, with a

recommended warn-text in English and a description of their meaning.

These warnings describe failures induced by the session description.

The first digit of warning codes beginning with "3" indicates

warnings specific to SIP. Warnings 300 through 329 are reserved for

indicating problems with keywords in the session description, 330

through 339 are warnings related to basic network services requested

in the session description, 370 through 379 are warnings related to

quantitative QoS parameters requested in the session description, and

390 through 399 are miscellaneous warnings that do not fall into one

of the above categories.

300 Incompatible network protocol: One or more network protocols

contained in the session description are not available.

301 Incompatible network address formats: One or more network

address formats contained in the session description are not

available.

302 Incompatible transport protocol: One or more transport

protocols described in the session description are not

available.

303 Incompatible bandwidth units: One or more bandwidth

measurement units contained in the session description were

not understood.

304 Media type not available: One or more media types contained in

the session description are not available.

305 Incompatible media format: One or more media formats contained

in the session description are not available.

306 Attribute not understood: One or more of the media attributes

in the session description are not supported.

307 Session description parameter not understood: A parameter

other than those listed above was not understood.

330 Multicast not available: The site where the user is located

does not support multicast.

331 Unicast not available: The site where the user is located does

not support unicast communication (usually due to the presence

of a firewall).

Rosenberg, et. al. Standards Track [Page 181]

RFC 3261 SIP: Session Initiation Protocol June 2002

370 Insufficient bandwidth: The bandwidth specified in the session

description or defined by the media exceeds that known to be

available.

399 Miscellaneous warning: The warning text can include arbitrary

information to be presented to a human user or logged. A

system receiving this warning MUST NOT take any automated

action.

1xx and 2xx have been taken by HTTP/1.1.

Additional "warn-code"s can be defined through IANA, as defined in

Section 27.2.

Examples:

Warning: 307 isi.edu "Session parameter 'foo' not understood"

Warning: 301 isi.edu "Incompatible network address type 'E.164'"

20.44 WWW-Authenticate

A WWW-Authenticate header field value contains an authentication

challenge. See Section 22.2 for further details on its usage.

Example:

WWW-Authenticate: Digest realm="atlanta.com",

domain="sip:boxesbybob.com", qop="auth",

nonce="f84f1cec41e6cbe5aea9c8e88d359",

opaque="", stale=FALSE, algorithm=MD5

21 Response Codes

The response codes are consistent with, and extend, HTTP/1.1 response

codes. Not all HTTP/1.1 response codes are appropriate, and only

those that are appropriate are given here. Other HTTP/1.1 response

codes SHOULD NOT be used. Also, SIP defines a new class, 6xx.

21.1 Provisional 1xx

Provisional responses, also known as informational responses,

indicate that the server contacted is performing some further action

and does not yet have a definitive response. A server sends a 1xx

response if it expects to take more than 200 ms to obtain a final

response. Note that 1xx responses are not transmitted reliably.

They never cause the client to send an ACK. Provisional (1xx)

responses MAY contain message bodies, including session descriptions.

Rosenberg, et. al. Standards Track [Page 182]

RFC 3261 SIP: Session Initiation Protocol June 2002

21.1.1 100 Trying

This response indicates that the request has been received by the

next-hop server and that some unspecified action is being taken on

behalf of this call (for example, a database is being consulted).

This response, like all other provisional responses, stops

retransmissions of an INVITE by a UAC. The 100 (Trying) response is

different from other provisional responses, in that it is never

forwarded upstream by a stateful proxy.

21.1.2 180 Ringing

The UA receiving the INVITE is trying to alert the user. This

response MAY be used to initiate local ringback.

21.1.3 181 Call Is Being Forwarded

A server MAY use this status code to indicate that the call is being

forwarded to a different set of destinations.

21.1.4 182 Queued

The called party is temporarily unavailable, but the server has

decided to queue the call rather than reject it. When the callee

becomes available, it will return the appropriate final status

response. The reason phrase MAY give further details about the

status of the call, for example, "5 calls queued; expected waiting

time is 15 minutes". The server MAY issue several 182 (Queued)

responses to update the caller about the status of the queued call.

21.1.5 183 Session Progress

The 183 (Session Progress) response is used to convey information

about the progress of the call that is not otherwise classified. The

Reason-Phrase, header fields, or message body MAY be used to convey

more details about the call progress.

21.2 Successful 2xx

The request was successful.

21.2.1 200 OK

The request has succeeded. The information returned with the

response depends on the method used in the request.

Rosenberg, et. al. Standards Track [Page 183]

RFC 3261 SIP: Session Initiation Protocol June 2002

21.3 Redirection 3xx

3xx responses give information about the user's new location, or

about alternative services that might be able to satisfy the call.

21.3.1 300 Multiple Choices

The address in the request resolved to several choices, each with its

own specific location, and the user (or UA) can select a preferred

communication end point and redirect its request to that location.

The response MAY include a message body containing a list of resource

characteristics and location(s) from which the user or UA can choose

the one most appropriate, if allowed by the Accept request header

field. However, no MIME types have been defined for this message

body.

The choices SHOULD also be listed as Contact fields (Section 20.10).

Unlike HTTP, the SIP response MAY contain several Contact fields or a

list of addresses in a Contact field. UAs MAY use the Contact header

field value for automatic redirection or MAY ask the user to confirm

a choice. However, this specification does not define any standard

for such automatic selection.

This status response is appropriate if the callee can be reached

at several different locations and the server cannot or prefers

not to proxy the request.

21.3.2 301 Moved Permanently

The user can no longer be found at the address in the Request-URI,

and the requesting client SHOULD retry at the new address given by

the Contact header field (Section 20.10). The requestor SHOULD

update any local directories, address books, and user location caches

with this new value and redirect future requests to the address(es)

listed.

21.3.3 302 Moved Temporarily

The requesting client SHOULD retry the request at the new address(es)

given by the Contact header field (Section 20.10). The Request-URI

of the new request uses the value of the Contact header field in the

response.

Rosenberg, et. al. Standards Track [Page 184]

RFC 3261 SIP: Session Initiation Protocol June 2002

The duration of the validity of the Contact URI can be indicated

through an Expires (Section 20.19) header field or an expires

parameter in the Contact header field. Both proxies and UAs MAY

cache this URI for the duration of the expiration time. If there is

no explicit expiration time, the address is only valid once for

recursing, and MUST NOT be cached for future transactions.

If the URI cached from the Contact header field fails, the Request-

URI from the redirected request MAY be tried again a single time.

The temporary URI may have become out-of-date sooner than the

expiration time, and a new temporary URI may be available.

21.3.4 305 Use Proxy

The requested resource MUST be accessed through the proxy given by

the Contact field. The Contact field gives the URI of the proxy.

The recipient is expected to repeat this single request via the

proxy. 305 (Use Proxy) responses MUST only be generated by UASs.

21.3.5 380 Alternative Service

The call was not successful, but alternative services are possible.

The alternative services are described in the message body of the

response. Formats for such bodies are not defined here, and may be

the subject of future standardization.

21.4 Request Failure 4xx

4xx responses are definite failure responses from a particular

server. The client SHOULD NOT retry the same request without

modification (for example, adding appropriate authorization).

However, the same request to a different server might be successful.

21.4.1 400 Bad Request

The request could not be understood due to malformed syntax. The

Reason-Phrase SHOULD identify the syntax problem in more detail, for

example, "Missing Call-ID header field".

21.4.2 401 Unauthorized

The request requires user authentication. This response is issued by

UASs and registrars, while 407 (Proxy Authentication Required) is

used by proxy servers.

Rosenberg, et. al. Standards Track [Page 185]

RFC 3261 SIP: Session Initiation Protocol June 2002

21.4.3 402 Payment Required

Reserved for future use.

21.4.4 403 Forbidden

The server understood the request, but is refusing to fulfill it.

Authorization will not help, and the request SHOULD NOT be repeated.

21.4.5 404 Not Found

The server has definitive information that the user does not exist at

the domain specified in the Request-URI. This status is also

returned if the domain in the Request-URI does not match any of the

domains handled by the recipient of the request.

21.4.6 405 Method Not Allowed

The method specified in the Request-Line is understood, but not

allowed for the address identified by the Request-URI.

The response MUST include an Allow header field containing a list of

valid methods for the indicated address.

21.4.7 406 Not Acceptable

The resource identified by the request is only capable of generating

response entities that have content characteristics not acceptable

according to the Accept header field sent in the request.

21.4.8 407 Proxy Authentication Required

This code is similar to 401 (Unauthorized), but indicates that the

client MUST first authenticate itself with the proxy. SIP access

authentication is explained in Sections 26 and 22.3.

This status code can be used for applications where access to the

communication channel (for example, a telephony gateway) rather than

the callee requires authentication.

21.4.9 408 Request Timeout

The server could not produce a response within a suitable amount of

time, for example, if it could not determine the location of the user

in time. The client MAY repeat the request without modifications at

any later time.

Rosenberg, et. al. Standards Track [Page 186]

RFC 3261 SIP: Session Initiation Protocol June 2002

21.4.10 410 Gone

The requested resource is no longer available at the server and no

forwarding address is known. This condition is expected to be

considered permanent. If the server does not know, or has no

facility to determine, whether or not the condition is permanent, the

status code 404 (Not Found) SHOULD be used instead.

21.4.11 413 Request Entity Too Large

The server is refusing to process a request because the request

entity-body is larger than the server is willing or able to process.

The server MAY close the connection to prevent the client from

continuing the request.

If the condition is temporary, the server SHOULD include a Retry-

After header field to indicate that it is temporary and after what

time the client MAY try again.

21.4.12 414 Request-URI Too Long

The server is refusing to service the request because the Request-URI

is longer than the server is willing to interpret.

21.4.13 415 Unsupported Media Type

The server is refusing to service the request because the message

body of the request is in a format not supported by the server for

the requested method. The server MUST return a list of acceptable

formats using the Accept, Accept-Encoding, or Accept-Language header

field, depending on the specific problem with the content. UAC

processing of this response is described in Section 8.1.3.5.

21.4.14 416 Unsupported URI Scheme

The server cannot process the request because the scheme of the URI

in the Request-URI is unknown to the server. Client processing of

this response is described in Section 8.1.3.5.

21.4.15 420 Bad Extension

The server did not understand the protocol extension specified in a

Proxy-Require (Section 20.29) or Require (Section 20.32) header

field. The server MUST include a list of the unsupported extensions

in an Unsupported header field in the response. UAC processing of

this response is described in Section 8.1.3.5.

Rosenberg, et. al. Standards Track [Page 187]

RFC 3261 SIP: Session Initiation Protocol June 2002

21.4.16 421 Extension Required

The UAS needs a particular extension to process the request, but this

extension is not listed in a Supported header field in the request.

Responses with this status code MUST contain a Require header field

listing the required extensions.

A UAS SHOULD NOT use this response unless it truly cannot provide any

useful service to the client. Instead, if a desirable extension is

not listed in the Supported header field, servers SHOULD process the

request using baseline SIP capabilities and any extensions supported

by the client.

21.4.17 423 Interval Too Brief

The server is rejecting the request because the expiration time of

the resource refreshed by the request is too short. This response

can be used by a registrar to reject a registration whose Contact

header field expiration time was too small. The use of this response

and the related Min-Expires header field are described in Sections

10.2.8, 10.3, and 20.23.

21.4.18 480 Temporarily Unavailable

The callee's end system was contacted successfully but the callee is

currently unavailable (for example, is not logged in, logged in but

in a state that precludes communication with the callee, or has

activated the "do not disturb" feature). The response MAY indicate a

better time to call in the Retry-After header field. The user could

also be available elsewhere (unbeknownst to this server). The reason

phrase SHOULD indicate a more precise cause as to why the callee is

unavailable. This value SHOULD be settable by the UA. Status 486

(Busy Here) MAY be used to more precisely indicate a particular

reason for the call failure.

This status is also returned by a redirect or proxy server that

recognizes the user identified by the Request-URI, but does not

currently have a valid forwarding location for that user.

21.4.19 481 Call/Transaction Does Not Exist

This status indicates that the UAS received a request that does not

match any existing dialog or transaction.

21.4.20 482 Loop Detected

The server has detected a loop (Section 16.3 Item 4).

Rosenberg, et. al. Standards Track [Page 188]

RFC 3261 SIP: Session Initiation Protocol June 2002

21.4.21 483 Too Many Hops

The server received a request that contains a Max-Forwards (Section

20.22) header field with the value zero.

21.4.22 484 Address Incomplete

The server received a request with a Request-URI that was incomplete.

Additional information SHOULD be provided in the reason phrase.

This status code allows overlapped dialing. With overlapped

dialing, the client does not know the length of the dialing

string. It sends strings of increasing lengths, prompting the

user for more input, until it no longer receives a 484 (Address

Incomplete) status response.

21.4.23 485 Ambiguous

The Request-URI was ambiguous. The response MAY contain a listing of

possible unambiguous addresses in Contact header fields. Revealing

alternatives can infringe on privacy of the user or the organization.

It MUST be possible to configure a server to respond with status 404

(Not Found) or to suppress the listing of possible choices for

ambiguous Request-URIs.

Example response to a request with the Request-URI

sip:lee@example.com:

SIP/2.0 485 Ambiguous

Contact: Carol Lee <sip:carol.lee@example.com>

Contact: Ping Lee <sip:p.lee@example.com>

Contact: Lee M. Foote <sips:lee.foote@example.com>

Some email and voice mail systems provide this functionality. A

status code separate from 3xx is used since the semantics are

different: for 300, it is assumed that the same person or service

will be reached by the choices provided. While an automated

choice or sequential search makes sense for a 3xx response, user

intervention is required for a 485 (Ambiguous) response.

21.4.24 486 Busy Here

The callee's end system was contacted successfully, but the callee is

currently not willing or able to take additional calls at this end

system. The response MAY indicate a better time to call in the

Retry-After header field. The user could also be available

Rosenberg, et. al. Standards Track [Page 189]

RFC 3261 SIP: Session Initiation Protocol June 2002

elsewhere, such as through a voice mail service. Status 600 (Busy

Everywhere) SHOULD be used if the client knows that no other end

system will be able to accept this call.

21.4.25 487 Request Terminated

The request was terminated by a BYE or CANCEL request. This response

is never returned for a CANCEL request itself.

21.4.26 488 Not Acceptable Here

The response has the same meaning as 606 (Not Acceptable), but only

applies to the specific resource addressed by the Request-URI and the

request may succeed elsewhere.

A message body containing a description of media capabilities MAY be

present in the response, which is formatted according to the Accept

header field in the INVITE (or application/sdp if not present), the

same as a message body in a 200 (OK) response to an OPTIONS request.

21.4.27 491 Request Pending

The request was received by a UAS that had a pending request within

the same dialog. Section 14.2 describes how such "glare" situations

are resolved.

21.4.28 493 Undecipherable

The request was received by a UAS that contained an encrypted MIME

body for which the recipient does not possess or will not provide an

appropriate decryption key. This response MAY have a single body

containing an appropriate public key that should be used to encrypt

MIME bodies sent to this UA. Details of the usage of this response

code can be found in Section 23.2.

21.5 Server Failure 5xx

5xx responses are failure responses given when a server itself has

erred.

21.5.1 500 Server Internal Error

The server encountered an unexpected condition that prevented it from

fulfilling the request. The client MAY display the specific error

condition and MAY retry the request after several seconds.

If the condition is temporary, the server MAY indicate when the

client may retry the request using the Retry-After header field.

Rosenberg, et. al. Standards Track [Page 190]

RFC 3261 SIP: Session Initiation Protocol June 2002

21.5.2 501 Not Implemented

The server does not support the functionality required to fulfill the

request. This is the appropriate response when a UAS does not

recognize the request method and is not capable of supporting it for

any user. (Proxies forward all requests regardless of method.)

Note that a 405 (Method Not Allowed) is sent when the server

recognizes the request method, but that method is not allowed or

supported.

21.5.3 502 Bad Gateway

The server, while acting as a gateway or proxy, received an invalid

response from the downstream server it accessed in attempting to

fulfill the request.

21.5.4 503 Service Unavailable

The server is temporarily unable to process the request due to a

temporary overloading or maintenance of the server. The server MAY

indicate when the client should retry the request in a Retry-After

header field. If no Retry-After is given, the client MUST act as if

it had received a 500 (Server Internal Error) response.

A client (proxy or UAC) receiving a 503 (Service Unavailable) SHOULD

attempt to forward the request to an alternate server. It SHOULD NOT

forward any other requests to that server for the duration specified

in the Retry-After header field, if present.

Servers MAY refuse the connection or drop the request instead of

responding with 503 (Service Unavailable).

21.5.5 504 Server Time-out

The server did not receive a timely response from an external server

it accessed in attempting to process the request. 408 (Request

Timeout) should be used instead if there was no response within the

period specified in the Expires header field from the upstream

server.

21.5.6 505 Version Not Supported

The server does not support, or refuses to support, the SIP protocol

version that was used in the request. The server is indicating that

it is unable or unwilling to complete the request using the same

major version as the client, other than with this error message.

Rosenberg, et. al. Standards Track [Page 191]

RFC 3261 SIP: Session Initiation Protocol June 2002

21.5.7 513 Message Too Large

The server was unable to process the request since the message length

exceeded its capabilities.

21.6 Global Failures 6xx

6xx responses indicate that a server has definitive information about

a particular user, not just the particular instance indicated in the

Request-URI.

21.6.1 600 Busy Everywhere

The callee's end system was contacted successfully but the callee is

busy and does not wish to take the call at this time. The response

MAY indicate a better time to call in the Retry-After header field.

If the callee does not wish to reveal the reason for declining the

call, the callee uses status code 603 (Decline) instead. This status

response is returned only if the client knows that no other end point

(such as a voice mail system) will answer the request. Otherwise,

486 (Busy Here) should be returned.

21.6.2 603 Decline

The callee's machine was successfully contacted but the user

explicitly does not wish to or cannot participate. The response MAY

indicate a better time to call in the Retry-After header field. This

status response is returned only if the client knows that no other

end point will answer the request.

21.6.3 604 Does Not Exist Anywhere

The server has authoritative information that the user indicated in

the Request-URI does not exist anywhere.

21.6.4 606 Not Acceptable

The user's agent was contacted successfully but some aspects of the

session description such as the requested media, bandwidth, or

addressing style were not acceptable.

A 606 (Not Acceptable) response means that the user wishes to

communicate, but cannot adequately support the session described.

The 606 (Not Acceptable) response MAY contain a list of reasons in a

Warning header field describing why the session described cannot be

supported. Warning reason codes are listed in Section 20.43.

Rosenberg, et. al. Standards Track [Page 192]

RFC 3261 SIP: Session Initiation Protocol June 2002

A message body containing a description of media capabilities MAY be

present in the response, which is formatted according to the Accept

header field in the INVITE (or application/sdp if not present), the

same as a message body in a 200 (OK) response to an OPTIONS request.

It is hoped that negotiation will not frequently be needed, and when

a new user is being invited to join an already existing conference,

negotiation may not be possible. It is up to the invitation

initiator to decide whether or not to act on a 606 (Not Acceptable)

response.

This status response is returned only if the client knows that no

other end point will answer the request.

22 Usage of HTTP Authentication

SIP provides a stateless, challenge-based mechanism for

authentication that is based on authentication in HTTP. Any time

that a proxy server or UA receives a request (with the exceptions

given in Section 22.1), it MAY challenge the initiator of the request

to provide assurance of its identity. Once the originator has been

identified, the recipient of the request SHOULD ascertain whether or

not this user is authorized to make the request in question. No

authorization systems are recommended or discussed in this document.

The "Digest" authentication mechanism described in this section

provides message authentication and replay protection only, without

message integrity or confidentiality. Protective measures above and

beyond those provided by Digest need to be taken to prevent active

attackers from modifying SIP requests and responses.

Note that due to its weak security, the usage of "Basic"

authentication has been deprecated. Servers MUST NOT accept

credentials using the "Basic" authorization scheme, and servers also

MUST NOT challenge with "Basic". This is a change from RFC 2543.

22.1 Framework

The framework for SIP authentication closely parallels that of HTTP

(RFC 2617 [17]). In particular, the BNF for auth-scheme, auth-param,

challenge, realm, realm-value, and credentials is identical (although

the usage of "Basic" as a scheme is not permitted). In SIP, a UAS

uses the 401 (Unauthorized) response to challenge the identity of a

UAC. Additionally, registrars and redirect servers MAY make use of

401 (Unauthorized) responses for authentication, but proxies MUST

NOT, and instead MAY use the 407 (Proxy Authentication Required)

Rosenberg, et. al. Standards Track [Page 193]

RFC 3261 SIP: Session Initiation Protocol June 2002

response. The requirements for inclusion of the Proxy-Authenticate,

Proxy-Authorization, WWW-Authenticate, and Authorization in the

various messages are identical to those described in RFC 2617 [17].

Since SIP does not have the concept of a canonical root URL, the

notion of protection spaces is interpreted differently in SIP. The

realm string alone defines the protection domain. This is a change

from RFC 2543, in which the Request-URI and the realm together

defined the protection domain.

This previous definition of protection domain caused some amount

of confusion since the Request-URI sent by the UAC and the

Request-URI received by the challenging server might be different,

and indeed the final form of the Request-URI might not be known to

the UAC. Also, the previous definition depended on the presence

of a SIP URI in the Request-URI and seemed to rule out alternative

URI schemes (for example, the tel URL).

Operators of user agents or proxy servers that will authenticate

received requests MUST adhere to the following guidelines for

creation of a realm string for their server:

o Realm strings MUST be globally unique. It is RECOMMENDED that

a realm string contain a hostname or domain name, following the

recommendation in Section 3.2.1 of RFC 2617 [17].

o Realm strings SHOULD present a human-readable identifier that

can be rendered to a user.

For example:

INVITE sip:bob@biloxi.com SIP/2.0

Authorization: Digest realm="biloxi.com", <...>

Generally, SIP authentication is meaningful for a specific realm, a

protection domain. Thus, for Digest authentication, each such

protection domain has its own set of usernames and passwords. If a

server does not require authentication for a particular request, it

MAY accept a default username, "anonymous", which has no password

(password of ""). Similarly, UACs representing many users, such as

PSTN gateways, MAY have their own device-specific username and

password, rather than accounts for particular users, for their realm.

While a server can legitimately challenge most SIP requests, there

are two requests defined by this document that require special

handling for authentication: ACK and CANCEL.

Rosenberg, et. al. Standards Track [Page 194]

RFC 3261 SIP: Session Initiation Protocol June 2002

Under an authentication scheme that uses responses to carry values

used to compute nonces (such as Digest), some problems come up for

any requests that take no response, including ACK. For this reason,

any credentials in the INVITE that were accepted by a server MUST be

accepted by that server for the ACK. UACs creating an ACK message

will duplicate all of the Authorization and Proxy-Authorization

header field values that appeared in the INVITE to which the ACK

corresponds. Servers MUST NOT attempt to challenge an ACK.

Although the CANCEL method does take a response (a 2xx), servers MUST

NOT attempt to challenge CANCEL requests since these requests cannot

be resubmitted. Generally, a CANCEL request SHOULD be accepted by a

server if it comes from the same hop that sent the request being

canceled (provided that some sort of transport or network layer

security association, as described in Section 26.2.1, is in place).

When a UAC receives a challenge, it SHOULD render to the user the

contents of the "realm" parameter in the challenge (which appears in

either a WWW-Authenticate header field or Proxy-Authenticate header

field) if the UAC device does not already know of a credential for

the realm in question. A service provider that pre-configures UAs

with credentials for its realm should be aware that users will not

have the opportunity to present their own credentials for this realm

when challenged at a pre-configured device.

Finally, note that even if a UAC can locate credentials that are

associated with the proper realm, the potential exists that these

credentials may no longer be valid or that the challenging server

will not accept these credentials for whatever reason (especially

when "anonymous" with no password is submitted). In this instance a

server may repeat its challenge, or it may respond with a 403

Forbidden. A UAC MUST NOT re-attempt requests with the credentials

that have just been rejected (though the request may be retried if

the nonce was stale).

22.2 User-to-User Authentication

When a UAS receives a request from a UAC, the UAS MAY authenticate

the originator before the request is processed. If no credentials

(in the Authorization header field) are provided in the request, the

UAS can challenge the originator to provide credentials by rejecting

the request with a 401 (Unauthorized) status code.

The WWW-Authenticate response-header field MUST be included in 401

(Unauthorized) response messages. The field value consists of at

least one challenge that indicates the authentication scheme(s) and

parameters applicable to the realm.

Rosenberg, et. al. Standards Track [Page 195]

RFC 3261 SIP: Session Initiation Protocol June 2002

An example of the WWW-Authenticate header field in a 401 challenge

is:

WWW-Authenticate: Digest

realm="biloxi.com",

qop="auth,auth-int",

nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",

opaque="5ccc069c403ebaf9f0171e9517f40e41"

When the originating UAC receives the 401 (Unauthorized), it SHOULD,

if it is able, re-originate the request with the proper credentials.

The UAC may require input from the originating user before

proceeding. Once authentication credentials have been supplied

(either directly by the user, or discovered in an internal keyring),

UAs SHOULD cache the credentials for a given value of the To header

field and "realm" and attempt to re-use these values on the next

request for that destination. UAs MAY cache credentials in any way

they would like.

If no credentials for a realm can be located, UACs MAY attempt to

retry the request with a username of "anonymous" and no password (a

password of "").

Once credentials have been located, any UA that wishes to

authenticate itself with a UAS or registrar -- usually, but not

necessarily, after receiving a 401 (Unauthorized) response -- MAY do

so by including an Authorization header field with the request. The

Authorization field value consists of credentials containing the

authentication information of the UA for the realm of the resource

being requested as well as parameters required in support of

authentication and replay protection.

An example of the Authorization header field is:

Authorization: Digest username="bob",

realm="biloxi.com",

nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",

uri="sip:bob@biloxi.com",

qop=auth,

nc=00000001,

cnonce="0a4f113b",

response="6629fae49393a05397450978507c4ef1",

opaque="5ccc069c403ebaf9f0171e9517f40e41"

When a UAC resubmits a request with its credentials after receiving a

401 (Unauthorized) or 407 (Proxy Authentication Required) response,

it MUST increment the CSeq header field value as it would normally

when sending an updated request.

Rosenberg, et. al. Standards Track [Page 196]

RFC 3261 SIP: Session Initiation Protocol June 2002

22.3 Proxy-to-User Authentication

Similarly, when a UAC sends a request to a proxy server, the proxy

server MAY authenticate the originator before the request is

processed. If no credentials (in the Proxy-Authorization header

field) are provided in the request, the proxy can challenge the

originator to provide credentials by rejecting the request with a 407

(Proxy Authentication Required) status code. The proxy MUST populate

the 407 (Proxy Authentication Required) message with a Proxy-

Authenticate header field value applicable to the proxy for the

requested resource.

The use of Proxy-Authenticate and Proxy-Authorization parallel that

described in [17], with one difference. Proxies MUST NOT add values

to the Proxy-Authorization header field. All 407 (Proxy

Authentication Required) responses MUST be forwarded upstream toward

the UAC following the procedures for any other response. It is the

UAC's responsibility to add the Proxy-Authorization header field

value containing credentials for the realm of the proxy that has

asked for authentication.

If a proxy were to resubmit a request adding a Proxy-Authorization

header field value, it would need to increment the CSeq in the new

request. However, this would cause the UAC that submitted the

original request to discard a response from the UAS, as the CSeq

value would be different.

When the originating UAC receives the 407 (Proxy Authentication

Required) it SHOULD, if it is able, re-originate the request with the

proper credentials. It should follow the same procedures for the

display of the "realm" parameter that are given above for responding

to 401.

If no credentials for a realm can be located, UACs MAY attempt to

retry the request with a username of "anonymous" and no password (a

password of "").

The UAC SHOULD also cache the credentials used in the re-originated

request.

The following rule is RECOMMENDED for proxy credential caching:

If a UA receives a Proxy-Authenticate header field value in a 401/407

response to a request with a particular Call-ID, it should

incorporate credentials for that realm in all subsequent requests

that contain the same Call-ID. These credentials MUST NOT be cached

across dialogs; however, if a UA is configured with the realm of its

local outbound proxy, when one exists, then the UA MAY cache

Rosenberg, et. al. Standards Track [Page 197]

RFC 3261 SIP: Session Initiation Protocol June 2002

credentials for that realm across dialogs. Note that this does mean

a future request in a dialog could contain credentials that are not

needed by any proxy along the Route header path.

Any UA that wishes to authenticate itself to a proxy server --

usually, but not necessarily, after receiving a 407 (Proxy

Authentication Required) response -- MAY do so by including a Proxy-

Authorization header field value with the request. The Proxy-

Authorization request-header field allows the client to identify

itself (or its user) to a proxy that requires authentication. The

Proxy-Authorization header field value consists of credentials

containing the authentication information of the UA for the proxy

and/or realm of the resource being requested.

A Proxy-Authorization header field value applies only to the proxy

whose realm is identified in the "realm" parameter (this proxy may

previously have demanded authentication using the Proxy-Authenticate

field). When multiple proxies are used in a chain, a Proxy-

Authorization header field value MUST NOT be consumed by any proxy

whose realm does not match the "realm" parameter specified in that

value.

Note that if an authentication scheme that does not support realms is

used in the Proxy-Authorization header field, a proxy server MUST

attempt to parse all Proxy-Authorization header field values to

determine whether one of them has what the proxy server considers to

be valid credentials. Because this is potentially very time-

consuming in large networks, proxy servers SHOULD use an

authentication scheme that supports realms in the Proxy-Authorization

header field.

If a request is forked (as described in Section 16.7), various proxy

servers and/or UAs may wish to challenge the UAC. In this case, the

forking proxy server is responsible for aggregating these challenges

into a single response. Each WWW-Authenticate and Proxy-Authenticate

value received in responses to the forked request MUST be placed into

the single response that is sent by the forking proxy to the UA; the

ordering of these header field values is not significant.

When a proxy server issues a challenge in response to a request,

it will not proxy the request until the UAC has retried the

request with valid credentials. A forking proxy may forward a

request simultaneously to multiple proxy servers that require

authentication, each of which in turn will not forward the request

until the originating UAC has authenticated itself in their

respective realm. If the UAC does not provide credentials for

Rosenberg, et. al. Standards Track [Page 198]

RFC 3261 SIP: Session Initiation Protocol June 2002

each challenge, the proxy servers that issued the challenges will

not forward requests to the UA where the destination user might be

located, and therefore, the virtues of forking are largely lost.

When resubmitting its request in response to a 401 (Unauthorized) or

407 (Proxy Authentication Required) that contains multiple

challenges, a UAC MAY include an Authorization value for each WWW-

Authenticate value and a Proxy-Authorization value for each Proxy-

Authenticate value for which the UAC wishes to supply a credential.

As noted above, multiple credentials in a request SHOULD be

differentiated by the "realm" parameter.

It is possible for multiple challenges associated with the same realm

to appear in the same 401 (Unauthorized) or 407 (Proxy Authentication

Required). This can occur, for example, when multiple proxies within

the same administrative domain, which use a common realm, are reached

by a forking request. When it retries a request, a UAC MAY therefore

supply multiple credentials in Authorization or Proxy-Authorization

header fields with the same "realm" parameter value. The same

credentials SHOULD be used for the same realm.

22.4 The Digest Authentication Scheme

This section describes the modifications and clarifications required

to apply the HTTP Digest authentication scheme to SIP. The SIP

scheme usage is almost completely identical to that for HTTP [17].

Since RFC 2543 is based on HTTP Digest as defined in RFC 2069 [39],

SIP servers supporting RFC 2617 MUST ensure they are backwards

compatible with RFC 2069. Procedures for this backwards

compatibility are specified in RFC 2617. Note, however, that SIP

servers MUST NOT accept or request Basic authentication.

The rules for Digest authentication follow those defined in [17],

with "HTTP/1.1" replaced by "SIP/2.0" in addition to the following

differences:

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

URI = SIP-URI / SIPS-URI

2. The BNF in RFC 2617 has an error in that the 'uri' parameter

of the Authorization header field for HTTP Digest

Rosenberg, et. al. Standards Track [Page 199]

RFC 3261 SIP: Session Initiation Protocol June 2002

authentication is not enclosed in quotation marks. (The

example in Section 3.5 of RFC 2617 is correct.) For SIP, the

'uri' MUST be enclosed in quotation marks.

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

digest-uri-value = Request-URI ; as defined in Section 25

4. The example procedure for choosing a nonce based on Etag does

not work for SIP.

5. The text in RFC 2617 [17] regarding cache operation does not

apply to SIP.

6. RFC 2617 [17] requires that a server check that the URI in the

request line and the URI included in the Authorization header

field point to the same resource. In a SIP context, these two

URIs may refer to different users, due to forwarding at some

proxy. Therefore, in SIP, a server MAY check that the

Request-URI in the Authorization header field value

corresponds to a user for whom the server is willing to accept

forwarded or direct requests, but it is not necessarily a

failure if the two fields are not equivalent.

7. As a clarification to the calculation of the A2 value for

message integrity assurance in the Digest authentication

scheme, implementers should assume, when the entity-body is

empty (that is, when SIP messages have no body) that the hash

of the entity-body resolves to the MD5 hash of an empty

string, or:

H(entity-body) = MD5("") =

"d41d8cd98f00b204e9800998ecf8427e"

8. RFC 2617 notes that a cnonce value MUST NOT be sent in an

Authorization (and by extension Proxy-Authorization) header

field if no qop directive has been sent. Therefore, any

algorithms that have a dependency on the cnonce (including

"MD5-Sess") require that the qop directive be sent. Use of

the "qop" parameter is optional in RFC 2617 for the purposes

of backwards compatibility with RFC 2069; since RFC 2543 was

based on RFC 2069, the "qop" parameter must unfortunately

remain optional for clients and servers to receive. However,

servers MUST always send a "qop" parameter in WWW-Authenticate

and Proxy-Authenticate header field values. If a client

receives a "qop" parameter in a challenge header field, it

MUST send the "qop" parameter in any resulting authorization

header field.

Rosenberg, et. al. Standards Track [Page 200]

RFC 3261 SIP: Session Initiation Protocol June 2002

RFC 2543 did not allow usage of the Authentication-Info header field

(it effectively used RFC 2069). However, we now allow usage of this

header field, since it provides integrity checks over the bodies and

provides mutual authentication. RFC 2617 [17] defines mechanisms for

backwards compatibility using the qop attribute in the request.

These mechanisms MUST be used by a server to determine if the client

supports the new mechanisms in RFC 2617 that were not specified in

RFC 2069.

23 S/MIME

SIP messages carry MIME bodies and the MIME standard includes

mechanisms for securing MIME contents to ensure both integrity and

confidentiality (including the 'multipart/signed' and

'application/pkcs7-mime' MIME types, see RFC 1847 [22], RFC 2630 [23]

and RFC 2633 [24]). Implementers should note, however, that there

may be rare network intermediaries (not typical proxy servers) that

rely on viewing or modifying the bodies of SIP messages (especially

SDP), and that secure MIME may prevent these sorts of intermediaries

from functioning.

This applies particularly to certain types of firewalls.

The PGP mechanism for encrypting the header fields and bodies of

SIP messages described in RFC 2543 has been deprecated.

23.1 S/MIME Certificates

The certificates that are used to identify an end-user for the

purposes of S/MIME differ from those used by servers in one important

respect - rather than asserting that the identity of the holder

corresponds to a particular hostname, these certificates assert that

the holder is identified by an end-user address. This address is

composed of the concatenation of the "userinfo" "@" and "domainname"

portions of a SIP or SIPS URI (in other words, an email address of

the form "bob@biloxi.com"), most commonly corresponding to a user's

address-of-record.

These certificates are also associated with keys that are used to

sign or encrypt bodies of SIP messages. Bodies are signed with the

private key of the sender (who may include their public key with the

message as appropriate), but bodies are encrypted with the public key

of the intended recipient. Obviously, senders must have

foreknowledge of the public key of recipients in order to encrypt

message bodies. Public keys can be stored within a UA on a virtual

keyring.

Rosenberg, et. al. Standards Track [Page 201]

RFC 3261 SIP: Session Initiation Protocol June 2002

Each user agent that supports S/MIME MUST contain a keyring

specifically for end-users' certificates. This keyring should map

between addresses of record and corresponding certificates. Over

time, users SHOULD use the same certificate when they populate the

originating URI of signaling (the From header field) with the same

address-of-record.

Any mechanisms depending on the existence of end-user certificates

are seriously limited in that there is virtually no consolidated

authority today that provides certificates for end-user applications.

However, users SHOULD acquire certificates from known public

certificate authorities. As an alternative, users MAY create self-

signed certificates. The implications of self-signed certificates

are explored further in Section 26.4.2. Implementations may also use

pre-configured certificates in deployments in which a previous trust

relationship exists between all SIP entities.

Above and beyond the problem of acquiring an end-user certificate,

there are few well-known centralized directories that distribute

end-user certificates. However, the holder of a certificate SHOULD

publish their certificate in any public directories as appropriate.

Similarly, UACs SHOULD support a mechanism for importing (manually or

automatically) certificates discovered in public directories

corresponding to the target URIs of SIP requests.

23.2 S/MIME Key Exchange

SIP itself can also be used as a means to distribute public keys in

the following manner.

Whenever the CMS SignedData message is used in S/MIME for SIP, it

MUST contain the certificate bearing the public key necessary to

verify the signature.

When a UAC sends a request containing an S/MIME body that initiates a

dialog, or sends a non-INVITE request outside the context of a

dialog, the UAC SHOULD structure the body as an S/MIME

'multipart/signed' CMS SignedData body. If the desired CMS service

is EnvelopedData (and the public key of the target user is known),

the UAC SHOULD send the EnvelopedData message encapsulated within a

SignedData message.

When a UAS receives a request containing an S/MIME CMS body that

includes a certificate, the UAS SHOULD first validate the

certificate, if possible, with any available root certificates for

certificate authorities. The UAS SHOULD also determine the subject

of the certificate (for S/MIME, the SubjectAltName will contain the

appropriate identity) and compare this value to the From header field

Rosenberg, et. al. Standards Track [Page 202]

RFC 3261 SIP: Session Initiation Protocol June 2002

of the request. If the certificate cannot be verified, because it is

self-signed, or signed by no known authority, or if it is verifiable

but its subject does not correspond to the From header field of

request, the UAS MUST notify its user of the status of the

certificate (including the subject of the certificate, its signer,

and any key fingerprint information) and request explicit permission

before proceeding. If the certificate was successfully verified and

the subject of the certificate corresponds to the From header field

of the SIP request, or if the user (after notification) explicitly

authorizes the use of the certificate, the UAS SHOULD add this

certificate to a local keyring, indexed by the address-of-record of

the holder of the certificate.

When a UAS sends a response containing an S/MIME body that answers

the first request in a dialog, or a response to a non-INVITE request

outside the context of a dialog, the UAS SHOULD structure the body as

an S/MIME 'multipart/signed' CMS SignedData body. If the desired CMS

service is EnvelopedData, the UAS SHOULD send the EnvelopedData

message encapsulated within a SignedData message.

When a UAC receives a response containing an S/MIME CMS body that

includes a certificate, the UAC SHOULD first validate the

certificate, if possible, with any appropriate root certificate. The

UAC SHOULD also determine the subject of the certificate and compare

this value to the To field of the response; although the two may very

well be different, and this is not necessarily indicative of a

security breach. If the certificate cannot be verified because it is

self-signed, or signed by no known authority, the UAC MUST notify its

user of the status of the certificate (including the subject of the

certificate, its signator, and any key fingerprint information) and

request explicit permission before proceeding. If the certificate

was successfully verified, and the subject of the certificate

corresponds to the To header field in the response, or if the user

(after notification) explicitly authorizes the use of the

certificate, the UAC SHOULD add this certificate to a local keyring,

indexed by the address-of-record of the holder of the certificate.

If the UAC had not transmitted its own certificate to the UAS in any

previous transaction, it SHOULD use a CMS SignedData body for its

next request or response.

On future occasions, when the UA receives requests or responses that

contain a From header field corresponding to a value in its keyring,

the UA SHOULD compare the certificate offered in these messages with

the existing certificate in its keyring. If there is a discrepancy,

the UA MUST notify its user of a change of the certificate

(preferably in terms that indicate that this is a potential security

breach) and acquire the user's permission before continuing to

Rosenberg, et. al. Standards Track [Page 203]

RFC 3261 SIP: Session Initiation Protocol June 2002

process the signaling. If the user authorizes this certificate, it

SHOULD be added to the keyring alongside any previous value(s) for

this address-of-record.

Note well however, that this key exchange mechanism does not

guarantee the secure exchange of keys when self-signed certificates,

or certificates signed by an obscure authority, are used - it is

vulnerable to well-known attacks. In the opinion of the authors,

however, the security it provides is proverbially better than

nothing; it is in fact comparable to the widely used SSH application.

These limitations are explored in greater detail in Section 26.4.2.

If a UA receives an S/MIME body that has been encrypted with a public

key unknown to the recipient, it MUST reject the request with a 493

(Undecipherable) response. This response SHOULD contain a valid

certificate for the respondent (corresponding, if possible, to any

address of record given in the To header field of the rejected

request) within a MIME body with a 'certs-only' "smime-type"

parameter.

A 493 (Undecipherable) sent without any certificate indicates that

the respondent cannot or will not utilize S/MIME encrypted messages,

though they may still support S/MIME signatures.

Note that a user agent that receives a request containing an S/MIME

body that is not optional (with a Content-Disposition header

"handling" parameter of "required") MUST reject the request with a

415 Unsupported Media Type response if the MIME type is not

understood. A user agent that receives such a response when S/MIME

is sent SHOULD notify its user that the remote device does not

support S/MIME, and it MAY subsequently resend the request without

S/MIME, if appropriate; however, this 415 response may constitute a

downgrade attack.

If a user agent sends an S/MIME body in a request, but receives a

response that contains a MIME body that is not secured, the UAC

SHOULD notify its user that the session could not be secured.

However, if a user agent that supports S/MIME receives a request with

an unsecured body, it SHOULD NOT respond with a secured body, but if

it expects S/MIME from the sender (for example, because the sender's

From header field value corresponds to an identity on its keychain),

the UAS SHOULD notify its user that the session could not be secured.

A number of conditions that arise in the previous text call for the

notification of the user when an anomalous certificate-management

event occurs. Users might well ask what they should do under these

circumstances. First and foremost, an unexpected change in a

certificate, or an absence of security when security is expected, are

Rosenberg, et. al. Standards Track [Page 204]

RFC 3261 SIP: Session Initiation Protocol June 2002

causes for caution but not necessarily indications that an attack is

in progress. Users might abort any connection attempt or refuse a

connection request they have received; in telephony parlance, they

could hang up and call back. Users may wish to find an alternate

means to contact the other party and confirm that their key has

legitimately changed. Note that users are sometimes compelled to

change their certificates, for example when they suspect that the

secrecy of their private key has been compromised. When their

private key is no longer private, users must legitimately generate a

new key and re-establish trust with any users that held their old

key.

Finally, if during the course of a dialog a UA receives a certificate

in a CMS SignedData message that does not correspond with the

certificates previously exchanged during a dialog, the UA MUST notify

its user of the change, preferably in terms that indicate that this

is a potential security breach.

23.3 Securing MIME bodies

There are two types of secure MIME bodies that are of interest to

SIP: use of these bodies should follow the S/MIME specification [24]

with a few variations.

o "multipart/signed" MUST be used only with CMS detached

signatures.

This allows backwards compatibility with non-S/MIME-

compliant recipients.

o S/MIME bodies SHOULD have a Content-Disposition header field,

and the value of the "handling" parameter SHOULD be "required."

o If a UAC has no certificate on its keyring associated with the

address-of-record to which it wants to send a request, it

cannot send an encrypted "application/pkcs7-mime" MIME message.

UACs MAY send an initial request such as an OPTIONS message

with a CMS detached signature in order to solicit the

certificate of the remote side (the signature SHOULD be over a

"message/sip" body of the type described in Section 23.4).

Note that future standardization work on S/MIME may define

non-certificate based keys.

o Senders of S/MIME bodies SHOULD use the "SMIMECapabilities"

(see Section 2.5.2 of [24]) attribute to express their

capabilities and preferences for further communications. Note

especially that senders MAY use the "preferSignedData"

Rosenberg, et. al. Standards Track [Page 205]

RFC 3261 SIP: Session Initiation Protocol June 2002

capability to encourage receivers to respond with CMS

SignedData messages (for example, when sending an OPTIONS

request as described above).

o S/MIME implementations MUST at a minimum support SHA1 as a

digital signature algorithm, and 3DES as an encryption

algorithm. All other signature and encryption algorithms MAY

be supported. Implementations can negotiate support for these

algorithms with the "SMIMECapabilities" attribute.

o Each S/MIME body in a SIP message SHOULD be signed with only

one certificate. If a UA receives a message with multiple

signatures, the outermost signature should be treated as the

single certificate for this body. Parallel signatures SHOULD

NOT be used.

The following is an example of an encrypted S/MIME SDP body

within a SIP message:

INVITE sip:bob@biloxi.com SIP/2.0

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

To: Bob <sip:bob@biloxi.com>

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 314159 INVITE

Max-Forwards: 70

Contact: <sip:alice@pc33.atlanta.com>

Content-Type: application/pkcs7-mime; smime-type=enveloped-data;

name=smime.p7m

Content-Disposition: attachment; filename=smime.p7m

handling=required

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Content-Type: application/sdp \*

\* \*

\* v=0 \*

\* o=alice 53655765 2353687637 IN IP4 pc33.atlanta.com \*

\* s=- \*

\* t=0 0 \*

\* c=IN IP4 pc33.atlanta.com \*

\* m=audio 3456 RTP/AVP 0 1 3 99 \*

\* a=rtpmap:0 PCMU/8000 \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Rosenberg, et. al. Standards Track [Page 206]

RFC 3261 SIP: Session Initiation Protocol June 2002

23.4 SIP Header Privacy and Integrity using S/MIME: Tunneling SIP

As a means of providing some degree of end-to-end authentication,

integrity or confidentiality for SIP header fields, S/MIME can

encapsulate entire SIP messages within MIME bodies of type

"message/sip" and then apply MIME security to these bodies in the

same manner as typical SIP bodies. These encapsulated SIP requests

and responses do not constitute a separate dialog or transaction,

they are a copy of the "outer" message that is used to verify

integrity or to supply additional information.

If a UAS receives a request that contains a tunneled "message/sip"

S/MIME body, it SHOULD include a tunneled "message/sip" body in the

response with the same smime-type.

Any traditional MIME bodies (such as SDP) SHOULD be attached to the

"inner" message so that they can also benefit from S/MIME security.

Note that "message/sip" bodies can be sent as a part of a MIME

"multipart/mixed" body if any unsecured MIME types should also be

transmitted in a request.

23.4.1 Integrity and Confidentiality Properties of SIP Headers

When the S/MIME integrity or confidentiality mechanisms are used,

there may be discrepancies between the values in the "inner" message

and values in the "outer" message. The rules for handling any such

differences for all of the header fields described in this document

are given in this section.

Note that for the purposes of loose timestamping, all SIP messages

that tunnel "message/sip" SHOULD contain a Date header in both the

"inner" and "outer" headers.

23.4.1.1 Integrity

Whenever integrity checks are performed, the integrity of a header

field should be determined by matching the value of the header field

in the signed body with that in the "outer" messages using the

comparison rules of SIP as described in 20.

Header fields that can be legitimately modified by proxy servers are:

Request-URI, Via, Record-Route, Route, Max-Forwards, and Proxy-

Authorization. If these header fields are not intact end-to-end,

implementations SHOULD NOT consider this a breach of security.

Changes to any other header fields defined in this document

constitute an integrity violation; users MUST be notified of a

discrepancy.

Rosenberg, et. al. Standards Track [Page 207]

RFC 3261 SIP: Session Initiation Protocol June 2002

23.4.1.2 Confidentiality

When messages are encrypted, header fields may be included in the

encrypted body that are not present in the "outer" message.

Some header fields must always have a plaintext version because they

are required header fields in requests and responses - these include:

To, From, Call-ID, CSeq, Contact. While it is probably not useful to

provide an encrypted alternative for the Call-ID, CSeq, or Contact,

providing an alternative to the information in the "outer" To or From

is permitted. Note that the values in an encrypted body are not used

for the purposes of identifying transactions or dialogs - they are

merely informational. If the From header field in an encrypted body

differs from the value in the "outer" message, the value within the

encrypted body SHOULD be displayed to the user, but MUST NOT be used

in the "outer" header fields of any future messages.

Primarily, a user agent will want to encrypt header fields that have

an end-to-end semantic, including: Subject, Reply-To, Organization,

Accept, Accept-Encoding, Accept-Language, Alert-Info, Error-Info,

Authentication-Info, Expires, In-Reply-To, Require, Supported,

Unsupported, Retry-After, User-Agent, Server, and Warning. If any of

these header fields are present in an encrypted body, they should be

used instead of any "outer" header fields, whether this entails

displaying the header field values to users or setting internal

states in the UA. They SHOULD NOT however be used in the "outer"

headers of any future messages.

If present, the Date header field MUST always be the same in the

"inner" and "outer" headers.

Since MIME bodies are attached to the "inner" message,

implementations will usually encrypt MIME-specific header fields,

including: MIME-Version, Content-Type, Content-Length, Content-

Language, Content-Encoding and Content-Disposition. The "outer"

message will have the proper MIME header fields for S/MIME bodies.

These header fields (and any MIME bodies they preface) should be

treated as normal MIME header fields and bodies received in a SIP

message.

It is not particularly useful to encrypt the following header fields:

Min-Expires, Timestamp, Authorization, Priority, and WWW-

Authenticate. This category also includes those header fields that

can be changed by proxy servers (described in the preceding section).

UAs SHOULD never include these in an "inner" message if they are not

Rosenberg, et. al. Standards Track [Page 208]

RFC 3261 SIP: Session Initiation Protocol June 2002

included in the "outer" message. UAs that receive any of these

header fields in an encrypted body SHOULD ignore the encrypted

values.

Note that extensions to SIP may define additional header fields; the

authors of these extensions should describe the integrity and

confidentiality properties of such header fields. If a SIP UA

encounters an unknown header field with an integrity violation, it

MUST ignore the header field.

23.4.2 Tunneling Integrity and Authentication

Tunneling SIP messages within S/MIME bodies can provide integrity for

SIP header fields if the header fields that the sender wishes to

secure are replicated in a "message/sip" MIME body signed with a CMS

detached signature.

Provided that the "message/sip" body contains at least the

fundamental dialog identifiers (To, From, Call-ID, CSeq), then a

signed MIME body can provide limited authentication. At the very

least, if the certificate used to sign the body is unknown to the

recipient and cannot be verified, the signature can be used to

ascertain that a later request in a dialog was transmitted by the

same certificate-holder that initiated the dialog. If the recipient

of the signed MIME body has some stronger incentive to trust the

certificate (they were able to validate it, they acquired it from a

trusted repository, or they have used it frequently) then the

signature can be taken as a stronger assertion of the identity of the

subject of the certificate.

In order to eliminate possible confusions about the addition or

subtraction of entire header fields, senders SHOULD replicate all

header fields from the request within the signed body. Any message

bodies that require integrity protection MUST be attached to the

"inner" message.

If a Date header is present in a message with a signed body, the

recipient SHOULD compare the header field value with its own internal

clock, if applicable. If a significant time discrepancy is detected

(on the order of an hour or more), the user agent SHOULD alert the

user to the anomaly, and note that it is a potential security breach.

If an integrity violation in a message is detected by its recipient,

the message MAY be rejected with a 403 (Forbidden) response if it is

a request, or any existing dialog MAY be terminated. UAs SHOULD

notify users of this circumstance and request explicit guidance on

how to proceed.

Rosenberg, et. al. Standards Track [Page 209]

RFC 3261 SIP: Session Initiation Protocol June 2002

The following is an example of the use of a tunneled "message/sip"

body:

INVITE sip:bob@biloxi.com SIP/2.0

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

To: Bob <sip:bob@biloxi.com>

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 314159 INVITE

Max-Forwards: 70

Date: Thu, 21 Feb 2002 13:02:03 GMT

Contact: <sip:alice@pc33.atlanta.com>

Content-Type: multipart/signed;

protocol="application/pkcs7-signature";

micalg=sha1; boundary=boundary42

Content-Length: 568

--boundary42

Content-Type: message/sip

INVITE sip:bob@biloxi.com SIP/2.0

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

To: Bob <bob@biloxi.com>

From: Alice <alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 314159 INVITE

Max-Forwards: 70

Date: Thu, 21 Feb 2002 13:02:03 GMT

Contact: <sip:alice@pc33.atlanta.com>

Content-Type: application/sdp

Content-Length: 147

v=0

o=UserA 2890844526 2890844526 IN IP4 here.com

s=Session SDP

c=IN IP4 pc33.atlanta.com

t=0 0

m=audio 49172 RTP/AVP 0

a=rtpmap:0 PCMU/8000

--boundary42

Content-Type: application/pkcs7-signature; name=smime.p7s

Content-Transfer-Encoding: base64

Content-Disposition: attachment; filename=smime.p7s;

handling=required

Rosenberg, et. al. Standards Track [Page 210]

RFC 3261 SIP: Session Initiation Protocol June 2002

ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6

4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj

n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4

7GhIGfHfYT64VQbnj756

--boundary42-

23.4.3 Tunneling Encryption

It may also be desirable to use this mechanism to encrypt a

"message/sip" MIME body within a CMS EnvelopedData message S/MIME

body, but in practice, most header fields are of at least some use to

the network; the general use of encryption with S/MIME is to secure

message bodies like SDP rather than message headers. Some

informational header fields, such as the Subject or Organization

could perhaps warrant end-to-end security. Headers defined by future

SIP applications might also require obfuscation.

Another possible application of encrypting header fields is selective

anonymity. A request could be constructed with a From header field

that contains no personal information (for example,

sip:anonymous@anonymizer.invalid). However, a second From header

field containing the genuine address-of-record of the originator

could be encrypted within a "message/sip" MIME body where it will

only be visible to the endpoints of a dialog.

Note that if this mechanism is used for anonymity, the From header

field will no longer be usable by the recipient of a message as an

index to their certificate keychain for retrieving the proper

S/MIME key to associated with the sender. The message must first

be decrypted, and the "inner" From header field MUST be used as an

index.

In order to provide end-to-end integrity, encrypted "message/sip"

MIME bodies SHOULD be signed by the sender. This creates a

"multipart/signed" MIME body that contains an encrypted body and a

signature, both of type "application/pkcs7-mime".

Rosenberg, et. al. Standards Track [Page 211]

RFC 3261 SIP: Session Initiation Protocol June 2002

In the following example, of an encrypted and signed message, the

text boxed in asterisks ("\*") is encrypted:

INVITE sip:bob@biloxi.com SIP/2.0

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

To: Bob <sip:bob@biloxi.com>

From: Anonymous <sip:anonymous@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 314159 INVITE

Max-Forwards: 70

Date: Thu, 21 Feb 2002 13:02:03 GMT

Contact: <sip:pc33.atlanta.com>

Content-Type: multipart/signed;

protocol="application/pkcs7-signature";

micalg=sha1; boundary=boundary42

Content-Length: 568

--boundary42

Content-Type: application/pkcs7-mime; smime-type=enveloped-data;

name=smime.p7m

Content-Transfer-Encoding: base64

Content-Disposition: attachment; filename=smime.p7m

handling=required

Content-Length: 231

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Content-Type: message/sip \*

\* \*

\* INVITE sip:bob@biloxi.com SIP/2.0 \*

\* Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8 \*

\* To: Bob <bob@biloxi.com> \*

\* From: Alice <alice@atlanta.com>;tag=1928301774 \*

\* Call-ID: a84b4c76e66710 \*

\* CSeq: 314159 INVITE \*

\* Max-Forwards: 70 \*

\* Date: Thu, 21 Feb 2002 13:02:03 GMT \*

\* Contact: <sip:alice@pc33.atlanta.com> \*

\* \*

\* Content-Type: application/sdp \*

\* \*

\* v=0 \*

\* o=alice 53655765 2353687637 IN IP4 pc33.atlanta.com \*

\* s=Session SDP \*

\* t=0 0 \*

\* c=IN IP4 pc33.atlanta.com \*

\* m=audio 3456 RTP/AVP 0 1 3 99 \*

\* a=rtpmap:0 PCMU/8000 \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Rosenberg, et. al. Standards Track [Page 212]

RFC 3261 SIP: Session Initiation Protocol June 2002

--boundary42

Content-Type: application/pkcs7-signature; name=smime.p7s

Content-Transfer-Encoding: base64

Content-Disposition: attachment; filename=smime.p7s;

handling=required

ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6

4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj

n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4

7GhIGfHfYT64VQbnj756

--boundary42-

24 Examples

In the following examples, we often omit the message body and the

corresponding Content-Length and Content-Type header fields for

brevity.

24.1 Registration

Bob registers on start-up. The message flow is shown in Figure 9.

Note that the authentication usually required for registration is not

shown for simplicity.

biloxi.com Bob's

registrar softphone

| |

| REGISTER F1 |

|<---------------|

| 200 OK F2 |

|--------------->|

Figure 9: SIP Registration Example

F1 REGISTER Bob -> Registrar

REGISTER sip:registrar.biloxi.com SIP/2.0

Via: SIP/2.0/UDP bobspc.biloxi.com:5060;branch=z9hG4bKnashds7

Max-Forwards: 70

To: Bob <sip:bob@biloxi.com>

From: Bob <sip:bob@biloxi.com>;tag=456248

Call-ID: 843817637684230@998sdasdh09

CSeq: 1826 REGISTER

Contact: <sip:bob@192.0.2.4>

Expires: 7200

Content-Length: 0

Rosenberg, et. al. Standards Track [Page 213]

RFC 3261 SIP: Session Initiation Protocol June 2002

The registration expires after two hours. The registrar responds

with a 200 OK:

F2 200 OK Registrar -> Bob

SIP/2.0 200 OK

Via: SIP/2.0/UDP bobspc.biloxi.com:5060;branch=z9hG4bKnashds7

;received=192.0.2.4

To: Bob <sip:bob@biloxi.com>;tag=2493k59kd

From: Bob <sip:bob@biloxi.com>;tag=456248

Call-ID: 843817637684230@998sdasdh09

CSeq: 1826 REGISTER

Contact: <sip:bob@192.0.2.4>

Expires: 7200

Content-Length: 0

24.2 Session Setup

This example contains the full details of the example session setup

in Section 4. The message flow is shown in Figure 1. Note that

these flows show the minimum required set of header fields - some

other header fields such as Allow and Supported would normally be

present.

F1 INVITE Alice -> atlanta.com proxy

INVITE sip:bob@biloxi.com SIP/2.0

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

Max-Forwards: 70

To: Bob <sip:bob@biloxi.com>

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 314159 INVITE

Contact: <sip:alice@pc33.atlanta.com>

Content-Type: application/sdp

Content-Length: 142

(Alice's SDP not shown)

Rosenberg, et. al. Standards Track [Page 214]

RFC 3261 SIP: Session Initiation Protocol June 2002

F2 100 Trying atlanta.com proxy -> Alice

SIP/2.0 100 Trying

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

;received=192.0.2.1

To: Bob <sip:bob@biloxi.com>

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 314159 INVITE

Content-Length: 0

F3 INVITE atlanta.com proxy -> biloxi.com proxy

INVITE sip:bob@biloxi.com SIP/2.0

Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

;received=192.0.2.1

Max-Forwards: 69

To: Bob <sip:bob@biloxi.com>

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 314159 INVITE

Contact: <sip:alice@pc33.atlanta.com>

Content-Type: application/sdp

Content-Length: 142

(Alice's SDP not shown)

F4 100 Trying biloxi.com proxy -> atlanta.com proxy

SIP/2.0 100 Trying

Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1

;received=192.0.2.2

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

;received=192.0.2.1

To: Bob <sip:bob@biloxi.com>

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 314159 INVITE

Content-Length: 0

Rosenberg, et. al. Standards Track [Page 215]

RFC 3261 SIP: Session Initiation Protocol June 2002

F5 INVITE biloxi.com proxy -> Bob

INVITE sip:bob@192.0.2.4 SIP/2.0

Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bK4b43c2ff8.1

Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1

;received=192.0.2.2

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

;received=192.0.2.1

Max-Forwards: 68

To: Bob <sip:bob@biloxi.com>

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 314159 INVITE

Contact: <sip:alice@pc33.atlanta.com>

Content-Type: application/sdp

Content-Length: 142

(Alice's SDP not shown)

F6 180 Ringing Bob -> biloxi.com proxy

SIP/2.0 180 Ringing

Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bK4b43c2ff8.1

;received=192.0.2.3

Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1

;received=192.0.2.2

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

;received=192.0.2.1

To: Bob <sip:bob@biloxi.com>;tag=a6c85cf

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

Contact: <sip:bob@192.0.2.4>

CSeq: 314159 INVITE

Content-Length: 0

F7 180 Ringing biloxi.com proxy -> atlanta.com proxy

SIP/2.0 180 Ringing

Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1

;received=192.0.2.2

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

;received=192.0.2.1

To: Bob <sip:bob@biloxi.com>;tag=a6c85cf

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

Contact: <sip:bob@192.0.2.4>

CSeq: 314159 INVITE

Content-Length: 0

Rosenberg, et. al. Standards Track [Page 216]

RFC 3261 SIP: Session Initiation Protocol June 2002

F8 180 Ringing atlanta.com proxy -> Alice

SIP/2.0 180 Ringing

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

;received=192.0.2.1

To: Bob <sip:bob@biloxi.com>;tag=a6c85cf

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

Contact: <sip:bob@192.0.2.4>

CSeq: 314159 INVITE

Content-Length: 0

F9 200 OK Bob -> biloxi.com proxy

SIP/2.0 200 OK

Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bK4b43c2ff8.1

;received=192.0.2.3

Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1

;received=192.0.2.2

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

;received=192.0.2.1

To: Bob <sip:bob@biloxi.com>;tag=a6c85cf

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 314159 INVITE

Contact: <sip:bob@192.0.2.4>

Content-Type: application/sdp

Content-Length: 131

(Bob's SDP not shown)

F10 200 OK biloxi.com proxy -> atlanta.com proxy

SIP/2.0 200 OK

Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1

;received=192.0.2.2

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

;received=192.0.2.1

To: Bob <sip:bob@biloxi.com>;tag=a6c85cf

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 314159 INVITE

Contact: <sip:bob@192.0.2.4>

Content-Type: application/sdp

Content-Length: 131

(Bob's SDP not shown)

Rosenberg, et. al. Standards Track [Page 217]

RFC 3261 SIP: Session Initiation Protocol June 2002

F11 200 OK atlanta.com proxy -> Alice

SIP/2.0 200 OK

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

;received=192.0.2.1

To: Bob <sip:bob@biloxi.com>;tag=a6c85cf

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 314159 INVITE

Contact: <sip:bob@192.0.2.4>

Content-Type: application/sdp

Content-Length: 131

(Bob's SDP not shown)

F12 ACK Alice -> Bob

ACK sip:bob@192.0.2.4 SIP/2.0

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds9

Max-Forwards: 70

To: Bob <sip:bob@biloxi.com>;tag=a6c85cf

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 314159 ACK

Content-Length: 0

The media session between Alice and Bob is now established.

Bob hangs up first. Note that Bob's SIP phone maintains its own CSeq

numbering space, which, in this example, begins with 231. Since Bob

is making the request, the To and From URIs and tags have been

swapped.

F13 BYE Bob -> Alice

BYE sip:alice@pc33.atlanta.com SIP/2.0

Via: SIP/2.0/UDP 192.0.2.4;branch=z9hG4bKnashds10

Max-Forwards: 70

From: Bob <sip:bob@biloxi.com>;tag=a6c85cf

To: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 231 BYE

Content-Length: 0

Rosenberg, et. al. Standards Track [Page 218]

RFC 3261 SIP: Session Initiation Protocol June 2002

F14 200 OK Alice -> Bob

SIP/2.0 200 OK

Via: SIP/2.0/UDP 192.0.2.4;branch=z9hG4bKnashds10

From: Bob <sip:bob@biloxi.com>;tag=a6c85cf

To: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 231 BYE

Content-Length: 0

The SIP Call Flows document [40] contains further examples of SIP

messages.

25 Augmented BNF for the SIP Protocol

All of the mechanisms specified in this document are described in

both prose and an augmented Backus-Naur Form (BNF) defined in RFC

2234 [10]. Section 6.1 of RFC 2234 defines a set of core rules that

are used by this specification, and not repeated here. Implementers

need to be familiar with the notation and content of RFC 2234 in

order to understand this specification. Certain basic rules are in

uppercase, such as SP, LWS, HTAB, CRLF, DIGIT, ALPHA, etc. Angle

brackets are used within definitions to clarify the use of rule

names.

The use of square brackets is redundant syntactically. It is used as

a semantic hint that the specific parameter is optional to use.

25.1 Basic Rules

The following rules are used throughout this specification to

describe basic parsing constructs. The US-ASCII coded character set

is defined by ANSI X3.4-1986.

alphanum = ALPHA / DIGIT

Rosenberg, et. al. Standards Track [Page 219]

RFC 3261 SIP: Session Initiation Protocol June 2002

Several rules are incorporated from RFC 2396 [5] but are updated to

make them compliant with RFC 2234 [10]. These include:

reserved = ";" / "/" / "?" / ":" / "@" / "&" / "=" / "+"

/ "$" / ","

unreserved = alphanum / mark

mark = "-" / "\_" / "." / "!" / "~" / "\*" / "'"

/ "(" / ")"

escaped = "%" HEXDIG HEXDIG

SIP header field values can be folded onto multiple lines if the

continuation line begins with a space or horizontal tab. All linear

white space, including folding, has the same semantics as SP. A

recipient MAY replace any linear white space with a single SP before

interpreting the field value or forwarding the message downstream.

This is intended to behave exactly as HTTP/1.1 as described in RFC

2616 [8]. The SWS construct is used when linear white space is

optional, generally between tokens and separators.

LWS = [\*WSP CRLF] 1\*WSP ; linear whitespace

SWS = [LWS] ; sep whitespace

To separate the header name from the rest of value, a colon is used,

which, by the above rule, allows whitespace before, but no line

break, and whitespace after, including a linebreak. The HCOLON

defines this construct.

HCOLON = \*( SP / HTAB ) ":" SWS

The TEXT-UTF8 rule is only used for descriptive field contents and

values that are not intended to be interpreted by the message parser.

Words of \*TEXT-UTF8 contain characters from the UTF-8 charset (RFC

2279 [7]). The TEXT-UTF8-TRIM rule is used for descriptive field

contents that are n t quoted strings, where leading and trailing LWS

is not meaningful. In this regard, SIP differs from HTTP, which uses

the ISO 8859-1 character set.

TEXT-UTF8-TRIM = 1\*TEXT-UTF8char \*(\*LWS TEXT-UTF8char)

TEXT-UTF8char = %x21-7E / UTF8-NONASCII

UTF8-NONASCII = %xC0-DF 1UTF8-CONT

/ %xE0-EF 2UTF8-CONT

/ %xF0-F7 3UTF8-CONT

/ %xF8-Fb 4UTF8-CONT

/ %xFC-FD 5UTF8-CONT

UTF8-CONT = %x80-BF

Rosenberg, et. al. Standards Track [Page 220]

RFC 3261 SIP: Session Initiation Protocol June 2002

A CRLF is allowed in the definition of TEXT-UTF8-TRIM only as part of

a header field continuation. It is expected that the folding LWS

will be replaced with a single SP before interpretation of the TEXT-

UTF8-TRIM value.

Hexadecimal numeric characters are used in several protocol elements.

Some elements (authentication) force hex alphas to be lower case.

LHEX = DIGIT / %x61-66 ;lowercase a-f

Many SIP header field values consist of words separated by LWS or

special characters. Unless otherwise stated, tokens are case-

insensitive. These special characters MUST be in a quoted string to

be used within a parameter value. The word construct is used in

Call-ID to allow most separators to be used.

token = 1\*(alphanum / "-" / "." / "!" / "%" / "\*"

/ "\_" / "+" / "`" / "'" / "~" )

separators = "(" / ")" / "<" / ">" / "@" /

"," / ";" / ":" / "\" / DQUOTE /

"/" / "[" / "]" / "?" / "=" /

"{" / "}" / SP / HTAB

word = 1\*(alphanum / "-" / "." / "!" / "%" / "\*" /

"\_" / "+" / "`" / "'" / "~" /

"(" / ")" / "<" / ">" /

":" / "\" / DQUOTE /

"/" / "[" / "]" / "?" /

"{" / "}" )

When tokens are used or separators are used between elements,

whitespace is often allowed before or after these characters:

STAR = SWS "\*" SWS ; asterisk

SLASH = SWS "/" SWS ; slash

EQUAL = SWS "=" SWS ; equal

LPAREN = SWS "(" SWS ; left parenthesis

RPAREN = SWS ")" SWS ; right parenthesis

RAQUOT = ">" SWS ; right angle quote

LAQUOT = SWS "<"; left angle quote

COMMA = SWS "," SWS ; comma

SEMI = SWS ";" SWS ; semicolon

COLON = SWS ":" SWS ; colon

LDQUOT = SWS DQUOTE; open double quotation mark

RDQUOT = DQUOTE SWS ; close double quotation mark

Rosenberg, et. al. Standards Track [Page 221]

RFC 3261 SIP: Session Initiation Protocol June 2002

Comments can be included in some SIP header fields by surrounding the

comment text with parentheses. Comments are only allowed in fields

containing "comment" as part of their field value definition. In all

other fields, parentheses are considered part of the field value.

comment = LPAREN \*(ctext / quoted-pair / comment) RPAREN

ctext = %x21-27 / %x2A-5B / %x5D-7E / UTF8-NONASCII

/ LWS

ctext includes all chars except left and right parens and backslash.

A string of text is parsed as a single word if it is quoted using

double-quote marks. In quoted strings, quotation marks (") and

backslashes (\) need to be escaped.

quoted-string = SWS DQUOTE \*(qdtext / quoted-pair ) DQUOTE

qdtext = LWS / %x21 / %x23-5B / %x5D-7E

/ UTF8-NONASCII

The backslash character ("\") MAY be used as a single-character

quoting mechanism only within quoted-string and comment constructs.

Unlike HTTP/1.1, the characters CR and LF cannot be escaped by this

mechanism to avoid conflict with line folding and header separation.

quoted-pair = "\" (%x00-09 / %x0B-0C

/ %x0E-7F)

SIP-URI = "sip:" [ userinfo ] hostport

uri-parameters [ headers ]

SIPS-URI = "sips:" [ userinfo ] hostport

uri-parameters [ headers ]

userinfo = ( user / telephone-subscriber ) [ ":" password ] "@"

user = 1\*( unreserved / escaped / user-unreserved )

user-unreserved = "&" / "=" / "+" / "$" / "," / ";" / "?" / "/"

password = \*( unreserved / escaped /

"&" / "=" / "+" / "$" / "," )

hostport = host [ ":" port ]

host = hostname / IPv4address / IPv6reference

hostname = \*( domainlabel "." ) toplabel [ "." ]

domainlabel = alphanum

/ alphanum \*( alphanum / "-" ) alphanum

toplabel = ALPHA / ALPHA \*( alphanum / "-" ) alphanum

Rosenberg, et. al. Standards Track [Page 222]

RFC 3261 SIP: Session Initiation Protocol June 2002

IPv4address = 1\*3DIGIT "." 1\*3DIGIT "." 1\*3DIGIT "." 1\*3DIGIT

IPv6reference = "[" IPv6address "]"

IPv6address = hexpart [ ":" IPv4address ]

hexpart = hexseq / hexseq "::" [ hexseq ] / "::" [ hexseq ]

hexseq = hex4 \*( ":" hex4)

hex4 = 1\*4HEXDIG

port = 1\*DIGIT

The BNF for telephone-subscriber can be found in RFC 2806 [9]. Note,

however, that any characters allowed there that are not allowed in

the user part of the SIP URI MUST be escaped.

uri-parameters = \*( ";" uri-parameter)

uri-parameter = transport-param / user-param / method-param

/ ttl-param / maddr-param / lr-param / other-param

transport-param = "transport="

( "udp" / "tcp" / "sctp" / "tls"

/ other-transport)

other-transport = token

user-param = "user=" ( "phone" / "ip" / other-user)

other-user = token

method-param = "method=" Method

ttl-param = "ttl=" ttl

maddr-param = "maddr=" host

lr-param = "lr"

other-param = pname [ "=" pvalue ]

pname = 1\*paramchar

pvalue = 1\*paramchar

paramchar = param-unreserved / unreserved / escaped

param-unreserved = "[" / "]" / "/" / ":" / "&" / "+" / "$"

headers = "?" header \*( "&" header )

header = hname "=" hvalue

hname = 1\*( hnv-unreserved / unreserved / escaped )

hvalue = \*( hnv-unreserved / unreserved / escaped )

hnv-unreserved = "[" / "]" / "/" / "?" / ":" / "+" / "$"

SIP-message = Request / Response

Request = Request-Line

\*( message-header )

CRLF

[ message-body ]

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

Request-URI = SIP-URI / SIPS-URI / absoluteURI

absoluteURI = scheme ":" ( hier-part / opaque-part )

hier-part = ( net-path / abs-path ) [ "?" query ]

net-path = "//" authority [ abs-path ]

abs-path = "/" path-segments

Rosenberg, et. al. Standards Track [Page 223]

RFC 3261 SIP: Session Initiation Protocol June 2002

opaque-part = uric-no-slash \*uric

uric = reserved / unreserved / escaped

uric-no-slash = unreserved / escaped / ";" / "?" / ":" / "@"

/ "&" / "=" / "+" / "$" / ","

path-segments = segment \*( "/" segment )

segment = \*pchar \*( ";" param )

param = \*pchar

pchar = unreserved / escaped /

":" / "@" / "&" / "=" / "+" / "$" / ","

scheme = ALPHA \*( ALPHA / DIGIT / "+" / "-" / "." )

authority = srvr / reg-name

srvr = [ [ userinfo "@" ] hostport ]

reg-name = 1\*( unreserved / escaped / "$" / ","

/ ";" / ":" / "@" / "&" / "=" / "+" )

query = \*uric

SIP-Version = "SIP" "/" 1\*DIGIT "." 1\*DIGIT

message-header = (Accept

/ Accept-Encoding

/ Accept-Language

/ Alert-Info

/ Allow

/ Authentication-Info

/ Authorization

/ Call-ID

/ Call-Info

/ Contact

/ Content-Disposition

/ Content-Encoding

/ Content-Language

/ Content-Length

/ Content-Type

/ CSeq

/ Date

/ Error-Info

/ Expires

/ From

/ In-Reply-To

/ Max-Forwards

/ MIME-Version

/ Min-Expires

/ Organization

/ Priority

/ Proxy-Authenticate

/ Proxy-Authorization

/ Proxy-Require

/ Record-Route

/ Reply-To

Rosenberg, et. al. Standards Track [Page 224]

RFC 3261 SIP: Session Initiation Protocol June 2002

/ Require

/ Retry-After

/ Route

/ Server

/ Subject

/ Supported

/ Timestamp

/ To

/ Unsupported

/ User-Agent

/ Via

/ Warning

/ WWW-Authenticate

/ extension-header) CRLF

INVITEm = %x49.4E.56.49.54.45 ; INVITE in caps

ACKm = %x41.43.4B ; ACK in caps

OPTIONSm = %x4F.50.54.49.4F.4E.53 ; OPTIONS in caps

BYEm = %x42.59.45 ; BYE in caps

CANCELm = %x43.41.4E.43.45.4C ; CANCEL in caps

REGISTERm = %x52.45.47.49.53.54.45.52 ; REGISTER in caps

Method = INVITEm / ACKm / OPTIONSm / BYEm

/ CANCELm / REGISTERm

/ extension-method

extension-method = token

Response = Status-Line

\*( message-header )

CRLF

[ message-body ]

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

Status-Code = Informational

/ Redirection

/ Success

/ Client-Error

/ Server-Error

/ Global-Failure

/ extension-code

extension-code = 3DIGIT

Reason-Phrase = \*(reserved / unreserved / escaped

/ UTF8-NONASCII / UTF8-CONT / SP / HTAB)

Informational = "100" ; Trying

/ "180" ; Ringing

/ "181" ; Call Is Being Forwarded

/ "182" ; Queued

/ "183" ; Session Progress

Rosenberg, et. al. Standards Track [Page 225]

RFC 3261 SIP: Session Initiation Protocol June 2002

Success = "200" ; OK

Redirection = "300" ; Multiple Choices

/ "301" ; Moved Permanently

/ "302" ; Moved Temporarily

/ "305" ; Use Proxy

/ "380" ; Alternative Service

Client-Error = "400" ; Bad Request

/ "401" ; Unauthorized

/ "402" ; Payment Required

/ "403" ; Forbidden

/ "404" ; Not Found

/ "405" ; Method Not Allowed

/ "406" ; Not Acceptable

/ "407" ; Proxy Authentication Required

/ "408" ; Request Timeout

/ "410" ; Gone

/ "413" ; Request Entity Too Large

/ "414" ; Request-URI Too Large

/ "415" ; Unsupported Media Type

/ "416" ; Unsupported URI Scheme

/ "420" ; Bad Extension

/ "421" ; Extension Required

/ "423" ; Interval Too Brief

/ "480" ; Temporarily not available

/ "481" ; Call Leg/Transaction Does Not Exist

/ "482" ; Loop Detected

/ "483" ; Too Many Hops

/ "484" ; Address Incomplete

/ "485" ; Ambiguous

/ "486" ; Busy Here

/ "487" ; Request Terminated

/ "488" ; Not Acceptable Here

/ "491" ; Request Pending

/ "493" ; Undecipherable

Server-Error = "500" ; Internal Server Error

/ "501" ; Not Implemented

/ "502" ; Bad Gateway

/ "503" ; Service Unavailable

/ "504" ; Server Time-out

/ "505" ; SIP Version not supported

/ "513" ; Message Too Large

Rosenberg, et. al. Standards Track [Page 226]

RFC 3261 SIP: Session Initiation Protocol June 2002

Global-Failure = "600" ; Busy Everywhere

/ "603" ; Decline

/ "604" ; Does not exist anywhere

/ "606" ; Not Acceptable

Accept = "Accept" HCOLON

[ accept-range \*(COMMA accept-range) ]

accept-range = media-range \*(SEMI accept-param)

media-range = ( "\*/\*"

/ ( m-type SLASH "\*" )

/ ( m-type SLASH m-subtype )

) \*( SEMI m-parameter )

accept-param = ("q" EQUAL qvalue) / generic-param

qvalue = ( "0" [ "." 0\*3DIGIT ] )

/ ( "1" [ "." 0\*3("0") ] )

generic-param = token [ EQUAL gen-value ]

gen-value = token / host / quoted-string

Accept-Encoding = "Accept-Encoding" HCOLON

[ encoding \*(COMMA encoding) ]

encoding = codings \*(SEMI accept-param)

codings = content-coding / "\*"

content-coding = token

Accept-Language = "Accept-Language" HCOLON

[ language \*(COMMA language) ]

language = language-range \*(SEMI accept-param)

language-range = ( ( 1\*8ALPHA \*( "-" 1\*8ALPHA ) ) / "\*" )

Alert-Info = "Alert-Info" HCOLON alert-param \*(COMMA alert-param)

alert-param = LAQUOT absoluteURI RAQUOT \*( SEMI generic-param )

Allow = "Allow" HCOLON [Method \*(COMMA Method)]

Authorization = "Authorization" HCOLON credentials

credentials = ("Digest" LWS digest-response)

/ other-response

digest-response = dig-resp \*(COMMA dig-resp)

dig-resp = username / realm / nonce / digest-uri

/ dresponse / algorithm / cnonce

/ opaque / message-qop

/ nonce-count / auth-param

username = "username" EQUAL username-value

username-value = quoted-string

digest-uri = "uri" EQUAL LDQUOT digest-uri-value RDQUOT

digest-uri-value = rquest-uri ; Equal to request-uri as specified

by HTTP/1.1

message-qop = "qop" EQUAL qop-value

Rosenberg, et. al. Standards Track [Page 227]

RFC 3261 SIP: Session Initiation Protocol June 2002

cnonce = "cnonce" EQUAL cnonce-value

cnonce-value = nonce-value

nonce-count = "nc" EQUAL nc-value

nc-value = 8LHEX

dresponse = "response" EQUAL request-digest

request-digest = LDQUOT 32LHEX RDQUOT

auth-param = auth-param-name EQUAL

( token / quoted-string )

auth-param-name = token

other-response = auth-scheme LWS auth-param

\*(COMMA auth-param)

auth-scheme = token

Authentication-Info = "Authentication-Info" HCOLON ainfo

\*(COMMA ainfo)

ainfo = nextnonce / message-qop

/ response-auth / cnonce

/ nonce-count

nextnonce = "nextnonce" EQUAL nonce-value

response-auth = "rspauth" EQUAL response-digest

response-digest = LDQUOT \*LHEX RDQUOT

Call-ID = ( "Call-ID" / "i" ) HCOLON callid

callid = word [ "@" word ]

Call-Info = "Call-Info" HCOLON info \*(COMMA info)

info = LAQUOT absoluteURI RAQUOT \*( SEMI info-param)

info-param = ( "purpose" EQUAL ( "icon" / "info"

/ "card" / token ) ) / generic-param

Contact = ("Contact" / "m" ) HCOLON

( STAR / (contact-param \*(COMMA contact-param)))

contact-param = (name-addr / addr-spec) \*(SEMI contact-params)

name-addr = [ display-name ] LAQUOT addr-spec RAQUOT

addr-spec = SIP-URI / SIPS-URI / absoluteURI

display-name = \*(token LWS)/ quoted-string

contact-params = c-p-q / c-p-expires

/ contact-extension

c-p-q = "q" EQUAL qvalue

c-p-expires = "expires" EQUAL delta-seconds

contact-extension = generic-param

delta-seconds = 1\*DIGIT

Content-Disposition = "Content-Disposition" HCOLON

disp-type \*( SEMI disp-param )

disp-type = "render" / "session" / "icon" / "alert"

/ disp-extension-token

Rosenberg, et. al. Standards Track [Page 228]

RFC 3261 SIP: Session Initiation Protocol June 2002

disp-param = handling-param / generic-param

handling-param = "handling" EQUAL

( "optional" / "required"

/ other-handling )

other-handling = token

disp-extension-token = token

Content-Encoding = ( "Content-Encoding" / "e" ) HCOLON

content-coding \*(COMMA content-coding)

Content-Language = "Content-Language" HCOLON

language-tag \*(COMMA language-tag)

language-tag = primary-tag \*( "-" subtag )

primary-tag = 1\*8ALPHA

subtag = 1\*8ALPHA

Content-Length = ( "Content-Length" / "l" ) HCOLON 1\*DIGIT

Content-Type = ( "Content-Type" / "c" ) HCOLON media-type

media-type = m-type SLASH m-subtype \*(SEMI m-parameter)

m-type = discrete-type / composite-type

discrete-type = "text" / "image" / "audio" / "video"

/ "application" / extension-token

composite-type = "message" / "multipart" / extension-token

extension-token = ietf-token / x-token

ietf-token = token

x-token = "x-" token

m-subtype = extension-token / iana-token

iana-token = token

m-parameter = m-attribute EQUAL m-value

m-attribute = token

m-value = token / quoted-string

CSeq = "CSeq" HCOLON 1\*DIGIT LWS Method

Date = "Date" HCOLON SIP-date

SIP-date = rfc1123-date

rfc1123-date = wkday "," 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 error-uri \*(COMMA error-uri)

Rosenberg, et. al. Standards Track [Page 229]

RFC 3261 SIP: Session Initiation Protocol June 2002

error-uri = LAQUOT absoluteURI RAQUOT \*( SEMI generic-param )

Expires = "Expires" HCOLON delta-seconds

From = ( "From" / "f" ) HCOLON from-spec

from-spec = ( name-addr / addr-spec )

\*( SEMI from-param )

from-param = tag-param / generic-param

tag-param = "tag" EQUAL token

In-Reply-To = "In-Reply-To" HCOLON callid \*(COMMA callid)

Max-Forwards = "Max-Forwards" HCOLON 1\*DIGIT

MIME-Version = "MIME-Version" HCOLON 1\*DIGIT "." 1\*DIGIT

Min-Expires = "Min-Expires" HCOLON delta-seconds

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 challenge

challenge = ("Digest" LWS digest-cln \*(COMMA digest-cln))

/ other-challenge

other-challenge = auth-scheme LWS auth-param

\*(COMMA auth-param)

digest-cln = realm / domain / nonce

/ opaque / stale / algorithm

/ qop-options / auth-param

realm = "realm" EQUAL realm-value

realm-value = quoted-string

domain = "domain" EQUAL LDQUOT URI

\*( 1\*SP URI ) RDQUOT

URI = absoluteURI / abs-path

nonce = "nonce" EQUAL nonce-value

nonce-value = quoted-string

opaque = "opaque" EQUAL quoted-string

stale = "stale" EQUAL ( "true" / "false" )

algorithm = "algorithm" EQUAL ( "MD5" / "MD5-sess"

/ token )

qop-options = "qop" EQUAL LDQUOT qop-value

\*("," qop-value) RDQUOT

qop-value = "auth" / "auth-int" / token

Proxy-Authorization = "Proxy-Authorization" HCOLON credentials

Rosenberg, et. al. Standards Track [Page 230]

RFC 3261 SIP: Session Initiation Protocol June 2002

Proxy-Require = "Proxy-Require" HCOLON option-tag

\*(COMMA option-tag)

option-tag = token

Record-Route = "Record-Route" HCOLON rec-route \*(COMMA rec-route)

rec-route = name-addr \*( SEMI rr-param )

rr-param = generic-param

Reply-To = "Reply-To" HCOLON rplyto-spec

rplyto-spec = ( name-addr / addr-spec )

\*( SEMI rplyto-param )

rplyto-param = generic-param

Require = "Require" HCOLON option-tag \*(COMMA option-tag)

Retry-After = "Retry-After" HCOLON delta-seconds

[ comment ] \*( SEMI retry-param )

retry-param = ("duration" EQUAL delta-seconds)

/ generic-param

Route = "Route" HCOLON route-param \*(COMMA route-param)

route-param = name-addr \*( SEMI rr-param )

Server = "Server" HCOLON server-val \*(LWS server-val)

server-val = product / comment

product = token [SLASH product-version]

product-version = token

Subject = ( "Subject" / "s" ) HCOLON [TEXT-UTF8-TRIM]

Supported = ( "Supported" / "k" ) HCOLON

[option-tag \*(COMMA option-tag)]

Timestamp = "Timestamp" HCOLON 1\*(DIGIT)

[ "." \*(DIGIT) ] [ LWS delay ]

delay = \*(DIGIT) [ "." \*(DIGIT) ]

To = ( "To" / "t" ) HCOLON ( name-addr

/ addr-spec ) \*( SEMI to-param )

to-param = tag-param / generic-param

Unsupported = "Unsupported" HCOLON option-tag \*(COMMA option-tag)

User-Agent = "User-Agent" HCOLON server-val \*(LWS server-val)

Rosenberg, et. al. Standards Track [Page 231]

RFC 3261 SIP: Session Initiation Protocol June 2002

Via = ( "Via" / "v" ) HCOLON via-parm \*(COMMA via-parm)

via-parm = sent-protocol LWS sent-by \*( SEMI via-params )

via-params = via-ttl / via-maddr

/ via-received / via-branch

/ via-extension

via-ttl = "ttl" EQUAL ttl

via-maddr = "maddr" EQUAL host

via-received = "received" EQUAL (IPv4address / IPv6address)

via-branch = "branch" EQUAL token

via-extension = generic-param

sent-protocol = protocol-name SLASH protocol-version

SLASH transport

protocol-name = "SIP" / token

protocol-version = token

transport = "UDP" / "TCP" / "TLS" / "SCTP"

/ other-transport

sent-by = host [ COLON port ]

ttl = 1\*3DIGIT ; 0 to 255

Warning = "Warning" HCOLON warning-value \*(COMMA warning-value)

warning-value = warn-code SP warn-agent SP warn-text

warn-code = 3DIGIT

warn-agent = hostport / pseudonym

; the name or pseudonym of the server adding

; the Warning header, for use in debugging

warn-text = quoted-string

pseudonym = token

WWW-Authenticate = "WWW-Authenticate" HCOLON challenge

extension-header = header-name HCOLON header-value

header-name = token

header-value = \*(TEXT-UTF8char / UTF8-CONT / LWS)

message-body = \*OCTET

26 Security Considerations: Threat Model and Security Usage

Recommendations

SIP is not an easy protocol to secure. Its use of intermediaries,

its multi-faceted trust relationships, its expected usage between

elements with no trust at all, and its user-to-user operation make

security far from trivial. Security solutions are needed that are

deployable today, without extensive coordination, in a wide variety

of environments and usages. In order to meet these diverse needs,

several distinct mechanisms applicable to different aspects and

usages of SIP will be required.

Rosenberg, et. al. Standards Track [Page 232]

RFC 3261 SIP: Session Initiation Protocol June 2002

Note that the security of SIP signaling itself has no bearing on the

security of protocols used in concert with SIP such as RTP, or with

the security implications of any specific bodies SIP might carry

(although MIME security plays a substantial role in securing SIP).

Any media associated with a session can be encrypted end-to-end

independently of any associated SIP signaling. Media encryption is

outside the scope of this document.

The considerations that follow first examine a set of classic threat

models that broadly identify the security needs of SIP. The set of

security services required to address these threats is then detailed,

followed by an explanation of several security mechanisms that can be

used to provide these services. Next, the requirements for

implementers of SIP are enumerated, along with exemplary deployments

in which these security mechanisms could be used to improve the

security of SIP. Some notes on privacy conclude this section.

26.1 Attacks and Threat Models

This section details some threats that should be common to most

deployments of SIP. These threats have been chosen specifically to

illustrate each of the security services that SIP requires.

The following examples by no means provide an exhaustive list of the

threats against SIP; rather, these are "classic" threats that

demonstrate the need for particular security services that can

potentially prevent whole categories of threats.

These attacks assume an environment in which attackers can

potentially read any packet on the network - it is anticipated that

SIP will frequently be used on the public Internet. Attackers on the

network may be able to modify packets (perhaps at some compromised

intermediary). Attackers may wish to steal services, eavesdrop on

communications, or disrupt sessions.

26.1.1 Registration Hijacking

The SIP registration mechanism allows a user agent to identify itself

to a registrar as a device at which a user (designated by an address

of record) is located. A registrar assesses the identity asserted in

the From header field of a REGISTER message to determine whether this

request can modify the contact addresses associated with the

address-of-record in the To header field. While these two fields are

frequently the same, there are many valid deployments in which a

third-party may register contacts on a user's behalf.

Rosenberg, et. al. Standards Track [Page 233]

RFC 3261 SIP: Session Initiation Protocol June 2002

The From header field of a SIP request, however, can be modified

arbitrarily by the owner of a UA, and this opens the door to

malicious registrations. An attacker that successfully impersonates

a party authorized to change contacts associated with an address-of-

record could, for example, de-register all existing contacts for a

URI and then register their own device as the appropriate contact

address, thereby directing all requests for the affected user to the

attacker's device.

This threat belongs to a family of threats that rely on the absence

of cryptographic assurance of a request's originator. Any SIP UAS

that represents a valuable service (a gateway that interworks SIP

requests with traditional telephone calls, for example) might want to

control access to its resources by authenticating requests that it

receives. Even end-user UAs, for example SIP phones, have an

interest in ascertaining the identities of originators of requests.

This threat demonstrates the need for security services that enable

SIP entities to authenticate the originators of requests.

26.1.2 Impersonating a Server

The domain to which a request is destined is generally specified in

the Request-URI. UAs commonly contact a server in this domain

directly in order to deliver a request. However, there is always a

possibility that an attacker could impersonate the remote server, and

that the UA's request could be intercepted by some other party.

For example, consider a case in which a redirect server at one

domain, chicago.com, impersonates a redirect server at another

domain, biloxi.com. A user agent sends a request to biloxi.com, but

the redirect server at chicago.com answers with a forged response

that has appropriate SIP header fields for a response from

biloxi.com. The forged contact addresses in the redirection response

could direct the originating UA to inappropriate or insecure

resources, or simply prevent requests for biloxi.com from succeeding.

This family of threats has a vast membership, many of which are

critical. As a converse to the registration hijacking threat,

consider the case in which a registration sent to biloxi.com is

intercepted by chicago.com, which replies to the intercepted

registration with a forged 301 (Moved Permanently) response. This

response might seem to come from biloxi.com yet designate chicago.com

as the appropriate registrar. All future REGISTER requests from the

originating UA would then go to chicago.com.

Prevention of this threat requires a means by which UAs can

authenticate the servers to whom they send requests.

Rosenberg, et. al. Standards Track [Page 234]

RFC 3261 SIP: Session Initiation Protocol June 2002

26.1.3 Tampering with Message Bodies

As a matter of course, SIP UAs route requests through trusted proxy

servers. Regardless of how that trust is established (authentication

of proxies is discussed elsewhere in this section), a UA may trust a

proxy server to route a request, but not to inspect or possibly

modify the bodies contained in that request.

Consider a UA that is using SIP message bodies to communicate session

encryption keys for a media session. Although it trusts the proxy

server of the domain it is contacting to deliver signaling properly,

it may not want the administrators of that domain to be capable of

decrypting any subsequent media session. Worse yet, if the proxy

server were actively malicious, it could modify the session key,

either acting as a man-in-the-middle, or perhaps changing the

security characteristics requested by the originating UA.

This family of threats applies not only to session keys, but to most

conceivable forms of content carried end-to-end in SIP. These might

include MIME bodies that should be rendered to the user, SDP, or

encapsulated telephony signals, among others. Attackers might

attempt to modify SDP bodies, for example, in order to point RTP

media streams to a wiretapping device in order to eavesdrop on

subsequent voice communications.

Also note that some header fields in SIP are meaningful end-to-end,

for example, Subject. UAs might be protective of these header fields

as well as bodies (a malicious intermediary changing the Subject

header field might make an important request appear to be spam, for

example). However, since many header fields are legitimately

inspected or altered by proxy servers as a request is routed, not all

header fields should be secured end-to-end.

For these reasons, the UA might want to secure SIP message bodies,

and in some limited cases header fields, end-to-end. The security

services required for bodies include confidentiality, integrity, and

authentication. These end-to-end services should be independent of

the means used to secure interactions with intermediaries such as

proxy servers.

26.1.4 Tearing Down Sessions

Once a dialog has been established by initial messaging, subsequent

requests can be sent that modify the state of the dialog and/or

session. It is critical that principals in a session can be certain

that such requests are not forged by attackers.

Rosenberg, et. al. Standards Track [Page 235]

RFC 3261 SIP: Session Initiation Protocol June 2002

Consider a case in which a third-party attacker captures some initial

messages in a dialog shared by two parties in order to learn the

parameters of the session (To tag, From tag, and so forth) and then

inserts a BYE request into the session. The attacker could opt to

forge the request such that it seemed to come from either

participant. Once the BYE is received by its target, the session

will be torn down prematurely.

Similar mid-session threats include the transmission of forged re-

INVITEs that alter the session (possibly to reduce session security

or redirect media streams as part of a wiretapping attack).

The most effective countermeasure to this threat is the

authentication of the sender of the BYE. In this instance, the

recipient needs only know that the BYE came from the same party with

whom the corresponding dialog was established (as opposed to

ascertaining the absolute identity of the sender). Also, if the

attacker is unable to learn the parameters of the session due to

confidentiality, it would not be possible to forge the BYE. However,

some intermediaries (like proxy servers) will need to inspect those

parameters as the session is established.

26.1.5 Denial of Service and Amplification

Denial-of-service attacks focus on rendering a particular network

element unavailable, usually by directing an excessive amount of

network traffic at its interfaces. A distributed denial-of-service

attack allows one network user to cause multiple network hosts to

flood a target host with a large amount of network traffic.

In many architectures, SIP proxy servers face the public Internet in

order to accept requests from worldwide IP endpoints. SIP creates a

number of potential opportunities for distributed denial-of-service

attacks that must be recognized and addressed by the implementers and

operators of SIP systems.

Attackers can create bogus requests that contain a falsified source

IP address and a corresponding Via header field that identify a

targeted host as the originator of the request and then send this

request to a large number of SIP network elements, thereby using

hapless SIP UAs or proxies to generate denial-of-service traffic

aimed at the target.

Similarly, attackers might use falsified Route header field values in

a request that identify the target host and then send such messages

to forking proxies that will amplify messaging sent to the target.

Rosenberg, et. al. Standards Track [Page 236]

RFC 3261 SIP: Session Initiation Protocol June 2002

Record-Route could be used to similar effect when the attacker is

certain that the SIP dialog initiated by the request will result in

numerous transactions originating in the backwards direction.

A number of denial-of-service attacks open up if REGISTER requests

are not properly authenticated and authorized by registrars.

Attackers could de-register some or all users in an administrative

domain, thereby preventing these users from being invited to new

sessions. An attacker could also register a large number of contacts

designating the same host for a given address-of-record in order to

use the registrar and any associated proxy servers as amplifiers in a

denial-of-service attack. Attackers might also attempt to deplete

available memory and disk resources of a registrar by registering

huge numbers of bindings.

The use of multicast to transmit SIP requests can greatly increase

the potential for denial-of-service attacks.

These problems demonstrate a general need to define architectures

that minimize the risks of denial-of-service, and the need to be

mindful in recommendations for security mechanisms of this class of

attacks.

26.2 Security Mechanisms

From the threats described above, we gather that the fundamental

security services required for the SIP protocol are: preserving the

confidentiality and integrity of messaging, preventing replay attacks

or message spoofing, providing for the authentication and privacy of

the participants in a session, and preventing denial-of-service

attacks. Bodies within SIP messages separately require the security

services of confidentiality, integrity, and authentication.

Rather than defining new security mechanisms specific to SIP, SIP

reuses wherever possible existing security models derived from the

HTTP and SMTP space.

Full encryption of messages provides the best means to preserve the

confidentiality of signaling - it can also guarantee that messages

are not modified by any malicious intermediaries. However, SIP

requests and responses cannot be naively encrypted end-to-end in

their entirety because message fields such as the Request-URI, Route,

and Via need to be visible to proxies in most network architectures

so that SIP requests are routed correctly. Note that proxy servers

need to modify some features of messages as well (such as adding Via

header field values) in order for SIP to function. Proxy servers

must therefore be trusted, to some degree, by SIP UAs. To this

purpose, low-layer security mechanisms for SIP are recommended, which

Rosenberg, et. al. Standards Track [Page 237]

RFC 3261 SIP: Session Initiation Protocol June 2002

encrypt the entire SIP requests or responses on the wire on a hop-

by-hop basis, and that allow endpoints to verify the identity of

proxy servers to whom they send requests.

SIP entities also have a need to identify one another in a secure

fashion. When a SIP endpoint asserts the identity of its user to a

peer UA or to a proxy server, that identity should in some way be

verifiable. A cryptographic authentication mechanism is provided in

SIP to address this requirement.

An independent security mechanism for SIP message bodies supplies an

alternative means of end-to-end mutual authentication, as well as

providing a limit on the degree to which user agents must trust

intermediaries.

26.2.1 Transport and Network Layer Security

Transport or network layer security encrypts signaling traffic,

guaranteeing message confidentiality and integrity.

Oftentimes, certificates are used in the establishment of lower-layer

security, and these certificates can also be used to provide a means

of authentication in many architectures.

Two popular alternatives for providing security at the transport and

network layer are, respectively, TLS [25] and IPSec [26].

IPSec is a set of network-layer protocol tools that collectively can

be used as a secure replacement for traditional IP (Internet

Protocol). IPSec is most commonly used in architectures in which a

set of hosts or administrative domains have an existing trust

relationship with one another. IPSec is usually implemented at the

operating system level in a host, or on a security gateway that

provides confidentiality and integrity for all traffic it receives

from a particular interface (as in a VPN architecture). IPSec can

also be used on a hop-by-hop basis.

In many architectures IPSec does not require integration with SIP

applications; IPSec is perhaps best suited to deployments in which

adding security directly to SIP hosts would be arduous. UAs that

have a pre-shared keying relationship with their first-hop proxy

server are also good candidates to use IPSec. Any deployment of

IPSec for SIP would require an IPSec profile describing the protocol

tools that would be required to secure SIP. No such profile is given

in this document.

Rosenberg, et. al. Standards Track [Page 238]

RFC 3261 SIP: Session Initiation Protocol June 2002

TLS provides transport-layer security over connection-oriented

protocols (for the purposes of this document, TCP); "tls" (signifying

TLS over TCP) can be specified as the desired transport protocol

within a Via header field value or a SIP-URI. TLS is most suited to

architectures in which hop-by-hop security is required between hosts

with no pre-existing trust association. For example, Alice trusts

her local proxy server, which after a certificate exchange decides to

trust Bob's local proxy server, which Bob trusts, hence Bob and Alice

can communicate securely.

TLS must be tightly coupled with a SIP application. Note that

transport mechanisms are specified on a hop-by-hop basis in SIP, thus

a UA that sends requests over TLS to a proxy server has no assurance

that TLS will be used end-to-end.

The TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA ciphersuite [6] MUST be supported at

a minimum by implementers when TLS is used in a SIP application. For

purposes of backwards compatibility, proxy servers, redirect servers,

and registrars SHOULD support TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA.

Implementers MAY also support any other ciphersuite.

26.2.2 SIPS URI Scheme

The SIPS URI scheme adheres to the syntax of the SIP URI (described

in 19), although the scheme string is "sips" rather than "sip". The

semantics of SIPS are very different from the SIP URI, however. SIPS

allows resources to specify that they should be reached securely.

A SIPS URI can be used as an address-of-record for a particular user

- the URI by which the user is canonically known (on their business

cards, in the From header field of their requests, in the To header

field of REGISTER requests). When used as the Request-URI of a

request, the SIPS scheme signifies that each hop over which the

request is forwarded, until the request reaches the SIP entity

responsible for the domain portion of the Request-URI, must be

secured with TLS; once it reaches the domain in question it is

handled in accordance with local security and routing policy, quite

possibly using TLS for any last hop to a UAS. When used by the

originator of a request (as would be the case if they employed a SIPS

URI as the address-of-record of the target), SIPS dictates that the

entire request path to the target domain be so secured.

The SIPS scheme is applicable to many of the other ways in which SIP

URIs are used in SIP today in addition to the Request-URI, including

in addresses-of-record, contact addresses (the contents of Contact

headers, including those of REGISTER methods), and Route headers. In

each instance, the SIPS URI scheme allows these existing fields to

Rosenberg, et. al. Standards Track [Page 239]

RFC 3261 SIP: Session Initiation Protocol June 2002

designate secure resources. The manner in which a SIPS URI is

dereferenced in any of these contexts has its own security properties

which are detailed in [4].

The use of SIPS in particular entails that mutual TLS authentication

SHOULD be employed, as SHOULD the ciphersuite

TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA. Certificates received in the

authentication process SHOULD be validated with root certificates

held by the client; failure to validate a certificate SHOULD result

in the failure of the request.

Note that in the SIPS URI scheme, transport is independent of TLS,

and thus "sips:alice@atlanta.com;transport=tcp" and

"sips:alice@atlanta.com;transport=sctp" are both valid (although

note that UDP is not a valid transport for SIPS). The use of

"transport=tls" has consequently been deprecated, partly because

it was specific to a single hop of the request. This is a change

since RFC 2543.

Users that distribute a SIPS URI as an address-of-record may elect to

operate devices that refuse requests over insecure transports.

26.2.3 HTTP Authentication

SIP provides a challenge capability, based on HTTP authentication,

that relies on the 401 and 407 response codes as well as header

fields for carrying challenges and credentials. Without significant

modification, the reuse of the HTTP Digest authentication scheme in

SIP allows for replay protection and one-way authentication.

The usage of Digest authentication in SIP is detailed in Section 22.

26.2.4 S/MIME

As is discussed above, encrypting entire SIP messages end-to-end for

the purpose of confidentiality is not appropriate because network

intermediaries (like proxy servers) need to view certain header

fields in order to route messages correctly, and if these

intermediaries are excluded from security associations, then SIP

messages will essentially be non-routable.

However, S/MIME allows SIP UAs to encrypt MIME bodies within SIP,

securing these bodies end-to-end without affecting message headers.

S/MIME can provide end-to-end confidentiality and integrity for

message bodies, as well as mutual authentication. It is also

possible to use S/MIME to provide a form of integrity and

confidentiality for SIP header fields through SIP message tunneling.

Rosenberg, et. al. Standards Track [Page 240]

RFC 3261 SIP: Session Initiation Protocol June 2002

The usage of S/MIME in SIP is detailed in Section 23.

26.3 Implementing Security Mechanisms

26.3.1 Requirements for Implementers of SIP

Proxy servers, redirect servers, and registrars MUST implement TLS,

and MUST support both mutual and one-way authentication. It is

strongly RECOMMENDED that UAs be capable initiating TLS; UAs MAY also

be capable of acting as a TLS server. Proxy servers, redirect

servers, and registrars SHOULD possess a site certificate whose

subject corresponds to their canonical hostname. UAs MAY have

certificates of their own for mutual authentication with TLS, but no

provisions are set forth in this document for their use. All SIP

elements that support TLS MUST have a mechanism for validating

certificates received during TLS negotiation; this entails possession

of one or more root certificates issued by certificate authorities

(preferably well-known distributors of site certificates comparable

to those that issue root certificates for web browsers).

All SIP elements that support TLS MUST also support the SIPS URI

scheme.

Proxy servers, redirect servers, registrars, and UAs MAY also

implement IPSec or other lower-layer security protocols.

When a UA attempts to contact a proxy server, redirect server, or

registrar, the UAC SHOULD initiate a TLS connection over which it

will send SIP messages. In some architectures, UASs MAY receive

requests over such TLS connections as well.

Proxy servers, redirect servers, registrars, and UAs MUST implement

Digest Authorization, encompassing all of the aspects required in 22.

Proxy servers, redirect servers, and registrars SHOULD be configured

with at least one Digest realm, and at least one "realm" string

supported by a given server SHOULD correspond to the server's

hostname or domainname.

UAs MAY support the signing and encrypting of MIME bodies, and

transference of credentials with S/MIME as described in Section 23.

If a UA holds one or more root certificates of certificate

authorities in order to validate certificates for TLS or IPSec, it

SHOULD be capable of reusing these to verify S/MIME certificates, as

appropriate. A UA MAY hold root certificates specifically for

validating S/MIME certificates.

Rosenberg, et. al. Standards Track [Page 241]

RFC 3261 SIP: Session Initiation Protocol June 2002

Note that is it anticipated that future security extensions may

upgrade the normative strength associated with S/MIME as S/MIME

implementations appear and the problem space becomes better

understood.

26.3.2 Security Solutions

The operation of these security mechanisms in concert can follow the

existing web and email security models to some degree. At a high

level, UAs authenticate themselves to servers (proxy servers,

redirect servers, and registrars) with a Digest username and

password; servers authenticate themselves to UAs one hop away, or to

another server one hop away (and vice versa), with a site certificate

delivered by TLS.

On a peer-to-peer level, UAs trust the network to authenticate one

another ordinarily; however, S/MIME can also be used to provide

direct authentication when the network does not, or if the network

itself is not trusted.

The following is an illustrative example in which these security

mechanisms are used by various UAs and servers to prevent the sorts

of threats described in Section 26.1. While implementers and network

administrators MAY follow the normative guidelines given in the

remainder of this section, these are provided only as example

implementations.

26.3.2.1 Registration

When a UA comes online and registers with its local administrative

domain, it SHOULD establish a TLS connection with its registrar

(Section 10 describes how the UA reaches its registrar). The

registrar SHOULD offer a certificate to the UA, and the site

identified by the certificate MUST correspond with the domain in

which the UA intends to register; for example, if the UA intends to

register the address-of-record 'alice@atlanta.com', the site

certificate must identify a host within the atlanta.com domain (such

as sip.atlanta.com). When it receives the TLS Certificate message,

the UA SHOULD verify the certificate and inspect the site identified

by the certificate. If the certificate is invalid, revoked, or if it

does not identify the appropriate party, the UA MUST NOT send the

REGISTER message and otherwise proceed with the registration.

When a valid certificate has been provided by the registrar, the

UA knows that the registrar is not an attacker who might redirect

the UA, steal passwords, or attempt any similar attacks.

Rosenberg, et. al. Standards Track [Page 242]

RFC 3261 SIP: Session Initiation Protocol June 2002

The UA then creates a REGISTER request that SHOULD be addressed to a

Request-URI corresponding to the site certificate received from the

registrar. When the UA sends the REGISTER request over the existing

TLS connection, the registrar SHOULD challenge the request with a 401

(Proxy Authentication Required) response. The "realm" parameter

within the Proxy-Authenticate header field of the response SHOULD

correspond to the domain previously given by the site certificate.

When the UAC receives the challenge, it SHOULD either prompt the user

for credentials or take an appropriate credential from a keyring

corresponding to the "realm" parameter in the challenge. The

username of this credential SHOULD correspond with the "userinfo"

portion of the URI in the To header field of the REGISTER request.

Once the Digest credentials have been inserted into an appropriate

Proxy-Authorization header field, the REGISTER should be resubmitted

to the registrar.

Since the registrar requires the user agent to authenticate

itself, it would be difficult for an attacker to forge REGISTER

requests for the user's address-of-record. Also note that since

the REGISTER is sent over a confidential TLS connection, attackers

will not be able to intercept the REGISTER to record credentials

for any possible replay attack.

Once the registration has been accepted by the registrar, the UA

SHOULD leave this TLS connection open provided that the registrar

also acts as the proxy server to which requests are sent for users in

this administrative domain. The existing TLS connection will be

reused to deliver incoming requests to the UA that has just completed

registration.

Because the UA has already authenticated the server on the other

side of the TLS connection, all requests that come over this

connection are known to have passed through the proxy server -

attackers cannot create spoofed requests that appear to have been

sent through that proxy server.

26.3.2.2 Interdomain Requests

Now let's say that Alice's UA would like to initiate a session with a

user in a remote administrative domain, namely "bob@biloxi.com". We

will also say that the local administrative domain (atlanta.com) has

a local outbound proxy.

The proxy server that handles inbound requests for an administrative

domain MAY also act as a local outbound proxy; for simplicity's sake

we'll assume this to be the case for atlanta.com (otherwise the user

agent would initiate a new TLS connection to a separate server at

this point). Assuming that the client has completed the registration

Rosenberg, et. al. Standards Track [Page 243]

RFC 3261 SIP: Session Initiation Protocol June 2002

process described in the preceding section, it SHOULD reuse the TLS

connection to the local proxy server when it sends an INVITE request

to another user. The UA SHOULD reuse cached credentials in the

INVITE to avoid prompting the user unnecessarily.

When the local outbound proxy server has validated the credentials

presented by the UA in the INVITE, it SHOULD inspect the Request-URI

to determine how the message should be routed (see [4]). If the

"domainname" portion of the Request-URI had corresponded to the local

domain (atlanta.com) rather than biloxi.com, then the proxy server

would have consulted its location service to determine how best to

reach the requested user.

Had "alice@atlanta.com" been attempting to contact, say,

"alex@atlanta.com", the local proxy would have proxied to the

request to the TLS connection Alex had established with the

registrar when he registered. Since Alex would receive this

request over his authenticated channel, he would be assured that

Alice's request had been authorized by the proxy server of the

local administrative domain.

However, in this instance the Request-URI designates a remote domain.

The local outbound proxy server at atlanta.com SHOULD therefore

establish a TLS connection with the remote proxy server at

biloxi.com. Since both of the participants in this TLS connection

are servers that possess site certificates, mutual TLS authentication

SHOULD occur. Each side of the connection SHOULD verify and inspect

the certificate of the other, noting the domain name that appears in

the certificate for comparison with the header fields of SIP

messages. The atlanta.com proxy server, for example, SHOULD verify

at this stage that the certificate received from the remote side

corresponds with the biloxi.com domain. Once it has done so, and TLS

negotiation has completed, resulting in a secure channel between the

two proxies, the atlanta.com proxy can forward the INVITE request to

biloxi.com.

The proxy server at biloxi.com SHOULD inspect the certificate of the

proxy server at atlanta.com in turn and compare the domain asserted

by the certificate with the "domainname" portion of the From header

field in the INVITE request. The biloxi proxy MAY have a strict

security policy that requires it to reject requests that do not match

the administrative domain from which they have been proxied.

Such security policies could be instituted to prevent the SIP

equivalent of SMTP 'open relays' that are frequently exploited to

generate spam.

Rosenberg, et. al. Standards Track [Page 244]

RFC 3261 SIP: Session Initiation Protocol June 2002

This policy, however, only guarantees that the request came from the

domain it ascribes to itself; it does not allow biloxi.com to

ascertain how atlanta.com authenticated Alice. Only if biloxi.com

has some other way of knowing atlanta.com's authentication policies

could it possibly ascertain how Alice proved her identity.

biloxi.com might then institute an even stricter policy that forbids

requests that come from domains that are not known administratively

to share a common authentication policy with biloxi.com.

Once the INVITE has been approved by the biloxi proxy, the proxy

server SHOULD identify the existing TLS channel, if any, associated

with the user targeted by this request (in this case

"bob@biloxi.com"). The INVITE should be proxied through this channel

to Bob. Since the request is received over a TLS connection that had

previously been authenticated as the biloxi proxy, Bob knows that the

From header field was not tampered with and that atlanta.com has

validated Alice, although not necessarily whether or not to trust

Alice's identity.

Before they forward the request, both proxy servers SHOULD add a

Record-Route header field to the request so that all future requests

in this dialog will pass through the proxy servers. The proxy

servers can thereby continue to provide security services for the

lifetime of this dialog. If the proxy servers do not add themselves

to the Record-Route, future messages will pass directly end-to-end

between Alice and Bob without any security services (unless the two

parties agree on some independent end-to-end security such as

S/MIME). In this respect the SIP trapezoid model can provide a nice

structure where conventions of agreement between the site proxies can

provide a reasonably secure channel between Alice and Bob.

An attacker preying on this architecture would, for example, be

unable to forge a BYE request and insert it into the signaling

stream between Bob and Alice because the attacker has no way of

ascertaining the parameters of the session and also because the

integrity mechanism transitively protects the traffic between

Alice and Bob.

26.3.2.3 Peer-to-Peer Requests

Alternatively, consider a UA asserting the identity

"carol@chicago.com" that has no local outbound proxy. When Carol

wishes to send an INVITE to "bob@biloxi.com", her UA SHOULD initiate

a TLS connection with the biloxi proxy directly (using the mechanism

described in [4] to determine how to best to reach the given

Request-URI). When her UA receives a certificate from the biloxi

proxy, it SHOULD be verified normally before she passes her INVITE

across the TLS connection. However, Carol has no means of proving

Rosenberg, et. al. Standards Track [Page 245]

RFC 3261 SIP: Session Initiation Protocol June 2002

her identity to the biloxi proxy, but she does have a CMS-detached

signature over a "message/sip" body in the INVITE. It is unlikely in

this instance that Carol would have any credentials in the biloxi.com

realm, since she has no formal association with biloxi.com. The

biloxi proxy MAY also have a strict policy that precludes it from

even bothering to challenge requests that do not have biloxi.com in

the "domainname" portion of the From header field - it treats these

users as unauthenticated.

The biloxi proxy has a policy for Bob that all non-authenticated

requests should be redirected to the appropriate contact address

registered against 'bob@biloxi.com', namely <sip:bob@192.0.2.4>.

Carol receives the redirection response over the TLS connection she

established with the biloxi proxy, so she trusts the veracity of the

contact address.

Carol SHOULD then establish a TCP connection with the designated

address and send a new INVITE with a Request-URI containing the

received contact address (recomputing the signature in the body as

the request is readied). Bob receives this INVITE on an insecure

interface, but his UA inspects and, in this instance, recognizes the

From header field of the request and subsequently matches a locally

cached certificate with the one presented in the signature of the

body of the INVITE. He replies in similar fashion, authenticating

himself to Carol, and a secure dialog begins.

Sometimes firewalls or NATs in an administrative domain could

preclude the establishment of a direct TCP connection to a UA. In

these cases, proxy servers could also potentially relay requests

to UAs in a way that has no trust implications (for example,

forgoing an existing TLS connection and forwarding the request

over cleartext TCP) as local policy dictates.

26.3.2.4 DoS Protection

In order to minimize the risk of a denial-of-service attack against

architectures using these security solutions, implementers should

take note of the following guidelines.

When the host on which a SIP proxy server is operating is routable

from the public Internet, it SHOULD be deployed in an administrative

domain with defensive operational policies (blocking source-routed

traffic, preferably filtering ping traffic). Both TLS and IPSec can

also make use of bastion hosts at the edges of administrative domains

that participate in the security associations to aggregate secure

tunnels and sockets. These bastion hosts can also take the brunt of

denial-of-service attacks, ensuring that SIP hosts within the

administrative domain are not encumbered with superfluous messaging.

Rosenberg, et. al. Standards Track [Page 246]

RFC 3261 SIP: Session Initiation Protocol June 2002

No matter what security solutions are deployed, floods of messages

directed at proxy servers can lock up proxy server resources and

prevent desirable traffic from reaching its destination. There is a

computational expense associated with processing a SIP transaction at

a proxy server, and that expense is greater for stateful proxy

servers than it is for stateless proxy servers. Therefore, stateful

proxies are more susceptible to flooding than stateless proxy

servers.

UAs and proxy servers SHOULD challenge questionable requests with

only a single 401 (Unauthorized) or 407 (Proxy Authentication

Required), forgoing the normal response retransmission algorithm, and

thus behaving statelessly towards unauthenticated requests.

Retransmitting the 401 (Unauthorized) or 407 (Proxy Authentication

Required) status response amplifies the problem of an attacker

using a falsified header field value (such as Via) to direct

traffic to a third party.

In summary, the mutual authentication of proxy servers through

mechanisms such as TLS significantly reduces the potential for rogue

intermediaries to introduce falsified requests or responses that can

deny service. This commensurately makes it harder for attackers to

make innocent SIP nodes into agents of amplification.

26.4 Limitations

Although these security mechanisms, when applied in a judicious

manner, can thwart many threats, there are limitations in the scope

of the mechanisms that must be understood by implementers and network

operators.

26.4.1 HTTP Digest

One of the primary limitations of using HTTP Digest in SIP is that

the integrity mechanisms in Digest do not work very well for SIP.

Specifically, they offer protection of the Request-URI and the method

of a message, but not for any of the header fields that UAs would

most likely wish to secure.

The existing replay protection mechanisms described in RFC 2617 also

have some limitations for SIP. The next-nonce mechanism, for

example, does not support pipelined requests. The nonce-count

mechanism should be used for replay protection.

Another limitation of HTTP Digest is the scope of realms. Digest is

valuable when a user wants to authenticate themselves to a resource

with which they have a pre-existing association, like a service

Rosenberg, et. al. Standards Track [Page 247]

RFC 3261 SIP: Session Initiation Protocol June 2002

provider of which the user is a customer (which is quite a common

scenario and thus Digest provides an extremely useful function). By

way of contrast, the scope of TLS is interdomain or multirealm, since

certificates are often globally verifiable, so that the UA can

authenticate the server with no pre-existing association.

26.4.2 S/MIME

The largest outstanding defect with the S/MIME mechanism is the lack

of a prevalent public key infrastructure for end users. If self-

signed certificates (or certificates that cannot be verified by one

of the participants in a dialog) are used, the SIP-based key exchange

mechanism described in Section 23.2 is susceptible to a man-in-the-

middle attack with which an attacker can potentially inspect and

modify S/MIME bodies. The attacker needs to intercept the first

exchange of keys between the two parties in a dialog, remove the

existing CMS-detached signatures from the request and response, and

insert a different CMS-detached signature containing a certificate

supplied by the attacker (but which seems to be a certificate for the

proper address-of-record). Each party will think they have exchanged

keys with the other, when in fact each has the public key of the

attacker.

It is important to note that the attacker can only leverage this

vulnerability on the first exchange of keys between two parties - on

subsequent occasions, the alteration of the key would be noticeable

to the UAs. It would also be difficult for the attacker to remain in

the path of all future dialogs between the two parties over time (as

potentially days, weeks, or years pass).

SSH is susceptible to the same man-in-the-middle attack on the first

exchange of keys; however, it is widely acknowledged that while SSH

is not perfect, it does improve the security of connections. The use

of key fingerprints could provide some assistance to SIP, just as it

does for SSH. For example, if two parties use SIP to establish a

voice communications session, each could read off the fingerprint of

the key they received from the other, which could be compared against

the original. It would certainly be more difficult for the man-in-

the-middle to emulate the voices of the participants than their

signaling (a practice that was used with the Clipper chip-based

secure telephone).

The S/MIME mechanism allows UAs to send encrypted requests without

preamble if they possess a certificate for the destination address-

of-record on their keyring. However, it is possible that any

particular device registered for an address-of-record will not hold

the certificate that has been previously employed by the device's

current user, and that it will therefore be unable to process an

Rosenberg, et. al. Standards Track [Page 248]

RFC 3261 SIP: Session Initiation Protocol June 2002

encrypted request properly, which could lead to some avoidable error

signaling. This is especially likely when an encrypted request is

forked.

The keys associated with S/MIME are most useful when associated with

a particular user (an address-of-record) rather than a device (a UA).

When users move between devices, it may be difficult to transport

private keys securely between UAs; how such keys might be acquired by

a device is outside the scope of this document.

Another, more prosaic difficulty with the S/MIME mechanism is that it

can result in very large messages, especially when the SIP tunneling

mechanism described in Section 23.4 is used. For that reason, it is

RECOMMENDED that TCP should be used as a transport protocol when

S/MIME tunneling is employed.

26.4.3 TLS

The most commonly voiced concern about TLS is that it cannot run over

UDP; TLS requires a connection-oriented underlying transport

protocol, which for the purposes of this document means TCP.

It may also be arduous for a local outbound proxy server and/or

registrar to maintain many simultaneous long-lived TLS connections

with numerous UAs. This introduces some valid scalability concerns,

especially for intensive ciphersuites. Maintaining redundancy of

long-lived TLS connections, especially when a UA is solely

responsible for their establishment, could also be cumbersome.

TLS only allows SIP entities to authenticate servers to which they

are adjacent; TLS offers strictly hop-by-hop security. Neither TLS,

nor any other mechanism specified in this document, allows clients to

authenticate proxy servers to whom they cannot form a direct TCP

connection.

26.4.4 SIPS URIs

Actually using TLS on every segment of a request path entails that

the terminating UAS must be reachable over TLS (perhaps registering

with a SIPS URI as a contact address). This is the preferred use of

SIPS. Many valid architectures, however, use TLS to secure part of

the request path, but rely on some other mechanism for the final hop

to a UAS, for example. Thus SIPS cannot guarantee that TLS usage

will be truly end-to-end. Note that since many UAs will not accept

incoming TLS connections, even those UAs that do support TLS may be

required to maintain persistent TLS connections as described in the

TLS limitations section above in order to receive requests over TLS

as a UAS.

Rosenberg, et. al. Standards Track [Page 249]

RFC 3261 SIP: Session Initiation Protocol June 2002

Location services are not required to provide a SIPS binding for a

SIPS Request-URI. Although location services are commonly populated

by user registrations (as described in Section 10.2.1), various other

protocols and interfaces could conceivably supply contact addresses

for an AOR, and these tools are free to map SIPS URIs to SIP URIs as

appropriate. When queried for bindings, a location service returns

its contact addresses without regard for whether it received a

request with a SIPS Request-URI. If a redirect server is accessing

the location service, it is up to the entity that processes the

Contact header field of a redirection to determine the propriety of

the contact addresses.

Ensuring that TLS will be used for all of the request segments up to

the target domain is somewhat complex. It is possible that

cryptographically authenticated proxy servers along the way that are

non-compliant or compromised may choose to disregard the forwarding

rules associated with SIPS (and the general forwarding rules in

Section 16.6). Such malicious intermediaries could, for example,

retarget a request from a SIPS URI to a SIP URI in an attempt to

downgrade security.

Alternatively, an intermediary might legitimately retarget a request

from a SIP to a SIPS URI. Recipients of a request whose Request-URI

uses the SIPS URI scheme thus cannot assume on the basis of the

Request-URI alone that SIPS was used for the entire request path

(from the client onwards).

To address these concerns, it is RECOMMENDED that recipients of a

request whose Request-URI contains a SIP or SIPS URI inspect the To

header field value to see if it contains a SIPS URI (though note that

it does not constitute a breach of security if this URI has the same

scheme but is not equivalent to the URI in the To header field).

Although clients may choose to populate the Request-URI and To header

field of a request differently, when SIPS is used this disparity

could be interpreted as a possible security violation, and the

request could consequently be rejected by its recipient. Recipients

MAY also inspect the Via header chain in order to double-check

whether or not TLS was used for the entire request path until the

local administrative domain was reached. S/MIME may also be used by

the originating UAC to help ensure that the original form of the To

header field is carried end-to-end.

If the UAS has reason to believe that the scheme of the Request-URI

has been improperly modified in transit, the UA SHOULD notify its

user of a potential security breach.

Rosenberg, et. al. Standards Track [Page 250]

RFC 3261 SIP: Session Initiation Protocol June 2002

As a further measure to prevent downgrade attacks, entities that

accept only SIPS requests MAY also refuse connections on insecure

ports.

End users will undoubtedly discern the difference between SIPS and

SIP URIs, and they may manually edit them in response to stimuli.

This can either benefit or degrade security. For example, if an

attacker corrupts a DNS cache, inserting a fake record set that

effectively removes all SIPS records for a proxy server, then any

SIPS requests that traverse this proxy server may fail. When a user,

however, sees that repeated calls to a SIPS AOR are failing, they

could on some devices manually convert the scheme from SIPS to SIP

and retry. Of course, there are some safeguards against this (if the

destination UA is truly paranoid it could refuse all non-SIPS

requests), but it is a limitation worth noting. On the bright side,

users might also divine that 'SIPS' would be valid even when they are

presented only with a SIP URI.

26.5 Privacy

SIP messages frequently contain sensitive information about their

senders - not just what they have to say, but with whom they

communicate, when they communicate and for how long, and from where

they participate in sessions. Many applications and their users

require that this sort of private information be hidden from any

parties that do not need to know it.

Note that there are also less direct ways in which private

information can be divulged. If a user or service chooses to be

reachable at an address that is guessable from the person's name and

organizational affiliation (which describes most addresses-of-

record), the traditional method of ensuring privacy by having an

unlisted "phone number" is compromised. A user location service can

infringe on the privacy of the recipient of a session invitation by

divulging their specific whereabouts to the caller; an implementation

consequently SHOULD be able to restrict, on a per-user basis, what

kind of location and availability information is given out to certain

classes of callers. This is a whole class of problem that is

expected to be studied further in ongoing SIP work.

In some cases, users may want to conceal personal information in

header fields that convey identity. This can apply not only to the

From and related headers representing the originator of the request,

but also the To - it may not be appropriate to convey to the final

destination a speed-dialing nickname, or an unexpanded identifier for

a group of targets, either of which would be removed from the

Request-URI as the request is routed, but not changed in the To

Rosenberg, et. al. Standards Track [Page 251]

RFC 3261 SIP: Session Initiation Protocol June 2002

header field if the two were initially identical. Thus it MAY be

desirable for privacy reasons to create a To header field that

differs from the Request-URI.

27 IANA Considerations

All method names, header field names, status codes, and option tags

used in SIP applications are registered with IANA through

instructions in an IANA Considerations section in an RFC.

The specification instructs the IANA to create four new sub-

registries under http://www.iana.org/assignments/sip-parameters:

Option Tags, Warning Codes (warn-codes), Methods and Response Codes,

added to the sub-registry of Header Fields that is already present

there.

27.1 Option Tags

This specification establishes the Option Tags sub-registry under

http://www.iana.org/assignments/sip-parameters.

Option tags are used in header fields such as Require, Supported,

Proxy-Require, and Unsupported in support of SIP compatibility

mechanisms for extensions (Section 19.2). The option tag itself is a

string that is associated with a particular SIP option (that is, an

extension). It identifies the option to SIP endpoints.

Option tags are registered by the IANA when they are published in

standards track RFCs. The IANA Considerations section of the RFC

must include the following information, which appears in the IANA

registry along with the RFC number of the publication.

o Name of the option tag. The name MAY be of any length, but

SHOULD be no more than twenty characters long. The name MUST

consist of alphanum (Section 25) characters only.

o Descriptive text that describes the extension.

27.2 Warn-Codes

This specification establishes the Warn-codes sub-registry under

http://www.iana.org/assignments/sip-parameters and initiates its

population with the warn-codes listed in Section 20.43. Additional

warn-codes are registered by RFC publication.

Rosenberg, et. al. Standards Track [Page 252]

RFC 3261 SIP: Session Initiation Protocol June 2002

The descriptive text for the table of warn-codes is:

Warning codes provide information supplemental to the status code in

SIP response messages when the failure of the transaction results

from a Session Description Protocol (SDP) (RFC 2327 [1]) problem.

The "warn-code" consists of three digits. A first digit of "3"

indicates warnings specific to SIP. Until a future specification

describes uses of warn-codes other than 3xx, only 3xx warn-codes may

be registered.

Warnings 300 through 329 are reserved for indicating problems with

keywords in the session description, 330 through 339 are warnings

related to basic network services requested in the session

description, 370 through 379 are warnings related to quantitative QoS

parameters requested in the session description, and 390 through 399

are miscellaneous warnings that do not fall into one of the above

categories.

27.3 Header Field Names

This obsoletes the IANA instructions about the header sub-registry

under http://www.iana.org/assignments/sip-parameters.

The following information needs to be provided in an RFC publication

in order to register a new header field name:

o The RFC number in which the header is registered;

o the name of the header field being registered;

o a compact form version for that header field, if one is

defined;

Some common and widely used header fields MAY be assigned one-letter

compact forms (Section 7.3.3). Compact forms can only be assigned

after SIP working group review, followed by RFC publication.

27.4 Method and Response Codes

This specification establishes the Method and Response-Code sub-

registries under http://www.iana.org/assignments/sip-parameters and

initiates their population as follows. The initial Methods table is:

Rosenberg, et. al. Standards Track [Page 253]

RFC 3261 SIP: Session Initiation Protocol June 2002

INVITE [RFC3261]

ACK [RFC3261]

BYE [RFC3261]

CANCEL [RFC3261]

REGISTER [RFC3261]

OPTIONS [RFC3261]

INFO [RFC2976]

The response code table is initially populated from Section 21, the

portions labeled Informational, Success, Redirection, Client-Error,

Server-Error, and Global-Failure. The table has the following

format:

Type (e.g., Informational)

Number Default Reason Phrase [RFC3261]

The following information needs to be provided in an RFC publication

in order to register a new response code or method:

o The RFC number in which the method or response code is

registered;

o the number of the response code or name of the method being

registered;

o the default reason phrase for that response code, if

applicable;

27.5 The "message/sip" MIME type.

This document registers the "message/sip" MIME media type in order to

allow SIP messages to be tunneled as bodies within SIP, primarily for

end-to-end security purposes. This media type is defined by the

following information:

Media type name: message

Media subtype name: sip

Required parameters: none

Optional parameters: version

version: The SIP-Version number of the enclosed message (e.g.,

"2.0"). If not present, the version defaults to "2.0".

Encoding scheme: SIP messages consist of an 8-bit header

optionally followed by a binary MIME data object. As such, SIP

messages must be treated as binary. Under normal circumstances

SIP messages are transported over binary-capable transports, no

special encodings are needed.

Rosenberg, et. al. Standards Track [Page 254]

RFC 3261 SIP: Session Initiation Protocol June 2002

Security considerations: see below

Motivation and examples of this usage as a security mechanism

in concert with S/MIME are given in 23.4.

27.6 New Content-Disposition Parameter Registrations

This document also registers four new Content-Disposition header

"disposition-types": alert, icon, session and render. The authors

request that these values be recorded in the IANA registry for

Content-Dispositions.

Descriptions of these "disposition-types", including motivation and

examples, are given in Section 20.11.

Short descriptions suitable for the IANA registry are:

alert the body is a custom ring tone to alert the user

icon the body is displayed as an icon to the user

render the body should be displayed to the user

session the body describes a communications session, for

example, as RFC 2327 SDP body

28 Changes From RFC 2543

This RFC revises RFC 2543. It is mostly backwards compatible with

RFC 2543. The changes described here fix many errors discovered in

RFC 2543 and provide information on scenarios not detailed in RFC

2543. The protocol has been presented in a more cleanly layered

model here.

We break the differences into functional behavior that is a

substantial change from RFC 2543, which has impact on

interoperability or correct operation in some cases, and functional

behavior that is different from RFC 2543 but not a potential source

of interoperability problems. There have been countless

clarifications as well, which are not documented here.

28.1 Major Functional Changes

o When a UAC wishes to terminate a call before it has been answered,

it sends CANCEL. If the original INVITE still returns a 2xx, the

UAC then sends BYE. BYE can only be sent on an existing call leg

(now called a dialog in this RFC), whereas it could be sent at any

time in RFC 2543.

o The SIP BNF was converted to be RFC 2234 compliant.

Rosenberg, et. al. Standards Track [Page 255]

RFC 3261 SIP: Session Initiation Protocol June 2002

o SIP URL BNF was made more general, allowing a greater set of

characters in the user part. Furthermore, comparison rules were

simplified to be primarily case-insensitive, and detailed handling

of comparison in the presence of parameters was described. The

most substantial change is that a URI with a parameter with the

default value does not match a URI without that parameter.

o Removed Via hiding. It had serious trust issues, since it relied

on the next hop to perform the obfuscation process. Instead, Via

hiding can be done as a local implementation choice in stateful

proxies, and thus is no longer documented.

o In RFC 2543, CANCEL and INVITE transactions were intermingled.

They are separated now. When a user sends an INVITE and then a

CANCEL, the INVITE transaction still terminates normally. A UAS

needs to respond to the original INVITE request with a 487

response.

o Similarly, CANCEL and BYE transactions were intermingled; RFC 2543

allowed the UAS not to send a response to INVITE when a BYE was

received. That is disallowed here. The original INVITE needs a

response.

o In RFC 2543, UAs needed to support only UDP. In this RFC, UAs

need to support both UDP and TCP.

o In RFC 2543, a forking proxy only passed up one challenge from

downstream elements in the event of multiple challenges. In this

RFC, proxies are supposed to collect all challenges and place them

into the forwarded response.

o In Digest credentials, the URI needs to be quoted; this is unclear

from RFC 2617 and RFC 2069 which are both inconsistent on it.

o SDP processing has been split off into a separate specification

[13], and more fully specified as a formal offer/answer exchange

process that is effectively tunneled through SIP. SDP is allowed

in INVITE/200 or 200/ACK for baseline SIP implementations; RFC

2543 alluded to the ability to use it in INVITE, 200, and ACK in a

single transaction, but this was not well specified. More complex

SDP usages are allowed in extensions.

Rosenberg, et. al. Standards Track [Page 256]

RFC 3261 SIP: Session Initiation Protocol June 2002

o Added full support for IPv6 in URIs and in the Via header field.

Support for IPv6 in Via has required that its header field

parameters allow the square bracket and colon characters. These

characters were previously not permitted. In theory, this could

cause interop problems with older implementations. However, we

have observed that most implementations accept any non-control

ASCII character in these parameters.

o DNS SRV procedure is now documented in a separate specification

[4]. This procedure uses both SRV and NAPTR resource records and

no longer combines data from across SRV records as described in

RFC 2543.

o Loop detection has been made optional, supplanted by a mandatory

usage of Max-Forwards. The loop detection procedure in RFC 2543

had a serious bug which would report "spirals" as an error

condition when it was not. The optional loop detection procedure

is more fully and correctly specified here.

o Usage of tags is now mandatory (they were optional in RFC 2543),

as they are now the fundamental building blocks of dialog

identification.

o Added the Supported header field, allowing for clients to indicate

what extensions are supported to a server, which can apply those

extensions to the response, and indicate their usage with a

Require in the response.

o Extension parameters were missing from the BNF for several header

fields, and they have been added.

o Handling of Route and Record-Route construction was very

underspecified in RFC 2543, and also not the right approach. It

has been substantially reworked in this specification (and made

vastly simpler), and this is arguably the largest change.

Backwards compatibility is still provided for deployments that do

not use "pre-loaded routes", where the initial request has a set

of Route header field values obtained in some way outside of

Record-Route. In those situations, the new mechanism is not

interoperable.

o In RFC 2543, lines in a message could be terminated with CR, LF,

or CRLF. This specification only allows CRLF.

Rosenberg, et. al. Standards Track [Page 257]

RFC 3261 SIP: Session Initiation Protocol June 2002

o Usage of Route in CANCEL and ACK was not well defined in RFC 2543.

It is now well specified; if a request had a Route header field,

its CANCEL or ACK for a non-2xx response to the request need to

carry the same Route header field values. ACKs for 2xx responses

use the Route values learned from the Record-Route of the 2xx

responses.

o RFC 2543 allowed multiple requests in a single UDP packet. This

usage has been removed.

o Usage of absolute time in the Expires header field and parameter

has been removed. It caused interoperability problems in elements

that were not time synchronized, a common occurrence. Relative

times are used instead.

o The branch parameter of the Via header field value is now

mandatory for all elements to use. It now plays the role of a

unique transaction identifier. This avoids the complex and bug-

laden transaction identification rules from RFC 2543. A magic

cookie is used in the parameter value to determine if the previous

hop has made the parameter globally unique, and comparison falls

back to the old rules when it is not present. Thus,

interoperability is assured.

o In RFC 2543, closure of a TCP connection was made equivalent to a

CANCEL. This was nearly impossible to implement (and wrong) for

TCP connections between proxies. This has been eliminated, so

that there is no coupling between TCP connection state and SIP

processing.

o RFC 2543 was silent on whether a UA could initiate a new

transaction to a peer while another was in progress. That is now

specified here. It is allowed for non-INVITE requests, disallowed

for INVITE.

o PGP was removed. It was not sufficiently specified, and not

compatible with the more complete PGP MIME. It was replaced with

S/MIME.

o Added the "sips" URI scheme for end-to-end TLS. This scheme is

not backwards compatible with RFC 2543. Existing elements that

receive a request with a SIPS URI scheme in the Request-URI will

likely reject the request. This is actually a feature; it ensures

that a call to a SIPS URI is only delivered if all path hops can

be secured.

Rosenberg, et. al. Standards Track [Page 258]

RFC 3261 SIP: Session Initiation Protocol June 2002

o Additional security features were added with TLS, and these are

described in a much larger and complete security considerations

section.

o In RFC 2543, a proxy was not required to forward provisional

responses from 101 to 199 upstream. This was changed to MUST.

This is important, since many subsequent features depend on

delivery of all provisional responses from 101 to 199.

o Little was said about the 503 response code in RFC 2543. It has

since found substantial use in indicating failure or overload

conditions in proxies. This requires somewhat special treatment.

Specifically, receipt of a 503 should trigger an attempt to

contact the next element in the result of a DNS SRV lookup. Also,

503 response is only forwarded upstream by a proxy under certain

conditions.

o RFC 2543 defined, but did no sufficiently specify, a mechanism for

UA authentication of a server. That has been removed. Instead,

the mutual authentication procedures of RFC 2617 are allowed.

o A UA cannot send a BYE for a call until it has received an ACK for

the initial INVITE. This was allowed in RFC 2543 but leads to a

potential race condition.

o A UA or proxy cannot send CANCEL for a transaction until it gets a

provisional response for the request. This was allowed in RFC

2543 but leads to potential race conditions.

o The action parameter in registrations has been deprecated. It was

insufficient for any useful services, and caused conflicts when

application processing was applied in proxies.

o RFC 2543 had a number of special cases for multicast. For

example, certain responses were suppressed, timers were adjusted,

and so on. Multicast now plays a more limited role, and the

protocol operation is unaffected by usage of multicast as opposed

to unicast. The limitations as a result of that are documented.

o Basic authentication has been removed entirely and its usage

forbidden.

Rosenberg, et. al. Standards Track [Page 259]

RFC 3261 SIP: Session Initiation Protocol June 2002

o Proxies no longer forward a 6xx immediately on receiving it.

Instead, they CANCEL pending branches immediately. This avoids a

potential race condition that would result in a UAC getting a 6xx

followed by a 2xx. In all cases except this race condition, the

result will be the same - the 6xx is forwarded upstream.

o RFC 2543 did not address the problem of request merging. This

occurs when a request forks at a proxy and later rejoins at an

element. Handling of merging is done only at a UA, and procedures

are defined for rejecting all but the first request.

28.2 Minor Functional Changes

o Added the Alert-Info, Error-Info, and Call-Info header fields for

optional content presentation to users.

o Added the Content-Language, Content-Disposition and MIME-Version

header fields.

o Added a "glare handling" mechanism to deal with the case where

both parties send each other a re-INVITE simultaneously. It uses

the new 491 (Request Pending) error code.

o Added the In-Reply-To and Reply-To header fields for supporting

the return of missed calls or messages at a later time.

o Added TLS and SCTP as valid SIP transports.

o There were a variety of mechanisms described for handling failures

at any time during a call; those are now generally unified. BYE

is sent to terminate.

o RFC 2543 mandated retransmission of INVITE responses over TCP, but

noted it was really only needed for 2xx. That was an artifact of

insufficient protocol layering. With a more coherent transaction

layer defined here, that is no longer needed. Only 2xx responses

to INVITEs are retransmitted over TCP.

o Client and server transaction machines are now driven based on

timeouts rather than retransmit counts. This allows the state

machines to be properly specified for TCP and UDP.

o The Date header field is used in REGISTER responses to provide a

simple means for auto-configuration of dates in user agents.

o Allowed a registrar to reject registrations with expirations that

are too short in duration. Defined the 423 response code and the

Min-Expires for this purpose.

Rosenberg, et. al. Standards Track [Page 260]

RFC 3261 SIP: Session Initiation Protocol June 2002

29 Normative References

[1] Handley, M. and V. Jacobson, "SDP: Session Description

Protocol", RFC 2327, April 1998.

[2] Bradner, S., "Key words for use in RFCs to Indicate Requirement

Levels", BCP 14, RFC 2119, March 1997.

[3] Resnick, P., "Internet Message Format", RFC 2822, April 2001.

[4] Rosenberg, J. and H. Schulzrinne, "SIP: Locating SIP Servers",

RFC 3263, June 2002.

[5] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource

Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[6] Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for

Transport Layer Security (TLS)", RFC 3268, June 2002.

[7] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC

2279, January 1998.

[8] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L.,

Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol --

HTTP/1.1", RFC 2616, June 1999.

[9] Vaha-Sipila, A., "URLs for Telephone Calls", RFC 2806, April

2000.

[10] Crocker, D. and P. Overell, "Augmented BNF for Syntax

Specifications: ABNF", RFC 2234, November 1997.

[11] Freed, F. and N. Borenstein, "Multipurpose Internet Mail

Extensions (MIME) Part Two: Media Types", RFC 2046, November

1996.

[12] Eastlake, D., Crocker, S. and J. Schiller, "Randomness

Recommendations for Security", RFC 1750, December 1994.

[13] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with

SDP", RFC 3264, June 2002.

[14] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August

1980.

[15] Postel, J., "DoD Standard Transmission Control Protocol", RFC

761, January 1980.

Rosenberg, et. al. Standards Track [Page 261]

RFC 3261 SIP: Session Initiation Protocol June 2002

[16] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer,

H., Taylor, T., Rytina, I., Kalla, M., Zhang, L. and V. Paxson,

"Stream Control Transmission Protocol", RFC 2960, October 2000.

[17] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S.,

Leach, P., Luotonen, A. and L. Stewart, "HTTP authentication:

Basic and Digest Access Authentication", RFC 2617, June 1999.

[18] Troost, R., Dorner, S. and K. Moore, "Communicating Presentation

Information in Internet Messages: The Content-Disposition Header

Field", RFC 2183, August 1997.

[19] Zimmerer, E., Peterson, J., Vemuri, A., Ong, L., Audet, F.,

Watson, M. and M. Zonoun, "MIME media types for ISUP and QSIG

Objects", RFC 3204, December 2001.

[20] Braden, R., "Requirements for Internet Hosts - Application and

Support", STD 3, RFC 1123, October 1989.

[21] Alvestrand, H., "IETF Policy on Character Sets and Languages",

BCP 18, RFC 2277, January 1998.

[22] Galvin, J., Murphy, S., Crocker, S. and N. Freed, "Security

Multiparts for MIME: Multipart/Signed and Multipart/Encrypted",

RFC 1847, October 1995.

[23] Housley, R., "Cryptographic Message Syntax", RFC 2630, June

1999.

[24] Ramsdell B., "S/MIME Version 3 Message Specification", RFC 2633,

June 1999.

[25] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC

2246, January 1999.

[26] Kent, S. and R. Atkinson, "Security Architecture for the

Internet Protocol", RFC 2401, November 1998.

30 Informative References

[27] R. Pandya, "Emerging mobile and personal communication systems,"

IEEE Communications Magazine, Vol. 33, pp. 44--52, June 1995.

[28] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson,

"RTP: A Transport Protocol for Real-Time Applications", RFC

1889, January 1996.

Rosenberg, et. al. Standards Track [Page 262]

RFC 3261 SIP: Session Initiation Protocol June 2002

[29] Schulzrinne, H., Rao, R. and R. Lanphier, "Real Time Streaming

Protocol (RTSP)", RFC 2326, April 1998.

[30] Cuervo, F., Greene, N., Rayhan, A., Huitema, C., Rosen, B. and

J. Segers, "Megaco Protocol Version 1.0", RFC 3015, November

2000.

[31] Handley, M., Schulzrinne, H., Schooler, E. and J. Rosenberg,

"SIP: Session Initiation Protocol", RFC 2543, March 1999.

[32] Hoffman, P., Masinter, L. and J. Zawinski, "The mailto URL

scheme", RFC 2368, July 1998.

[33] E. M. Schooler, "A multicast user directory service for

synchronous rendezvous," Master's Thesis CS-TR-96-18, Department

of Computer Science, California Institute of Technology,

Pasadena, California, Aug. 1996.

[34] Donovan, S., "The SIP INFO Method", RFC 2976, October 2000.

[35] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April

1992.

[36] Dawson, F. and T. Howes, "vCard MIME Directory Profile", RFC

2426, September 1998.

[37] Good, G., "The LDAP Data Interchange Format (LDIF) - Technical

Specification", RFC 2849, June 2000.

[38] Palme, J., "Common Internet Message Headers", RFC 2076,

February 1997.

[39] Franks, J., Hallam-Baker, P., Hostetler, J., Leach, P.,

Luotonen, A., Sink, E. and L. Stewart, "An Extension to HTTP:

Digest Access Authentication", RFC 2069, January 1997.

[40] Johnston, A., Donovan, S., Sparks, R., Cunningham, C., Willis,

D., Rosenberg, J., Summers, K. and H. Schulzrinne, "SIP Call

Flow Examples", Work in Progress.

[41] E. M. Schooler, "Case study: multimedia conference control in a

packet-switched teleconferencing system," Journal of

Internetworking: Research and Experience, Vol. 4, pp. 99--120,

June 1993. ISI reprint series ISI/RS-93-359.

Rosenberg, et. al. Standards Track [Page 263]

RFC 3261 SIP: Session Initiation Protocol June 2002

[42] H. Schulzrinne, "Personal mobility for multimedia services in

the Internet," in European Workshop on Interactive Distributed

Multimedia Systems and Services (IDMS), (Berlin, Germany), Mar.

1996.

[43] Floyd, S., "Congestion Control Principles", RFC 2914, September

2000.

Rosenberg, et. al. Standards Track [Page 264]

RFC 3261 SIP: Session Initiation Protocol June 2002

A Table of Timer Values

Table 4 summarizes the meaning and defaults of the various timers

used by this specification.

Timer Value Section Meaning

----------------------------------------------------------------------

T1 500ms default Section 17.1.1.1 RTT Estimate

T2 4s Section 17.1.2.2 The maximum retransmit

interval for non-INVITE

requests and INVITE

responses

T4 5s Section 17.1.2.2 Maximum duration a

message will

remain in the network

Timer A initially T1 Section 17.1.1.2 INVITE request retransmit

interval, for UDP only

Timer B 64\*T1 Section 17.1.1.2 INVITE transaction

timeout timer

Timer C > 3min Section 16.6 proxy INVITE transaction

bullet 11 timeout

Timer D > 32s for UDP Section 17.1.1.2 Wait time for response

0s for TCP/SCTP retransmits

Timer E initially T1 Section 17.1.2.2 non-INVITE request

retransmit interval,

UDP only

Timer F 64\*T1 Section 17.1.2.2 non-INVITE transaction

timeout timer

Timer G initially T1 Section 17.2.1 INVITE response

retransmit interval

Timer H 64\*T1 Section 17.2.1 Wait time for

ACK receipt

Timer I T4 for UDP Section 17.2.1 Wait time for

0s for TCP/SCTP ACK retransmits

Timer J 64\*T1 for UDP Section 17.2.2 Wait time for

0s for TCP/SCTP non-INVITE request

retransmits

Timer K T4 for UDP Section 17.1.2.2 Wait time for

0s for TCP/SCTP response retransmits

Table 4: Summary of timers

Rosenberg, et. al. Standards Track [Page 265]

RFC 3261 SIP: Session Initiation Protocol June 2002

Acknowledgments

We wish to thank the members of the IETF MMUSIC and SIP WGs for their

comments and suggestions. Detailed comments were provided by Ofir

Arkin, Brian Bidulock, Jim Buller, Neil Deason, Dave Devanathan,

Keith Drage, Bill Fenner, Cedric Fluckiger, Yaron Goland, John

Hearty, Bernie Hoeneisen, Jo Hornsby, Phil Hoffer, Christian Huitema,

Hisham Khartabil, Jean Jervis, Gadi Karmi, Peter Kjellerstedt, Anders

Kristensen, Jonathan Lennox, Gethin Liddell, Allison Mankin, William

Marshall, Rohan Mahy, Keith Moore, Vern Paxson, Bob Penfield, Moshe

J. Sambol, Chip Sharp, Igor Slepchin, Eric Tremblay, and Rick

Workman.

Brian Rosen provided the compiled BNF.

Jean Mahoney provided technical writing assistance.

This work is based, inter alia, on [41,42].

Rosenberg, et. al. Standards Track [Page 266]

RFC 3261 SIP: Session Initiation Protocol June 2002

Authors' Addresses

Authors addresses are listed alphabetically for the editors, the

writers, and then the original authors of RFC 2543. All listed

authors actively contributed large amounts of text to this document.

Jonathan Rosenberg

dynamicsoft

72 Eagle Rock Ave

East Hanover, NJ 07936

USA

EMail: jdrosen@dynamicsoft.com

Henning Schulzrinne

Dept. of Computer Science

Columbia University

1214 Amsterdam Avenue

New York, NY 10027

USA

EMail: schulzrinne@cs.columbia.edu

Gonzalo Camarillo

Ericsson

Advanced Signalling Research Lab.

FIN-02420 Jorvas

Finland

EMail: Gonzalo.Camarillo@ericsson.com

Alan Johnston

WorldCom

100 South 4th Street

St. Louis, MO 63102

USA

EMail: alan.johnston@wcom.com

Rosenberg, et. al. Standards Track [Page 267]

RFC 3261 SIP: Session Initiation Protocol June 2002

Jon Peterson

NeuStar, Inc

1800 Sutter Street, Suite 570

Concord, CA 94520

USA

EMail: jon.peterson@neustar.com

Robert Sparks

dynamicsoft, Inc.

5100 Tennyson Parkway

Suite 1200

Plano, Texas 75024

USA

EMail: rsparks@dynamicsoft.com

Mark Handley

International Computer Science Institute

1947 Center St, Suite 600

Berkeley, CA 94704

USA

EMail: mjh@icir.org

Eve Schooler

AT&T Labs-Research

75 Willow Road

Menlo Park, CA 94025

USA

EMail: schooler@research.att.com

Rosenberg, et. al. Standards Track [Page 268]

RFC 3261 SIP: Session Initiation Protocol June 2002

Full Copyright Statement

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

This document and translations of it may be copied and furnished to

others, and derivative works that comment on or otherwise explain it

or assist in its implementation may be prepared, copied, published

and distributed, in whole or in part, without restriction of any

kind, provided that the above copyright notice and this paragraph are

included on all such copies and derivative works. However, this

document itself may not be modified in any way, such as by removing

the copyright notice or references to the Internet Society or other

Internet organizations, except as needed for the purpose of

developing Internet standards in which case the procedures for

copyrights defined in the Internet Standards process must be

followed, or as required to translate it into languages other than

English.

The limited permissions granted above are perpetual and will not be

revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an

"AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING

TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING

BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION

HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF

MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

Funding for the RFC Editor function is currently provided by the

Internet Society.

Rosenberg, et. al. Standards Track [Page 269]