SIP

Understanding the Session Initiation Protocol

Third Edition

Alan B. Johnston

SIP: Understanding the Session Initiation Protocol

Third Edition

For a complete listing of titles in the
Artech House Telecommunications Series,
turn to the back of this book.

SIP: Understanding the Session Initiation Protocol

Third Edition

Alan B. Johnston

ARTECH
HOUSE

BOSTON | LONDON
artechhouse.com

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the U.S. Library of Congress.

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library.

Cover design by Yekaterina Ratner
Cover art by Lisa Johnston

ISBN 13: 978-1-60783-995-8

685 Canton Street
Norwood, MA 02062

10 9 8 7 6 5 4 3 2 1

For Lisa

Contents

Foreword to the First Edition *xxi*

Preface to the Third Edition *xxiii*

Preface to the Second Edition *xxv*

Preface to the First Edition *xxvii*

1 SIP and the Internet 1

1.1 Signaling Protocols 1

1.2 Internet Multimedia Protocol Stack 2

1.2.1 Physical Layer 2

1.2.2 Data/Link Layer 2

1.2.3 Network Layer 3

1.2.4 Transport Layer 4

1.2.5 Application Layer 9

1.2.6 Utility Applications 9

1.2.7 Multicast 10

1.3 Internet Names 11

1.4 URLs, URIs, and URNs 11

1.5 Domain Name Service 13

1.5.1 DNS Resource Records 14

1.5.2 Address Resource Records (A or AAAA) 15

1.5.3 Service Resource Records (SRV) 15
1.5.4 Naming Authority Pointer Resource Records (NAPTR) 16
1.5.5 DNS Resolvers 16
1.6 Global Open Standards 17
1.7 Internet Standards Process 18
1.8 A Brief History of SIP 20
1.9 Conclusion 21
References 21

2 Introduction to SIP 23

2.1 A Simple Session Establishment Example 23
2.2 SIP Call with a Proxy Server 31
2.3 SIP Registration Example 36
2.4 SIP Presence and Instant Message Example 38
2.5 Message Transport 43
2.5.1 UDP Transport 43
2.5.2 TCP Transport 45
2.5.3 TLS Transport 46
2.5.4 SCTP Transport 46
2.6 Transport Protocol Selection 47
2.7 Conclusion 48
2.8 Questions 48
References 50

3 SIP Clients and Servers 51

3.1 SIP User Agents 51
3.2 Presence Agents 52
3.3 Back-to-Back User Agents 53
3.4 SIP Gateways 54
3.5 SIP Servers 56
3.5.1 Proxy Servers 56
3.5.2 Redirect Servers 61
3.5.3 Registrar Servers 63
3.6 Uniform Resource Indicators 64
3.7 Acknowledgment of Messages 65

3.8 Reliability 66
3.9 Multicast Support 68
3.10 Conclusion 69
3.11 Questions 69
References 72

4 SIP Request Messages 73

4.1 Methods 73
4.1.1 INVITE 73
4.1.2 REGISTER 76
4.1.3 BYE 78
4.1.4 ACK 78
4.1.5 CANCEL 81
4.1.6 OPTIONS 82
4.1.7 SUBSCRIBE 84
4.1.8 NOTIFY 87
4.1.9 PUBLISH 88
4.1.10 REFER 91
4.1.11 MESSAGE 94
4.1.12 INFO 96
4.1.13 PRACK 97
4.1.14 UPDATE 99
4.2 URI and URL Schemes Used by SIP 100
4.2.1 SIP and SIPS URIs 101
4.2.2 Telephone URLs 102
4.2.3 Presence and Instant Messaging URLs 104
4.3 Tags 104
4.4 Message Bodies 105
4.5 Conclusion 107
4.6 Questions 107
References 108

5 SIP Response Messages 111

5.1 Informational 112
5.1.1 100 Trying 112
5.1.2 180 Ringing 113

5.1.3 181 Call is Being Forwarded 113
5.1.4 182 Call Queued 113
5.1.5 183 Session Progress 113

5.2 Success 114
5.2.1 200 OK 114
5.2.2 202 Accepted 115
5.2.3 204 No Notification 115

5.3 Redirection 115
5.3.1 300 Multiple Choices 115
5.3.2 301 Moved Permanently 116
5.3.3 302 Moved Temporarily 116
5.3.4 305 Use Proxy 116
5.3.5 380 Alternative Service 116

5.4 Client Error 116
5.4.1 400 Bad Request 116
5.4.2 401 Unauthorized 117
5.4.3 402 Payment Required 117
5.4.4 403 Forbidden 117
5.4.5 404 Not Found 118
5.4.6 405 Method Not Allowed 118
5.4.7 406 Not Acceptable 118
5.4.8 407 Proxy Authentication Required 118
5.4.9 408 Request Timeout 119
5.4.10 409 Conflict 119
5.4.11 410 Gone 119
5.4.12 411 Length Required 119
5.4.13 412 Conditional Request Failed 119
5.4.14 413 Request Entity Too Large 120
5.4.15 414 Request-URI Too Long 120
5.4.16 415 Unsupported Media Type 120
5.4.17 416 Unsupported URI Scheme 120
5.4.18 417 Unknown Resource Priority 120
5.4.19 420 Bad Extension 121
5.4.20 421 Extension Required 121
5.4.21 422 Session Timer Interval Too Small 121
5.4.22 423 Interval Too Brief 121

5.4.23 428 Use Identity Header 121
5.4.24 429 Provide Referror Identity 122
5.4.25 430 Flow Failed 122
5.4.26 433 Anonymity Disallowed 122
5.4.27 436 Bad Identity-Info Header 122
5.4.28 437 Unsupported Certificate 122
5.4.29 438 Invalid Identity Header 123
5.4.30 439 First Hop Lacks Outbound Support 123
5.4.31 440 Max-Breadth Exceeded 123
5.4.32 470 Consent Needed 123
5.4.33 480 Temporarily Unavailable 123
5.4.34 481 Dialog/Transaction Does Not Exist 123
5.4.35 482 Loop Detected 124
5.4.36 483 Too Many Hops 124
5.4.37 484 Address Incomplete 125
5.4.38 485 Ambiguous 125
5.4.39 486 Busy Here 126
5.4.40 487 Request Terminated 126
5.4.41 488 Not Acceptable Here 126
5.4.42 489 Bad Event 126
5.4.43 491 Request Pending 126
5.4.44 493 Request Undecipherable 127
5.4.45 494 Security Agreement Required 127
5.5 Server Error 128
5.5.1 500 Server Internal Error 128
5.5.2 501 Not Implemented 128
5.5.3 502 Bad Gateway 128
5.5.4 503 Service Unavailable 128
5.5.5 504 Gateway Timeout 128
5.5.6 505 Version Not Supported 129
5.5.7 513 Message Too Large 129
5.5.8 580 Preconditions Failure 129
5.6 Global Error 129
5.6.1 600 Busy Everywhere 129
5.6.2 603 Decline 129
5.6.3 604 Does Not Exist Anywhere 130

5.6.4 606 Not Acceptable 130
5.7 Questions 130
References 131

6 SIP Header Fields 133

6.1 Request and Response Header Fields 134
6.1.1 Accept 134
6.1.2 Accept-Encoding 134
6.1.3 Accept-Language 136
6.1.4 Alert-Info 136
6.1.5 Allow 137
6.1.6 Allow-Events 137
6.1.7 Answer-Mode 137
6.1.8 Call-ID 137
6.1.9 Contact 138
6.1.10 CSeq 140
6.1.11 Date 141
6.1.12 Encryption 141
6.1.13 Expires 141
6.1.14 From 141
6.1.15 History Info 142
6.1.16 Organization 143
6.1.17 Path 143
6.1.18 Priv-Answer-Mode 143
6.1.19 Record-Route 144
6.1.20 Recv-Info 144
6.1.21 Refer-Sub 144
6.1.22 Retry-After 145
6.1.23 Subject 145
6.1.24 Supported 146
6.1.25 Timestamp 147
6.1.26 To 147
6.1.27 User-Agent 147
6.1.28 Via 148
6.2 Request Header Fields 149
6.2.1 Accept-Contact 149

6.2.2 Authorization 150
6.2.3 Call-Info 150
6.2.4 Event 150
6.2.5 Hide 151
6.2.6 Identity 151
6.2.7 Identity-Info 151
6.2.8 In-Reply-To 151
6.2.9 Info-Package 152
6.2.10 Join 152
6.2.11 Priority 153
6.2.12 Privacy 153
6.2.13 Proxy-Authorization 153
6.2.14 Proxy-Require 154
6.2.15 P-OSP-Auth-Token 155
6.2.16 P-Asserted-Identity 155
6.2.17 P-Preferred-Identity 155
6.2.18 Max-Breadth 155
6.2.19 Max-Forwards 156
6.2.20 Reason 156
6.2.21 Refer-To 156
6.2.22 Referred-By 157
6.2.23 Reply-To 157
6.2.24 Replaces 158
6.2.25 Reject-Contact 158
6.2.26 Request-Disposition 159
6.2.27 Require 159
6.2.28 Resource-Priority 160
6.2.29 Response-Key 160
6.2.30 Route 160
6.2.31 RAck 161
6.2.32 Security-Client 161
6.2.33 Security-Verify 162
6.2.34 Session-Expires 162
6.2.35 SIP-If-Match 162
6.2.36 Subscription-State 162
6.2.37 Suppress-If-Match 163
6.2.38 Target-Dialog 163

6.2.39 Trigger-Consent 163

6.3 Response Header Fields 163

6.3.1 Accept-Resource-Priority 163

6.3.2 Authentication-Info 164

6.3.3 Error-Info 164

6.3.4 Flow-Timer 165

6.3.5 Min-Expires 165

6.3.6 Min-SE 165

6.3.7 Permission-Missing 165

6.3.8 Proxy-Authenticate 166

6.3.9 Security-Server 166

6.3.10 Server 166

6.3.11 Service-Route 166

6.3.12 SIP-ETag 167

6.3.13 Unsupported 167

6.3.14 Warning 167

6.3.15 WWW-Authenticate 168

6.3.16 RSeq 168

6.4 Message Body Header Fields 169

6.4.1 Content-Encoding 169

6.4.2 Content-Disposition 169

6.4.3 Content-Language 170

6.4.4 Content-Length 170

6.4.5 Content-Type 170

6.4.6 MIME-Version 171

6.5 Questions 171

References 172

7 Wireless, Mobility, and IMS 177

7.1 IP Mobility 177

7.2 SIP Mobility 178

7.3 IMS and SIP 184

7.4 IMS Header Fields 186

7.5 Conclusion 186

7.6 Questions 187

References 187

8 Presence and Instant Messaging 189

8.1 Introduction 189

8.2 History of IM and Presence 189

8.3 SIMPLE 191

8.4 Presence with SIMPLE 191

8.4.1 SIP Events Framework 191

8.4.2 Presence Bodies 192

8.4.3 Resource Lists 194

8.4.4 Filtering 200

8.4.5 Conditional Event Notifications and ETags 201

8.4.6 Partial Publication 202

8.4.7 Presence Documents Summary 204

8.5 Instant Messaging with SIMPLE 205

8.5.1 Page Mode Instant Messaging 205

8.5.2 Common Profile for Instant Messaging 205

8.5.3 Instant Messaging Delivery Notification 206

8.5.4 Message Composition Indication 208

8.5.5 Multiple Recipient Messages 209

8.5.6 Session Mode Instant Messaging 210

8.6 Jabber 213

8.6.1 Standardization as Extensible Messaging and Presence Protocol 213

8.6.2 Interworking with SIMPLE 214

8.6.3 Jingle 214

8.6.4 Future Standardization of XMPP 214

8.7 Conclusion 214

8.8 Questions 215

References 216

9 Services in SIP 219

9.1 Gateway Services 219

9.2 SIP Trunking 221

9.3 SIP Service Examples 221

9.4 Voicemail 223

9.5 SIP Video 225

9.6 Facsimile 226
9.7 Conferencing 227
9.7.1 Focus 227
9.7.2 Mixer 228
9.7.3 Non-SIP Conference Control 228
9.8 Application Sequencing 229
9.9 Other SIP Service Architectures 230
9.9.1 Service Oriented Architecture 231
9.9.2 Servlets 231
9.9.3 Service Delivery Platform 231
9.10 Conclusion 232
9.11 Questions 232
References 232

10 Network Address Translation 235

10.1 Introduction to NAT 235
10.2 Advantages of NAT 236
10.3 Disadvantages of NAT 237
10.4 How NAT Works 238
10.5 Types of NAT 239
10.5.1 Endpoint Independent Mapping NAT 240
10.5.2 Address Dependent Mapping NAT 240
10.5.3 Address and Port Dependent Mapping NAT 241
10.5.4 Hairpinning Support 241
10.5.5 IP Address Pooling Options 242
10.5.6 Port Assignment Options 242
10.5.7 Mapping Refresh 242
10.5.8 Filtering Modes 243
10.6 NAT Mapping Examples 244
10.7 NATs and SIP 245
10.8 Properties of a Friendly NAT or How a NAT Should BEHAVE 247
10.9 STUN Protocol 248
10.10 UNSAF Requirements 249
10.11 SIP Problems with NAT 249

10.11.1 Symmetric SIP 250
10.11.2 Connection Reuse 250
10.11.3 SIP Outbound 251
10.12 Media NAT Traversal Solutions 251
10.12.1 Symmetric RTP 251
10.12.2 RTCP Attribute 253
10.12.3 Self-Fixing Approach 253
10.13 Hole Punching 253
10.14 TURN: Traversal Using Relays Around NAT 257
10.15 ICE: Interactive Connectivity Establishment 258
10.16 Conclusion 259
10.17 Questions 260
References 261

11 Related Protocols 263

11.1 PSTN Protocols 263
11.1.1 Circuit Associated Signaling 263
11.1.2 ISDN Signaling 264
11.1.3 ISUP Signaling 264
11.2 SIP for Telephones 264
11.3 Media Gateway Control Protocols 265
11.4 H.323 266
11.4.1 Introduction to H.323 266
11.4.2 Example of H.323 268
11.4.3 Versions 271
References 271

12 Media Transport 273

12.1 Real-Time Transport Protocol (RTP) 273
12.2 RTP Control Protocol (RTCP) 278
12.2.1 RTCP Reports 279
12.2.2 RTCP Extended Reports 279
12.3 Compression 280
12.4 RTP Audio Video Profiles 281
12.4.1 Audio Codecs 282

12.4.2 Video Codecs 283
12.5 Conferencing 284
12.6 ToIP—Conversational Text 285
12.7 DTMF Transport 285
12.8 Questions 286
References 287

13 Negotiating Media Sessions 289

13.1 Session Description Protocol (SDP) 289
13.1.1 Protocol Version 291
13.1.2 Origin 291
13.1.3 Session Name and Information 292
13.1.4 URI 292
13.1.5 E-Mail Address and Phone Number 292
13.1.6 Connection Data 292
13.1.7 Bandwidth 293
13.1.8 Time, Repeat Times, and Time Zones 293
13.1.9 Encryption Keys 293
13.1.10 Media Announcements 293
13.1.11 Attributes 294
13.2 SDP Extensions 296
13.3 The Offer Answer Model 297
13.3.1 Rules for Generating an Offer 299
13.3.2 Rules for Generating an Answer 299
13.3.3 Rules for Modifying a Session 299
13.3.4 Special Case—Call Hold 299
13.4 Static and Dynamic Payloads 300
13.5 SIP Offer Answer Exchanges 300
13.6 Conclusion 301
13.7 Questions 301
References 304

14 SIP Security 307

14.1 Basic Security Concepts 307
14.1.1 Encryption 308
14.1.2 Public Key Cryptography 309

14.1.3 Diffie-Hellman Cryptography 309
14.1.4 Message Authentication 309
14.1.5 Digital Certificates 310
14.2 Threats 311
14.3 Security Protocols 312
14.3.1 IPSec 312
14.3.2 TLS 313
14.3.3 DNSSec 313
14.3.4 Secure MIME 314
14.4 SIP Security Model 314
14.4.1 SIP Digest Authentication 314
14.4.2 SIP Authentication Using TLS 316
14.4.3 Secure SIP 317
14.4.4 Identity 317
14.4.5 Enhanced SIP Identity 318
14.5 SIP Certificate Service 319
14.6 Media Security 322
14.6.1 Non-RTP Media 322
14.6.2 Secure RTP 323
14.6.3 Keying SRTP 323
14.6.4 Best Effort Encryption 325
14.6.5 ZRTP 326
14.7 Questions 327
References 328

15 Peer-to-Peer SIP 331

15.1 P2P Properties 331
15.2 P2P Properties of SIP 332
15.3 P2P Overlays 333
15.4 RELOAD 336
15.5 Host Identity Protocol 338
15.6 Conclusion 339
15.7 Questions 340
References 341

16 Call Flow Examples 343

16.1 SIP Call with Authentication, Proxies, and Record-Route 343

16.2 SIP Call with Stateless and Stateful Proxies with Called Party Busy 349

16.3 SIP to PSTN Call Through Gateways 352

16.4 PSTN to SIP Call Through a Gateway 356

16.5 Parallel Search 359

16.6 Call Setup with Two Proxies 363

16.7 SIP Presence and Instant Message Example 365

References 368

17 Future Directions 369

17.1 Bug Fixes and Clarifications 370

17.2 More Extensions 370

17.3 Better Identity 371

17.4 Interdomain SIP 371

17.5 Making Features Work Better 371

17.6 Emergency Calling 372

17.7 More SIP Trunking 372

17.8 P2P and HIP 372

17.9 Improved NAT Traversal 372

17.10 Security Deployment 372

17.11 Better Interoperability 373

References 373

Appendix Introduction to ABNF and XML 375

A.1 ABNF Rules 375

A.2 Introduction to XML 377

References 379

About the Author 381

Index 383

Foreword to the First Edition

The Internet now challenges the close to $1 trillion world telecom industry. A renaissance in communications is taking place on the Internet. At its source are new communication protocols that would be impractical on the centralized control systems of circuit-switched networks used in telecommunications. The Internet and the World Wide Web can be technically defined only by their protocols. Similarly, IP telephony and the wider family of IP communications are defined by several key protocols, most notably by the Session Initiation Protocol, or SIP.

The previously closed door of telecommunications is now wide open to web developers because of SIP and its relation to the web HTTP 1.1 protocol and the e-mail SMTP protocol. IP communications include voice/video, presence, instant messaging, mobility, conferencing, and even games. We believe many other communication areas are yet to be invented. The integration of all types of communications on the Internet may represent the next "killer application" and generate yet another wave of Internet growth.

As explained in this book, SIP is a close relative of the HTTP 1.1 and SMTP protocols. This represents a revolution in communications because it abandons the telecom signaling and control models developed for telephony over many years in favor of Internet and web-based protocols. Users and service providers obtain not only seamless integration of telephony and conferencing with many other World Wide Web and messaging applications, but also benefit from new forms of communications, such as presence and instant messaging.

Mobility can also be managed across various networks and devices using SIP. Location management is now under user control, so that incoming "calls" can be routed to any network and device that the called party may prefer. Users may even move across the globe to another service provider and maintain not only their URL "number", but also their personal tailored services and preferences. The end user gains control over all possible preferences, depending on various parameters such as who the other party is, what network he is on and what devices he is using, as well as time of day, subject, and other variables.

The new dimension in communications called "presence" enables users for the first time to indulge in "polite calling" by first sensing presence and preferences of the other party, before making a call. In its turn, presence can trigger location- and time-dependent user preferences. Users may want to be contacted in different ways, depending on their location and type of network access.

E-commerce will also benefit from IP communications. Extremely complex telecom applications, as found in call centers, have become even more complex when integrated with e-mail and web applications for e-commerce. Such applications, however, are quite straightforward to implement using SIP, due to its common structure with the web and e-mail. For example, both call routing and e-mail routing to agents—based on various criteria such as queue length, skill set, time of day, customer ID, the web page the customer is looking at, and customer history—can be reduced to simple XML scripts when using SIP and another IETF standard, the Call Processing Language (CPL). These examples are in no way exhaustive, but are mentioned here as a way of introduction.

This book starts with a short summary of the Internet, the World Wide Web, and its core protocols and addressing. Though familiar to many readers, these chapters provide useful focus on issues for the topics ahead. The introduction to SIP is made easy and understandable by examples that illustrate the protocol architecture and message details. Finally, in the core of the book, a methodical and complete explanation of SIP is provided. We refer the reader to the Table of Contents for a better overview and navigation through the topics.

Alan Johnston has made significant contributions toward the use of SIP for communications over the Internet. I had the privilege of watching Alan in meetings with some of the largest telecom vendors as he went methodically line by line over hundreds of call flows, which were then submitted as an Internet Draft to the Internet Engineering Task Force (IETF) and implemented in commercial systems. Alan combines in this book his expertise and methodical approach with page turning narrative and a discreet sense of humor.

I could not help reading the book manuscript page by page, since everything from Internet basics, protocols, and SIP itself is explained so well, in an attractive and concise manner.

Henry Sinnreich
Distinguished Member of Engineering
WorldCom
Richardson, Texas
July 2000

Preface to the Third Edition

Like the SIP protocol, this book continues to expand and grow in new directions. While the core SIP protocol is essentially the same as it was in 2000, in the years since then, many SIP extensions have been proposed and standardized in the IETF. Important areas such as NAT traversal, SIP and media security, and peer-to-peer (P2P) applications have made many changes in the protocol and the protocol suite. As before, this book will help you sort through all the RFCs and fully understand this important protocol.

Chapter 1 is an introduction for those unfamiliar with the basics of the Internet and the TCP/IP protocol suite. Readers with familiarity in this area can probably skip straight to Chapter 2 for a quick introduction to the SIP protocol. For the rest of the readers, you will find an overview of the various layers of the protocol stack and an introduction to such concepts as the Domain Name Service (DNS) and Uniform Resource Locators (URLs) and an introduction to the Internet standards process in the Internet Engineering Task Force (IETF). Also for the beginner, a new appendix introduces the basics of ABNF and XML, essential for understanding SIP syntax.

Chapters 2 and 3 give you the basics of the protocol, the various messages, elements, transports, and applications. Chapters 4 and 5 explain the SIP request types or methods, with one new method introduced in this edition. Chapters 5 and 6 detail all the SIP response messages and header field types. There are literally dozens and dozens of new responses and header fields in this edition; Chapter 6 alone now has 50 references, five times as many as the first edition. Chapter 7 covers mobility and wireless aspects of SIP. Chapter 8 covers the important presence and instant messaging applications and extensions for SIP, known as SIMPLE. While the basics for these applications were in previous editions, the full suite of functions and operations is now standardized and covered with examples in this chapter. Chapter 9 covers SIP services from VoIP to SIP trunking to conferencing, fax, and video.

Chapter 10 represents many years of hard work in the industry for solving the Network Address Translation (NAT) traversal issue. A large impediment to SIP and VoIP deployment has been NAT traversal, and the scalability of many older solutions has been a major headache for deployments. This chapter explains how NAT works, new classifications schemes for understanding the types of NAT. Then, hole punching is explained, with detailed examples of various types of NAT. NAT traversal protocols such as STUN, TURN, and ICE are explained, and the ways that SIP uses them to scaleably solve the NAT traversal problem are described. This chapter represents the latest technology from the field, presented in this edition for the first time.

This edition also has more information about related protocols such as SDP, H.323, Jabber or XMPP, and Real-Time Transport Protocol (RTP) for media. Chapter 13 discusses how SIP negotiates various types of multimedia sessions.

Chapter 14 is a concise summary of SIP security, starting with the basics of security and encryption, attacks, security protocols, and SIP authentication mechanisms. This chapter also has detailed information about delivering secure media (Secure RTP or SRTP) with SIP. New media keying protocols including ZRTP are also covered.

Chapter 15 is new, covering peer-to-peer (P2P) technologies and how they apply to SIP. The new RELOAD protocol being developed in the IETF is covered, along with other approaches including Host Identity Protocol or HIP.

Chapter 16 includes the detailed call flows for which this book is famous, and Chapter 17 discusses the future of SIP and areas of development and innovation in the years to come. Perhaps these topics will find their way into another future edition.

Also new in this edition are the review questions at the end of most chapters. This has evolved from the classes I have taught on Internet communications at Washington University in St. Louis over the past few years. In fact, much of the new material was generated and developed for my teaching. As a result, I thank my past (and future) students for their interest and attention. I also thank Dr. John Corrigan and Dr. Tom Bush for their support and encouragement of this class. I would like to thank Anwar Siddiqui, Harvey Waxman, and Mun Yuen Leong at Avaya for their support of my SIP work. I especially thank my wife Lisa for her excellent cover artwork and figure design.

In closing, I thank everyone at Artech for giving me the opportunity once again to write about my favorite topic. I also thank you, my past and current readers, for your interest, support, and enthusiasm, and I wish you the best of luck in all your endeavors with the Session Initiation Protocol.

Preface to the Second Edition

Much has changed in the 2.5 years since the first edition of *SIP: Understanding the Session Initiation Protocol* was published. In 2001, SIP was a relatively unknown quantity, an upstart in the voice over IP (VoIP) and multimedia communications industry. Today, SIP is seen as the future of call signaling and telephony. It has been widely deployed by service providers and enterprises and is used casually every day by users of the dominant PC operating system. The full range of possibilities enabled by SIP is just now being glimpsed, and many more possibilities are yet to come.

One reason for this rapid acceptance is that SIP is an incredibly powerful call control protocol. It allows intelligent end points to implement the entire suite of telephony, Private Branch Exchange (PBX), Class, and Centrex services without a service provider, and without a controller or switch, for example.

The biggest driver for SIP on the Internet, however, has less to do with SIP's signaling and call control capabilities. Instead, it is due to the extensions of SIP that turn it into a powerful "rendezvous" protocol that leverages mobility and presence to allow users to communicate using different devices, modes, and services anywhere they are connected to the Internet. SIP applications provide support for presence—the ability to find out the status or location of a user without attempting to set up a session.

Another major change in the past few years is the adoption of wireless SIP to enable multimedia IP communications. As described in the chapter on wireless, SIP is now being used both in its standard form over 802.11 wireless networks and in planned commercial Third Generation Partnership Project (3GPP) rollouts in the coming years. SIP is ideally suited for this key application.

Since 2001 SIP has also grown in terms of the specification itself. Initially, SIP was described by a single RFC with a few related RFCs and a couple of RFC extensions. In Chapter 6 alone, more than 20 SIP-related RFCs are referenced. This book attempts to put all those documents together and provide a single reference for the protocol and all its extensions. Even SIP headers and responses that were standardized in the past but are now removed (deprecated) are listed in this text, providing useful context and background. Many others are discussed that are in the final stages of standardization prior to publication as RFCs, providing an up-to-date insider's view of the future of the protocol. In closing, I again thank my colleagues in the Internet Engineering Task Force (IETF) and at MCI for all their contributions to the development of this protocol—it has been a privilege to be a part of a group of people that have created the SIP industry. Finally, I'd like to tip my hat to two of the key inventors of SIP who continue to develop and propel its implementation: Henning Schulzrinne and Jonathan Rosenberg.

Preface to the First Edition

When I began looking into the Session Initiation Protocol (SIP) in October 1998, I had prepared a list of a half dozen protocols relating to Voice over IP and Next Generation Networking. It was only a few days into my study that my list narrowed to just one: SIP. My background was in telecommunications, so I was familiar with the complex suite of protocols used for signaling in the Public Switched Telephone Network. It was readily apparent to me that SIP would be revolutionary in the telecommunications industry. Only a few weeks later I remember describing SIP to a colleague as the "SS7 of future telephony"—quite a bold statement for a protocol that almost no one had heard of, and that was not even yet a proposed standard!

Nearly 2 years later, I have continued to work almost exclusively with SIP since that day in my position with WorldCom, giving seminars and teaching the protocol to others. This book grew out of those seminars and my work on various Internet-Drafts.

This revolutionary protocol was also the discovery of a radical standards body—the Internet Engineering Task Force (IETF). Later, I attended my first IETF meeting, which was for me a career changing event. To interact with this dedicated band of engineers and developers, who have quietly taken the Internet from obscurity into one of the most important technological developments of the late 20th century, for the first time was truly exciting.

Just a few short years later, SIP has taken the telecommunications industry by storm. The industry press contains announcement after announcement of SIP product and service support from established vendor startups, and from established carriers. As each new group and company joins the dialog, the protocol has been able to adapt and grow without becoming unwieldy or overly complex. In the future, I believe that SIP, along with a TCP/IP stack, will find its way into practically every intelligent electronic device that has a need to communicate with the outside world.

With my telecommunications background, it is not surprising that I rely on telephone examples and analogies throughout this book to explain and illustrate SIP. This is also consistent with the probability that telecommunications is the first widely deployed use of the protocol. SIP stacks will soon be in multimedia PCs, laptops, palmtops, and in dedicated SIP telephones. The protocol will be used by telephone switches, gateways, wireless devices, and mobile phones. One of the key features of SIP, however, is its flexibility; as a result, the protocol is likely to be used in a whole host of applications that have little or nothing to do with telephony. Quite possibly one of these applications, such as instant messaging, may become the next "killer application" of the Internet. However, the operation and concepts of the protocol are unchanged regardless of the application, and the telephone analogies and examples are, I feel, easy to follow and comprehend.

The book begins with a discussion of the Internet, the IETF, and the Internet Multimedia Protocol Stack, of which SIP is a part. From there, the protocol is introduced by examples. Next, the elements of a SIP network are discussed, and the details of the protocol in terms of message types, headers, and response codes are covered. In order to make up a complete telephony system, related protocols, including Session Description Protocol (SDP) and Real-Time Transport Protocol (RTP), are covered. SIP is then compared to another signaling protocol, H.323, with the key advantages of SIP highlighted. Finally, the future direction of the evolution of the protocol is examined.

Two of the recurring themes of this book are the simplicity and stateless nature of the protocol. Simplicity is a hallmark of SIP due to its text-encoded, highly readable messages, and its simple transactional model with few exceptions and special conditions. Statelessness relates to the ability of SIP servers to store minimal (or no) information about the state or existence of a media session in a network. The ability of a SIP network to use stateless servers that do not need to record transactions, keep logs, fill and empty buffers, etc., is, I believe, a seminal step in the evolution of communications systems. I hope that these two themes become apparent as you read this book and learn about this exciting new protocol.

The text is filled with examples and sample SIP messages. I had to invent a whole set of IP addresses, domain names, and URLs. Please note that they are all fictional—do not try to send anything to them.

I would first like to thank the group of current and former engineers at WorldCom who shared their knowledge of this protocol and gave me the opportunity to author my first Internet-Draft document. I particularly thank Henry Sinnreich, Steve Donovan, Dean Willis, and Matt Cannon. I also thank Robert Sparks, who I first met at the first seminar on SIP that I ever presented. Throughout the whole 3-hour session I kept wondering about the guy with the pony tail who seemed to know more than me about this brand new protocol! Robert and I have spent countless hours discussing fine points of the protocol. In addition, I would like to thank him for his expert review of this manuscript prior to publication—it is a better book due to his thoroughness and attention to detail. I also thank everyone on the IETF SIP list who has assisted me with the protocol and added to my understanding of it.

A special thanks to my wife Lisa for the terrific cover artwork and the cool figures throughout the book.

Finally, I thank my editor Jon Workman, the series editor and reviewer, and the whole team at Artech for helping me in this, my first adventure in publishing.

1

SIP and the Internet

The Session Initiation Protocol (SIP) is a signaling, presence, and instant messaging protocol developed to set up, modify, and tear down multimedia sessions; request and deliver presence; and send and receive instant messages [1]. When teaching or lecturing about SIP, I begin by explaining that SIP is an Internet protocol. This actually means much more than just that SIP runs *over* the Internet. This means that SIP uses and takes advantage of the Internet architecture and protocol suite. This chapter will introduce the TCP/IP protocol suite that is the foundation for the Internet and SIP. First, some of the basic concepts of Internet protocols such as Transmission Control Protocol (TCP), Internet Protocol (IP), User Datagram Protocol (UDP), and the Domain Name System (DNS) will be covered. SIP, along with many other Internet protocols, has been developed by the Internet Engineering Task Force (IETF). The processes, steps, and life cycle involved in the development of Internet standards will also be covered. This chapter ends with a brief history of SIP.

1.1 Signaling Protocols

This book is about the Session Initiation Protocol (SIP). As the name implies, the protocol allows two end points to establish media sessions with each other. The main signaling functions of the protocol are as follows:

- Location of an end point;
- Contacting an end point to determine willingness to establish a session;
- Exchange of media information to allow a session to be established;

- Modification of existing media sessions;
- Teardown of existing media sessions.

SIP has also been extended to request and deliver presence information (online/off-line status and location information such as that contained in a buddy list) as well as instant message sessions. These functions include:

- Publishing and uploading of presence information;
- Requesting delivery of presence information;
- Presence and other event notification;
- Transporting of instant messages.

While some of the examples discuss SIP from a telephony perspective, there will be many nontelephony uses for SIP. SIP will likely be used to establish a set of session types that bear almost no resemblance to a telephone call.

The following section will introduce the Internet multimedia protocol stack and discuss these protocols at a high level.

1.2 Internet Multimedia Protocol Stack

Figure 1.1 shows the five layer Internet multimedia protocol stack. The layers shown and protocols identified will be discussed.

1.2.1 Physical Layer

The physical layer is the lowest layer of the protocol stack. It shows how devices are physically connected with each other. Common physical layer methods include copper (coax, twisted pair, or other wired connections), photons (fiber optics, laser light, or other photonic sources), or phonons (radio waves, microwaves, or other electromagnetic transmissions).

1.2.2 Data/Link Layer

The next layer is the data/link layer, which could be an Ethernet local area network (LAN); a telephone line (V.90 or 56k modem) running Point-to-Point Protocol (PPP); a digital subscriber line (DSL); or even a wireless 802.11 network. This layer performs such functions as symbol exchange, frame synchronization, and physical interface specification. Ethernet typically adds a 13-octet header and a 3-octet footer to every packet sent. Note that an octet is 8 bits of data, sometimes called a *byte*.

1.2.3 Network Layer

The next layer in Figure 1.1 is the network or Internet layer. Internet Protocol (IP) [2] is used at this layer to route a packet across the network using the destination IP address. IP is a connectionless, best-effort packet delivery protocol. IP packets can be lost, delayed, or received out of sequence. Each packet is routed on its own using the IP header appended to the physical packet. Most IP address examples in this book use the older version of IP, version 4 (IPv4). IPv4 addresses are four octets long, usually written in "dotted decimal" notation (for example, `207.134.3.5`). At the IP layer, packets are not acknowledged. A checksum is calculated to detect corruption in the IP header, which could cause a packet to become misrouted. Corruption or errors in the IP payload, however, are not detected; a higher layer must perform this function if necessary, and it is usually done at the transport layer. IP uses a single-octet protocol number in the packet header to identify the transport layer protocol that should receive the packet.

IP version 6 (IPv6) [3] was developed by the IETF as a replacement for IPv4. It has been slowly gaining support and is supported now by most operating systems. The biggest initial networks of IPv6 are wireless telephony carriers who need the most important advantage of IPv6 over IPv4—a much enlarged addressing space. IPv6 increases the addressing space from 32 bits in IPv4 to 128 bits, providing for over 4 billion IPv6 addresses. An IPv6 address is typically written as a sequence of eight hexadecimal numbers separated by colons. For example, `0:0:0:0:aaaa:bbbb:cccc:dddd` is an IPv6 address written in this format. It is also common to drop sequences of zeros with a single double colon. This same address can then be written as `::aaaa:bbbb:cccc:dddd`. SIP can use either IPv4 or IPv6.

IP addresses used over the public Internet are assigned in blocks by regional internet registries (RIR). For example, the American Registry for Internet

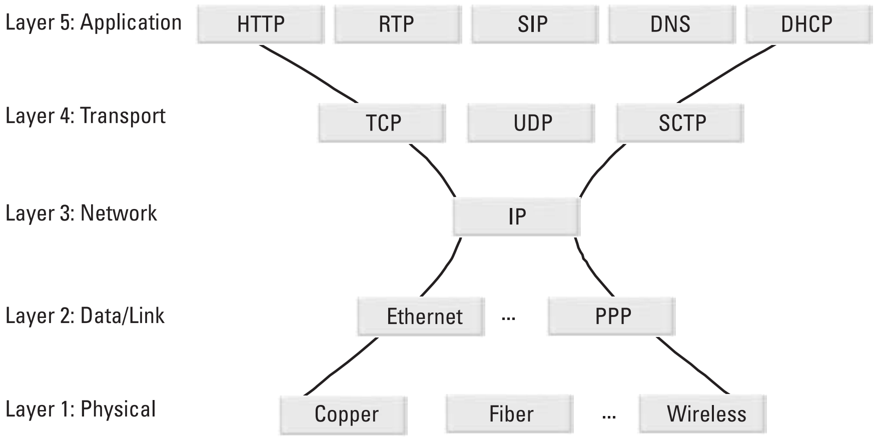


Figure 1.1 The Internet multimedia protocol stack.

Numbers (ARIN) allocates addresses in North America while Réseaux IP Européens Network Coordination Centre (RIP ENCC) allocates addresses in Europe. The Internet Assigned Number Association (IANA) manages the overall IP address pool, delegating blocks to the RIRs. Individual end users and enterprises use IP addresses allocated to them by their Internet Service Provider (ISP) from a regional registry.

As a result of this centralized assignment, IP addresses are globally unique. This enables a packet to be routed across the public Internet using only the destination IP address. Various protocols are used to route packets over an IP network, but they are outside of the scope of this book. Subnetting and other aspects of the structure of IP addresses are also not covered here. There are other excellent sources [4] that cover the entire suite of TCP/IP protocols in more detail.

Private IP addresses are addresses that are not routable on the public Internet but can be routable on a stub network LAN. A router performing network address translation (NAT) is used when a host with a private IP address needs to access resources on the public Internet. NAT temporarily binds or maps a host's private IP address, which is only routable within the LAN with a public IP address that has been allocated to the NAT. The NAT rewrites IP packets as they pass through in both directions, allowing connections. A detailed description of NAT and how it affects SIP and Internet communications can be found in Chapter 10. There are three IPv4 address blocks which have been allocated for private addresses in [5]:

```
10.0.0.0 - 10.255.255.255 or 10/8
172.16.0.0 - 172.31.255.255 or 172.16/12
192.168.0.0 - 192.168.255.255 or 192.168/16
```

Configuration information for Internet Protocol can be manually configured in a host or it can be learned automatically. Typically a host needs to know its own IP address, default gateway, subnet mask, and DNS server addresses. One common protocol for this is Dynamic Host Configuration Protocol (DHCP) which is defined by RFC 2132 [6]. DHCP allows a host to autodiscover all these parameters upon initialization of the IP stack. There are various DHCP extensions which have been defined to autoconfigure other protocols, including SIP.

1.2.4 Transport Layer

The next layer shown in Figure 1.1 is the transport layer. It uses a two-octet port number from the application layer to deliver the datagram or segment to the correct application layer protocol at the destination IP address. There are two commonly used transport layer protocols: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). In addition, there are two uncommon transport protocols: Stream Control Transmission Protocol (SCTP) and Datagram Congestion Control Protocol (DCCP), which are beginning to be used on the Internet. There is also Transport Layer Security (TLS) which provides security on top of TCP. These protocols are introduced in the following sections.

1.2.4.1 Transmission Control Protocol

Transmission Control Protocol (TCP) [7] provides reliable, connection-oriented transport over IP. A TCP connection between two hosts over an IP network is sometimes known as a *socket.* TCP is a client/server protocol. Servers "listen" on a specific port number for an incoming request to open a socket. A client sends a request to open a new socket to the server on the well-known port. The combination of the source IP address, source port, destination IP address, and destination port identifies the socket connection. As such, it is possible for two hosts to have multiple TCP connections open between them.

TCP uses sequence numbers and positive acknowledgments to ensure that each block of data, called a segment, has been received. Lost segments are retransmitted until they are successfully received. Figure 1.2 shows the message exchange to establish and tear down a TCP connection. A TCP server listens on a well-known port for a TCP SYN (synchronization) message to open the connection. The SYN message contains the initial sequence number the client will use during the connection. The server responds with an ACK message to acknowledge the SYN with an acknowledgment number, and then follows up with its own SYN message containing its own initial sequence number. Often, these two messages are combined into one SYN-ACK message that does both functions. The client completes the three-way handshake with an ACK or a DATA packet with the AK flag set to the server acknowledging the server's sequence number. Now that the connection is open, either client or server can send data in DATA packets (segments). The connection is closed when either side sends a FIN packet that receives an ACK. This exchange is shown in Figure 1.2.

TCP sends data in units called segments. The maximum segment size (MSS) is negotiated between the hosts during the handshake, and is usually based on the maximum transmission unit (MTU) of the local network. In general, the larger the segment size the more efficient the transport, except when packet loss is present when smaller segments can result in fewer retransmissions. A typical MTU value for the Internet is 1,500 octets.

TCP uses cumulative acknowledgements for reliability. The recipient sends ACK packets including the next sequence number it expects to receive. If a sender does not receive an ACK within a certain time period, the segment is resent. An example is shown in Figure 1.3.

TCP also has built in flow control. Flow control is used by a receiver to slow down the rate of transmission to allow the receiver to properly process or buffer incoming segments. TCP uses a sliding window for end-to-end control.

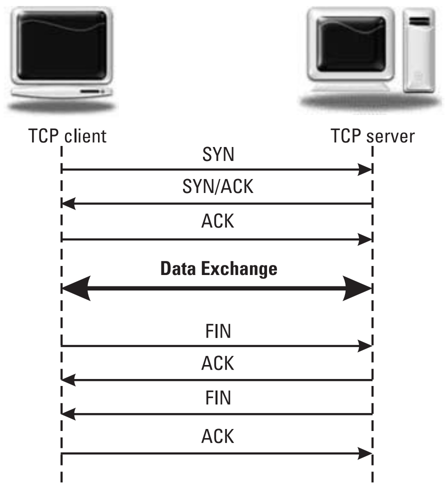


Figure 1.2 TCP handshake example.

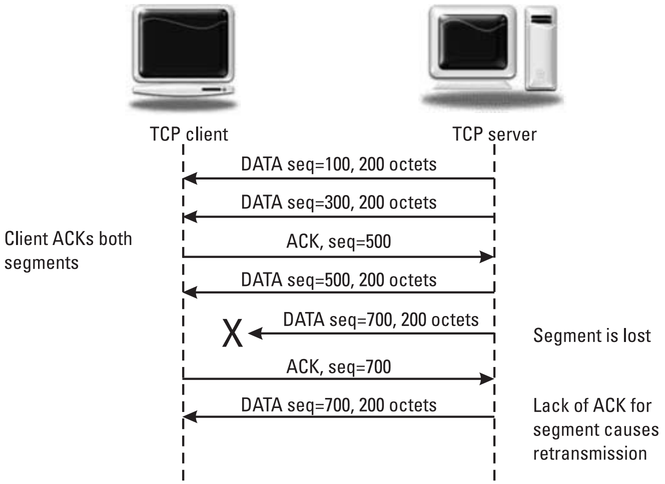


Figure 1.3 TCP reliability example.

Senders can only send the number of octets in the window before waiting for an ACK. A receiver can reduce the size of the window in ACK messages, even setting it to 0 to cause the sender to stop sending. Once the receiver has caught up, another ACK can be sent to increase the window size and resume the flow of segments. This is shown in Figure 1.4.

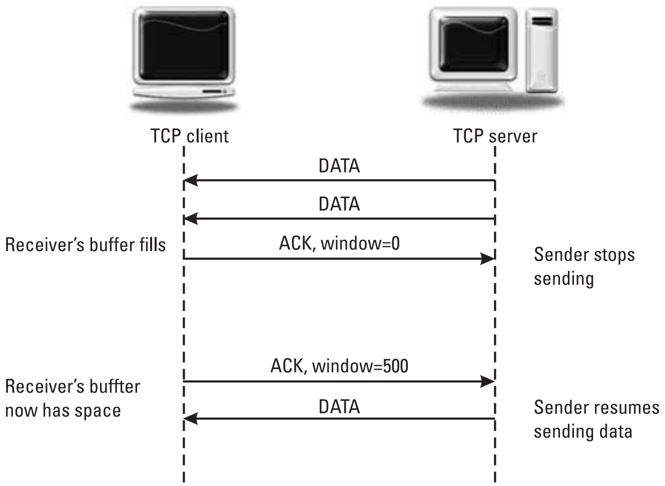


Figure 1.4 TCP flow control example.

TCP also has built in congestion control. TCP uses a slow-start algorithm to attempt to avoid congestion. When congestion occurs, TCP uses a fast retransmit and a fast recovery. The details of how these algorithms work can be found in any good TCP/IP reference such as [4].

TCP adds a 20-octet header field to each packet, and is a stream-oriented transport. An application using TCP to send messages must provide its own framing or separation between messages. Error segments are detected by a checksum covering both the TCP header and payload.

1.2.4.2 Transport Port Numbers

Ports numbers are used by the transport layer to multiplex and demultiplex multiple connections on a single host. Otherwise a pair of hosts could only have a single connection between them. Also, messages for different protocols can be separated by using different port numbers. Often these port numbers are associated with a specific protocol. Others are registered to a particular protocol. Ports are a 16 bit integer. Ports in the range `0` to `1024` are called well-known ports. Ports in the range of `1024` through `49151` are known as registered ports. Ports in the range of `49152` through `65535` are known as dynamic, private, or ephemeral ports. For example, Web servers use the well known port of `80`, SIP uses the registered ports of `5060` and `5061`, while RTP usually uses a dynamic port.

1.2.4.3 User Datagram Protocol

User Datagram Protocol (UDP) [8] provides unreliable transport across the Internet. It is a best-effort delivery service, since there is no acknowledgment of sent datagrams. Most of the complexity of TCP is not present, including sequence numbers, acknowledgments, and window sizes. UDP does detect datagrams with errors with a checksum. It is up to higher layer protocols to detect this datagram loss and initiate a retransmission if desired.

UDP does not provide congestion control or flow control—if any of these functions are needed, they must be built into the application layer protocol. UDP is best suited for short, single packet exchanges such as DNS or routing queries. It is also good for real-time, low latency transports protocols such as SIP and RTP.

UDP adds an 8 octet header field to datagrams. Applications and protocols that use UDP must do their own framing—they must break up information into individual UDP packets. For a message oriented protocol, this typically means one message or request per UDP datagram.

1.2.4.4 Transmission Layer Security

Transmission Layer Security (TLS) [9] is based on the Secure Sockets Layer (SSL) protocol first used in Web browsers. TLS uses TCP for transport although it has recently been extended to also run over UDP. TLS is commonly used today on the Internet for secure Web sites using the secure HTTP (https) URI scheme.

The TLS protocol has two layers: the TLS Transport Protocol and the TLS Handshake Protocol. The TLS Transport Protocol is used to provide a reliable and private transport mechanism. Data sent using the TLS Transport Protocol is encrypted so that a third party cannot intercept the data. A third party also cannot modify the transported data without one of the parties discovering this. The TLS Handshake Protocol is used to establish the connection, negotiate the encryption keys used by the TLS Transport Protocol, and provide authentication.

The key agreement scheme selects an encryption algorithm and generates a one-time key based on a secret passed between the two sides. During the handshake, the parties exchange certificates, which can be used for authentication. The cryptographic computations for a TLS connection are not trivial, and the multiple round trips needed to open a connection can add to message latency. Also, certificate verification can introduce processing delays. However, TLS transport has clear security advantages over UDP or TCP. TLS is widely supported due to its use in secure Web browsers and servers. TLS will be discussed more in Chapter 14.

1.2.4.5 Stream Control Transport Protocol

The Stream Control Transmission Protocol (SCTP) [10] is similar to TCP in that it provides reliable stream-based transport. However, it has some advantages over TCP transport for a message-based protocol. First, it has built-in message segmentation, so that individual messages are separated at the transport layer. Another advantage is that SCTP avoids the so-called “head of line blocking” problem of TCP. This is a TCP problem in which a dropped segment with a large window causes the entire window's worth of messages to wait in a buffer (that is, be blocked) until the dropped segment is retransmitted.

SCTP also supports multihoming, so if one of a pair of load balancing servers fails, the other can immediately begin receiving the messages without even requiring a DNS or other database lookup.

As a transport protocol, SCTP requires operating system level support to be used, which will initially delay its use in the Internet. Also, as we shall see in Chapter 10 on NAT traversal, the use of new transports on the Internet is severely limited by their support in middleboxes such as NAT. Also, note that the advantages of SCTP over TCP only occur during packet loss. In a zero loss network, the performance of the two is identical. SCTP is not commonly supported in Internet hosts today.

1.2.4.6 Datagram Congestion Control Protocol

Datagram Congestion Control Protocol (DCCP) [11] is another new transport protocol that tries to provide congestion and flow control similar to TCP but without the reliability or in-order delivery of TCP. It shows some promise for use as a real-time transport. However, its support is very limited today, and limited NAT support will delay its adoption.

1.2.5 Application Layer

The top layer shown in Figure 1.1 is the application layer. This includes signaling protocols such as SIP and media transport protocols such as Real-Time Transport Protocol (RTP), which is introduced in Chapter 12. HTTP, SMTP, FTP, and Telnet are all examples of application layer protocols. SIP can theoretically use any transport protocol, although it is currently standardized to run over TCP, UDP, and SCTP. The use of TCP, TLS, SCTP, and UDP transport for SIP will be discussed in the next chapter.

1.2.6 Utility Applications

Two Internet utility applications are also shown in Figure 1.1. The most common use of the DNS (well-known port number 53) is to resolve a symbolic name (such as domain.com, which is easy to remember) into an IP address (which is required by IP to route the packet). Also shown is the Dynamic Host Configuration Protocol (DHCP). DHCP allows an IP device to download configuration information upon initialization. Common fields include a dynamically assigned IP address, DNS addresses, subnet masks, maximum transmission unit (MTU), or maximum packet size, and server addresses for e-mail and Web browsing. SIP has a DHCP extension for configuration [12].

Another utility is Internet Control Message Protocol (ICMP), a control and diagnostic protocol that runs between single IP routing hops—between routers and between routers and hosts. It runs directly on top of the IP layer without a transport protocol, using protocol 1. The most common ICMP message is a Type 8 Echo Request or ping. Ping tests can be used to verify connectivity. A Type 0 Echo Reply is a ping response whose latency is often measured and displayed. Other ICMP error messages include a Type 3 Destination Unreachable message or a Type 11 TTL Exceeded for Datagram where TTL stands for Time to Live. ICMP is also used to provide the traceroute (tracert on Windows) Internet utility used to discover IP hops between hosts.

1.2.7 Multicast

In normal Internet packet routing, or unicast routing, a packet is routed to a single destination. In multicast routing, a single packet is routed to a set of destinations. Single LAN segments running a protocol such as Ethernet offer the capability for packet broadcast, where a packet is sent to every node on the network. Scaling this to a larger network with routers is a recipe for disaster, as broadcast traffic can quickly cause congestion. An alternative approach for this type of packet distribution is to use a packet reflector that receives packets and forwards copies to all destinations that are members of a broadcast group. For a number of years, the Internet Multicast Backbone Network (MBONE), an overlay of the public Internet, has used multicast routing for high-bandwidth broadcast sessions. Participants who wish to join a multicast session send a request to join the session to their local MBONE router using a protocol known as Internet Group Management Protocol (IGMP). That router will then begin to broadcast the multicast session on that LAN segment.

Additional requests to join the session from others in the same LAN segment will result in no additional multicast packets being sent, since the packets are already being broadcast. If the router is not aware of any multicast participants on its segment, it will not forward any of the packets. Routing of multicast packets between routers uses special multicast routing protocols to ensure that packet traffic on the backbone is kept to a minimum. Multicast IPv4 Internet addresses are reserved in the range 224.0.0.0 to 239.255.255.255.

Multicast transport is always UDP, since the handshake and acknowledgments of TCP are not possible. Certain addresses have been defined for certain protocols and applications. The scope or extent of a multicast session can be limited using the time to live (TTL) field in the IP header. This field is decremented by each router that forwards the packet, which limits the number of hops the packet takes. SIP support for multicast will be discussed in Section 3.8. Multicast is slowly becoming a part of the public Internet as service providers begin supporting it, and it is finding an important application today in the streaming of real-time video to set top boxes sometimes known as IPTV. There is another approach known as application layer multicast which uses peer-to-peer technology, which does not require any changes at the IP layer. This will be discussed in Chapter 15.

1.3 Internet Names

Internet addresses, covered in Chapter 2, are used to route individual datagrams over the Internet. However, they are not very friendly for humans to use. IPv4 addresses can be as long as 12 digits while IPv6 addresses can be as long as 32 hexadecimal digits long. A given Internet host with only one IP address may have many identities. Also, some Internet identities are that of the human user, not the actual host. For example, an e-mail address identifies a user, not a particular host on the Internet. The user may utilize multiple Internet hosts to access e-mail.

Internet names began with RFC 822 [13] which defined the `user@host` format that is so familiar today with email addresses. These text based names were defined to enable a piece of software known as a *parser* to be able to extract the various parts of the address and any parameters. In addition, some of the first Internet applications such as e-mail used a text-based way of encoding protocol messages. The method of encoding both Internet names and messages was defined in RFC 822 as Backus Naur Format (BNF). BNF was developed by John Backus to define the early programming language ALGOL. Today, many Internet protocols, including SIP, are defined using Augmented Backus Naur Form (ABNF) [14] which is based on BNF. The appendix has an introduction to ABNF.

1.4 URLs, URIs, and URNs

Uniform Resource Locators (URLs) are an addressing scheme developed for the World Wide Web (WWW). It is defined in RFC 1738 [15], and is a syntax for representing a resource available on the Internet. The general form is:

```
scheme:scheme-specific-part
```

for example, consider:

```
http://www.artechhouse.com/Default.aspx
```

The token `http` identifies the scheme or protocol to be used, in this case HTTP. The specifier follows the ":" and contains a domain name (`www.artechhouse.com`), which can be resolved into an IP address and a file name

(/Default.aspx). URLs can also contain additional parameters or qualifiers relating to transport, but they can never contain spaces. For example, telnet://host.company.com:24 indicates that the Telnet Protocol should be used to access host.company.com using port 24. New schemes for URLs for new protocols are easily constructed, and dozens have been defined, such as mailto, tel, and https. The sip and sips schemes will be introduced in Section 4.2.

Most protocols reference URLs, but with SIP we mainly reference Uniform Resource Indicators (URIs). This is due to the mobility aspects of SIP, which means that a particular address (URI) is not tied to a single physical device but instead is a logical entity that may move around and change its location in the Internet. However, the terms URL and URI are often used almost interchangeably in other contexts.

Some other examples include:

```
http://www.ese.wustl.edu
sip:barney@fwd.rubble.com
mailto:help@example.com?Subject=Help!
```

The initial set of URL schemes defined in RFC 1738 are in Table 1.1.

Common SIP and Internet Communications URL and URI schemes are listed in Table 1.2. The details of SIP and SIPS URIs are covered in Section 4.2.

Uniform Resource Names or URNs are defined by [16]. A URN provides a standard name for a resource but does not provide any information for how to access the resource. An example URN namespace is book ISBNs (International Standard Book Numbers):

Table 1.1
Initial Set of URL Schemes

Scheme	Protocol
http	Hypertext Transfer Protocol
ftp	File Transfer Protocol
gopher	The Gopher Protocol
mailto	Electronic mail address
news	USENET news
nntp	USENET news using NNTP access
telnet	Remote login
wais	Wide Area Information Servers
file	Host-specific file name
prospero	Prospero Directory Service

Table 1.2
URL and URI Schemes Used in Internet Communications

Scheme	Protocol
sip	SIP
sips	Secure SIP (TLS)
tel	Telephone number and dial string
im	Instant messaging inbox
pres	Presence
xmpp	Jabber IM and presence
h323	H.323

```
URN:ISBN:1-60783-995-4
```

URN namespaces are often used to identify XML extensions.

1.5 Domain Name Service

The Domain Name Service [17] is used on the Internet to map a symbolic name (such as www.amazon.com) to an IP address (such as 100.101.102.103 which is an example IPv4 address). DNS is also used to obtain information needed to route e-mail messages and, in the future, SIP messages. The use of names instead of numerical addresses is one of the Internet's greatest strengths because it gives the Internet a human, friendly feel. Domain names are organized in a hierarchy. Each level of the name is separated by a dot, with the highest level domain on the right side. (Note that the dots in a domain name have no correspondence to the dots in an IP address written in dotted decimal notation.) General top-level domains are shown in Table 1.3 (see http://www.icann.org/tlds for the latest list). Some such as com, net, and edu are commonly encountered, while others such as aero and coop are rare. There is also a set of country domains such as: us (United States), uk (United Kingdom), ca (Canada), and au (Australia). Each of these top-level domains has just one authority that assigns that domain to a user or group.

Once a domain name has been assigned, the authority places a link in their DNS server to the DNS server of the user or group who has been assigned the domain. For example, when company.com is allocated to a company, the authoritative DNS server for the top-level com domain entry for company contains the IP address of the company's DNS server(s). A name can then be further qualified by entries in the company's DNS server to point to individual servers in their network. For example, the company's DNS server may contain entries for www.company.com, ftp.company.com, and smtp.company.com. A number of types of DNS record types are defined. The DNS records used to resolve a host name into an IP address are called address records, or A records. Other types of records

Table 1.3
Generic Top Level Domains (gTLDs)

Domain	Description
com	Company
net	Network
int	Internet
org	Not-for-profit organization
edu	University or college
gov	U.S. government
mil	U.S. military
arpa	ARPAnet
info	Information
biz	Business
museum	Museum
name	Name
pro	Professional
aero	Air transport industry
coop	Cooperatives

include CNAME (canonical name or alias records), MX (mail exchange records), SRV (service records, used by SIP and other protocols), and TXT (free-form text records). Another type of DNS record is a PTR, or pointer record, used for reverse lookups. Reverse lookups are used to map an IP address back to a domain name. These records can be used to generate server logs that show not only the IP addresses of clients served, but also their domain name. Web browsing provides an example of the use of the DNS system. Another type of DNS record is known as a Naming Authority Pointer (NAPTR) record that can be used by a protocol known as ENUM [18] to map global telephone numbers into Internet URLs.

1.5.1 DNS Resource Records

DNS resource records are text records that are stored in DNS servers and retrieved by DNS resolvers. Each record has a minimum of four fields: name, type, class, and time to live (TTL). The name is the owner of the record and the resource being identified. The type is the type of resource records such as A, AAAA, MX, SRV, NAPTR, PTR, or TXT. Class is the class of address, which is IN for Internet addresses (all other network addresses are CH for Chaos). The time to live field is a 32-bit integer count of the number of seconds the record should be cached before being discarded.

Common DNS resource record types are listed in Table 1.4. The resource records used by SIP are discussed in detail in the following sections.

Table 1.4
Common DNS Resource Record Types

RR Type	Reference	Description
A	RFC 1035	Address IPv4, See Section 1.5.2
AAAA	RFC 3596	Address IPv6, See Section 1.5.2
CNAME	RFC 1035	Canonical name, used for aliases
MX	RFC 1035	Mail exchange
NAPTR	RFC 2915	Naming authority pointer record; see Section 1.5.4
NS	RFC 1035	Name server records
PTR	RFC 1035	Reverse domain name lookup
SOA	RFC 1035	Start of authority
SPF	RFC 4408	Sender policy framework for authorizing e-mail
SRV	RFC 2782	Services; see Section 1.5.3
TXT	RFC 1035	Text records

1.5.2 Address Resource Records (A or AAAA)

The most common type of resource record is an address record. As the name suggests, it provides an address for a resource or a host name. Besides the name, TTL, class, and type fields it includes a target field, which is an IP address. An A record provides [17] an IPv4 address while an AAAA record [19] provides an IPv6 address. An AAAA record is usually called a "quad A" record.

Structure:

```
Name TTL Class A Target
```

For example,

```
ese.wustl.edu. 3600 IN A 128.252.168.2
ietf.org. 300 IN AAAA 2610:a0:c779:b::d1ad:35b4
```

are examples of an A record and an AAAA record. Notice the "." used after the host name—this is to indicate that the address is absolute.

1.5.3 Service Resource Records (SRV)

Service resource records or SRV records [20] are used to lookup a host that provides a particular service. A number of services have been defined for SRV records including SIP service. SRV records use an underscore (_) in the service name to distinguish it from normal host names which may not include an underscore.

Structure:

```
Service.Proto.Name TTL Class SRV Priority Weight Port Target
```

The priority field is used to set the relative priority of this record, as an SRV query might return several SRV resource records. The priority is a 16 bit unsigned integer. Resolvers should use the lowest priority record (highest priority). The weight is a relative weight used to select between records with the same priority. It is also a 16 bit unsigned integer. The port is the transport port number that should be used for this service. This allows multiple instances of the same service to be run on the same host—each can utilize a different port number. The target is the domain name of the host. To reach the desired service, the target address and port number should be used.

For example,

```
_sip._udp.avaya.com. 300 IN SRV 0 100 5060 sip.avaya.com.
```

This example resource record is for SIP service using UDP as the transport protocol. The priority of this record is 0, indicating that it is the highest priority. The port number is 5060, the registered port for SIP. The target is sip.avaya.com, which will require an address (A or AAAA) lookup to resolve to an IP address. The use of SRV records in SIP will be discussed in Section 2.6.

1.5.4 Naming Authority Pointer Resource Records (NAPTR)

Naming authority pointer resource records or NAPTR records [21] are used to point to another record or URI. They are used by SIP to discover which transport protocols a given domain or server supports. In a protocol known as ENUM, they are used to resolve a telephone number into a URI. The usage of NAPTR records in SIP are discussed in Section 2.6.

Structure:

```
Domain TTL Class Type Order Preference Flags Service Regexp Replacement
```

Example:

```
columbia.edu. 3600 IN NAPTR 1 0 "s" "SIP+D2U" "" _sip._udp.columbia.edu.
```

1.5.5 DNS Resolvers

DNS resolvers are the software in an Internet host that looks up DNS records and sends DNS requests. When a user types in a Web address, such as www.artechhouse.com, the name must be resolved to an IP address before the browser can send the request for the index Web page from the Artech House Web server. The Web browser first launches a DNS query to the IP address for its DNS server, which has been manually configured or set up using DHCP.

The first step of the DNS resolver is to check the local DNS cache to see if the desired host name is already cached. If so, it returns that value without performing a lookup. If the value is not cached, then a query will be launched. If the DNS resolver is configured with the IP address of one or more DNS servers, it may simply forward the request to that DNS server and let that server take care of the request. Alternatively, it could resolve the address on its own using the following steps. The DNS resolver would check to see if the authoritative DNS server for the top level domain (e.g., com) is stored in its cache. If not, it will have to query a DNS root server (e.g., ".") for this information. DNS resolvers are configured with the IP addresses of the 16 DNS root servers. One of these will be selected and the query sent. With the authoritative DNS server name for the top level domain, the resolver will then query that DNS server for the next level domain name (e.g., "example.com"). Once it has the authoritative DNS server for that domain, another query will be launched for the next level domain name (e.g., "www.example.com"). This continues until the actual host name records are retrieved. At every stage, DNS servers can return cached values instead of launching a new query.

If the DNS server happens to have the name's `A` record stored locally (cached) from a recent query, it will return the IP address. If not, the DNS root server will then be queried to locate the authoritative DNS server for Artech House, which must contain the `A` records for the `artechhouse.com` domain. The HTTP `GET` request is then sent to that IP address, and the Web browsing session begins. There is only one authoritative DNS server for a domain, and it is operated by the owner of the domain name. Due to a very efficient caching scheme built into DNS, a DNS request often does not have to route all the way to this server. DNS is also used by an SMTP server to deliver an e-mail message. An SMTP server with an e-mail message to deliver initiates a DNS request for the `MX` record of the domain name in the destination e-mail address. The response to the request allows the SMTP server to contact the destination SMTP server and transfer the message. A similar process is defined for locating a SIP server using `SRV`, or service, DNS records.

1.6 Global Open Standards

SIP is an example of a global, open, Internet standard. Global means that the same protocol is used regardless of the country. A Web browser or Web server works the same if it is located in the United States, Europe, or Asia. This contrasts with telephony protocols, which have historically been highly regionalized. For example, ISDN User Part (ISUP), which forms the basis for today's telephone network, has numerous national and regional varieties. Telephone equipment can only be used in the country for which it was manufactured. Switching gear used to handle international traffic to many different countries is some of the most complicated and expensive telephone equipment. As a result, features, services, and innovation have spread very slowly over the PSTN. Contrast this to the Internet where new applications and services are immediately available worldwide to anyone with Internet connectivity.

SIP is an example of an open standard. This means that any individual or company can access the standard, participate in the standards process, and have their voices heard and their issues discussed. Again, this contrasts with other standards bodies, which have closed membership, expensive fees, or geographical restrictions. Internet standards have always been freely available to download over the Internet.

SIP is a standard because it provides a definition for how different vendors, providers, and users can interconnect and interoperate their communication equipment. As such, SIP defines message formats and state machines (i.e., "bits on the wire"). SIP does not specify architectures, business models, exact service definitions, or user interfaces.

1.7 Internet Standards Process

The Internet Engineering Task Force (IETF) [22] is the body that develops standards for the Internet. It is a loosely organized group of implementers, vendors, service providers, and academics who work together to solve Internet problems and develop new protocols. Anyone can participate in the IETF standards process. The first step is to find the working group in the area of interest and join their e-mail mailing list. Most of the IETF work is done over e-mail with exchanges of ideas and discussions about the standards. The basic operation and principles of the IETF are described in the well-written "Tao of the IETF" [23].

Standards begin life as a working document known as an Internet Draft. These documents are only valid for 6 months. After this time, they are either updated or they expire. A document that gains support from participants and is heading in the right direction can be adopted by a working group as an official working group document. A working group is a group of interested parties within the IETF who have been chartered to work on a particular problem or protocol. Document which have been adopted by a working group typically have a file name beginning `draft-ietf-wg-`. . . where wg is the name of the working group. Individual drafts usually have a file name beginning `draft-authorlast-name-`. . . . The document will continue to be refined and improved until the working group chairs determine that it is ready for a final review, known as Last Call. There is a Working Group Last Call (WGLC) and an IETF Last Call (IETF LC). Current Internet Drafts are listed at http://www.ietf.org/internet-drafts.

At this point, the document is considered and discussed by a group known as the Internet Engineering Steering Group (IESG) [24], which consists of 15 members. This body discusses and then votes on approval of the document. Once approved, the document is published as part of the Request for Comments (RFC) Series. RFC documents are protocol standards, informational documents, or best current practices (BCP). RFC documents are identified by their numbers, for example, SIP is RFC 3261. RFC numbers are sequential—as of early 2009, new RFCs were being published with numbers in the 5400s. RFCs have been published since the early 1980s, and the number of new RFCs has increased since the turn of the century. There is even a tradition of publishing "April 1st RFCs," which are essentially jokes. These RFCs appear completely serious and cannot be distinguished from normal RFCs, except that the subject matter is usually ridiculous and the RFC will be dated April 1. More than a few humorless engineers have been caught off guard by these documents, and some even show up on vendor specification sheets and request for proposals! RFCs are archived by the RFC editor [25]. RFCs can replace (obsolete) or add to (update) existing RFCs—this information is available on the RFC editor's Web site and is crucial information for developers and implementers. For example, the original SIP specification was published as RFC 2543. Then, RFC 3261 was published, which *obsoletes* RFC 2543. An implementor finding and coding to RFC 2543 will not interoperate with current SIP implementations on the Internet.

Inside the IETF is another body known as the Internet Architecture Board or IAB [26]. This group does not vote to approve standards but addresses architectural and high level issues affecting the Internet. There is a companion group to the IETF known as the Internet Research Task Force (IRTF) [27]. The IRTF looks at topics and protocols that won't be deployed on the Internet for more than a few years. There is also the Internet Assigned Names Association (IANA) [28]. IANA allocates and manages the pool of Internet addresses. IANA allocates addresses to regional bodies who then allocate them to Internet service providers (ISPs), enterprises, governments, and universities. The Internet Corporation for Assigned Names and Numbers or ICANN [29] manages the top level Internet domains. The World Wide Web Consortium or W3C [30] is responsible for all things relating to the Web, including Hypertext Markup Language (HTML) and Extensible Markup Language (XML), introduced in the appendix. The W3C is also developing standards for applications to use the Web to interact; this is known as Web services.

The International SIP Forum [31] is an organization that promotes the use of the SIP protocol. They are a membership nonprofit organization that holds regular meetings and discusses issues of interest to vendors, service providers, and users of SIP. Currently there are over 50 member companies. The SIP Forum has recently started publishing their own SIP recommendations, which describe how to utilized IETF SIP standards. For example, in 2007, version 1.0

of the SIPConnect SIP trunking recommendation [32] was published, which is discussed in Section 9.2.

1.8 A Brief History of SIP

SIP was originally developed by the IETF Multi-Party Multimedia Session Control Working Group (MMUSIC). Version 1.0 was submitted as an Internet Draft in 1997. Significant changes were made to the protocol and resulted in a second version, version 2.0, which was submitted as an Internet Draft in 1998. The protocol achieved proposed standard status in March 1999 and was published as RFC 2543 [33] in April 1999. In September 1999, the now closed SIP Working Group was established by the IETF to meet the growing interest in the protocol. An Internet Draft containing bug fixes and clarifications to SIP was submitted in July 2000, referred to as RFC 2543 "bis." This document was eventually published as RFC 3261 [1], which obsoletes (or replaces) the original RFC 2543 specification. In addition, many SIP extension RFC documents have been published.

The popularity of SIP in the IETF has lead to the formation of other SIP-related working groups. The now closed Session Initiation Protocol Investigation (SIPPING) working group was formed to investigate applications of SIP, develop requirements for SIP extensions, and publish best current practice (BCP) documents about the use of SIP. Currently, the SIPCORE working group is responsible for the core SIP standards. The SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) working group was formed to standardize related protocols for presence and instant messaging applications. Other working groups that make use of SIP include the PSTN and Internet Internetworking (PINT) working group and the Service in the PSTN/IN Requesting Internet Services (SPIRITS) working group.

To advance from proposed standard to draft standard, a protocol must have multiple independent interworking implementations and limited operational experience. Since the early days of RFC 2543, SIP interoperability test events, called SIPit (formerly called "bakeoffs"), have been held a few times per year. For the latest information about SIPit, visit http://www.sipit.net. (Note that the SIP Forum is a marketing/promotion organization for SIP and does not have any standardization function.) The final level, standard, is achieved after operational success has been demonstrated. With the documented interoperability from SIPit, the plan is to move SIP from proposed standard to draft standard status in the future.

SIP incorporates elements of two widely used Internet protocols: Hyper Text Transport Protocol (HTTP) used for Web browsing and Simple Mail Transport Protocol (SMTP) used for e-mail. From HTTP, SIP borrowed a client-server design and the use of URLs and URIs. From SMTP, SIP borrowed a text-encoding scheme and header style. For example, SIP reuses SMTP headers such as To, From, Date, and Subject.

SIP is also a protocol still under development. A number of key extensions are still being developed. As a result, this book will contain some references to Internet Drafts instead of RFCs. Implementers must be very careful when working from an Internet Draft as there may be an issued RFC that replaces it, or another Internet Draft may come out in the future to replace it. In short, an understanding of the IETF process and the stage of development of a particular extension may be needed in some areas. In this book, only stable and mature SIP extensions are discussed; the few that have Internet Draft references will likely be published as RFCs around the same time this edition is published.

1.9 Conclusion

This chapter has introduced the Internet, Internet architecture, and standards processes. Various bodies involved in Internet standards including the IETF, IRTF, IESG, and IAB have been discussed, as well as the basics of the Domain Name Service and Uniform Resource Locators and Indicators.

References

[1] Rosenberg, J., H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, “SIP: Session Initiation Protocol,” RFC 3261, June 2002.

[2] “Internet Protocol,” RFC 791, 1981.

[3] Deering, S., and R. Hinden, “Internet Protocol, Version 6 (IPv6) Specification,” RFC 1883, 1995.

[4] Wilder, F., *A Guide to the TCP/IP Protocol Suite*, Norwood, MA: Artech House, 1998.

[5] Rekhter, Y., et al., “Address Allocation for Private Internets,” RFC 1918, February 1996.

[6] Alexander, S., and R. Droms, “DHCP Options and BOOTP Vendor Extensions,” RFC 2132, March 1997.

[7] “Transmission Control Protocol,” RFC 793, 1981.

[8] Postal, J., “User Datagram Protocol,” RFC 768, 1980.

[9] Dierks, T., et al., “The TLS Protocol Version 1.0,” RFC 2246, 1999.

[10] Stewart, R., et al., “Stream Control Transmission Protocol,” RFC 2960, 1999.

[11] Phelan, T., “Datagram Transport Layer Security (DTLS) over the Datagram Congestion Control Protocol (DCCP),” RFC 5238, May 2008.

[12] Schulzrinne, H., “Dynamic Host Configuration Protocol (DHCP-for-IPv4) Option for Session Initiation Protocol (SIP) Servers,” RFC 3361, August 2002.

[13] Crocker, D., "Standard for the Format of ARPA Internet Text Messages," STD 11, RFC 822, August 1982.

[14] Crocker, D., and P. Overell, "Augmented BNF for Syntax Specifications: ABNF," STD 68, RFC 5234, January 2008.

[15] Berners-Lee, T., L. Masinter, and M. McCahill, "Uniform Resource Locators (URL)," RFC 1738, December 1994.

[16] Moats, R., "URN Syntax," RFC 2141, May 1997.

[17] Mockapetris, P., "Domain Names—Implementation and Specification," STD 13, RFC 1035, November 1987.

[18] Faltstrom, P., and M. Mealling, "The E.164 to Uniform Resource Identifiers (URI) Dynamic Delegation Discovery System (DDDS) Application (ENUM)," RFC 3761, April 2004.

[19] Thomson, S., et al., "DNS Extensions to Support IP Version 6," RFC 3596, October 2003.

[20] Gulbrandsen, A., P. Vixie, and L. Esibov, "A DNS RR for Specifying the Location of Services (DNS SRV)," RFC 2782, February 2000.

[21] Mealling, M., and R. Daniel, "The Naming Authority Pointer (NAPTR) DNS Resource Record," RFC 2915, September 2000.

[22] http://www.ietf.org.

[23] Hoffman, P., and S. Harris, "The Tao of IETF—A Novice's Guide to the Internet Engineering Task Force," RFC 4677, September 2006.

[24] http://www.iesg.org.

[25] http://www.rfc-editor.org.

[26] http://www.iab.org.

[27] http://www.irtf.org.

[28] http://www.iana.org.

[29] http://www.icann.org.

[30] http://www.w3c.org.

[31] http://www.sipforum.org.

[32] http://www.sipforum.org/sipconnect.

[33] Handley, M., et al., "SIP: Session Initiation Protocol," RFC 2543, March 1999.

2

Introduction to SIP

Often the best way to learn a protocol is to look at examples of its use. While the terminology, structures, and format of a new protocol can be confusing at first read, an example message flow can give a quick grasp of some of the key concepts of a protocol. The example message exchanges in this chapter will introduce SIP as defined by RFC 3261 [1].

The first example shows the basic message exchange between two SIP devices to establish and tear down a session; the second example shows the message exchange when a SIP proxy server is used. The third example shows SIP registration, and the fourth shows a SIP presence and instant message example. The chapter concludes with a discussion of SIP message transmission using UDP, TCP, TLS, and SCTP.

The examples will be introduced using call flow diagrams between a called and calling party, along with the details of each message. Each arrow in the figures represents a SIP message, with the arrowhead indicating the direction of transmission. The thick lines in the figures indicate the media stream. In these examples, the media will be assumed to be RTP [2] packets containing audio, but it could be another protocol. Details of RTP are covered in Chapter 12.

2.1 A Simple Session Establishment Example

Figure 2.1 shows the SIP message exchange between two SIP-enabled devices. The two devices could be SIP phones, phone clients running on a laptop or PC (known as *softclients*), PDAs, or mobile phones. It is assumed that both devices are connected to an IP network such as the Internet and know each other's IP address.

The calling party, Tesla, begins the message exchange by sending a SIP `INVITE` message to the called party, Marconi. The `INVITE` contains the details of

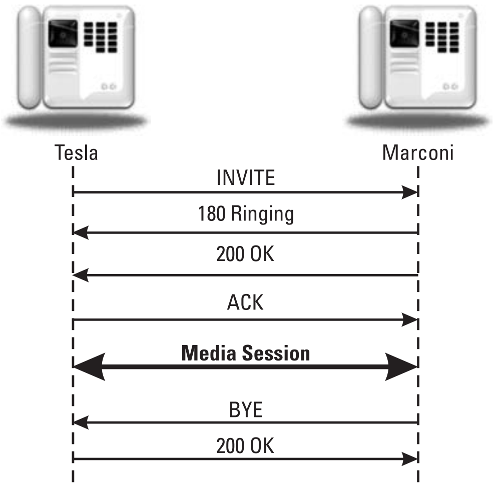


Figure 2.1 A simple SIP session establishment example.

the type of session or call that is requested. It could be a simple voice (audio) session, a multimedia session such as a videoconference, or a gaming session. The `INVITE` message contains the following fields:

```
INVITE sip:Marconi@radio.org SIP/2.0
Via: SIP/2.0/UDP lab.high-voltage.org:5060;branch=z9hG4bKfw19b
Max-Forwards: 70
To: G. Marconi <sip:Marconi@radio.org>
From: Nikola Tesla <sip:n.tesla@high-voltage.org>;tag=76341
Call-ID: j2qu348ek2328ws
CSeq: 1 INVITE
Subject: About That Power Outage...
Contact: <sip:n.tesla@lab.high-voltage.org>
Content-Type: application/sdp
Content-Length: 158

v=0
o=Tesla 2890844526 2890844526 IN IP4 lab.high-voltage.org
s=Phone Call
c=IN IP4 100.101.102.103
t=0 0
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Since SIP is a text-encoded protocol, this is actually what the SIP message would look like "on the wire" as a UDP datagram being transported over, for example, Ethernet.

The fields listed in the `INVITE` message are called header fields. They have the form `Header: Value` CRLF. The first line of the request message, called the start line, lists the method, which is `INVITE`, the Request-URI, then the SIP version number (2.0), all separated by spaces. Each line of a SIP message is terminated by a CRLF (Carriage Return Line Feed). The Request-URI is a special form of SIP URI and indicates the resource to which the request is being sent, also known as the request target. SIP URIs are discussed in more detail in Section 4.2.

The first header field following the start line shown is a `Via` header field. Each SIP device that originates or forwards a SIP message stamps its own address in a `Via` header field, usually written as a host name that can be resolved into an IP address using a DNS query. The `Via` header field contains the SIP version number (2.0), a "/", then `UDP` for UDP transport, a space, the hostname or address, a colon, then a port number (in this example the "well-known" SIP port number `5060`). Transport of SIP using TCP, UDP, TLS, and SCTP and the use of port numbers are covered later in this chapter. The `branch` parameter is a transaction identifier. Responses relating to this request can be correlated because they will contain this same transaction identifier.

The next header field shown is the `Max-Forwards`. It is initialized to some large integer and decremented by each SIP server, which receives and forwards the request, providing simple loop detection.

The next header fields are the `To` and `From` header fields, which show the originator and destination of the SIP request. SIP requests are routed based on the Request-URI instead of the `To` URI. This is because the Request-URI can be changed and rewritten as a request is forwarded, while the `To` URI generally stays the same. When a name label is used, as in this example, the SIP URI is enclosed in brackets <>. The name label could be displayed during alerting, for example, but is not used by the protocol.

The `Call-ID` header field is an identifier used to keep track of a particular SIP session. The originator of the request creates a locally unique string. Some older implementations also add an "@" and its host name to the string. In addition to the `Call-ID`, each party in the session also contributes a random identifier, unique for each call. These identifiers, called `tags`, are included in the `To` and `From` header fields as the session is established. The initial `INVITE` shown contains a `From` tag but no `To` tag.

The initiator of the session that generates the establishing `INVITE` generates the unique `Call-ID` and `From` tag. In the response to the `INVITE`, the user agent answering the request will generate the `To` tag. The combination of the local tag (contained in the `From` header field), remote tag (contained in the `To` header field), and the `Call-ID` uniquely identifies the established session, known as a *dialog*. This dialog identifier is used by both parties to identify this call because there could be multiple calls set up between them. Subsequent requests within the established session will use this dialog identifier, as will be shown in the following examples.

The next header field shown is the `CSeq`, or command sequence. It contains a number, followed by the method name, `INVITE` in this case. This number is incremented for each new request sent. In this example, the command sequence number is initialized to 1, but it could start at another integer value.

The `Via` header fields plus the `Max-Forwards`, `To`, `From`, `Call-ID`, and `CSeq` header fields represent the minimum required header field set in any SIP request message. Other header fields can be included as optional additional information, or information needed for a specific request type. A `Contact` header field is also required in this `INVITE` message, which contains the SIP URI of Tesla's communication device, known as a user agent (UA); this URI can be used to route messages directly to Tesla. The optional `Subject` header field is present in this example. It is not used by the protocol, but could be displayed during alerting to aid the called party in deciding whether to accept the call. The same sort of useful prioritization and screening commonly performed using the `Subject` and `From` header fields in an e-mail message is also possible with a SIP `INVITE` request.

The `Content-Type` and `Content-Length` header fields indicate that the message body is Session Description Protocol or SDP [3] and contains 158 octets of data. The basis for the octet count of 158 is shown in Table 2.1, where the CR LF at the end of each line is shown as a ©® and the octet count for each line is shown on the right-hand side. A blank line separates the message body from the header field list, which ends with the `Content-Length` header field. In this case, there are seven lines of SDP data describing the media attributes that the caller Tesla desires for the call. This media information is needed because SIP makes no assumptions about the type of media session to be established—the caller must specify exactly what type of session (audio, video, gaming) that he wishes to establish. The SDP field names are listed in Table 2.2, and will be discussed in detail in Chapter 13, but a quick review of the lines shows the basic information necessary to establish a session.

Table 2.2 includes the:

Table 2.1
`Content-Length` Calculation Example

Line	Total
`v=0©®`	05
`o=Tesla 2890844526 2890844526 IN IP4 lab.high-voltage.org©®`	59
`s=Phone Call©®`	14
`c=IN IP4 100.101.102.103©®`	26
`t=0 0©®`	07
`m=audio 49170 RTP/AVP 0©®`	25
`a=rtpmap:0 PCMU/8000©®`	22
	158

Table 2.2
SDP Data from Example

SDP Parameter	Parameter Name
`v=0`	Version number
`o=Tesla 2890844526 2890844526 IN IP4 lab.high-voltage.org`	Origin
`s=-`	Call subject
`c=IN IP4 100.101.102.103`	Connection
`t=0 0`	Time
`m=audio 49170 RTP/AVP 0`	Media
`a=rtpmap:0 PCMU/8000`	Attributes

- Connection IP address (100.101.102.103);
- Media format (audio);
- Port number (49170);
- Media transport protocol (RTP);
- Media encoding (PCM μ Law);
- Sampling rate (8,000 Hz).

`INVITE` is an example of a SIP request message. There are five other methods or types of SIP requests defined in the core SIP specification RFC 3261 with others defined in extension RFCs, which update RFC 3261. The next message in Figure 2.1 is a `180 Ringing` message sent in response to the `INVITE`. This message indicates that the called party, Marconi, has received the `INVITE` and that alerting is taking place. The alerting could be ringing a phone, a flashing message on a screen, or any other method of attracting the attention of the called party, Marconi.

The `180 Ringing` is an example of a SIP response message. Responses are numerical and are classified by the first digit of the number. A `180` response is an *informational class* response, identified by the first digit being a 1. Informational responses are used to convey noncritical information about the progress of the call. Many SIP response codes were based on HTTP version 1.1 response codes with some extensions and additions. Anyone who has ever browsed the World Wide Web has likely received a "`404 Not Found`" response from a Web server when a requested page was not found. `404 Not Found` is also a valid SIP *client error class* response in a request to an unknown user. The other classes of SIP responses are covered in Chapter 5.

The response code number in SIP alone determines the way the response is interpreted by the server or the user. The reason phrase, `Ringing` in this case, is suggested in the standard, but any text can be used to convey more information.

For instance, `180 Hold your horses, I’m trying to wake him up!` is a perfectly valid SIP response and has the same meaning as a `180 Ringing` response.

The `180 Ringing` response has the following structure:

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP lab.high-voltage.org:5060;branch=z9hG4bKfw19b
 ;received=100.101.102.103
To: G. Marconi <sip:marconi@radio.org>;tag=a53e42
From: Nikola Tesla <sip:n.tesla@high-voltage.org>;tag=76341
Call-ID: j2qu348ek2328ws
CSeq: 1 INVITE
Contact: <sip:marconi@tower.radio.org>
Content-length: 0
```

The message was created by copying many of the header fields from the `INVITE` message, including the `Via`, `To`, `From`, `Call-ID`, and `CSeq`, then adding a response start line containing the SIP version number, the response code, and the reason phrase. This approach simplifies the message processing for responses.

The `Via` header field contains the original `branch` parameter but also has an additional `received` parameter. This parameter contains the literal IP address that the request was received from (`100.101.102.103`), which typically is the same address that the URI in the `Via` resolves using DNS (`lab.high-voltage.org`).

Note that the `To` and `From` header fields are not reversed in the response message as one might expect them to be. Even though this message is sent to Marconi from Tesla, the header fields read the opposite. This is because the `To` and `From` header fields in SIP are defined to indicate the direction of the request, not the direction of the message. Since Tesla initiated this request, all responses to this `INVITE` will read `To:` Marconi `From:` Tesla.

The `To` header field now contains a tag that was generated by Marconi. All future requests and responses in this session or dialog will contain both the tag generated by Tesla and the tag generated by Marconi.

The response also contains a `Contact` header field, which contains an address at which Marconi can be contacted directly once the session is established.

When the called party, Marconi, decides to accept the call (i.e., the phone is answered), a `200 OK` response is sent. This response also indicates that the type of media session proposed by the caller is acceptable. The `200 OK` is an example of a *success class* response. The `200 OK` message body contains Marconi’s media information:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP lab.high-voltage.org:5060;branch=z9hG4bKfw19b
 ;received=100.101.102.103
To: G. Marconi <sip:marconi@radio.org>;tag=a53e42
From: Nikola Tesla <sip:n.tesla@high-voltage.org>;tag=76341
Call-ID: j2qu348ek2328ws
CSeq: 1 INVITE
Contact: <sip:marconi@tower.radio.org>
```

```
Content-Type: application/sdp
Content-Length: 155

v=0
o=Marconi 2890844528 2890844528 IN IP4 tower.radio.org
s=Phone Call
c=IN IP4 200.201.202.203
t=0 0
m=audio 60000 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

This response is constructed the same way as the `180 Ringing` response and contains the same `To` tag and `Contact` URI. The media capabilities, however, must be communicated in a SDP message body added to the response. From the same SDP fields as Table 2.2, the SDP contains:

- End-point IP address (200.201.202.203);
- Media format (audio);
- Port number (60000);
- Media transport protocol (RTP);
- Media encoding (PCM μ-Law);
- Sampling rate (8,000 Hz).

The final step is to confirm the media session with an *acknowledgment* request. The confirmation means that Tesla has successfully received Marconi's response. This exchange of media information allows the media session to be established using another protocol: RTP in this example.

```
ACK sip:marconi@tower.radio.org SIP/2.0
Via: SIP/2.0/UDP lab.high-voltage.rg:5060;branch=z9hG4bK321g
Max-Forwards: 70
To: G. Marconi <sip:marconi@radio.org>;tag=a53e42
From: Nikola Tesla <sip:n.tesla@high-voltage.org>;tag=76341
Call-ID: j2qu348ek2328ws
CSeq: 1 ACK
Content-Length: 0
```

The command sequence, `CSeq`, has the same number as the `INVITE`, but the method is set to `ACK`. At this point, the media session begins using the media information carried in the SIP messages. The media session takes place using another protocol, typically RTP. The `branch` parameter in the `Via` header field contains a newer transaction identifier than the `INVITE`, since an `ACK` sent to acknowledge a `200 OK` is considered a separate transaction.

This message exchange shows that SIP is an end-to-end signaling protocol. A SIP network or SIP server is not required for the protocol to be used. Two end points running a SIP protocol stack and knowing each other's IP addresses can use SIP to set up a media session between them. Although less obvious, this example also shows the client-server nature of the SIP protocol. When Tesla originates the INVITE request, he is acting as a SIP client. When Marconi responds to the request, he is acting as a SIP server. After the media session is established, Marconi originates the BYE request and acts as the SIP client, while Tesla acts as the SIP server when he responds. This is why a SIP-enabled device must contain both SIP user agent server and SIP user agent client software—during a typical session, both are needed. This is quite different from other client-server Internet protocols such as HTTP or FTP. The Web browser is always an HTTP client, and the Web server is always an HTTP server, and similarly for FTP. In SIP, an end point will switch back and forth during a session between being a client and a server.

In Figure 2.1, a BYE request is sent by Marconi to terminate the media session:

```
BYE sip:n.tesla@lab.high-voltage.org SIP/2.0
Via: SIP/2.0/UDP tower.radio.org:5060;branch=z9hG4bK392kf
Max-Forwards: 70
To: Nikola Tesla <sip:n.tesla@high-voltage.org>;tag=76341
From: G. Marconi <sip:marconi@radio.org>;tag=a53e42
Call-ID: j2qu348ek2328ws
CSeq: 1392 BYE
Content-Length: 0
```

The Via header field in this example is populated with Marconi's host address and contains a new transaction identifier since the BYE is considered a separate transaction from the INVITE or ACK transactions shown previously. The To and From header fields reflect that this request is originated by Marconi, as they are reversed from the messages in the previous transaction. Tesla, however, is able to identify the dialog using the presence of the same local and remote tags and Call-ID as the INVITE, and tear down the correct media session.

Notice that all of the branch IDs shown in the example so far begin with the string z9hG4bK. This is a special string that indicates that the branch ID has been calculated using strict rules defined in RFC 3261 and is as a result usable as a transaction identifier.[1]

The confirmation response to the BYE is a 200 OK:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP tower.radio.org:5060;branch=z9hG4bK392kf
 ;received=200.201.202.203
To: Nikola Tesla <sip:n.tesla@high-voltage.org>;tag=76341
From: G. Marconi <sip:marconi@radio.org>;tag=a53e42
Call-ID: j2qu348ek2328ws
```

[1] This string is needed because branch IDs generated by user agents prior to RFC 3261 may have constructed branch IDs which are not suitable as transaction identifiers. In this case, a client must construct its own transaction identifier using the To tag, From tag, Call-ID, and CSeq.

```
CSeq: 1392 BYE
Content-Length: 0
```

The response echoes the CSeq of the original request: 1392 BYE. No ACK is sent since ACK is only sent in response to INVITE requests.

2.2 SIP Call with a Proxy Server

In the SIP message exchange of Figure 2.1, Tesla knew the IP address of Marconi and was able to send the INVITE directly to that address. This will not be the case in general—an IP address cannot be used like a telephone number. One reason is that IP addresses are often dynamically assigned due to the shortage of IPv4 addresses. For example, when a PC dials in to an Internet service provider (ISP) modem bank, an IP address is assigned using DHCP to the PC from a pool of available addresses allocated to the ISP. For the duration of the session, the IP address does not change, but it is different for each dial-in session. Even for an "always on" Internet connection such as a DSL line, a different IP address can be assigned after each reboot of the PC. Also, an IP address does not uniquely identify a user, but identifies a node on a particular physical IP network. You have one IP address at your office, another at home, and still another when you log on remotely while traveling. Ideally, there would be one address that would identify you wherever you are. In fact, there is an Internet protocol that does exactly that with e-mail. SMTP uses a host or system independent name (an e-mail address) that does not correspond to a particular IP address. It allows e-mail messages to reach you regardless of what your IP address is and where you are logged onto the Internet.

In addition, a request routed using only IP addresses will reach only one end point—only one device. Since communication is typically user-to-user instead of device-to-device, a more useful addressing scheme would allow a particular user to call another particular user, which would result in the request reaching the target user regardless of which device they are currently using, or if they have multiple devices.

SIP uses e-mail-like names for addresses. The addressing scheme is part of a family of Internet addresses known as URIs, as described in Section 1.4. SIP URIs can also handle telephone numbers, transport parameters, and a number of other items. A full description, including examples, can be found in Section 4.2. For now, the key point is that a SIP URI is a name that is resolved to an IP address by using a SIP proxy server and DNS lookups at the time of the call, as will be seen in the next example. Figure 2.2 shows an example of a more typical SIP call with a type of SIP server called a *proxy server*. In this example, the caller Schroedinger calls Heisenberg through a SIP proxy server. A SIP proxy operates in a similar way to a proxy in HTTP and other Internet protocols. A SIP proxy

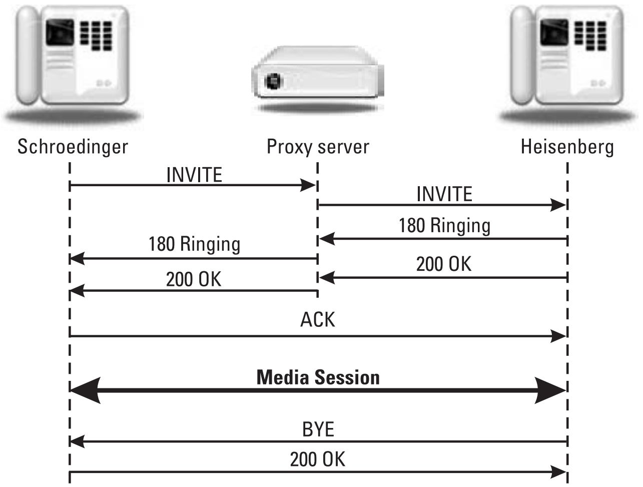


Figure 2.2 SIP call example with a proxy server.

does not set up or terminate sessions, but sits in the middle of a SIP message exchange, receiving messages and forwarding them. This example shows one proxy, but there can be multiple proxies in a signaling path.

SIP has two broad categories of URIs: ones that correspond to a user, and ones that correspond to a single device or end point. The user URI is known as an address of record (AOR), and a request sent to an address of record will require database lookups and service and feature operations, which can result in the request being sent to one or more end devices. A device URI is known as a contact, and typically does not require database lookups. An address of record URI is usually used in `To` and `From` header fields, as this is the general way to reach a person and is suitable for storing in address books and in returning missed calls. A device URI is usually used in a `Contact` header field and is associated with a particular user for a shorter period of time. The method of relating (or binding) a contact URI with an address of record URI will be discussed in Section 2.3.

Because Schroedinger does not know exactly where Heisenberg is currently logged on and what device he is currently using, a SIP proxy server is used to route the `INVITE`. First, a DNS lookup of Heisenberg's SIP URI domain name (`munich.de`) is performed (see Section 2.6 for the details), which returns the IP address of the proxy server `proxy.munich.de`, which handles that domain. The `INVITE` is then sent to that IP address:

```
INVITE sip:werner.heisenberg@munich.de SIP/2.0
Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bKmp17a
Max-Forwards: 70
To: Heisenberg <sip:werner.heisenberg@munich.de>
From: E. Schroedinger <sip:schroed5244@wave.org>;tag=42
```

```
Call-ID: 4827311-391-32934
CSeq: 1 INVITE
Subject: Where are you exactly?
Contact: <sip:schroed5244@pc33.wave.org>
Content-Type: application/sdp
Content-Length: 159

v=0
o=schroed5244 2890844526 2890844526 IN IP4 100.101.102.103
s=Phone Call
t=0 0
c=IN IP4 100.101.102.103
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

The proxy looks up the SIP URI in the Request-URI (`sip:werner.heisenberg@munich.de`) in its database and locates Heisenberg. This completes the two-step process of:

1. DNS lookup by user agent to locate the IP address of the proxy. Database lookup is performed by the proxy to locate the IP address.
2. The `INVITE` is then forwarded to Heisenberg's IP address with the addition of a second Via header field stamped with the address of the proxy.

```
INVITE sip:werner.heisenberg@200.201.202.203
Via: SIP/2.0/UDP proxy.munich.de:5060;branch=z9hG4bK83842.1
Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bKmp17a
Max-Forwards: 69
To: Heisenberg <sip:werner.heisenberg@munich.de>
From: E. Schroedinger <sip:schroed5244@wave.org>;tag=42
Call-ID: 4827311-391-32934
CSeq: 1 INVITE
Contact: <sip:schroed5244@pc33.wave.org>
Content-Type: application/sdp
Content-Length: 159

v=0
o=schroed5244 2890844526 2890844526 IN IP4 100.101.102.103
s=Phone Call
c=IN IP4 100.101.102.103
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

From the presence of two `Via` header fields, Heisenberg knows that the `INVITE` has been routed through a proxy server. Having received the `INVITE`, a `180 Ringing` response is sent by Heisenberg to the proxy:

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP proxy.munich.de:5060;branch=z9hG4bK83842.1
 ;received=100.101.102.105
Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bKmp17a
To: Heisenberg <sip:werner.heisenberg@munich.de>;tag=314159
```

```
From: E. Schroedinger <sip:schroed5244@wave.org>;tag=42
Call-ID: 4827311-391-32934
CSeq: 1 INVITE
Contact: <sip:werner.heisenberg@200.201.202.203>
Content-Length: 0
```

Again, this response contains the `Via` header fields, and the `To`, `From`, `Call-ID`, and `CSeq` header fields from the `INVITE` request. The response is then sent to the address in the first `Via` header field, `proxy.munich.de` to the port number listed in the `Via` header field: `5060`, in this case. Notice that the `To` header field now has a `tag` added to it to identify this particular dialog. Only the first `Via` header field contains a `received` parameter, since the second `Via` header already contains the literal IP address in the URI. The `Contact` header field contains the device URI of Heisenberg.

The proxy receives the response, checks that the first `Via` header field has its own address (`proxy.munich.de`), uses the transaction identifier in the `Via` header to locate the transaction, removes that `Via` header field, then forwards the response to the address in the next `Via` header field: IP address `100.101.102.103`, port `5060`. The resulting response sent by the proxy to Schroedinger is:

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bKmp17a
To: Heisenberg <sip:werner.heisenberg@munich.de>;tag=314159
From: E. Schroedinger <sip:schroed5244@wave.org>;tag=42
Call-ID: 4827311-391-32934
CSeq: 1 INVITE
Contact: <sip:werner.heisenberg@200.201.202.203>
Content-Length: 0
```

The use of `Via` header fields in routing and forwarding SIP messages reduces complexity in message forwarding. The request required a database lookup by the proxy to be routed. The response requires no lookup because the routing is imbedded in the message in the `Via` header fields. This ensures that responses route back through the same set of proxies as the request. The call is accepted by Heisenberg, who sends a `200 OK` response:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP proxy.munich.de:5060;branch=z9hG4bK83842.1
 ;received=100.101.102.105
Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bKmp17a
To: Heisenberg <sip:werner.heisenberg@munich.de>;tag=314159
From: E. Schroedinger <sip:schroed5244@wave.org>;tag=42
Call-ID: 4827311-391-32934
CSeq: 1 INVITE
Contact: <sip:werner.heisenberg@200.201.202.203>
Content-Type: application/sdp
Content-Length: 159

v=0
o=heisenberg 2890844526 2890844526 IN IP4 200.201.202.203
s=Phone Call
```

```
c=IN IP4 200.201.202.203
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

The proxy forwards the `200 OK` message to Schroedinger after removing the first `Via` header field:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bKmp17a
To: Heisenberg <sip:werner.heisenberg@munich.de>;tag=314159
From: E. Schroedinger <sip:schroed5244@wave.org>;tag=42
Call-ID: 4827311-391-32934
CSeq: 1 INVITE
Contact: <sip:werner.heisenberg@200.201.202.203>
Content-Type: application/sdp
Content-Length: 159

v=0
o=heisenberg 2890844526 2890844526 IN IP4 200.201.202.203
s=phone call
c=IN IP4 200.201.202.203
t=0 0
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

The presence of the `Contact` header field with the SIP URI address of Heisenberg in the `200 OK` allows Schroedinger to send the `ACK` directly to Heisenberg, bypassing the proxy. (Note that the Request-URI is set to Heisenberg's `Contact` URI and not the URI in the `To` header field.) This request, and all future requests, continue to use the `tag` in the `To` header field:

```
ACK sip:werner.heisenberg@200.201.202.203
Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bKka42
Max-Forwards: 70
To: Heisenberg <sip:werner.heisenberg@munich.de>;tag=314159
From: E. Schroedinger <sip:schroed5244@wave.org>;tag=42
Call-ID: 4827311-391-32934
CSeq: 1 ACK
Content-Length: 0
```

This shows that the proxy server is not really "in the call." It facilitates the two end points locating and contacting each other, but it can drop out of the signaling path as soon as it no longer adds any value to the exchange. This role of helping the two user agents locate each other is sometimes called *rendezvous* and is a key function of the SIP protocol. A proxy server can force further messaging to route through it by inserting a `Record-Route` header field, which is described in Section 6.119. In addition, it is possible to have a proxy server that does not retain any knowledge of the fact that there is a session established between Schroedinger and Heisenberg (referred to as *call state information*). This is discussed in Section 2.3.1. Note that the media is always end-to-end and not through the proxy.

In SIP the path of the signaling messages is totally independent of the path of the media. In telephony, this is described as the separation of control channel and bearer channel.

The media session is ended when Heisenberg sends a BYE message:

```
BYE sip:schroed5244@pc33.wave.org SIP/2.0
Via: SIP/2.0/UDP 200.201.202.203:5060;branch=z9hG4bK4332
Max-Forwards: 70
To: E. Schroedinger <sip:schroed5244@wave.org>;tag=42
From: Heisenberg <sip:werner.heisenberg@munich.de>
 ;tag=314159
Call-ID: 4827311-391-32934
CSeq: 2000 BYE
Content-Length: 0
```

Note that Heisenberg's CSeq was initialized to 2000. Each SIP device maintains its own independent CSeq number space. This is explained in some detail in Section 6.1.5. The Request-URI is set to Schroedinger's Contact URI. Schroedinger confirms with a 200 OK response:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 200.201.202.203:5060;branch=z9hG4bK4332
To: E. Schroedinger <sip:schroed5244@wave.org>;tag=42
From: Heisenberg <sip:werner.heisenberg@munich.de>
 ;tag=314159
Call-ID: 4827311-391-32934
CSeq: 2000 BYE
Content-Length: 0
```

Not discussed in the previous example is the question of how the database accessed by the proxy contained Heisenberg's current IP address. There are many ways this could be done using SIP or other protocols. The mechanism for accomplishing this using SIP is called *registration* and is discussed in the next section.

2.3 SIP Registration Example

In this example, shown in Figure 2.3, Heisenberg sends a SIP REGISTER request to a type of SIP server known as a registrar server. The SIP registrar server receives the message and uses the information in the request to update the database used by proxies to route SIP requests. Contained in the REGISTER message To header is the SIP URI address of Heisenberg. This is Heisenberg's well-known address, perhaps printed on his business card or published on a Web page or in a directory. Also contained in the REGISTER is a Contact URI, which represents the current device (and its IP address) that the user Heisenberg is currently using. The

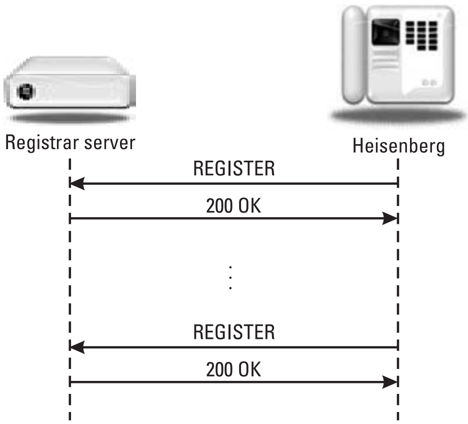


Figure 2.3 SIP registration example.

registrar binds the SIP URI of Heisenberg and the IP address of the device in a database that can be used, for example, by the proxy server in Figure 2.2 to locate Heisenberg. When a proxy server with access to the database receives an `INVITE` request addressed to Heisenberg's URI (i.e., an incoming call), the request will be proxied to the `Contact` URI of the currently registered device.

This registration has no real counterpart in the telephone network, but it is very similar to the registration a wireless phone performs when it is turned on. A mobile phone sends its identity to the base station (BS), which then forwards the location and phone number of the mobile phone to a home location register (HLR). When the mobile switching center (MSC) receives an incoming call, it consults the HLR to get the current location of the mobile phone. Further aspects of SIP mobility are discussed in Chapter 7.

The `REGISTER` message sent by Heisenberg to the SIP registrar server has the form:

```
REGISTER sip:registrar.munich.de SIP/2.0
Via: SIP/2.0/UDP 200.201.202.203:5060;branch=z9hG4bKus19
Max-Forwards: 70
To: Werner Heisenberg <sip:werner.heisenberg@munich.de>
From: Werner Heisenberg <sip:werner.heisenberg@munich.de>
 ;tag=3431
Call-ID: 73764291
CSeq: 1 REGISTER
Contact: sip:werner.heisenberg@200.201.202.203
Content-Length: 0
```

The Request-URI in the start line of the message contains the address of the registrar server. In a `REGISTER` request, the `To` header field contains the URI that is being registered, in this case `sip:werner.heisenberg@munich.de`. This results in the `To` and `From` header fields usually being the same, although an example of third-party registration is given in Section 4.1.2. The SIP URI in the Contact address is stored by the registrar.

The registrar server acknowledges the successful registration by sending a 200 OK response to Heisenberg. The response echoes the Contact information that has just been stored in the database and includes a To tag:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 200.201.202.203:5060;branch=z9hG4bKus19
To: Werner Heisenberg <sip:werner.heisenberg@munich.de>
 ;tag=8771
From: Werner Heisenberg <sip:werner.heisenberg@munich.de>
 ;tag=3431
Call-ID: 73764291
CSeq: 1 REGISTER
Contact: <sip:werner.heisenberg@munich.de>;expires=3600
Content-Length: 0
```

The Contact URI is returned along with an expires parameter, which indicates how long the registration is valid (in this case, 1 hour, or 3,600 seconds). If Heisenberg wants the registration to be valid beyond that interval, he must send another REGISTER request within the expiration interval.

Registration is typically performed automatically on initialization of a SIP device and at regular intervals determined by the expiration interval chosen by the registrar server. Registration is an additive process—more than one device can be registered against a SIP URI. If more than one device is registered, a proxy may forward the request to either or both devices in a sequential or parallel search. Additional register operations can be used to clear registrations or retrieve a list of currently registered devices.

2.4 SIP Presence and Instant Message Example

This example shows how SIP is used in a presence and instant messaging application. Presence information can be thought of as the state of a user or device, or willingness to communicate at a particular instant. It can be as simple as whether a particular user is signed in or not, whether they are active at their station, or idle or away. For a mobile device, presence information can include the actual location in terms of coordinates, or in general terms such as “in the office,” “traveling,” or “in the lab.” Presence information can even include information about the status or mood of the user, whether they are working, relaxing, or socializing. For all these examples, a presence protocol is mainly concerned about establishing subscriptions or long-term relationships between devices about transferring status information, and the delivery of that information. The actual information transferred, and how that information is presented to the user, is application dependent. In terms of the SIP protocol, SUBSCRIBE is used to request status or presence updates from the presence server (or *presentity*), and `NOTIFY` is used to deliver that information to the requestor or presence *watcher*. SIP presence uses the SIP Events extensions [4, 5].

In this example, Chebychev wishes to communicate with Poisson. The message flow is shown in Figure 2.4. To find out the status of Poisson, Chebychev subscribes to Poisson's presence information by sending a `SUBSCRIBE` message to Poisson. The request looks like:

```
SUBSCRIBE sip:poisson@probability.org SIP/2.0
Via SIP/2.0/TCP lecturehall21.academy.ru:5060
 ;branch=z9hG4bK348471123
Max-Forwards: 70
To: M. Poisson <sip:poisson@probability.org>
From: P. L. Chebychev <sip:chebychev@academy.ru>;tag=21171
Call-ID: 58dkfj34924lk34452k592520
CSeq: 3412 SUBSCRIBE
Allow-Events: presence
Allow: ACK, INVITE, CANCEL, BYE, NOTIFY, SUBSCRIBE, MESSAGE
Contact: <sip:pafnuty@lecturehall21.academy.ru:37129;transport=tcp>
Event: presence
Content-Length: 0
```

In this example, TCP is used as the transport for the SIP messages as indicated in the `Via` header field and in the `transport=tcp` parameter in the

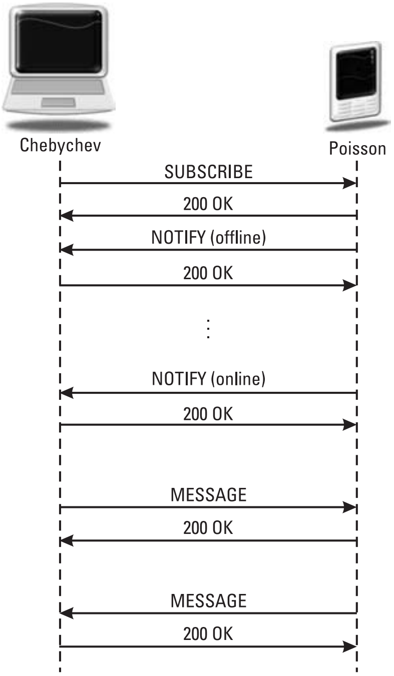


Figure 2.4 SIP presence and instant messaging example.

`Contact` URI. Also note that a nondefault port number, port `37129` is used for this `Contact` URI. This request also contains `Allow` and `Allow-Events` header fields, which are used to advertise capabilities. In this example, Chebychev is indicating support for receiving seven methods listed in the `Allow` header field, and also presence subscriptions in the `Allow-Event` header field. As this `SUBSCRIBE` is creating a dialog (in an analogous way that an `INVITE` created a dialog in the earlier examples), the `From` contains a tag but the `To` header field does not yet contain a tag.

Poisson accepts the subscription request by sending a `200 OK` response back to Chebychev:

```
SIP/2.0 200 OK
Via SIP/2.0/TCP lecturehall21.academy.ru:5060
 ;branch=z9hG4bK348471123;received=19.34.3.1
To: M. Poisson <sip:poisson@probability.org>;tag=25140
From: P. L. Chebychev <sip:chebychev@academy.ru>;tag=21171
Call-ID: 58dkfj349241k34452k592520
CSeq: 3412 SUBSCRIBE
Allow-Events: dialog, presence
Allow: ACK, INVITE, CANCEL, BYE, NOTIFY, SUBSCRIBE, MESSAGE
Contact: <sip:s.possion@dist.probability.org;transport=tcp>
Event: presence
Expires: 3600
Content-Length: 0
```

In this example, there are no proxy servers between Chebychev's watcher and Poisson's presence server, although there could be any number. The `Expires` header field indicates that the subscription expires in 1 hour. The actual subscription is begun by Poisson sending the first `NOTIFY` back to Chebychev:

```
NOTIFY sip:pafnuty@lecturehall21.academy.ru:37129 SIP/2.0
Via SIP/2.0/TCP dist.probablilty.org:5060
 ;branch=z9hG4bK4321
Max-Forwards: 70
To: P. L. Chebychev <sip:chebychev@academy.ru>;tag=21171
From: M. Poisson <sip:poisson@probability.org>;tag=25140
Call-ID: 58dkfj349241k34452k592520
CSeq: 1026 NOTIFY
Allow: ACK, INVITE, CANCEL, BYE, NOTIFY, SUBSCRIBE, MESSAGE
Allow-Events: dialog, presence
Contact: <sip:s.possion@dist.probability.org;transport=tcp>
Subscription-State: active;expires=3600
Event: presence
Content-Type: application/pidf+xml
Content-Length: 244

<?xml version=”1.0” encoding=”UTF-8”?>
<presence xmlns=”urn:ietf:params:xml:ns:pidf”
 entity=”sip:poisson@probability.org”>
 <tuple id=”452426775”>
 <status>
  <basic>closed</basic>
 </status>
 </tuple>
</presence>
```

Note that this NOTIFY is sent within the dialog established with the SUBSCRIBE—it uses the same dialog identifier (Call-ID, local and remote tags)—and the request is sent to the Contact URI provided by Chebychev in the subscription request. The Subscription-State header field indicates that the subscription has been authorized and activated and that it will expire in 1 hour unless refreshed by Chebychev (using another SUBSCRIBE request).

The Common Presence and Instant Message Presence Information Data Format (CPIM PIDF) [6] XML message body contains the status information that Poisson is currently off-line (closed).

Chebychev sends a 200 OK response to the NOTIFY to confirm that it has been received:

```
SIP/2.0 200 OK
Via SIP/2.0/TCP dist.probablilty.org:5060
 ;branch=z9hG4bK4321;received=24.32.1.3
To: P. L. Chebychev <sip:chebychev@academy.ru>;tag=21171
From: M. Poisson <sip:poisson@probability.org>;tag=25140
Call-ID: 58dkfj34924lk34452k592520
CSeq: 1026 NOTIFY
Content-Length: 0
```

Later, when Poisson does sign in, this information is provided in a second NOTIFY containing the change in status:

```
NOTIFY sip:pafnuty@lecturehall21.academy.ru SIP/2.0
Via SIP/2.0/TCP dist.probablilty.org:5060
 ;branch=z9hG4bK334241
Max-Forwards: 70
To: P. L. Chebychev <sip:chebychev@academy.ru>;tag=21171
From: M. Poisson <sip:poisson@probability.org>;tag=25140
Call-ID: 58dkfj34924lk34452k592520
CSeq: 1027 NOTIFY
Allow: ACK, INVITE, CANCEL, BYE, NOTIFY, SUBSCRIBE, MESSAGE
Allow-Events: presence
Contact: <sip:s.possion@dist.probability.org;transport=tcp>
Subscription-State: active;expires=1800
Event: presence
Content-Type: application/pidf+xml
Content-Length: 325

<?xml version=″1.0″ encoding=″UTF-8″?>
<presence xmlns=″urn:ietf:params:xml:ns:pidf″
 entity=″sip:poisson@probability.org″>
 <tuple id=″452426775″>
 <status>
  <basic>open</basic>
 </status>
  <contact>sip:s.possion@dist.probability.org;transport=tcp
  </contact>
 </tuple>
</presence>
```

The expiration time indicated in the Subscription-State header field indicates that 30 minutes have passed since the subscription was established. The CPIM PIDF XML message body now indicates that Poisson is online (open) and can be reached via the URI:

```
sip:s.possion@dist.probability.org;transport=tcp
```

Chebychev confirms receipt of the NOTIFY with a 200 OK response:

```
SIP/2.0 200 OK
Via SIP/2.0/TCP dist.probablilty.org:5060
 ;branch=z9hG4bK334241;received=24.32.1.3
To: P. L. Chebychev <sip:chebychev@academy.ru>;tag=21171
From: M. Poisson <sip:poisson@probability.org>;tag=25140
Call-ID: 58dkfj34924lk34452k592520
CSeq: 1027 NOTIFY
Content-Length: 0
```

Now that Chebychev knows that Poisson is online, he sends an instant message to him using the Contact URI from the NOTIFY:

```
MESSAGE sip:s.possion@dist.probability.org SIP/2.0
Via SIP/2.0/TCP lecturehall21.academy.ru:5060
 ;branch=z9hG4bK3gtr2
Max-Forwards: 70
To: M. Poisson <sip:s.possion@dist.probability.org>
From: P. L. Chebychev <sip:chebychev@academy.ru>;tag=4542
Call-ID: 9dkei93vjq1ei3
CSeq: 15 MESSAGE
Allow: ACK, INVITE, CANCEL, BYE, NOTIFY, SUBSCRIBE, MESSAGE
Content-Type: text/plain
Content-Length: 11

Hi There!
```

Notice that this MESSAGE is sent outside the dialog. Instant messages sent using the MESSAGE method in SIP are like page messages—they are not part of any dialog. As a result, each message contains a new Call-ID and From tag. The 200 OK response is used to acknowledge receipt of the instant message.

```
SIP/2.0 200 OK
Via SIP/2.0/TCP lecturehall21.academy.ru:5060
 ;branch=z9hG4bK3gtr2;received=19.34.3.1
To: M. Poisson <sip:s.possion@dist.probability.org>;tag=2321
From: P. L. Chebychev <sip:chebychev@academy.ru>;tag=4542
Call-ID: 9dkei93vjq1ei3
CSeq: 15 MESSAGE
Content-Length: 0
```

Poison answers with a reply, which is also sent outside of any dialog, with a new `Call-ID` and `From tag` (an instant message response is never sent in a `200 OK` reply to a `MESSAGE` request).

```
MESSAGE sip:chebychev@academy.ru SIP/2.0
Via SIP/2.0/TCP dist.probablilty.org:5060
 ;branch=z9hG4bK4526245
Max-Forwards: 70
To: P. L. Chebychev <sip:chebychev@academy.ru>
From: M. Poisson <sip:s.possion@dist.probability.org>
 ;tag=14083
Call-ID: lk34452k592520
CSeq: 2321 MESSAGE
Allow: ACK, INVITE, CANCEL, BYE, NOTIFY, SUBSCRIBE, MESSAGE
Content-Type: text/plain
Content-Length: 2

Well, hello there to you, too!
```

This receives a `200 OK` reply:

```
SIP/2.0 200 OK
Via SIP/2.0/TCP dist.probablilty.org:5060
 ;branch=z9hG4bK4526245;received=24.32.1.3
To: P. L. Chebychev <sip:chebychev@academy.ru>
 ;tag=mc3bg5q77wms
From: M. Poisson <sip:s.possion@dist.probability.org>;
 tag=14083
Call-ID: lk34452k592520
CSeq: 2321 MESSAGE
Content-Length: 0
```

Other presence packages define other sets of information that can be requested by watchers from presence servers. Further examples of SIP presence and instant messaging can be found in Chapter 8.

2.5 Message Transport

As discussed in Chapter 1, SIP is an application layer protocol in the Internet Multimedia Protocol stack shown in Figure 1.1. RFC 3261 defines the use of TCP, UDP, or TLS transport. An extension document describes how SCTP can be used. How a SIP message is transported using these four protocols will be described in the following sections. The compression of SIP for transport over low bandwidth connections, such as wireless, is discussed in Chapter 7.

2.5.1 UDP Transport

When using UDP, each SIP request or response message is carried in a single UDP datagram or packet. For a particularly large message body, there is a compact form of SIP that saves space in representing some header fields with a single character. This is discussed in Chapter 6. Figure 2.5 shows a SIP `BYE` request exchange during an established SIP session using UDP.

The source port is chosen from a pool of available port numbers (above `49172`), or the default SIP port of `5060` can be used. The lack of handshaking or acknowledgment in UDP transport means that a datagram could be lost along with a SIP message. The checksum, however, enables UDP to discard errored datagrams, allowing SIP to assume that a received message is complete and error-free. The reliability mechanisms built into SIP to handle message retransmissions are described in Section 3.7. The reply is also sent to port `5060`, or the port number listed in the top `Via` header field.

UDP provides the simplest transport for user agents and servers, and allows them to operate without the transport layer state. However, UDP offers no congestion control. A series of lost packets on a heavily loaded IP link can cause retransmissions, which in turn produce more lost packets and can push the link into congestion collapse. Also, UDP may only be used for SIP when the message (and its response) is known to be less than the Message Transport Unit (MTU) size of the IP network. For simple SIP messages, this is not a problem. However, for large messages containing multiple message bodies and large header fields, this can be a problem. In this case, TCP must be used, since SIP does not support fragmentation at the SIP layer.

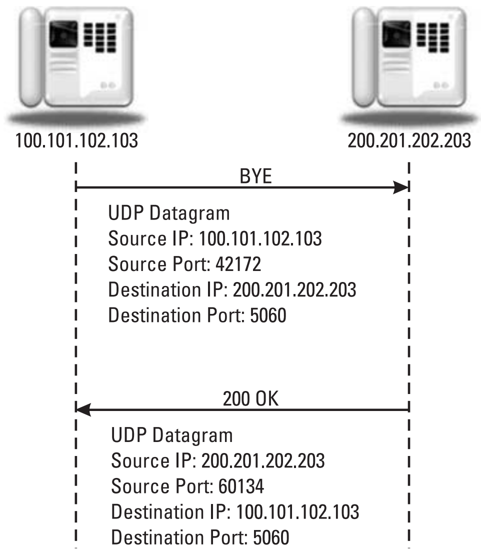


Figure 2.5 Transmission of a SIP message using UDP.

2.5.2 TCP Transport

TCP provides a reliable transport layer, but at a cost of complexity and transmission delay over the network. The use of TCP for transport in a SIP message exchange is shown in Figure 2.6. This example shows an `INVITE` sent by a user agent at `100.101.103.103` to a type of SIP server called a redirect server at `200.201.202.203`. A SIP redirect server does not forward `INVITE` requests like a proxy, but looks up the destination address and instead returns that address in a redirection class (`3xx`) response. The `302 Moved Temporarily` response is acknowledged by the user agent with an `ACK` message. Not shown in this figure is the next step, where the `INVITE` would be resent to the address returned by the redirect server. As in the UDP example, the well-known SIP port number of `5060` is chosen for the destination port, and the source port is chosen from an available pool of port numbers. Before the message can be sent, however, the TCP connection must be opened between the two end points. This transport layer datagram exchange is shown in Figure 2.6 as a single arrow, but it is actually a three-way handshake between the end points as shown in Figure 1.2. Once the connection is established, the messages are sent in the stream.

The `Content-Length` header field is critical when TCP is used to transport SIP, since it is used to find the end of one message and the start of the next. When TCP or another stream-based transport is used, `Content-Length` is a required header field in all requests and responses.

To send the `302 Moved Temporarily` response, the server typically opens a new TCP connection in the reverse direction, using `5060` (or the port listed in

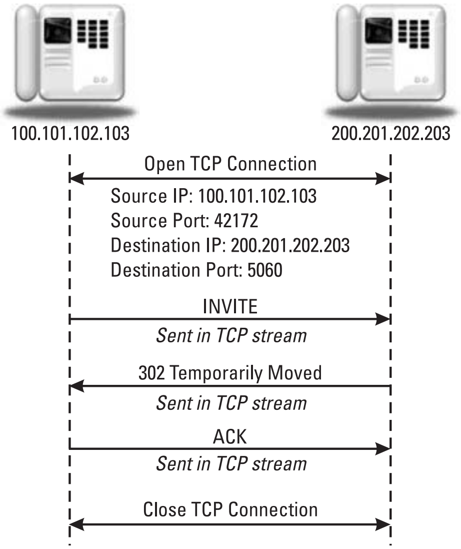


Figure 2.6 Transmission of a SIP message using TCP.

the top Via header field) as the destination port. The acknowledgment ACK is sent in the TCP stream used for the INVITE. Because this concludes the SIP session, the connection is then closed. If a TCP connection closes during a dialog, a new one can be opened to send a request within the dialog, such as a BYE request to terminate the media session.

As previously mentioned, TCP provides reliable transport and congestion control. It can also transport SIP messages of arbitrary sizes. The disadvantages of TCP include the setup delay in establishing the connection and the need for servers to maintain this connection state at the transport layer.

2.5.3 TLS Transport

SIP can use TLS [7] over TCP the same way as for encrypted transport, with the additional capabilities of authentication. In Section 4.2.1 the secure SIP URI scheme (sips) will be discussed, which uses TLS transport. The default SIP port number for TLS transport is port 5061.

If TLS is used between two proxies, each proxy may have a certificate allowing mutual authentication. However, if a client does not have a certificate, TLS can be used in conjunction with another authentication mechanism, such as SIP digest, to allow mutual authentication.

The SIP use of TLS takes advantage of both the encryption and authentication services. However, the encryption and authentication is only useful on a single hop. If a SIP request takes multiple hops (i.e., includes one or more proxy servers), TLS is not useful for end-to-end authentication. SIP proxies must support TLS and will likely use TLS for long-lived connections. TLS will be covered more in Chapter 14.

2.5.4 SCTP Transport

An extension to SIP defines the use of SCTP [8] with SIP to provide reliable stream-based transport with some advantages over TCP transport for a message-based protocol such as SIP. First, it has built-in message segmentation, so that individual SIP messages are separated at the transport layer. With TCP, the SIP protocol must use the Content-Length calculation to delineate messages. If a TCP connection is being shared by a number of SIP transactions and dialogs, the "head of line blocking" problem discussed in Section 1.2.4.5 can cause the buffer to contain valid SIP messages that could be processed by the server while the retransmission takes place. Due to its message level delineation, SCTP is able to continue to forward messages to the application layer while simultaneously requesting a retransmission of a dropped message. Note that this is only a problem when multiple applications are multiplexed over a single TCP connection. An example of this is a TCP link between two signaling proxy servers. For a user agent to proxy TCP connection, this is usually not a problem unless the two have many simultaneous dialogs established.

SCTP also supports multihoming, so if one of a pair of load balancing SIP proxies fails, the other can immediately begin receiving the messages without even requiring a DNS or other database lookup. The SIP usage of SCTP is described in [9].

2.6 Transport Protocol Selection

Since SIP supports multiple transport protocols, it must have a way of managing them. The full SIP usage of DNS is defined in [10], but the basic steps for a client are listed here. There are two ways that this is acheived. The first is through explicit indications in a SIP URI. The presence of a `transport=tcp` or `transport=sctp` [9] indicates that the particular transport protocol should be used for this URI. Note that for TLS transport, the SIPS URI scheme should be used, although some implementations use the nonstandard `transport=tls` parameter. UDP is used if `transport=udp` is included. When no `transport` parameter is included, the following rules are followed:

1. If the URI has a numeric IP address, then UDP should be used for a SIP URI and TCP for a SIPS URI.
2. If the URI does not have a numeric address but has a numeric port number, then UDP should be used for a SIP URI and TCP for a SIPS URI.
3. If the URI does not have a numeric IP address or port, and NAPTR DNS queries are supported, then a DNS NAPTR query should be performed on the host part of the URI. The NAPTR service fields are “SIP+D2U” for UDP, “SIP+D2T” for TCP, and “SIP+D2S” for SCTP transport. The result of the NAPTR regex replacement will be a URI, which is used for an SRV lookup described in the next step. The preference field indicates the relative preference if multiple transports are supported. If no NAPTR records are returned, an SRV query should be performed.
4. The SIP usage of DNS SRV records uses “_sip” or “_sips” for the protocol and “_udp”, “_tcp”, and “_sctp” for UDP, TCP, and SCTP transports. The results of the SRV query will be a target hostname and port number. The request should be sent to that address and port. Full details on SRV record usage are in [11].

5. If no SRV records are found, then an address query for A or an AAAA DNS query should be performed, and UDP should be used for a SIP URI and TCP for a SIPS URI.

A slightly different set of rules are followed by proxy servers, as described in [10].

For example, consider the DNS lookup performed by Schroedinger in Figure 2.2. The URI is `sip:werner.heisenberg@munich.de`, which does not contain a numeric IP address or port, so steps 1 and 2 are not followed. Schroedinger then follows step 3 and performs a NAPTR query on `munich.de` which returns the following record:

```
munich.de. 360 IN NAPTR 100 50 "s" "SIPS+DTU" "" _sip._udp.munich.de
```

Since only UDP transport is supported, step 4 results in an SRV query on `_sip._udp.munich.de`, which returns the following record:

```
_sip._udp.munich.de. 300 IN SRV 0 100 5060 proxy.munich.de
```

Finally, an A lookup is performed on `proxy.munich.de` which returns:

```
proxy.munich.de. 3600 IN A 100.101.102.105
```

As a result, Schroedinger sends the `INVITE` to `100.101.102.105:5060` as shown in Figure 2.2.

2.7 Conclusion

This chapter introduced the Session Initiation Protocol using some common call flow examples including a basic call, call through a proxy server, registration, and presence and instant messaging. The next chapter will explore further the details of SIP and the behavior or standard elements such as user agents, proxies, redirect servers, and registrars.

2.8 Questions

Q2.1 Define a SIP dialog.

Q2.2 What SIP parameter carries the SIP transaction identifier?

Q2.3 Compare proxy, registrar, and redirect servers.

Q2.4 Which SIP methods create dialogs? Which SIP methods end dialogs?

Q2.5 Explain the purpose of the `Contact` header field in an `INVITE`.

Q2.6 Is the `Content-Length` header field mandatory for TCP transport? Why or why not?

Q2.7 What is the purpose of `Via` header fields?

Q2.8 The DNS application Dig has returned the following values. What are the three types of DNS Resource returned? Explain the meaning of each field of the record for `_sip._tcp.iptel.org`. What IP address and port would a SIP request (Service = sip) be sent to at the `iptel.org` domain, assuming TCP transport (Proto=tcp)?

```
$ dig _sip._tcp.iptel.org in srv

; <<>> DiG 9.3.4 <<>> _sip._tcp.iptel.org in srv
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id:
15807
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 5,
ADDITIONAL: 5

;; QUESTION SECTION:
;_sip._tcp.iptel.org.      IN   SRV

;; ANSWER SECTION:
_sip._tcp.iptel.org.  86400  IN   SRV   0 0 15060 sip.
iptel.org.

;; AUTHORITY SECTION:
iptel.org.       51797  IN   NS   ns4.mydyndns.org.
iptel.org.       51797  IN   NS   ns5.mydyndns.org.
iptel.org.       51797  IN   NS   ns.iptel.org.
iptel.org.       51797  IN   NS   ns2.mydyndns.org.
iptel.org.       51797  IN   NS   ns3.mydyndns.org.

;; ADDITIONAL SECTION:
sip.iptel.org.     86400  IN   A    213.192.59.75
ns2.mydyndns.org.    41406  IN   A    204.13.249.76
ns3.mydyndns.org.    41406  IN   A    208.78.69.76
ns4.mydyndns.org.    41406  IN   A    91.198.22.76
ns5.mydyndns.org.    43021  IN   A    203.62.195.76

;; Query time: 178 msec
;; SERVER: 192.168.0.1#53(192.168.0.1)
;; WHEN: Wed Feb 6 09:56:07 2008
;; MSG SIZE rcvd: 248
```

Q2.9 Explain the difference between the Request-URI and the `To` URI in a SIP `INVITE`.

Q2.10 Explain the meaning of each of the parameters in the following `Via` header field:

```
Via: SIP/2.0/SCTP room42.lib.edu:4213
 ;branch=z9hG4bK3423;received=13.34.3.1
```

References

[1] Rosenberg, J., H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261, June 2002.

[2] Schulzrinne, H., et al., "RTP: A Transport Protocol for Real-Time Applications," STD 64, RFC 3550, July 2003.

[3] Handley, M., V. Jacobson, and C. Perkins, "SDP: Session Description Protocol," RFC 4566, July 2006.

[4] Roach, A., "Session Initiation Protocol (SIP)—Specific Event Notification," RFC 3265, June 2002.

[5] Campbell, B., et al., "Session Initiation Protocol (SIP) Extension for Instant Messaging," RFC 3428, December 2002.

[6] Sugano, H., et al., "Presence Information Data Format (PIDF)," RFC 3863, August 2004.

[7] Dierks, T., and C. Allen, "The TLS Protocol Version 1.0," RFC 2246, January 1999.

[8] Stewart, R., et al., "Stream Control Transmission Protocol," RFC 2960, October 2000.

[9] Rosenberg, J., H. Schulzrinne, and G. Camarillo, "The Stream Control Transmission Protocol (SCTP) as a Transport for the Session Initiation Protocol (SIP)," RFC 4168, October 2005.

[10] Rosenberg, J., and H. Schulzrinne, "SIP: Session Initiation Protocol," RFC 3263, June 2002.

[11] Gulbrandsen, A., P. Vixie, and L. Esibov, "A DNS RR for Specifying the Location of Services (DNS SRV)," RFC2782, February 2000.

3

SIP Clients and Servers

The client-server nature of SIP has been introduced in the example message flows of Chapter 2. In this chapter, the types of clients and servers in a SIP network will be introduced and defined.

3.1 SIP User Agents

A SIP-enabled end device is called a SIP user agent (UA) [1]. One purpose of SIP is to enable sessions to be established between user agents. As the name implies, a user agent takes direction or input from a user and acts as an agent on their behalf to set up and tear down media sessions with other user agents. In most cases the user will be a human, but the user could also be another protocol, as in the case of a gateway (described in Section 3.4). A UA must be capable of establishing a media session with another UA.

A UA must maintain the state on calls that it initiates or participates in. A minimum call state set includes the local and remote `tags`, `Call-ID`, local and remote `CSeq` header fields, along with the route set and any state information necessary for the media. This information is used to store the dialog information and for reliability. The remote `CSeq` storage is necessary to distinguish between a new request and a retransmission of an old request. A re-`INVITE` is used to change the session parameters of an existing or pending call. It uses the same `Call-ID` and `tags` as the original `INVITE/200 OK` exchange, but the `CSeq` is incremented because it is a new request. A retransmitted `INVITE` will contain the same `Call-ID` and `CSeq` as a previous `INVITE`. Even after a call has been terminated, the call state must be maintained by a user agent for at least 32 seconds in case of lost messages in the call tear down.

User agents silently discard an `ACK` for an unknown dialog. Requests to an unknown URI receive a `404 Not Found` response. A user agent receiving a request for an unknown dialog responds with a `481 Dialog/Transaction Does Not Exist`. Responses from an unknown dialog are also silently discarded. These silent discards are necessary for security. Otherwise, a malicious user agent could gain information about other SIP user agents by spamming fake requests or responses.

A minimal implementation must to be able to interpret any unknown response based on the class (first digit of the number) of the response, but it is not required to understand every response code defined. That is, if an undefined `498 Wrong Phase of the Moon` response is received, it must be treated as a `400 Client Error`.

A user agent responds to an unsupported request with a `501 Not Implemented` response. For example, a UA receiving a method that it does not support would return a `501` response. A SIP UA must support UDP and TCP transport if it sends messages greater than 1,000 octets in size.

A SIP user agent contains both a client application and a server application. The two parts are a user agent client (UAC) and user agent server (UAS). The UAC initiates requests while the UAS generates responses. During a session, a user agent will usually operate as both a UAC and a UAS.

A SIP user agent must also support Session Description Protocol (SDP) for media description. Other types of media description protocols can be used in bodies, but SDP support is mandatory. Details of SDP are in Section 13.1.

A UA must understand any extensions listed in a `Require` header field in a request. Unknown header fields may be ignored by a UA. A UA should advertise its capabilities and features in any request it sends. This allows other UAs to learn them without having to make an explicit capabilities query. For example, the methods that a UA supports should be listed in an `Allow` header field. SIP extensions should be listed in a `Supported` header field. Message body types that are supported should be listed in an `Accept` header field.

UAs typically register with a proxy server in their domain.

3.2 Presence Agents

A presence agent (PA) [2] is a SIP UA that is capable of receiving subscription requests and generating state notifications as defined by the SIP Events specification [3]. An example of a presence agent can be found in Section 3.4. A presence agent supports the presence event package [2], responds to `SUBSCRIBE` requests, and sends `NOTIFY` requests. A presence agent also sometimes publishes event state to an event state compositor (ESC) using `PUBLISH` requests, as described in Section 4.1.9.

A presence agent can collect presence information from a number of devices. Presence information can come from a SIP device registering, a SIP device publishing presence information [4], or many other non-SIP sources.

A presence server is also a presence UA that can supply presence information about a number of users, and can also act as a proxy, forwarding `SUBSCRIBE` requests to another presence agents.

A presence agent first authenticates a subscription request. If the authentication passes, it establishes a dialog and sends the notifications over that dialog. The subscription can be refreshed by receiving new `SUBSCRIBE` requests.

Chapter 8 has a complete description of presence agents.

3.3 Back-to-Back User Agents

A back-to-back user agent (B2BUA) is a type of SIP UA that receives a SIP request, then reformulates the request and sends it out as a new request. As such, some B2BUAs act like a proxy but do not follow proxy routing rules. For example, a B2BUA device can be used to implement an anonymizer service in which two SIP UAs can communicate without either party learning the other party's URI, IP address, or other information. To achieve this, an anonymizer B2BUA would reformulate an incoming request with an entirely new `From`, `Via`, `Contact`, `Call-ID`, and SDP media information, also removing any other SIP header fields that might contain information about the calling party. The response returned would also change the `Contact` and SDP media information from the called party. The modified SDP would point to the B2BUA itself, which would forward RTP media packets from the called party to the calling party and vice versa. In this way, neither end point learns any identifying information about the other party during the session establishment. (Of course, the calling party needs to know the called party's URI in order for the call to take place.)

Sometimes B2BUAs are employed to implement other SIP services. However, they break the end-to-end nature of an Internet protocol such as SIP. Also, a B2BUA is a call-stateful single point of failure in a network, which means their use will reduce the reliability of SIP sessions over the Internet. The relayed media suffers from increased latency and increased probability of packet loss, which can reduce the quality of the media session. Geographic distribution of B2BUAs can reduce these effects, but the problem of selecting the best B2BUA for a particular session is a very difficult one since the source and destination IP addresses of the media are not known until the session is actually established (with a `200 OK`).

B2BUAs can be a part of many devices. For example, many private branch exchange (PBX) enterprise telephone systems incorporate B2BUA logic. Conference bridges and mixers also use B2BUA logic. Another type of B2BUA present in some SIP networks is application layer gateways (ALG). Some firewalls have

ALG functionality built in, which allows a firewall to permit SIP and media traffic while still maintaining a high level of security. Another common type of B2BUA is known as a Session Border Controller (SBC). Some common functions of a SBC are listed in Table 3.1 which is from [5]. Note that many of these functions break the end-to-end security properties of SIP and SIP security.

3.4 SIP Gateways

A SIP gateway is an application that interfaces a SIP network to a network utilizing another signaling protocol. In terms of the SIP protocol, a gateway is just a special type of user agent, where the user agent acts on behalf of another protocol rather than a human. A gateway terminates the signaling path and can also terminate the media path, although this is not always the case. For example, a SIP to H.323 gateway terminates the SIP signaling path and converts the signaling to H.323, but the SIP user agent and H.323 terminal can exchange RTP media information directly with each other without going through the gateway. An example of this is described in Section 11.4.

A SIP to Public Switched Telephone Network (PSTN) gateway terminates both the signaling and media paths. SIP can be translated into, or interwork with, common PSTN protocols such as Integrated Services Digital Network (ISDN), ISDN User Part (ISUP), and other circuit associated signaling (CAS) protocols, which are briefly described in Section 11.1. A PSTN gateway also converts the RTP media stream in the IP network into a standard telephony trunk or line. The conversion of signaling and media paths allows calling to and from the PSTN using SIP. Examples of these gateways are described in Section 16.2. Figure 3.1 shows a SIP network connected via gateways with the PSTN and an H.323 network. The SIP/H.323 interworking function is described in [6].

In Figure 3.1, the SIP network, PSTN network, and H.323 networks are shown as clouds, which obscure the underlying details. Connecting to the SIP

Table 3.1
Session Border Controller Functions

Function	Use
Topology hiding	Hiding all internal IP addresses to conceal internal topology.
Media traffic management	Controlling which media types and codecs are used.
Fixing capability mismatches	Ensuring interop when multiple ways of implementing features happens (e.g., transfer with REFER or 3PCC).
Maintaining NAT mappings	Keeping SIP-related UDP NAT Mappings alive.
Access control	Authenticating and challenging requests.
Protocol repair	Fixing known SIP interoperability failures in devices.
Media encryption	Allows SRTP in external network but RTP in internal network.

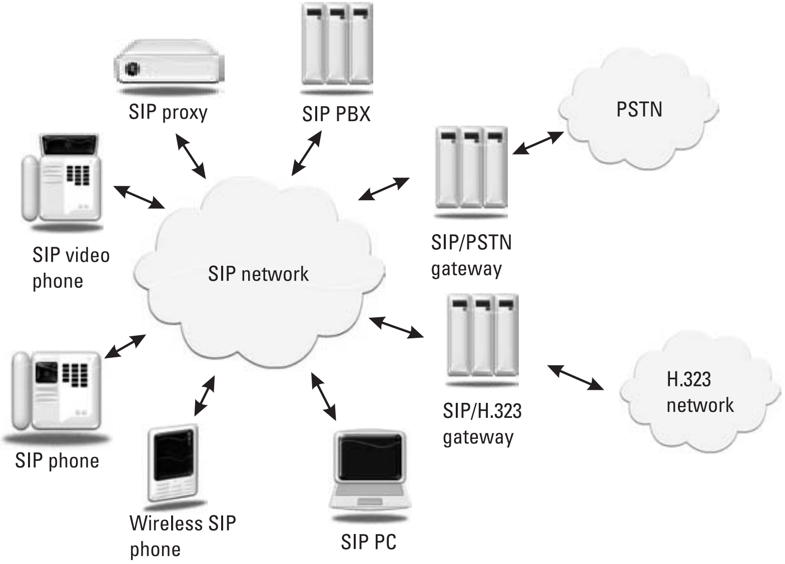


Figure 3.1 A SIP network with gateways.

cloud are SIP IP telephones, SIP-enabled PCs, and corporate SIP gateways with attached telephones. The clouds are connected by gateways. H.323 terminals and H.323-enabled PCs are attached to the H.323 network. The PSTN cloud connects to ordinary analog black telephones (so-called because of the original color of their shell), digital ISDN telephones, and corporate private branch exchanges (PBXs). PBXs connect to the PSTN using shared trunks and provide line interfaces for either analog or digital telephones.

Gateways are sometimes decomposed into a media gateway (MG) and a media gateway controller (MGC). An MGC is sometimes called a call agent because it manages call control protocols (signaling), while the MG manages the media connection. This decomposition is transparent to SIP; the protocols used to decompose a gateway are described in Section 11.3.

Another difference between a user agent and a gateway is the number of users supported. While a user agent typically supports a single user (although perhaps with multiple lines), a gateway can support hundreds or thousands of users. A PSTN gateway could support a large corporate customer, or an entire geographic area. As a result, a gateway does not `REGISTER` every user it supports in the same way that a user agent might. Instead, a non-SIP protocol can be used to inform proxies about gateways and assist in routing. One protocol that has been proposed for this is the Telephony Routing over IP (TRIP) protocol [7], which allows an interdomain routing table of gateways to be developed. Another protocol called Telephony Gateway Registration Protocol (TGREP) [8] has also been developed to allow a gateway to register with a proxy server within a domain.

3.5 SIP Servers

SIP servers are applications that accept SIP requests and respond to them. A SIP server should not be confused with a user agent server or the client-server nature of the protocol, which describe operation in terms of clients (originators of requests) and servers (originators of responses to requests). A SIP server is a different type of entity; the types of SIP servers discussed in this section are logical entities. Actual SIP server implementations may contain a number of server types, or may operate as a different type of server under different conditions. Because servers provide services and features to user agents, they must support both TCP and UDP for transport. Figure 3.2 shows the interaction of user agents, servers, and a location service. Note that the protocol used between a server and the location service or database is generally not SIP and is not discussed in this book.

3.5.1 Proxy Servers

A SIP proxy server receives a SIP request from a user agent or another proxy and acts on behalf of the user agent in forwarding or responding to the request. Just as a router forwards IP packets at the IP layer, a SIP proxy forwards SIP messages at the application layer. A proxy is not a B2BUA since it is only allowed to modify requests and responses according to strict rules set out in RFC 3261. These rules preserve the end-to-end transparency of the SIP signaling while still allowing a proxy server to perform valuable services and functions for user agents.

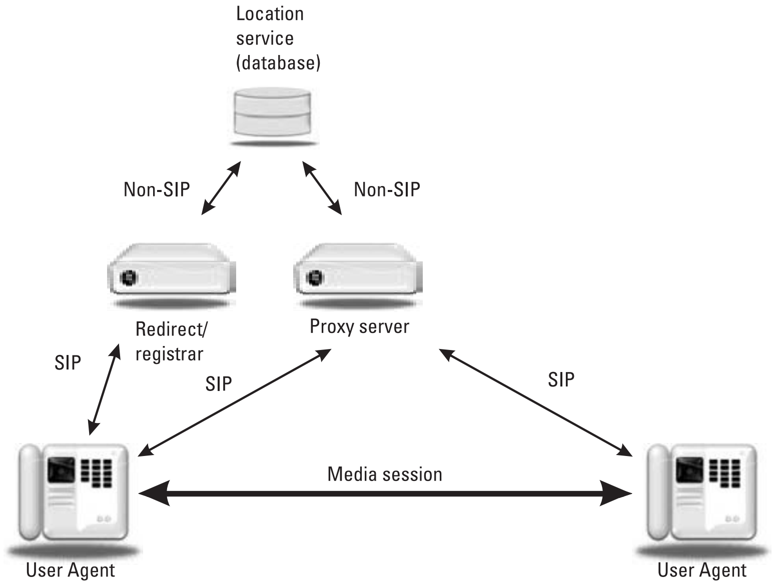


Figure 3.2 SIP user agent, server, and location service interaction.

A proxy server typically has access to a database or a location service to aid it in processing the request (determining the next hop). The interface between the proxy and the location service is not defined by the SIP protocol. A proxy can use any number of types of databases to aid in processing a request. Databases could contain SIP registrations, presence information, or any other type of information about where a user is located. The example in Figure 2.2 introduced a proxy server as a facilitator of SIP message exchange, providing user location services to the caller.

A proxy does not need to understand a SIP request in order to forward it—any unknown request type is assumed to use the non-`INVITE` transaction model. A proxy should not change the order of header fields or in general modify or delete header fields.

A proxy server is different from a user agent or gateway in three key ways:

1. A proxy server does not issue requests; it only responds to requests from a user agent. (`CANCEL` and `ACK` requests are an exception to this rule.)
2. A proxy server has no media capabilities.
3. A proxy server does not parse message bodies; it relies exclusively on SIP header fields.

Figure 3.3 shows a common network topology known as the SIP Trapezoid. In this topology, a pair of user agents in different domains establishes a session using a pair of proxy servers, one in each domain. The trapezoid refers to the shape formed by the signaling and media messages. In this configuration, each user agent is configured with a default outbound proxy server, to which it sends all requests. This proxy server typically will authenticate the user agent and may pull up a profile of the user and apply outbound routing services. In an interdomain exchange, DNS `SRV` queries will be used to locate a proxy server in the other domain. This proxy, sometimes called an inbound proxy, may apply inbound routing services on behalf of the called party. This proxy also has access to the current registration information for the user, and can route the request to the called party. In general, future SIP requests will be sent directly between the two user agents, unless one or both proxies insert a `Record-Route` header field.

A proxy server can be either stateless or stateful. A stateless proxy server processes each SIP request or response based solely on the message contents. Once the message has been parsed, processed, and forwarded or responded to, no information (such as dialog information) about the message is stored. A stateless proxy never retransmits a message, and does not use any SIP timers. Note that the stateless loop detection using `Via` header fields described in RFC 2543 has been deprecated (removed) in RFC 3261 in favor of the use of a mandatory `Max-Forwards` header field in all requests.

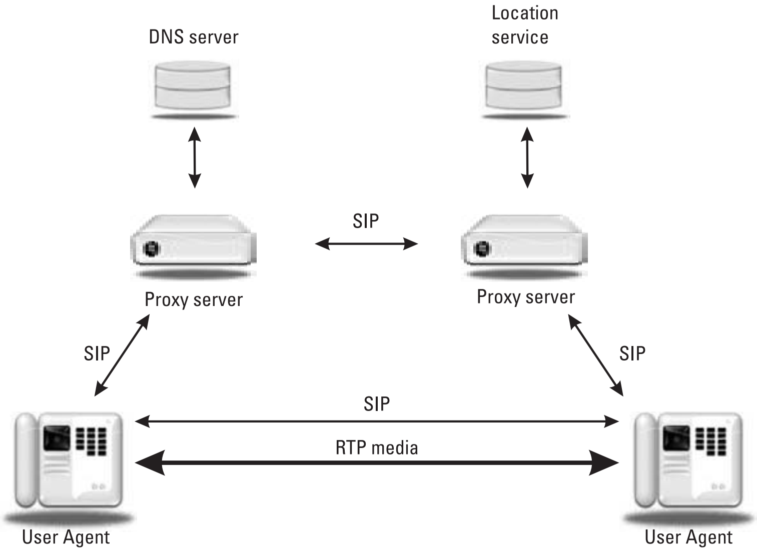


Figure 3.3 A SIP trapezoid.

A stateful proxy server keeps track of requests and responses received in the past, and uses that information in processing future requests and responses. For example, a stateful proxy server starts a timer when a request is forwarded. If no response to the request is received within the timer period, the proxy will retransmit the request, relieving the user agent of this task. Also, a stateful proxy can require user agent authentication, as described in Chapter 14.

The most common type of SIP proxy is a transaction stateful proxy. A transaction stateful proxy keeps state about a transaction but only for the duration of the pending request. For example, a transaction stateful proxy will keep state when it receives an `INVITE` request until it receives a `200 OK` or a final failure response (e.g., `404 Not Found`). After that, it would destroy the state information. This allows a proxy to perform useful search services but minimize the amount of state storage required.

One such example of a search service is a proxy server that receives an `INVITE` request, then forwards it to a number of locations at the same time, or forks the request. This *forking* proxy server keeps track of each of the outstanding requests and the response to each, as shown in Figure 3.4. This is useful if the location service or database lookup returns multiple possible locations for the called party that need to be tried.

In the example of Figure 3.4, the `INVITE` contains:

```
INVITE sip:support@chaos.info SIP/2.0
Via: SIP/2.0/UDP 45.2.32.1:5060 ;branch=z9hG4bK67865
Max-Forwards: 70
To: <sip:support@chaos.info>
```

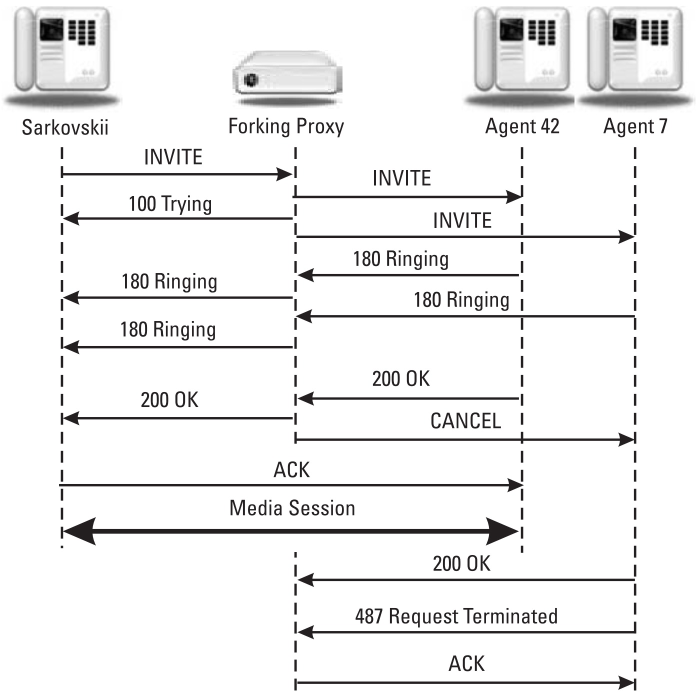


Figure 3.4 Forking proxy operation.

```
From: A. N. Sarkovskii <sip:sarkovskii@45.2.32.1>;tag=7643545
Call-ID: 0140092501
CSeq: 1 INVITE
Subject: Bifurcation Question
Contact: <sip:sarkovskii@45.2.32.1>
Content-Type: application/sdp
Content-Length: ...

(SDP not shown)
```

The `INVITE` is received by the `chaos.info` proxy server, which forks to two user agents. Each user agent begins alerting, sending two provisional responses back to Sarkovskii. They are:

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP 45.2.32.1:5060;branch=z9hG4bK67865
To: <sip:support@chaos.info>;tag=343214112
From: A. N. Sarkovskii <sip:sarkovskii@45.2.32.1>;tag=7643545
Call-ID: 0140092501
CSeq: 1 INVITE
Contact: <sip:agent42@67.42.2.1>
Content-Length: 0
```

and:

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP 45.2.32.1:5060;branch=z9hG4bK67865
To: <sip:support@chaos.info>;tag=a5ff34d9ee201
From: A. N. Sarkovskii <sip:sarkovskii@45.2.32.1>;tag=7643545
Call-ID: 0140092501
CSeq: 1 INVITE
Contact: <sip:agent7@67.42.2.32>
Content-Length: 0
```

The two responses are identical except for having different `To tags` and `Contact` URIs. Finally, one of the two UAs answers and sends a `200 OK` response:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 45.2.32.1:5060;branch=z9hG4bK67865
To: <sip:support@chaos.info>;tag=343214112
From: A. N. Sarkovskii <sip:sarkovskii@45.2.32.1>;tag=7643545
Call-ID: 0140092501
CSeq: 1 INVITE
Contact: <sip:agent42@67.42.2.1>
Content-Type: application/sdp
Content-Length: ...

(SDP not shown)
```

The forking proxy server sends a `CANCEL` to the second UA to stop that phone alerting. If both UAs had answered, the forking proxy would have forwarded both `200 OK` responses back to the caller who then would have had to choose which one, most likely by accepting one and sending a `BYE` to the other.

A stateful proxy usually sends a `100 Trying` response when it receives an `INVITE`. A stateless proxy never sends a `100 Trying` response. A `100 Trying` response received by a proxy is never forwarded—it is a single hop only response. A proxy handling a TCP request must be stateful, since a user agent will assume reliable transport and rely on the proxy for retransmissions on any UDP hops in the signaling path.[1]

The only limit to the number of proxies that can forward a message is controlled by the `Max-Forwards` header field, which is decremented by each proxy that touches the request. If the `Max-Forwards` count goes to zero, the proxy discards the message and sends a `483 Too Many Hops` response back to the originator.

The SIP session timer extension [9] limits the time period over which a stateful proxy must maintain state information without a refresh re-`INVITE`. In the initial `INVITE` request, a `Session-Expires` header field indicates a timer interval after which stateful proxies may discard state information about the session. User agents must tear down the call after the expiration of the timer. The caller can send re-`INVITE`s to refresh the timer, enabling a "keep alive" mechanism for

[1] TCP usually provides end-to-end reliability for applications. In SIP, however, TCP only provides single-hop reliability. End-to-end reliability is only achieved by a chain of TCP hops or TCP hops interleaved with UDP hops and stateful proxies.

SIP. This solves the problem of how long to store state information in cases where a BYE request is lost or misdirected, or in other security cases described in later sections. The details of this implementation are described in Section 6.2.34.

3.5.2 Redirect Servers

A redirect server was introduced in Figure 2.6 as a type of SIP server that responds to, but does not forward, requests. Like a proxy server, a redirect server uses a database or location service to lookup a user. The location information, however, is sent back to the caller in a redirection class response (3xx), which, after the ACK, concludes the transaction. Figure 3.5 shows a call flow that is very similar to the example in Figure 3.2, except the server uses redirection instead of proxying to assist Schroedinger in locating Heisenberg.

The INVITE from Figure 3.5 contains:

```
INVITE sip:werner.heisenberg@munich.de SIP/2.0
Via: SIP/2.0/UDP 100.101.102.103:5060 ;branch=z9hG4bK54532
Max-Forwards: 70
To: Heisenberg <sip:werner.heisenberg@munich.de>
From: E. Schroedinger <sip:schroed5244@wave.org>;tag=4313413
Call-ID: 734224912341371927319032
CSeq: 1 INVITE
```

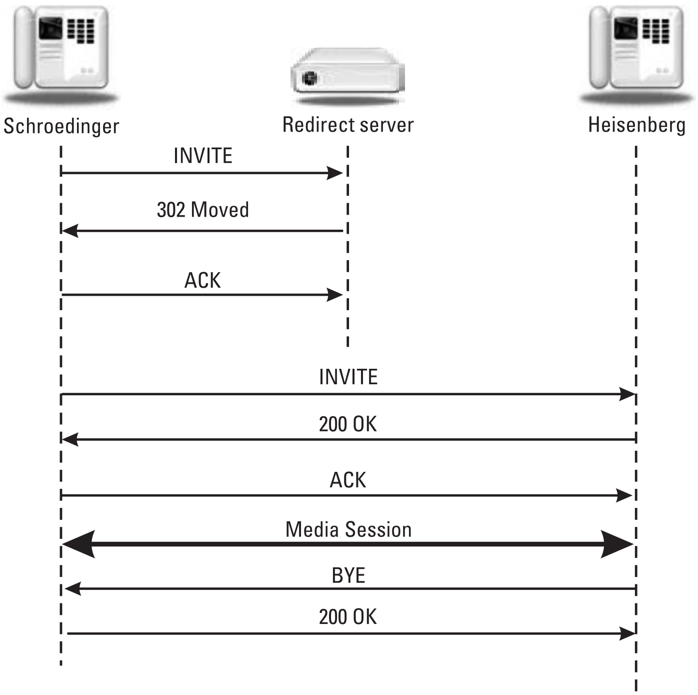


Figure 3.5 Example with redirect server.

```
Subject: Where are you exactly?
Contact: <sip:schroed5244@pc33.wave.org>
Content-Type: application/sdp
Content-Length: 150

v=0
o=schroed5244 2890844526 2890844526 IN IP4 100.101.102.103
s=-
t=0 0
c=IN IP4 100.101.102.103
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

The redirection response to the INVITE is sent by the redirect server:

```
SIP/2.0 302 Moved Temporarily
Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bK54532
To: Heisenberg <sip:werner.heisenberg@munich.de>;tag=052500
From: E. Schroedinger <sip:schroed5244@wave.org>;tag=4313413
Call-ID: 734224912341371927319032
CSeq: 1 INVITE
Contact: sip:werner.heisenberg@200.201.202.203
Content-Length: 0
```

Schroedinger acknowledges the response:

```
ACK sip:werner.heisenberg@munich.de SIP/2.0
Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bK54532
Max-Forwards: 70
To: Heisenberg <sip:werner.heisenberg@munich.de>;tag=052500
From: E. Schroedinger <sip:schroed5244@wave.org>;tag=4313413
Call-ID: 734224912341371927319032
CSeq: 1 ACK
Content-Length: 0
```

Notice that the ACK request reuses the same branch ID as the INVITE and the 302 response. This is because an ACK to a non-2xx final response is considered to be part of the same transaction as the INVITE. Only an ACK sent in response to a 200 OK is considered a separate transaction with a unique branch ID. Also, an ACK to a non-2xx final response is a hop-by-hop response, not an end-to-end response as discussed in Section 3.6.

This exchange completes this call attempt, so a new INVITE is generated with a new Call-ID and sent directly to the location obtained from the Contact header field in the 302 response from the redirect server.

```
INVITE sip:werner.heisenberg@200.201.202.203 SIP/2.0
Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bK92313
Max-Forwards: 70
To: Heisenberg <sip:werner.heisenberg@munich.de>
From: E. Schroedinger <sip:schroed5244@wave.org>;tag=13473
Call-ID: 54-67-45-23-13
CSeq: 1 INVITE
Subject: Where are you exactly?
```

```
Contact: <sip:schroed5244@pc33.wave.org>
Content-Type: application/sdp
Content-Length: 150

v=0
o=schroed5244 2890844526 2890844526 IN IP4
00.101.102.103
s=-
t=0 0
c=IN IP4 100.101.102.103
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

The call then proceeds in the same way as Figure 3.2, with the messages being identical. Note that in Figure 3.5, a `180 Ringing` response is not sent; instead, the `200 OK` response is sent right away. Since `1xx` informational responses are optional, this is a perfectly valid response by the UAS if Heisenberg responded to the alerting immediately and accepted the call. In the PSTN, this scenario is called fast answer.

3.5.3 Registrar Servers

A SIP registrar server was introduced in the example of Figure 3.3. A registrar server, also known as a registration server, accepts SIP `REGISTER` requests; all other requests receive a `501 Not Implemented` response. The contact information from the request is then made available to other SIP servers within the same administrative domain, such as proxies and redirect servers. In a registration request, the `To` header field contains the name of the resource being registered, and the `Contact` header fields contain the contact or device URIs. The registration server creates a temporary binding between the address of record (AOR) URI in the `To` and the device URI in the `Contact` header field.

Registration servers usually require the registering user agent to be authenticated, using the means described in Chapter 14, so that incoming calls cannot be hijacked by an unauthorized user. This could be accomplished by an unauthorized user registering someone else's SIP URI to point to their own UA. Incoming calls to that URI would then ring the wrong UA. Depending on the header fields present, a `REGISTER` request can be used by a user agent to retrieve a list of current registrations, clear all registrations, or add a registration URI to the list. These types of requests are described in Section 4.1.2.

There are a number of ways in which a proxy may know to fork a request to a set of UAs. One way is through manual configuration, such as entering the information in a Web page or database. Another way is to have multiple registrations for the same AOR. If multiple UAs register against the same AOR, the proxy can fork an incoming request to all of them. The priority of multiple registrations is governed by the `q`-value included in the `Contact` header field. For contacts of the same priority, a proxy can fork the request to all of them at the same time. For contacts with different priorities, a proxy can do sequential forking, sending the request in the order specified by the q-values.

For full registration security, TLS must be used as HTTP digest does not provide the needed integrity protection. Otherwise, an attacker can modify the Contact URI in an authenticated REGISTER to point to another UA.

3.6 Uniform Resource Indicators

SIP uses a number of Uniform Resource Identifiers. Common URIs are shown in Table 3.2.

SIP URIs will be discussed in Section 4.2. SIPS will be covered in Chapter 14. Telephony URI is covered in Section 4.2.2. Presence and IM URIs are covered in Chapter 8, along with the XMPP URI. H.323 URIs are covered in Section 11.4. Web URIs are defined in [10].

SIP uses Uniform Resource Indicators or URIs for most addresses. URIs and URLs were introduced in Section 1.4. For SIP, the URI scheme is either sip for a normal SIP URI or sips for a Secure SIP URI. Secure SIP means that a SIP message sent using this URI will be protected using TLS across each hop. SIP URIs must contain either a host name or an IP address. They usually contain a user part, although they do not have to. For example, a URI for a proxy server typically will not have a user part. URIs also may contain parameters. SIP URI parameters are listed in Table 3.3. In this table, URI means any valid URI while URN means any valid URN.

The following is a list of some examples of SIP URIs.

```
sip:fred@flintstone.org
sip:vilma@flintstone.org;transport=tcp
sip:the%20great%one@whalers.org
sip:7325551212@gw.gateway.com
sip:192.0.3.4:44352
sip:proxy34.sipstation.com
```

Table 3.2
Common URIs Used with SIP

URI Scheme	Use	Specification
sip	SIP	RFC 3261
sips	Secure SIP	RFC 3261
tel	Telephony	RFC 3966
pres	Presence	RFC 3861
im	Instant messaging	RFC 3861
xmpp	XMPP (Jabber)	RFC 4622
h323	H.323	RFC 3508
http	Web	RFC 2616

Table 3.3
SIP URI Parameters

Parameter	Specification	Example	Common Usage
`cause`	RFC 4458	`cause=486`	Voicemail
`comp`	RFC 3486	`comp=sigcomp`	Sigcomp compression used
`content-type`	RFC 4240	`content-type=audio`	Media server control
`delay`	RFC 4240	`delay=10`	Media server control
`duration`	RFC 4240	`duration=60`	Media server control
`gr`	[14]	`gr`	Globally routable UA URI
`local`	RFC 4240	`local=en`	Media server control
`lr`	RFC 3261	`lr`	Loose route parameter used in `Route` and `Record-Route`
`maddr`	RFC 3261	`maddr=1.2.3.4`	Multicast address
`method`	RFC 3261	`method=INVITE`	Used to escape a method into a URI
`ob`	[15]	`ob`	SIP outbound
`param[n]`	RFC 4240	`param1="today"`	Media server control [11]
`play`	RFC 4240	`play=URI`	Media server control [11]
`repeat`	RFC 4240	`repeat=forever`	Media server control [11]
`sigcomp-id`	RFC 5049	`sigcomp-id=URN`	Sigcomp ID [12]
`target`	RFC 4240	`target=IRO`	Media server control [11]
`target`	RFC 4458	`target=URI`	Voicemail [13]
`ttl`	RFC 3261	`ttl=224`	Time to live for a multicast address
`transport`	RFC 3261	`transport=tcp`	Transport protocol
`user`	RFC 3261	`user=phone`	Telephone digits in user part
`voicexml`	RFC 4240	`voicexml=URI`	Media server control [11]

```
sip:r3.example.com;lr
sip:+43321232;user=phone@sp.serviceprovider.org
```

SIP URIs can also be used to encoded telephone numbers. Sometimes, this includes the `user=phone` parameter.

3.7 Acknowledgment of Messages

Most SIP requests are end-to-end messages between user agents. Proxies between the two user agents simply forward the messages they receive and rely on the user agents to generate acknowledgments or responses.

There are some exceptions to this general rule. The `CANCEL` method (used to terminate pending calls or searches and discussed in detail in Section 4.1.5) is a hop-by-hop request. A proxy receiving a `CANCEL` immediately sends a `200 OK` response back to the sender and generates a new `CANCEL`, which is then forwarded in the next hop to the same set of destinations as the original request. (The order of sending the 200 OK and forwarding the CANCEL is not important.) This is shown in Figure 3.4.

Other exceptions to this rule include 3xx, 4xx, 5xx, and 6xx responses to an INVITE request. While an ACK to a 2xx response is generated by the end point, a 3xx, 4xx, 5xx, or 6xx response is acknowledged on a hop-by-hop basis. A proxy server receiving one of these responses immediately generates an ACK back to the sender and forwards the response to the next hop. This type of hop-by-hop acknowledgment is shown in Figure 4.2.

ACK messages are only sent to acknowledge responses to INVITE requests. For responses to all other request types, there is no acknowledgment. A lost response is detected by the UAS when the request is retransmitted.

3.8 Reliability

SIP has reliability mechanisms defined, which allow the use of unreliable transport layer protocols such as UDP. When SIP uses TCP, these mechanisms are not used, since it is assumed that TCP will retransmit the message if it is lost and inform the client if the server is unreachable.

For SIP transport using UDP, there is always the possibility of messages being lost or even received out of sequence, because UDP guarantees only that the datagram is error free. A UAS validates and parses a SIP request to make sure that the UAC has not errored by creating a request missing required header fields or other syntax violations. Reliability mechanisms in SIP include:

- Retransmission timers;
- Increasing command sequence CSeq numbers;
- Positive acknowledgments.

How SIP handles retransmissions depends on the method. One retransmission scheme is defined for INVITEs, known as INVITE transactions, and another is defined for all other methods, known as a non-INVITE transaction.

For non-INVITE transactions, a SIP timer, T1, is started by a UAC or a stateful proxy server when a new request is generated or sent. If no response to the request (as identified by a response containing the identical local tag, remote tag, Call-ID, and CSeq) is received when T1 expires, the request is resent. After a request is retransmitted, the next timer period is doubled until T2 is reached. If a provisional (informational class 1xx) response is received, the UAC or stateful proxy server immediately switches to timer T2. After that, the remaining retransmissions occur at T2 intervals. This capped exponential backoff process is continued until a 64*T1, after which the request is declared dead. A stateful proxy server that receives a retransmission of a request discards the retransmission and continues its retransmission schedule based on its own timers. Typically, it will resend the last provisional response. This retransmission scheme for non-INVITE is shown in Figure 3.6 for a REFER request.

For an INVITE transaction, the retransmission scheme is slightly different. INVITEs are retransmitted starting at T1, and then the timer is doubled after each retransmission. The INVITE is retransmitted until 64*T1 after which the request is declared dead. After a provisional (1xx) response is received, the INVITE is never retransmitted. This retransmission scheme is shown in Figure 3.7. A proxy may discard the transaction state after 3 minutes. A stateful proxy must store a forwarded request or generated response message for 32 seconds. Suggested default values for T1 and T2 are 500 ms and 4 seconds, respectively. Timer T1 is supposed to be an estimate of the roundtrip time (RTT) in the network. Longer values are allowed, but not shorter ones, because this will generate more message retransmissions. See Table 4 in RFC 3261 [1] for a summary of SIP timers.

Note that gaps in CSeq number do not always indicate a lost message. In the authentication examples, not every request (and hence CSeq) generated by

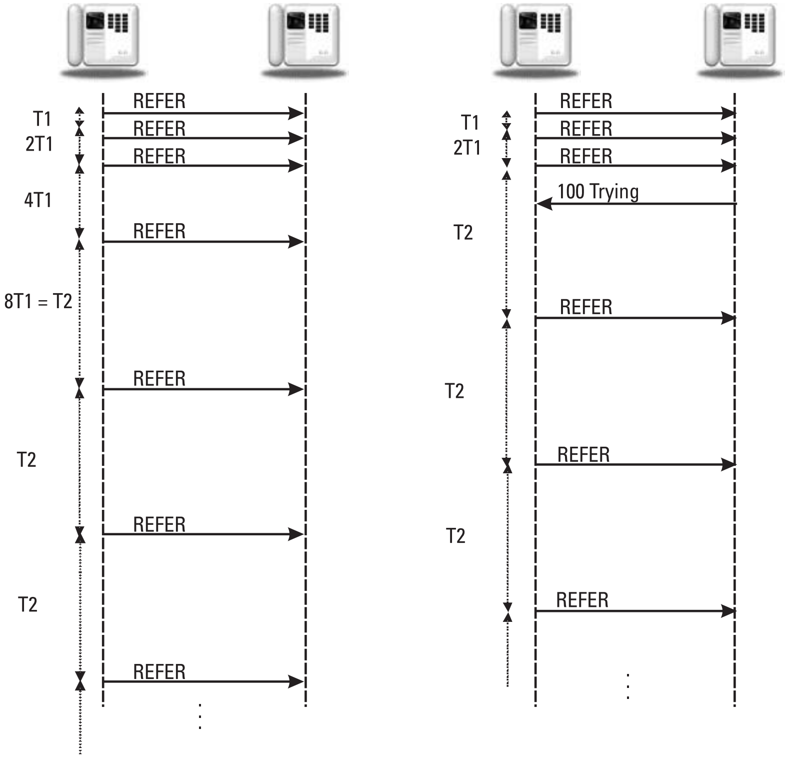


Figure 3.6 A SIP reliability example of a non-INVITE transaction.

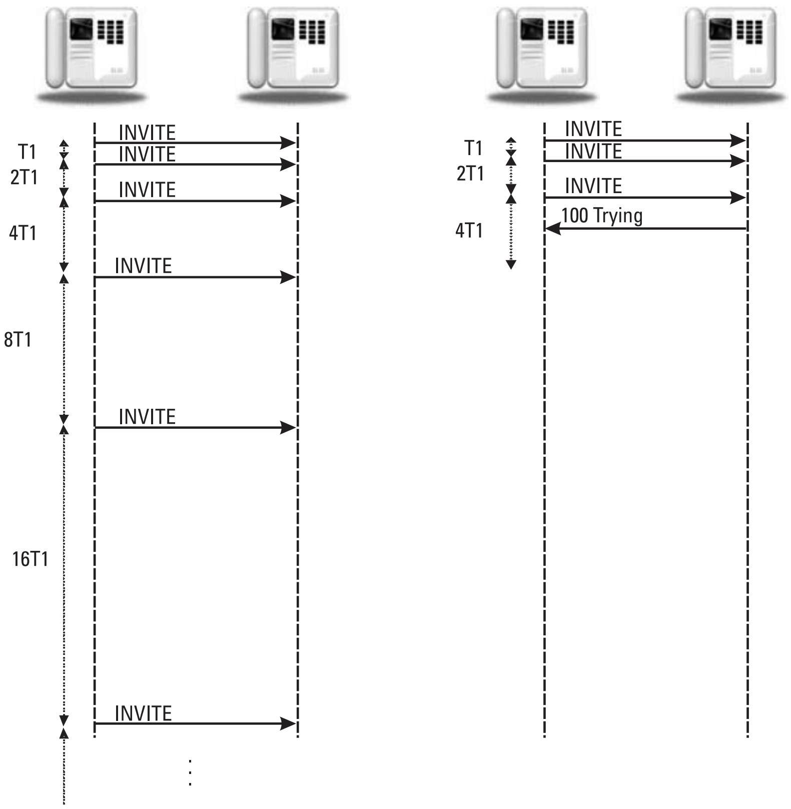


Figure 3.7 A SIP reliability example of an `INVITE` transaction.

the UAC will reach the UAS if authentication challenges occur by proxies in the path.

3.9 Multicast Support

SIP support for UDP multicast has been mentioned in previous sections. There are two main uses for multicast in SIP.

SIP registration can be done using multicast, by sending the `REGISTER` message to the well-known “All SIP Servers” URI `sip:sip.mcast.net` at IP address `224.0.1.75` for IPv4. The `ttl` parameter is usually set to 1 to indicate that only a single hop should be used.

RFC 2543 defined sending other SIP messages, including `INVITE`, over multicast. However, this was not included in RFC 3261 and is no longer considered standard SIP.

The use of a multicast address is indicated by the `maddr` parameter in a URI or in a SIP message using the `maddr` parameter in the `Via` header field.

3.10 Conclusion

This chapter introduced SIP clients and servers, discussing user agents, gateways, proxies, redirect servers, and registrars. SIP URIs, reliability, and retransmissions were also covered.

3.11 Questions

Q3.1 Fill in the missing messages in the call flow in Figure 3.8 with two UAs and one proxy.

Q3.2 A UA sends an `OPTIONS` to another UA, which does not respond. Assume T1 = 500 ms and T2 = 4 seconds. Show the timing of the retransmissions relative to $t = 0$ when the first `OPTIONS` is sent. How many messages are sent all together?

Q3.3 Fill in the `CSeq` header fields (number *and* method) for each of the messages in Figure 3.9.

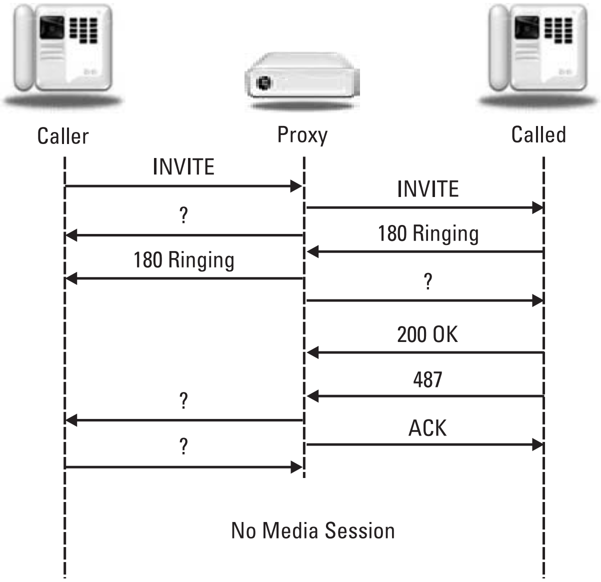


Figure 3.8 Call flow for Question Q3.1.

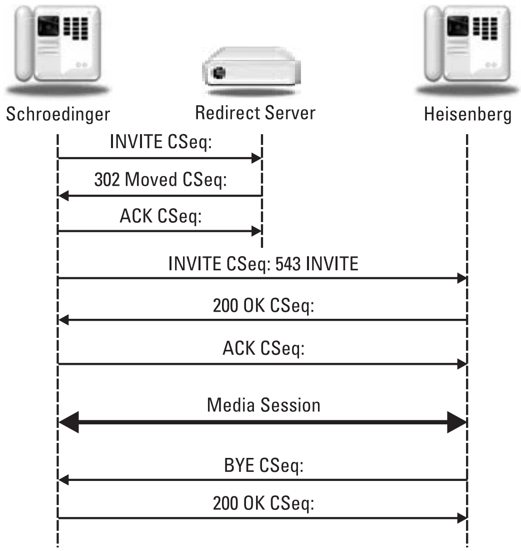


Figure 3.9 Call flow for Question Q3.3.

Q3.4 Add the missing SIP messages and responses for the call flow in Figure 3.10. Assume the proxy does not `Record-Route` (Hint: there are 6 missing messages that will result in just a single media session between Alice and Bob.)

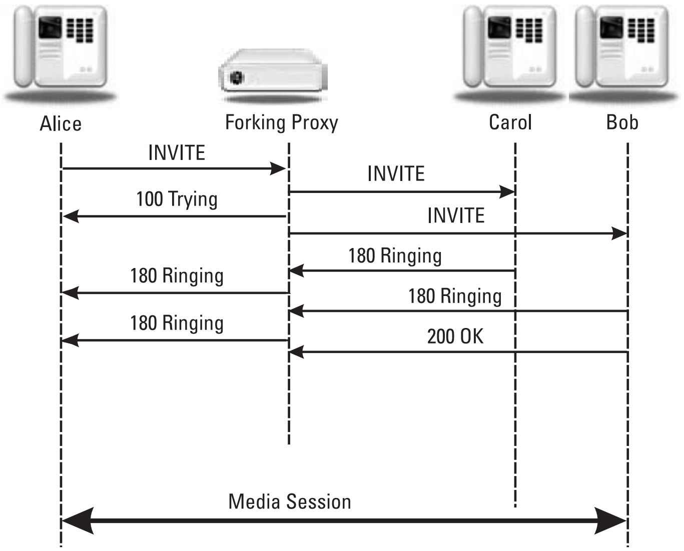


Figure 3.10 Call flow for Question Q3.4.

Q3.5 Fill in the time intervals for the retransmission example in Figure 3.11.

Q3.6 What are two ways that a proxy knows to fork a request?

Q3.7 When does a proxy generate an ACK to a response, and when does it just forward the response without generating an ACK?

Q3.8 What is the difference between a redirect server and a proxy server?

Q3.9 What is the purpose of a SIP registrar server?

Q3.10 A UA sends a REGISTER. After 2.3 seconds, a 100 Trying response is received. After another 0.7 second, a 200 OK response is received. In total, how many times was the REGISTER request sent?

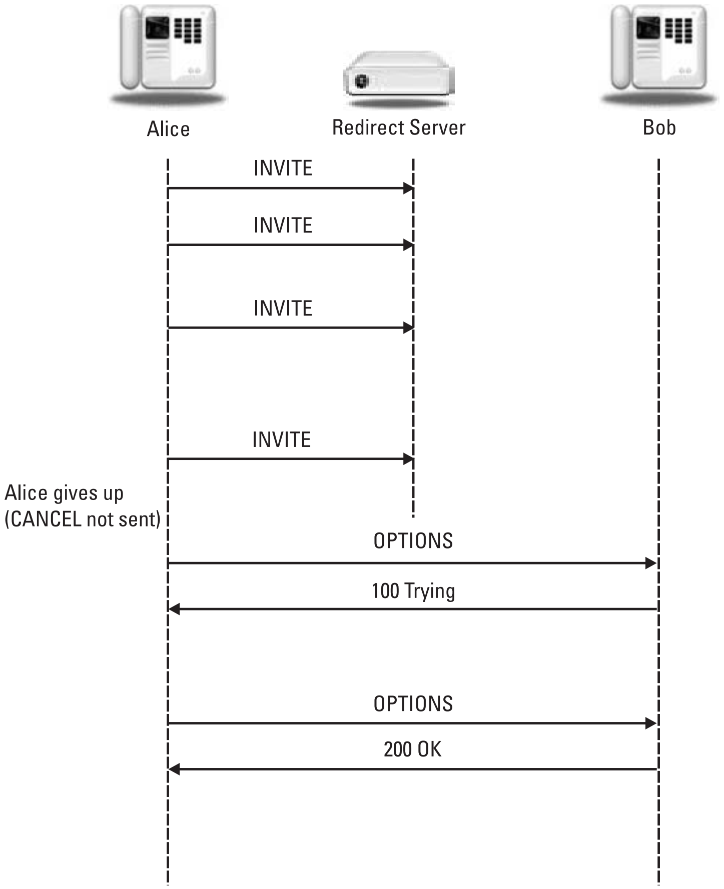


Figure 3.11 Call flow for Question Q3.6.

References

[1] Rosenberg, J., H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261, June 2002.

[2] Rosenberg, J., "A Presence Event Package for the Session Initiation Protocol (SIP)," RFC 3856, August 2004.

[3] Roach, A., "Session Initiation Protocol (SIP)—Specific Event Notification," RFC 3265, June 2002.

[4] Niemi, A., "Session Initiation Protocol (SIP) Extension for Event State Publication," RFC 3903, October 2004.

[5] Hautakorpi, J., et al., "Requirements from SIP (Session Initiation Protocol) Session Border Control Deployments," draft-ietf-sipping-sbc-funcs-08 (work in progress), January 2009.

[6] Schulzrinne, H., and C. Agboh, "Session Initiation Protocol (SIP)-H.323 Interworking Requirements," RFC 4123, July 2005.

[7] Rosenberg, J., H. Salama, and M. Squire, "Telephony Routing over IP (TRIP)," RFC 3219, January 2002.

[8] Bangalore, M., et al., "A Telephony Gateway Registration Protocol (TGREP)," RFC 5140, March 2008.

[9] Donovan, S., and J. Rosenberg, "Session Timers in the Session Initiation Protocol (SIP)," RFC 4028, April 2005.

[10] Fielding, R., et al., "Hypertext Transfer Protocol—HTTP/1.1," RFC 2616, June 1999.

[11] Burger, E., J. Van Dyke, and A. Spitzer, "Basic Network Media Services with SIP," RFC 4240, December 2005.

[12] Bormann, C., et al., "Applying Signaling Compression (SigComp) to the Session Initiation Protocol (SIP)," RFC 5049, December 2007.

[13] Jennings, C., F. Audet, and J. Elwell, "Session Initiation Protocol (SIP) URIs for Applications such as Voicemail and Interactive Voice Response (IVR)," RFC 4458, April 2006.

[14] Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)," draft-ietf-sip-gruu-15 (work in progress), October 2007.

[15] Jennings, C., and R. Mahy, "Managing Client Initiated Connections in the Session Initiation Protocol (SIP)," draft-ietf-sip-outbound-20 (work in progress), June 2009.

4

SIP Request Messages

This chapter covers the types of SIP requests called *methods*. Six are described in the SIP specification document RFC 3261 [1]. Eight more methods are described in separate RFC documents. After discussing the methods, this chapter concludes with a discussion of SIP URLs and URIs, `tags`, and message bodies.

4.1 Methods

SIP requests or methods are considered "verbs" in the protocol, since they request a specific action to be taken by another user agent or server. The `INVITE`, `REGISTER`, `BYE`, `ACK`, `CANCEL`, and `OPTIONS` methods are the original six methods in SIP. The `REFER`, `SUBSCRIBE`, `NOTIFY`, `PUBLISH`, `MESSAGE`, `UPDATE`, `INFO`, and `PRACK` methods are described in separate RFCs.

A user agent (UA) receiving a method it does not support replies with a `501 Not Implemented` response. Method names are case sensitive and conventionally use all uppercase for visual clarity to distinguish them from header fields, which use both upper and lower case. Note that a proxy does not need to understand a request method in order to forward the request. A proxy treats an unknown method as if it were an `OPTIONS`; that is, it forwards the request to the destination if it can. This allows new features and methods useful for user agents to be introduced without requiring support from proxies that may be in the middle. UAs should indicate which methods they support in an `Allow` header field in requests and responses.

4.1.1 INVITE

The `INVITE` method is used to establish media sessions between user agents. In telephony, it is similar to a `Setup` message in ISDN or an initial address message (IAM) in ISUP. (PSTN protocols are briefly introduced in Section 11.1.)

Responses to INVITEs are always acknowledged with the ACK method described in Section 4.1.4. Examples of the use of the INVITE method are described in Chapter 2.

An INVITE usually has a message body containing the media information of the caller. The message body can also contain other session information, such as a resource list. If an INVITE does not contain media information, the ACK contains the media information of the UAC. An example of this call flow is shown in Figure 4.1. If the media information contained in the ACK is not acceptable, then the called party must send a BYE to cancel the session—a CANCEL cannot be sent because the session is already established. A media session is considered established when the INVITE, 200 OK, and ACK messages have been exchanged between the UAC and the UAS. A successful INVITE request establishes a dialog between the two user agents, which continues until a BYE is sent by either party to end the session, as described in Section 4.1.3.

A UAC that originates an INVITE to establish a dialog creates a globally unique Call-ID that is used for the duration of the call. A CSeq count is initialized (which need not be set to 1, but must be an integer) and incremented for each new request for the same Call-ID. The To and From headers are populated with the remote and local addresses. A From tag is included in the INVITE, and the UAS includes a To tag in any responses, as described in Section 4.3. A To tag in a 200 OK response to an INVITE is used in the To header field of the ACK and all future requests within the dialog. The combination of the To tag, From tag, and Call-ID is the unique identifier for the dialog.

An INVITE sent for an existing dialog references the same Call-ID as the original INVITE and contains the same To and From tags. Sometimes called a re-INVITE, the request is used to change the session characteristics or refresh the state of the dialog. The CSeq command sequence number is incremented so

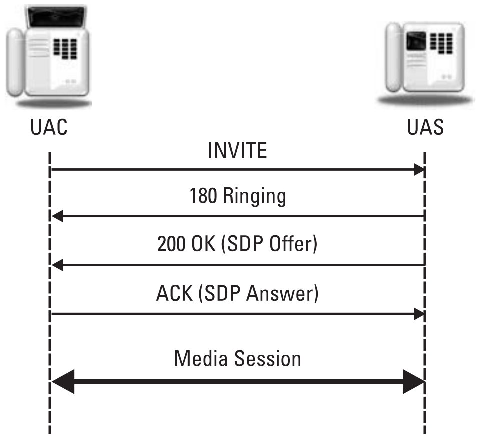


Figure 4.1 INVITE without an SDP offer.

that a UAS can distinguish the re-INVITE from a retransmission of the original INVITE.

If a re-INVITE is refused or fails in any way, the session continues as if the INVITE had never been sent. A re-INVITE must not be sent by a UAC until a final response to the initial INVITE has been received—instead, an UPDATE request can be sent, as described in Section 4.1.14. There is an additional case where two UAs simultaneously send re-INVITEs to each other. This is handled in the same way with a Retry-After header. This condition is called *glare* in telephony (see Figure 5.3) and occurs when both ends of a trunk group seize the same trunk at the same time.

An Expires header in an INVITE indicates to the UAS how long the call request is valid. For example, the UAS could leave an unanswered INVITE request displayed on a screen for the duration specified in the Expires header. Once a session is established, the Expires header has no meaning—the expiration of the time does not terminate the media session. Instead, a Session-Expires header can be used to place a time limit on an established session without a re-INVITE or UPDATE refresh.

An example INVITE request with a SDP message body is shown here:

```
INVITE sip:411@salzburg.at;user=phone SIP/2.0
Via: SIP/2.0/UDP salzburg.edu.at:5060;branch=z9hG4bK1d32hr4
Max-Forwards:70
To: <sip:411@salzburg.at;user=phone>
From: Christian Doppler <sip:c.doppler@salzburg.edu.at>
   ;tag=817234
Call-ID: 12-45-A5-46-F5-43-32-F3-C2
CSeq: 1 INVITE
Subject: Train Timetables
Allow: INVITE, ACK, CANCEL, BYE, OPTIONS, REFER, SUBSCRIBE,
 NOTIFY
Contact: sip:c.doppler@salzburg.edu.at
Content-Type: application/sdp
Content-Length: 195

v=0
o=doppler 2890842326 2890844532 IN IP4 salzburg.edu.at
s=-
c=IN IP4 50.61.72.83
t=0 0
m=audio 49172 RTP/AVP 97 98 0
a=rtpmap:97 iLBC/8000
a=rtpmap:98 SPEEX/8000
a=rtpmap:0 PCMU/8000
```

In addition to the required headers, this request contains the optional Subject and Allow header fields. Note that this Request-URI contains a phone number. Phone number support in SIP URIs is described in Section 4.2.

The mandatory and header fields in an INVITE request are shown in Table 4.1.

Table 4.1
Mandatory Header Fields in an INVITE

Via
To
From
Call-ID
CSeq
Contact
Max-Forwards

4.1.2 REGISTER

The REGISTER method is used by a UA to notify a SIP network of its current Contact URI (IP address) and the URI that should have requests routed to this Contact. As mentioned in Section 3.5.3, SIP registration bears some similarity to cell phone registration on initialization. Registration is not required to enable a user agent to use a proxy server for outgoing calls. It is necessary, however, for a user agent to register to receive incoming calls from proxies that serve that domain, unless some non-SIP mechanism is used by the location service to populate the SIP URIs and Contacts of end points. A REGISTER request may contain a message body, although its use is not defined in the standard. Depending on the use of the Contact and Expires headers in the REGISTER request, the registrar server will take different actions. Examples of this are shown in Table 4.2. If no expires parameter or Expires header is present, a SIP URI will expire in 1 hour. The presence of an Expires header sets the expiration of Contacts with no expires parameter. If an expires parameter is present, it sets the expiration time for that Contact only. Non-SIP URIs have no default expiration time.

The CSeq is incremented for a REGISTER request. The use of the Request-URI, To, From, and Call-ID headers in a REGISTER request is slightly different

Table 4.2
Example Registration Contact URIs

Request Headers	Registrar Action
Contact: * Expires: 0	Cancel all registrations.
Contact: sip:galvani@bologna.edu.it ;expires=1800	Add Contact to current registrations; registration expires in 30 minutes.
Contact: sip:sgalvani@192.34.3.1 Expires: 1800	Add Contact to current registrations; registration expires in 30 minutes.
Contact: mailto:galvani@bologna.edu.it ;q=0.1	Add email URL, which doesn't expire.
No Contact header present	Return all current registrations in response.

than for other requests. The Request-URI contains only the domain of the registrar server with no user portion. The REGISTER request may be forwarded or proxied until it reaches the authoritative registrar server for the specified domain. The To header contains the SIP URI of the AOR (address of record) of the user agent that is being registered. The From contains the SIP URI of the sender of the request, usually the same as the To header. It is recommended that the same Call-ID be used for all registrations by a user agent.

A user agent sending a REGISTER request may receive a 3xx redirection or 4xx failure response containing a Contact header of the location to which registrations should be sent.

A third-party registration occurs when the party sending the registration request is not the party that is being registered. In this case, the From header will contain the URI of the party submitting the registration on behalf of the party identified in the To header. Chapter 2 contains an example of a first-party registration. An example third-party registration request for the user Euclid is shown here:

```
REGISTER sip:registrar.athens.gr SIP/2.0
Via: SIP/2.0/UDP 201.202.203.204:5060;branch=z9hG4bK313
Max-Forwards:70
To: sip:euclid@athens.gr
From: <sip:secretary@academy.athens.gr>;tag=543131
Call-ID: 48erl8132409wqer
CSeq: 1 REGISTER
Contact: sip:euclid@parthenon.athens.gr
Contact: mailto:euclid@geometry.org
Content-Length: 0
```

In some cases, the Contact URI provided by a UA in a registration may not be routable. For example, if the UA is behind a NAT, or if a firewall is configured to block incoming requests from arbitrary hosts. If this Contact URI is used outside a SIP dialog (for example, in sending a REFER or performing attended transfer), then call control operations might fail. An extension mechanism has been developed in which a UA can request from a registrar a so-called Globally Routable User Agent URI or GRUU [2]. This URI can be used in Contact header fields and other places the device wants to be directly reachable. A UA includes a Supported:gruu header field in a REGISTER request and a sip.instance feature tag, and if the registrar supports the mechanism, a GRUU will be returned in the 200 OK to the register in the pub-gruu and temp-gruu Contact header field parameters. The temp-gruu changes each time a registration is refreshed, while the pub-gruu is valid as long as the registration is refreshed. An example Contact header field containing a GRUU is shown here:

```
Contact: <sip:euclid@201.202.203.204>
 ;pub-gruu=″sip:euclid@athens.gr;gr=urn:uuid:00a0dc91e6bdf6″
 ;temp-gruu=″sip:k20flasdf2da@athens.gr;gr″
```

```
    ;+sip.instance="<urn:uuid:00a0dc91e6bdf6>"
    ;expires=1800
```

The mandatory headers in a REGISTER request are shown in Table 4.3.

4.1.3 BYE

The BYE method is used to terminate an established media session. In telephony, it is similar to a release message. A session is considered established if an INVITE has received a success class response (2xx) or an ACK has been sent. A BYE is sent only by UAs participating in the session, never by proxies or other third parties. It is an end-to-end method, so responses are only generated by the other UA. A UA responds with a 481 Dialog/Transaction Does Not Exist to a BYE for an unknown dialog.

A BYE cannot be used to cancel pending INVITEs because it will not be forked like an INVITE and may not reach the same set of UAs as the INVITE. An example BYE request looks like the following:

```
BYE sip:info@hypotenuse.org SIP/2.0
Via: SIP/2.0/TCP port443.hotmail.com:54212;branch=z9hG4bK312bc
Max-Forwards:70
To: <sip:info@hypotenuse.org>;tag=63124
From: <sip:pythag42@hotmail.com>;tag=9341123
Call-ID: 34283291273
CSeq: 47 BYE
Content-Length: 0
```

The mandatory headers in a BYE request are shown in Table 4.4.

4.1.4 ACK

The ACK method is used to acknowledge final responses to INVITE requests. Final responses to all other requests are never acknowledged. Final responses are defined as 2xx, 3xx, 4xx, 5xx, or 6xx class responses. The CSeq number is never incremented for an ACK, but the CSeq method is changed to ACK. This is so that a UAS can match the CSeq number of the ACK with the number of the corresponding INVITE.

Table 4.3
Mandatory Header Fields in a REGISTER

Via
To
From
Call-ID
CSeq
Max-Forwards

Table 4.4
Mandatory Header Fields in a BYE

Via
To
From
Call-ID
CSeq
Max-Forwards

An ACK may contain an application/sdp message body. This is permitted if the initial INVITE did not contain a SDP message body. If the INVITE contained an SDP offer message body, the ACK may not contain an SDP message body. The ACK may not be used to modify a media description that has already been sent in the initial INVITE; a re-INVITE or UPDATE must be used for this purpose. SDP in an ACK is used in some interworking scenarios with other protocols where the media characteristics may not be known when the initial INVITE is generated and sent.

For 2xx responses, the ACK is end-to-end, but for all other final responses it is done on a hop-by-hop basis when stateful proxies are involved. As a result, a proxy will generate an ACK for a 3xx, 4xx, 5xx, or 6xx response to an INVITE, as well as forwarding the response. The end-to-end nature of ACKs to 2xx responses allows a message body to be transported. An ACK generated in a hop-by-hop acknowledgment will contain just a single Via header with the address of the proxy server generating the ACK. The difference between hop-by-hop acknowledgments and response end-to-end acknowledgments is shown in the message fragments of Figure 4.2.

A hop-by-hop ACK reuses the same branch ID as the INVITE since it is considered part of the same transaction. An end-to-end ACK uses a different branch ID as it is considered a new transaction.

A stateful proxy receiving an ACK message must determine whether or not the ACK should be forwarded downstream to another proxy or user agent. That is, is the ACK a hop-by-hop ACK or an end-to-end ACK? This is done by comparing the branch ID for a match pending transaction branch IDs. If there is not an exact match, the ACK is proxied toward the UAS. Otherwise, the ACK is for this hop and is not forwarded by the proxy. Here is an example ACK containing SDP:

```
ACK sip:laplace@mathematica.org SIP/2.0
Via: SIP/2.0/TCP 128.5.2.1:5060;branch=z9hG4bK1834
Max-Forwards:70
To: Marquis de Laplace <sip:laplace@mathematica.org>
 ;tag=90210
From: Nathaniel Bowditch <sip:n.bowditch@salem.ma.us>
 ;tag=887865
Call-ID: 152-45-32-N-32-23-47-W
```

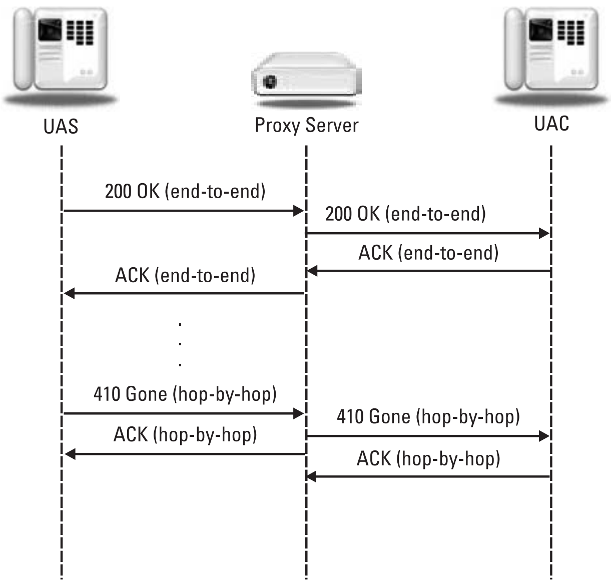


Figure 4.2 End-to-end versus hop-by-hop acknowledgments.

```
CSeq: 3 ACK
Content-Type: application/sdp
Content-Length: 172

v=0
o=bowditch 2590844326 2590944532 IN IP4 salem.ma.us
s=Bearing
c=IN IP4 salem.ma.us
t=0 0
m=audio 32852 RTP/AVP 96 0
a=rtpmap:96 SPEEX/8000
a=rtpmap:0 PCMU/8000
```

The mandatory and optional headers in an ACK message are shown in Table 4.5.

Table 4.5
Mandatory Header Fields in an ACK

Via
To
From
Call-ID
CSeq
Max-Forwards

4.1.5 CANCEL

The CANCEL method is used to terminate pending INVITEs or call attempts. It can be generated by either user agents or proxy servers provided that a 1xx response containing a tag has been received, but no final response has been received. A UA uses the method to cancel a pending call attempt it had initiated earlier. A forking proxy can use the method to cancel pending parallel branches after a successful response has been proxied back to the UAC. CANCEL is a hop-by-hop request and receives a response generated by the next stateful element. The difference between a hop-by-hop request and an end-to-end request is shown in Figure 4.3. The CSeq is not incremented for this method so that proxies and user agents can match the CSeq of the CANCEL with the CSeq of the pending INVITE to which it corresponds.

The branch ID for a CANCEL matches the INVITE that it is canceling. A CANCEL only has meaning for an INVITE since only an INVITE may take several seconds (or minutes) to complete. All other SIP requests complete immediately (that is, a UAS must immediately generate a final response). Consequently, the final result will always be generated before the CANCEL is received.

A proxy receiving a CANCEL forwards the CANCEL to the same set of locations with pending requests that the initial INVITE was sent. A proxy does not wait for responses to the forwarded CANCEL requests, but responds immediately. A UA confirms the cancellation with a 200 OK response to the CANCEL and replies to the INVITE with a 487 Request Terminated response.

If a final response has already been received, a UA will need to send a BYE to terminate the session. This is also the case in the race condition where a CANCEL

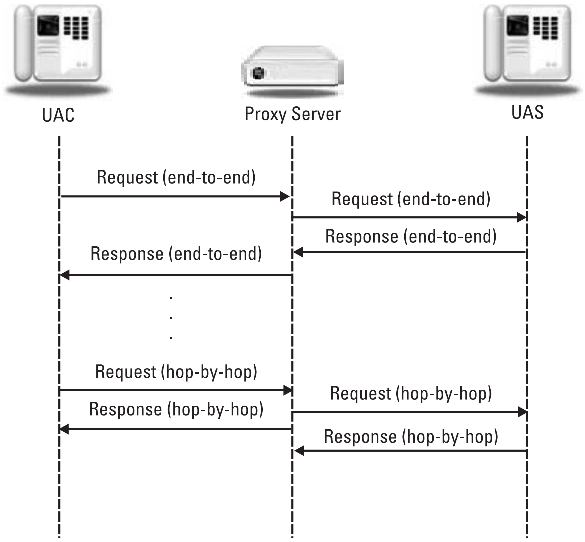


Figure 4.3 End-to-end versus hop-by-hop requests.

and a final response cross in the network, as shown in Figure 4.4. In this example, the CANCEL and 200 OK response messages cross between the proxy and the UAS. The proxy still replies to the CANCEL with a 200 OK, but then also forwards the 200 OK response to the INVITE. The 200 OK response to the CANCEL sent by the proxy only means that the CANCEL request was received and has been forwarded—the UAC must still be prepared to receive further final responses. No 487 response is sent in this scenario. The session is canceled by the UAC sending an ACK then a BYE in response to the 200 OK.

Since it is a hop-by-hop request, a CANCEL may not contain a message body. An example CANCEL request contains:

```
CANCEL sip:i.newton@cambridge.edu.gb SIP/2.0
Via: SIP/2.0/UDP 10.downing.gb:5060
 ;branch=z9hG4bK3134134
Max-Forwards:70
To: Isaac Newton <sip:i.newton@cambridge.edu.gb>
From: Rene Descartes <sip:visitor@10.downing.gb>;tag=034323
Call-ID: 23d8e0e4e2e505329299e288bbd4155a
CSeq: 32156 CANCEL
Content-Length: 0
```

The mandatory header fields in a CANCEL request are shown in Table 4.6.

4.1.6 OPTIONS

The OPTIONS method is used to query a user agent or server about its capabilities and discover its current availability. The response to the request lists the capabili-

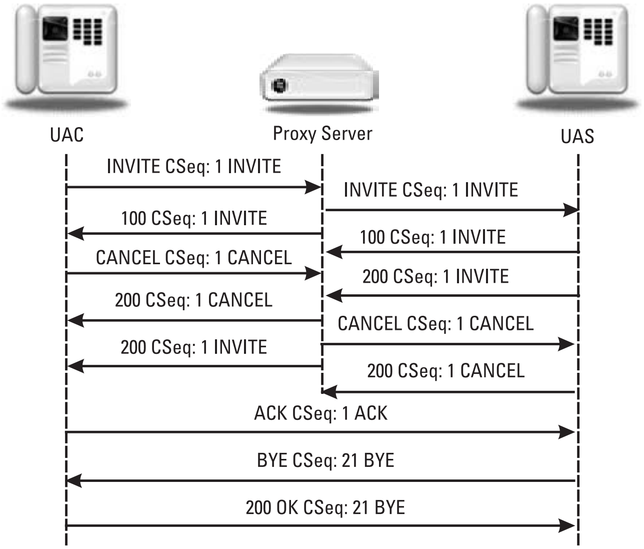


Figure 4.4 Race condition in call cancellation.

Table 4.6
Mandatory Header Fields in a CANCEL

Via
To
From
Call-ID
CSeq
Max-Forwards

ties of the user agent or server. A proxy never generates an OPTIONS request. A user agent or server responds to the request as it would to an INVITE (i.e., if it is not accepting calls, it would respond with a 4xx or 6xx response). A success class (2xx) response can contain Allow, Accept, Accept-Encoding, Accept-Language, and Supported headers indicating its capabilities. Feature tags (such as audio, video [3], and isfocus [4]) should be included with the Contact header field.

An OPTIONS request may not contain a message body. A proxy determines if an OPTIONS request is for itself by examining the Request-URI. If the Request-URI contains the address or host name of the proxy, the request is for the proxy. Otherwise, the OPTIONS is for another proxy or user agent and the request is forwarded. An example OPTIONS request and response contains:

```
OPTIONS sip:user@carrier.com SIP/2.0
Via: SIP/2.0/UDP cavendish.kings.cambridge.edu.uk
 ;branch=z9hG4bK1834
Max-Forwards:70
To: <sip:wiliamhopkins@cam.ac.uk>
From: J.C. Maxwell <sip:james.maxwell@kings.cambridge.edu.uk>
 ;tag=34
Call-ID: 747469e729acd305
CSeq: 29 OPTIONS
Content-Length: 0

SIP/2.0 200 OK
Via: SIP/2.0/UDP cavendish.kings.cambridge.edu.uk;tag=512A6
 ;branch=z9hG4bK0834 ;received=192.0.0.2
To: <sip:wiliamhopkins@cam.ac.uk>;tag=432
From: J.C. Maxwell <sip:james.maxwell@kings.cambridge.edu.uk>
 ;tag=34
Call-ID: 747469e729acd305
CSeq: 29 OPTIONS
Contact: <sip:william@tutors.cam.ac.uk>;audio;video
Allow: INVITE, OPTIONS, ACK, BYE, CANCEL, REFER
Supported: replaces, join
Accept-Language: en, de, fr
Content-Type: application/sdp
Content-Length: 170

v=0
o=jc 2590845378 2590945578 IN IP4 tutors.cam.ac.uk
s=-
c=IN IP4 tutors.cam.ac.uk
```

```
t=0 0
m=audio 32852 RTP/AVP 96 0
a=rtpmap:96 SPEEX/8000
a=rtpmap:0 PCMU/8000
m=video 82852 RTP/AVP 34
a=rtpmap:34 H263/90000
```

The mandatory headers in an OPTIONS request is listed in Table 4.7.

4.1.7 SUBSCRIBE

The SUBSCRIBE method [5] is used by a UA to establish a subscription for the purpose of receiving notifications (via the NOTIFY method) about a particular event. A successful subscription establishes a dialog between the UAC and the UAS. The subscription request contains an Expires header field, which indicates the desired duration of the existence of the subscription. After this time period passes, the subscription is automatically terminated. The subscription can be refreshed by sending another SUBSCRIBE within the dialog before the expiration time. A server accepting a subscription returns a 200 OK response also containing an Expires header field. The expiration timer can be the same as the request, or the server may shorten the interval, but it may not lengthen the interval. There is no UNSUBSCRIBE method used in SIP—instead a SUBSCRIBE with Expires:0 requests the termination of a subscription and hence the dialog. A terminated subscription (either due to timeout out or a termination request) will result in a final NOTIFY indicating that the subscription has been terminated (see Section 4.1.8 on NOTIFY). A 202 Accepted response to a SUBSCRIBE does not indicate whether the subscription has been authorized—it merely means it has been understood by the server.

The basic call flow is shown in Figure 4.5. The client sends a SUBSCRIBE, which is successful, and receives NOTIFYs as the requested events occur at the server. Before the expiration of the subscription time, the client re-SUBSCRIBEs to extend the subscription and hence receives more notifications.

Note that a client must be prepared to receive a NOTIFY before receiving a 200 OK response to the SUBSCRIBE. Also, due to forking, a client must be prepared to receive NOTIFYs from multiple servers (the NOTIFYs will have different To tags

Table 4.7
Mandatory Header Fields in an OPTIONS

Via
To
From
Call-ID
CSeq
Max-Forwards

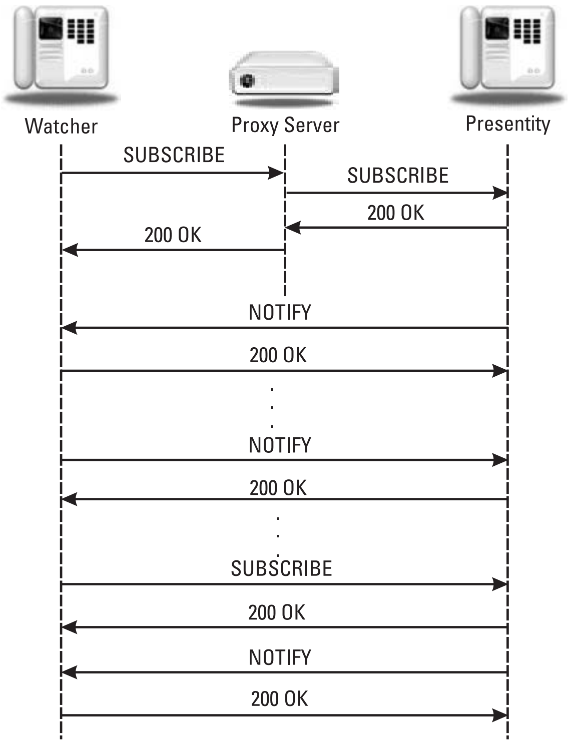


Figure 4.5 Example SUBSCRIBE and NOTIFY call flow.

and hence will establish separate dialogs), although only one 200 OK response to the SUBSCRIBE may be received.

An example SUBSCRIBE request is shown below:

```
SUBSCRIBE sip:ptolemy@rosettastone.org SIP/2.0
Via SIP/2.0/UDP proxy.elasticity.co.uk:5060
 ;branch=z9hG4bK348471123
Via SIP/2.0/UDP parlour.elasticity.co.uk:5060
 ;branch=z9hG4bKABDA ;received=192.0.3.4
Max-Forwards: 69
To: <sip:Ptolemy@rosettastone.org>
From: Thomas Young <sip:tyoung@elasticity.co.uk>;tag=1814
Call-ID: 452k59252058dkfj34924lk34
CSeq: 3412 SUBSCRIBE
Allow-Events: dialog
Contact: <sip:tyoung@parlour.elasticity.co.uk>
Event: dialog
Content-Length: 0
```

The type of event subscription is indicated by the required Event header field in the SUBSCRIBE request. Each application of the SIP Events framework [5] defines a package with a unique event tag. Each package defines the following:

- Default subscription expiration interval;
- Expected SUBSCRIBE message bodies;
- What events cause a NOTIFY to be sent, and what message body is expected in the NOTIFY;
- Whether the NOTIFY contains complete state or increments (deltas);
- Maximum notification rate.

A protocol called PSTN and Internet Interworking (PINT) [6] defines methods SUBSCRIBE, NOTIFY, and UNSUBSCRIBE, which have a similar semantic to SIP. A server can distinguish a PINT SUBSCRIBE request from a SIP SUBSCRIBE by the absence of an Event header field in the PINT request. A server should indicate which event packages it supports by listing them in an Allow-Events header field.

If a SUBSCRIBE refresh is sent within a dialog but receives a 481 Dialog Does Not Exist response, this means that the server has already terminated the subscription. The client should consider the dialog and subscription terminated and send a SUBSCRIBE to establish a new dialog and subscription.

An event template package is a special type that can be applied to any other package. The application of a template package to a package is shown by separating the package and template package names with a "." as in presence.winfo, [7] which is the application of the watcher info template package to the presence package. Table 4.8 lists the current set of SIP event and template packages.

Table 4.9 lists the mandatory header fields in a SUBSCRIBE request. Packages are standardized in the IETF based on the requirements in [5].

Table 4.8
SIP Event Packages

Package Name	Use	Specification
conference	Conferencing [7]	RFC 4579
consent-pending-additions	Consent framework [8]	RFC 5362
dialog	SIP dialog information [9]	RFC 4235
kpml	Key Press Markup Language [10]	RFC 4730
message-summary	Voicemail [11]	RFC 3842
presence	Presence [12]	RFC 3845
reg	Registration [13]	RFC 3680
refer	Refer [14]	RFC 3515
.winfo	Watcher template [15]	RFC 3857
vq-rtcpx	RTCP VoIP summary [16]	Draft

Table 4.9
Mandatory Header Fields in a SUBSCRIBE

Via
To
From
Call-ID
CSeq
Max-Forwards
Contact
Event
Allow-Events

4.1.8 NOTIFY

The NOTIFY method [5] is used by a user agent to convey information about the occurrence of a particular event. A NOTIFY is always sent within a dialog when a subscription exists between the subscriber and the notifier. However, it is possible for a subscription to be established using non-SIP means (no SUBSCRIBE is sent) and may also be implicit in another SIP request type (for example, a REFER establishes an implicit subscription). Since it is sent within a dialog, the NOTIFY will contain a To tag, From tag, and existing Call-ID. A basic call flow showing NOTIFY is in Figure 4.5.

A NOTIFY request normally receives a 200 OK response to indicate that it has been received. If a 481 Dialog/Transaction Does Not Exist response is received, the subscription is automatically terminated and no more NOTIFYs are sent.

NOTIFY requests contain an Event header field indicating the package and a Subscription-State header field indicating the current state of the subscription. The Event header field will contain the package name used in the subscription. Currently defined packages are listed in Table 4.8. The Subscription-State header field will either be active, pending, or terminated.

A NOTIFY is always sent at the start of a subscription and at the termination of a subscription. If a NOTIFY contains incremental (delta) state information, the message body will contain a state version number that will be incremented by 1 for each NOTIFY sent. This way, the receiver of the NOTIFY can tell if information is missing or received out of sequence.

An example NOTIFY request is shown here:

```
NOTIFY sip:tyoung@parlour.elasticity.co.uk SIP/2.0
Via SIP/2.0/UDP cartouche.rosettastone.org:5060
 ;branch=z9hG4bK3841323
Max-Forwards: 70
To: Thomas Young <sip:tyoung@elasticity.co.uk>;tag=1814
From: <sip:ptolemy@rosettastone.org>;tag=5363956k
Call-ID: 452k59252058dkfj34924lk34
CSeq: 3 NOTIFY
Contact: <sip:ptolemy@cartouche.rosettastone.org>
```

```
Event: dialog
Subscription-State: active;expires=180
Allow-Events: dialog
Content-Type: application/xml+dialog
Content-Length: ...

(XML Message body not shown...)
```

Table 4.10 lists the mandatory header fields in a NOTIFY request.

4.1.9 PUBLISH

The PUBLISH method [17] is used by a user agent to send (or publish) event state information to a server known as an event state compositor (ESC). PUBLISH is most useful when there are multiple sources of event information, such as a number of devices sharing the same AOR. In this case, to find the complete state, another UA would need to subscribe individually to all the devices. Instead, the UA can subscribe to the ESC. Individual UAs send PUBLISHes to the ESC, which processes them and puts them together, generating NOTIFYs to watchers as shown in Figure 4.6.

An example PUBLISH request is shown here:

```
PUBLISH sip:percy@lowell.edu SIP/2.0
Via SIP/2.0/UDP telescope32.lowell.edu:54620
 ;branch=z9hG4bK43d132s3
Max-Forwards: 70
To: <sip:percy@lowell.edu>
From: <sip:percy@lowell.edu>;tag=5645fg432f
Call-ID: 34jdUhwiQhd72e
CSeq: 352 PUBLISH
Contact: <sip:percy@telescope32.lowell.edu:54620>
Event: presence
Min-Expires: 1800
Expires: 3600
Allow-Events: presence
Content-Type: application/xml+pidf
```

Table 4.10
Mandatory Header Fields in a NOTIFY

To
Via
To
From
Call-ID
CSeq
Max-Forwards
Event
Allow-Events
Subscription-State

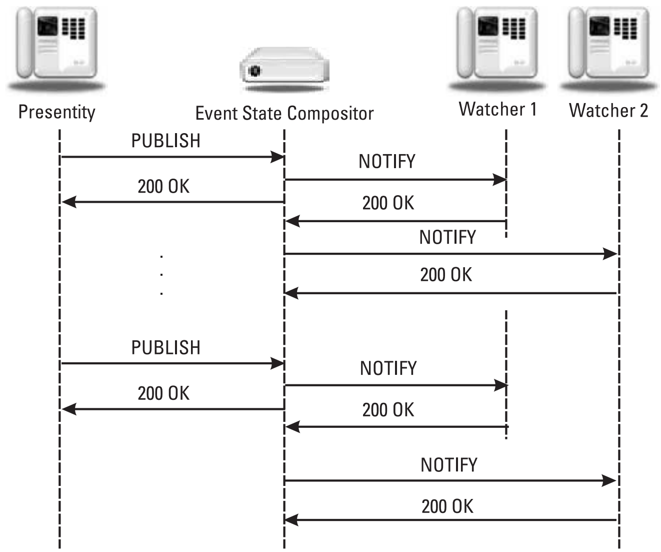


Figure 4.6 PUBLISH call flow example.

```
Content-Length: ...

(XML Message body not shown...)
```

A PUBLISH request is similar to a NOTIFY, except it is not sent in a dialog. A PUBLISH request must contain an Expires header field and a Min-Expires header field. The Expires header field indicates the maximum time when the ESC may discard the event state information, unless it is updated or refreshed. The Min-Expires header field indicates the minimum expiration that the ESC may choose. In the 200 OK response to the PUBLISH, the Expires header field will indicate the value chosen by the ESC, which must be between the Min-Expires and Expires intervals. If the Min-Expires interval is too short, the ESC may respond with a 423 Interval Too Brief response containing a Min-Expires interval acceptable to the ESC. The presence UA must then republish the information using this interval.

When an ESC receives and processes a PUBLISH it generates an *entity-tag*, a unique identifier for this piece of event state information, and returns the tag in a SIP-ETag header field of the 200 OK response. For example, here is a 200 OK response to the above PUBLISH:

```
SIP/2.0 200 OK
Via SIP/2.0/UDP telescope32.lowell.edu:54620
 ;branch=z9hG4bK43d132s3;received=173.34.3.1
Max-Forwards: 70
To: <sip:percy@lowell.edu>;tag=23211d
From: <sip:percy@lowell.edu>;tag=5645fg432f
```

```
Call-ID: 34jdUhwiQhd72e
CSeq: 352 PUBLISH
Event: presence
Expires: 1800
SIP-ETag: dkfiei4RIUOwqwe23
Allow-Events: presence
Content-Length: 0
```

Using the entity-tag, the publisher can update a previously published state. In this case, to refresh the previously published state, a `PUBLISH` containing a `SIP-If-Match` header field is used containing the assigned entity-tag, but not containing a body. In this way, the state can be refreshed without having to send the information again. For example, this `PUBLISH` could be used to refresh the first publication, provided it is received within the 30 minute interval:

```
PUBLISH sip:percy@lowell.edu SIP/2.0
Via SIP/2.0/UDP telescope32.lowell.edu:54620
 ;branch=z9hG4bK743d32s3a
Max-Forwards: 70
To: <sip:percy@lowell.edu>
From: <sip:percy@lowell.edu>;tag=458234kdf
Call-ID: 739238dkd2df
CSeq: 352 PUBLISH
Contact: <sip:percy@telescope32.lowell.edu:54620>
CSeq: 353 PUBLISH
Event: presence
Min-Expires: 1800
Expires: 3600
Allow-Events: presence
SIP-If-Match: dkfiei4RIUOwqwe23
Content-Length: 0
```

If the state had expired, or been updated such that the entity-tag no longer matches or is not valid, the ESC would return a `412 Conditional Request`

Table 4.11
Mandatory Header Fields in a `PUBLISH`

To
Via
To
From
Call-ID
CSeq
Max-Forwards
Contact
Event
Allow-Events
Expires
Min-Expires

Failed response. To modify the existing state, a PUBLISH would be sent containing the SIP-If-Match header field and a new message body. To remove the published state, a PUBLISH with the SIP-If-Match header field with an Expires:0 and no message body would result in the information being removed. Figure 8.7 shows an example of conditional publications.

Note that entity-tags are defined in HTTP [18]. However, the syntax and exact meaning are slightly different for SIP than HTTP.

Multiple presence UAs can publish for the same AOR to an ESC. In this case, the ESC will put the information together in an event specific way before generating notifications to subscribers. This process is referred to as state aggregation or composition.

4.1.10 REFER

The REFER method [14] is used by a user agent to request another user agent to access a URI or URL resource. The resource is identified by a URI or URL in the required Refer-To header field. Note that the URI or URL can be any type of URI: sip, sips, http, pres, and so forth. When the URI is a sip or sips URI, the REFER is probably being used to implement a call transfer service. REFER can also be used to implement peer-to-peer call control.

A REFER request can be sent either inside or outside an existing dialog. A typical call flow is shown in Figure 4.7. In this example, a UAC sends a REFER to a UAS. The UAS, after performing an authentication and authorization decides to accept the REFER and responds with a 202 Accepted response. Note that this response is sent immediately without waiting for the triggered request to complete. This is important because REFER uses the non-INVITE method state machine,

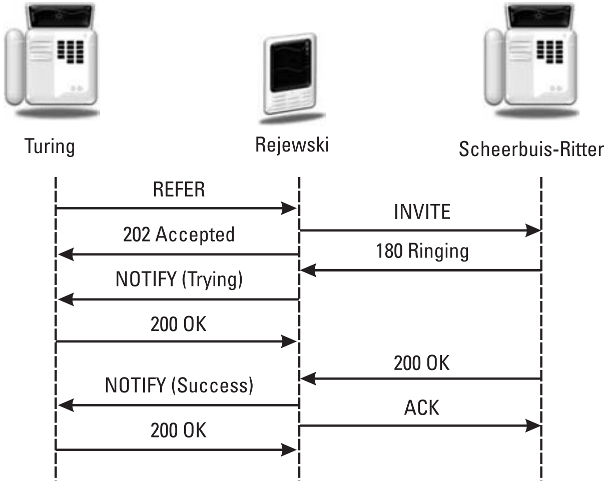


Figure 4.7 REFER example call flow.

which requires an immediate final response, unlike an INVITE which may take several seconds (or even minutes) to complete. Since the Refer-To URI in this example is a sip URI, the UAC sends an INVITE setting the Request-URI to the Refer-To URI. This INVITE is successful since it receives a 200 OK response. This successful outcome is communicated back to the UAC using a NOTIFY method (described in Section 4.1.8). The message body of the NOTIFY contains a partial copy of the final response to the triggered request. In this case, it contains the start-line SIP/2.0 200 OK. This part of a SIP message is described in the Content-Type header field as a message/sipfrag [19]. Note that this implicit subscription can be cancelled by including the Refer-Sub: false [20] header field in the REFER. If the 2xx response to the REFER also contains the Refer-Sub: false header field, no NOTIFYs will be sent.

An example of a REFER message is shown here:

```
REFER sip:m.rejewski@biuroszyfrow.pl SIP/2.0
Via SIP/2.0/UDP lab34.bletchleypark.co.uk:5060
 ;branch=z9hG4bK932039
Max-Forwards: 69
To: <sip:m.rejewski@biuroszyfrow.pl>;tag=ACEBDC
From: Alan Turing <sip:turing@bletchleypark.co.uk>
 ;tag=213424
Call-ID: 3419fak3kFD23s1A9dkl
CSeq: 5412 REFER
Refer-To: <sip:info@scherbius-ritter.com>
Content-Length: 0
```

Another example is the use of REFER to "push" a Web page. In the example of Figure 4.8, the UAC sends a REFER to the UAS with a Refer-To set to an HTTP URL or a Web page. This causes the UAS to send a 202 Accepted then

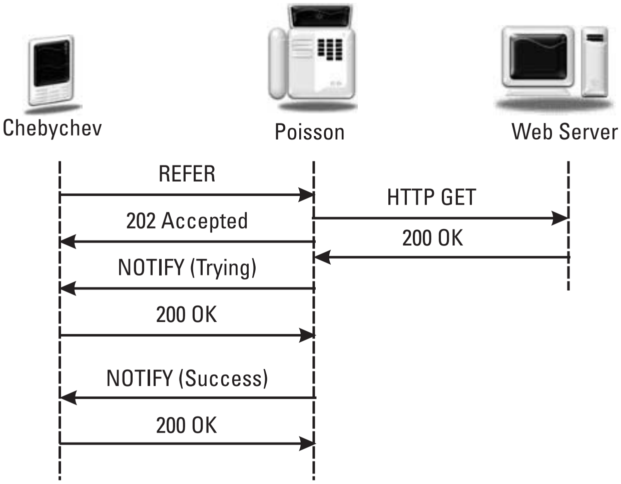


Figure 4.8 REFER example showing Web page push.

send an HTTP GET request to the Web server identified by the URL. After the Web page has loaded, the UAS sends a NOTIFY containing a body and HTTP/1.0 200 OK.

A REFER and the SIP request triggered by the REFER may contain the Referred-By header field, which contains information about who requested the request.

Figure 4.9 shows an advanced use of REFER to implement a common PSTN or PBX feature known as attended transfer [21]. In this feature, the transferor is assumed to be in a dialog (in a session) with the transferee. The transferor sends an INVITE to another party, called the transfer target. After the session is established between the transferor and the transfer target, the transferor sends a REFER to the transferee, which causes the transferee to generate a new INVITE (called

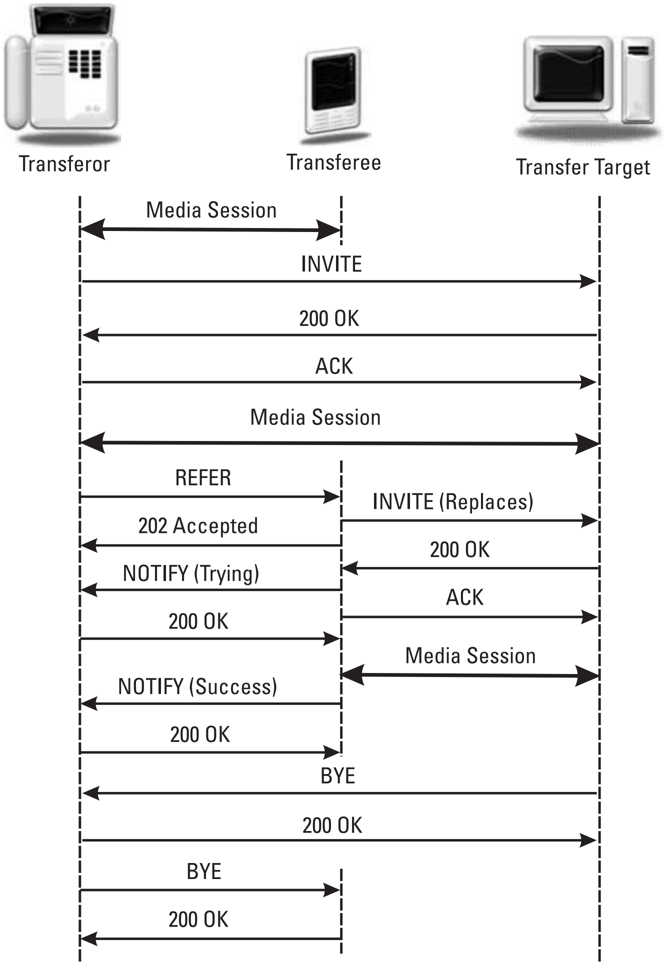


Figure 4.9 Use of REFER and Replaces to perform attended transfer feature.

a *triggered* INVITE) to the target. The successful INVITE replaces the existing session between the transferor and the transfer target. When the transferee receives notification that the transfer was successful, the session between the transferor and the transferee is terminated with a BYE. This application uses escaped header fields in the Refer-To URI. That is, certain SIP header fields are specified and prepopulated in the URI, which are then copied into the triggered INVITE. In this case, the transferor generates the Replaces header field necessary in the triggered INVITE to make the transfer succeed. The transferee copies the escaped Replaces header and places it in the INVITE.

The acceptance of a REFER with a 202 Accepted response creates an implicit subscription (a subscription without sending a SUBSCRIBE request; see Section 4.1.7). After sending the 202 Accepted, the target must send an immediate NOTIFY with the status 100 Trying and Subscription-State: active;expires=60, which indicates that the subscription will expire in 60 seconds (the expiration value is chosen by the notifier). The Subscription-State header contains the expiration time of the subscription. If that time period expires before the triggered request has completed, both sides terminate the subscription, with the notifier sending a final notification as discussed next.

The subscription is terminated when the transfer target (the party that accepted the REFER) sends a final notification (a NOTIFY with Subscription-State: terminated;reason=noresource). Usually, this is after the transfer target has received a final response to the triggered request. However, a transfer target that does not wish to establish a subscription and provide a final result of the REFER may send an immediate NOTIFY indicating that the subscription has been terminated. Each REFER sent creates a separate subscription. If more than one REFER is sent within a dialog, the resulting notifications (and subscriptions) are identified by an id parameter in the Event header field. The id parameter is optional in REFER triggered NOTIFYs except when multiple REFERs have been accepted, in which case it is mandatory.

The optional Referred-By header field can be included in a REFER request. The Refer-To header field can also contain feature tags [22] which tell the REFER recipient about the resource being referenced. Table 4.12 lists the mandatory header fields in a REFER request.

4.1.11 MESSAGE

The MESSAGE method [23] is used to transport instant messages (IM) using SIP. IMs usually consists of short messages exchanged in near-real time by participants engaged in a text conversation. MESSAGEs may be sent within a dialog or outside a dialog, but they do not establish a dialog by themselves. The actual message content is carried in the message body as a MIME attachment. All UAs

Table 4.12
Mandatory Header Fields in a REFER

Via
To
From
Call-ID
CSeq
Max-Forwards
Refer-To

that support the MESSAGE method must support plain/text format; they may also support other formats such as message/cpim [24] or text/html, as shown in Table 8.10.

A MESSAGE request normally receives a 200 OK response to indicate that the message has been delivered to the final destination. An IM response should not be sent in the message body of a 200 OK, but rather a separate MESSAGE request sent to the original sender. A 202 Accepted response indicates that the request has reached a store-and-forward device and will likely eventually be delivered to the final destination. In neither case does the 2xx response confirm that the message content has been rendered to the user. For this, the delivery notification mechanism is used, which will be discussed later in Section 8.5.2.

A MESSAGE request may use the im (instant message) URI scheme [25] in a Request-URI, although a client should try to resolve to a sip or sips URI.

An example MESSAGE call flow is shown in Figure 4.10.

Note that the MESSAGE method is not the only application of instant messaging with SIP. It is also possible to use SIP to establish an instant message

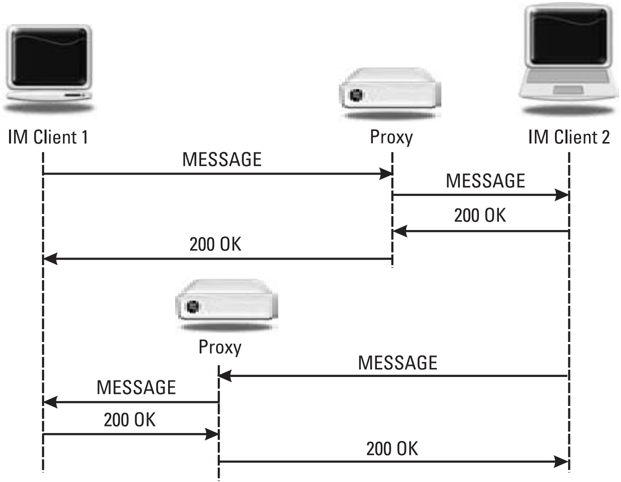


Figure 4.10 A SIP MESSAGE call flow showing instant message transport.

session in a completely analogous way that SIP is commonly used to establish a media session. An INVITE could be used to establish the session with a SDP body that describes the instant message protocol to be used directly between the two users. For example, Message Sessions Relay Protocol (MSRP), covered in Section 8.5.5, can be used for this.

An example MESSAGE request is shown here:

```
MESSAGE sip:editor@rcs.org SIP/2.0
Via SIP/2.0/UDP lab.mendeleev.org:5060;branch=z9hG4bK3
Max-Forwards: 70
To: <editor@rcs.org>
From: "D. I. Mendeleev" <dmitry@mendeleev.org>;tag=1865
Call-ID: 93847197172049343
CSeq: 5634 MESSAGE
Subject: First Row
Contact: <sip:dmitry@lab.mendeleev.org>
Content-Type: text/plain
Content-Length: 7

H, He
```

Table 4.13 lists the mandatory header fields in a MESSAGE request.

4.1.12 INFO

The INFO [26] method is used by a UA to send call signaling information to another UA with which it has an established media session. The request is end-to-end and is never initiated by proxies. A proxy will always forward an INFO request—it is up to the UAS to check to see if the dialog is valid. INFO requests for unknown dialogs receive a 481 Transaction/Dialog Does Not Exist response.

An INFO method typically contains a message body. The contents may be signaling information, a midcall event, or some sort of stimulus. INFO has been proposed to carry certain PSTN midcall signaling information such as ISUP (ISDN User Part) USR messages.

The INFO method always increments the CSeq. An example INFO method is:

Table 4.13
Mandatory Header Fields in a MESSAGE

To
Via
To
From
Call-ID
CSeq
Max-Forwards

```
INFO sip:poynting@mason.edu.uk SIP/2.0
Via: SIP/2.0/UDP cavendish.kings.cambridge.edu.uk
 ;branch=z9hG4bK24555
Max-Forwards: 70
To: John Poynting <sip:nting@mason.edu.uk> ;tag=3432
From: J.C. Maxwell <sip:james.maxwell@kings.cambridge.edu.uk>
 ;tag=432485820183
Call-ID: e71facaa7f7c0a29276054fe4951a9b6
Content-Type: application/ISUP
Content-Length: ...

(Binary message body not shown)
```

The base INFO specification does not have any mechanisms for negotiating which types of INFO bodies are acceptable. An extension [27] is being developed to add this capability. The extension defines packages for INFO usages, and a mechanism to discover and declare support for packages. The Recv-Info header field is included in requests and responses listing the INFO packages that the UA is willing to receive. The Info-Package header field is included in INFO requests indicating which package is being used.

The mandatory headers in an INFO request are shown in Table 4.14. Note that for backwards compatibility reasons, INFO without Info-Package will need to be accepted.

4.1.13 PRACK

The PRACK [28] method is used to acknowledge receipt of reliably transported provisional responses (1xx). The reliability of 2xx, 3xx, 4xx, 5xx, and 6xx responses to INVITEs is achieved using the ACK method. However, in cases where a provisional response, such as 180 Ringing, is critical in determining the call state, it may be necessary for the receipt of a provisional response to be confirmed. The PRACK method applies to all provisional responses except the 100 Trying response, which is never reliably transported.

A PRACK is generated by a UAC when a provisional response has been received containing an RSeq reliable sequence number and a Supported: 100rel

Table 4.14
Mandatory Header Fields in an INFO

To
Via
To
From
Call-ID
CSeq
Max-Forwards
Info-Package

header. The PRACK echoes the number in the RSeq and the CSeq of the response in a RAck header. The message flow is as shown in Figure 4.11. In this example, the UAC sends the 180 Ringing response reliably by including the RSeq header. When no PRACK is received from the UAC after the expiration of a timer (an “X” is used to represent a lost message), the response is retransmitted. The receipt of the PRACK confirms the delivery of the response and stops all further transmissions. The 200 OK response to the PRACK stops retransmissions of the PRACK request.

Reliable responses are retransmitted using the same exponential backoff mechanism used for final responses to an INVITE. The combination of Call-ID, CSeq number, and RAck number allows the UAC to match the PRACK to the provisional response it is acknowledging. As shown in Figure 4.11, the PRACK receives a 200 OK response, which can be distinguished from the 200 OK to the INVITE by the method contained in the CSeq header.

The PRACK method always increments the CSeq. A PRACK may contain a message body and may be used for offer/answer exchanges. An example exchange contains:

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP lucasian.trinity.cambridge.edu.uk
 ;branch=z9hG4bK452352;received=1.2.3.4
To: Descartes <sip:rene.descartes@metaphysics.org>;tag=12323
From: Newton <sip:newton@kings.cambridge.edu.uk>;tag=981
```

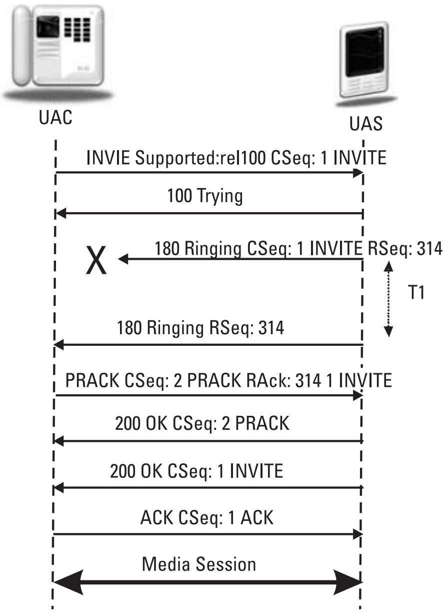

UAC
UAS
INVIE Supported:rel100 CSeq: 1 INVITE
100 Trying
X
180 Ringing CSeq: 1 INVITE RSeq: 314
T1
180 Ringing RSeq: 314
PRACK CSeq: 2 PRACK RAck: 314 1 INVITE
200 OK CSeq: 2 PRACK
200 OK CSeq: 1 INVITE
ACK CSeq: 1 ACK
Media Session

Figure 4.11 PRACK call flow example showing reliable provisional responses.

```
Call-ID: da6fa909f1c0188c539feb08d4496eb7
RSeq: 314
CSeq: 10 INVITE
Content-Length: 0

PRACK sip:rene.descartes@metaphysics.org SIP/2.0
Via: SIP/2.0/UDP lucasian.trinity.cambridge.edu.uk
 ;branch=z9hG4bKdtyw
Max-Forwards: 70
To: Descartes <sip:rene.descartes@metaphysics.org>;tag=12323
From: Newton <sip:newton@kings.cambridge.edu.uk>;tag=981
Call-ID: da6fa909f1c0188c539feb08d4496eb7
CSeq: 2 PRACK
RAck: 314 10 INVITE
Content-Length: 0

SIP/2.0 200 OK
Via: SIP/2.0/UDP lucasian.trinity.cambridge.edu.uk
 ;branch=z9hG4bKdtyw ;received=1.2.3.4
To: Descartes <sip:rene.descartes@metaphysics.org>;tag=12323
From: Newton <sip:newton@kings.cambridge.edu.uk>;tag=981
Call-ID: da6fa909f1c0188c539feb08d4496eb7
CSeq: 2 PRACK
Content-Length: 0
```

The mandatory header fields in a PRACK request are shown in Table 4.15.

4.1.14 UPDATE

The UPDATE method [29] is used to modify the state of a session without changing the state of the dialog. A session is established in SIP using an INVITE request (see Section 4.1.1) in an offer/answer manner, described in Chapter 13. Typically, a session offer is made in the INVITE and an answer is made in a response to the INVITE. In an established session, a re-INVITE is used to update session parameters. However, neither party in a pending session (INVITE sent but no final response received) may re-INVITE—instead, the UPDATE method is used.

Possible uses of UPDATE include muting or placing on hold pending media streams, performing QoS, or other end-to-end attribute negotiation prior to session establishment.

Table 4.15
Mandatory Header Fields in a PRACK

To
Via
To
From
Call-ID
CSeq
Max-Forwards
RAck

Figure 4.12 and the following show an example of an `UPDATE` message.

```
UPDATE sips:beale@bufords.bedford.va.us SIP/2.0
Via SIP/2.0/TLS client.crypto.org:5061;branch=z9hG4bK342
Max-Forwards: 70
To: T. Beale <sips:beale@bufords.bedford.va.us>;tag=71
From: Blaise Vigenere <sips:bvigenere@crypto.org>;tag=19438
Call-ID: 170189761183162948
CSeq: 94 UPDATE
Contact: <sips:client.crypto.org>
Content-Type: application/sdp
Content-Length: ...

(SDP Message body not shown...)
```

Table 4.16 lists the mandatory header fields in an `UPDATE` request.

4.2 URI and URL Schemes Used by SIP

SIP supports a number of URI and URL schemes including `sip`, `sips`, `tel`, `pres`, and `im` for SIP, secure SIP, telephone, presence, and instant message URIs as described in the following sections. In addition, other URI schemes can be present in SIP header fields as listed in Table 1.2.

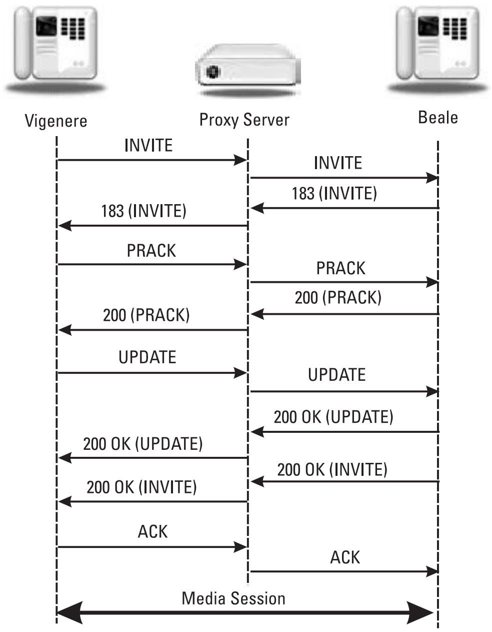


Figure 4.12 An `UPDATE` call flow example showing an offer/answer exchange.

Table 4.16
Mandatory Header Fields in an `UPDATE`

To
Via
To
From
Call-ID
CSeq
Max-Forwards
Contact

4.2.1 SIP and SIPS URIs

The addressing scheme of SIP URLs and URIs has been previously mentioned. SIP URIs are used in a number of places including the `To`, `From`, and `Contact` headers, as well as in the Request-URI, which indicates the destination. SIP URIs are similar to the `mailto` URL [29] and can be used in hyperlinks on Web pages, for example. They can also include telephone numbers. The information in a SIP URI indicates the way in which the resource (user) should be contacted using SIP.

An example SIP URI contains the scheme `sip` a “:”, then a `username@host` or IPv4 or IPv6 address followed by an optional “:”, then the port number, or a list of “;” separated URI parameters:

```
sip:joseph.fourier@transform.org:5060;transport=udp;user=ip;m
ethod=INVITE;ttl=1;maddr=240.101.102.103?Subject=FFT
```

Note that URIs may not contain spaces or line breaks, so this example would be on a single line. Some SIP URIs, such as a `REGISTER` Request-URI do not have a username, but begin with the host or IP address. In this example, the port number is shown as `5060`, the well-known port number for SIP. For a SIP URI, if the port number is not present `5060` is assumed. For a SIPS URI, port number `5061` is assumed. The `transport` parameter indicates UDP is to be used, which is the default. TCP, TLS, and SCTP are alternative transport parameters.

The `user` parameter is used by parsers to determine if a telephone number is present in the username portion of the URI. The assumed default is that it is not, indicated by the value `ip`. If a telephone number is present, it is indicated by the value `phone`. This parameter must not be used to guess at the characteristics or capabilities of the user agent. For example, the presence of a `user=phone` parameter must not be interpreted that the user agent is a SIP telephone (which may have limited display and processing capabilities). In a telephony environment, IP telephones and IP/PSTN gateways may in fact use the reverse assumption, interpreting any digits in a username as digits regardless of the presence of user=phone.

The method parameter is used to indicate the method to be used. The default is INVITE. This parameter has no meaning in To or From header fields or in a Request-URI but can be used in Contact headers for registration, for example, or in a Refer-To header field.

The ttl parameter is the time-to-live, which must only be used if the maddr parameter contains a multicast address and the transport parameter contains udp. The default value is 1. This value scopes the multicast session broadcast.

The maddr usually contains the multicast address to which the request should be directed, overriding the address in the host portion of the URI. It can also contain a unicast address of an alternative server for requests.

The method, maddr, ttl, and header parameters must not be included in To or From headers, but may be included in Contact headers or in Request-URIs. In addition to these parameters, a SIP URI may contain other user-defined parameters.

Following the "?" parameter, names can be specified to be included in the request. This is similar to the operation of the mailto URL, which allows Subject and Priority to be set for the request. Additional headers can be specified, separated by an "&". The header name body indicates that the contents of a message body for an INVITE request are being specified in the URI.

If the parameter user=phone is present, then the username portion of the URI can be interpreted as a telephone number. This allows additional parameters in the username portion of the URI, which allows the parameters and structure of a tel URL [30] to be present in the user part of the SIP URI as described in the next section.

The sips URI scheme has the same structure as the sip URI but begins with the sips scheme name. Note that a sips URI is not equivalent to a sip URI with transport=tls, since the sip URI does not have the same security requirements as the sips URI. The requirement is that TLS transport is used end-to-end for the SIP path. The only exception is a hop between the final proxy and the UAS, which may use another security mechanism besides TLS (IPSec, for example).

Not shown in the example is the loose route parameter lr, which can be present in sip or sips Record-Route and Route URIs to indicate that the proxy server identified by the URI supports loose routing.

4.2.2 Telephone URLs

The telephone URI scheme, tel, [30] can be used to represent a resource identified by a telephone number. Telephone numbers can be of two general forms, local or global. A local number is only valid in a particular geographic area and has only local significance. If the number is used outside of this area, it will either fail or return the wrong resource. A global telephone number, also called an E.164 number, is one that is, in principle, valid anywhere. It contains enough information about the country, region, and locality for the PSTN network to route calls to the correct resource. An example of a local phone number is:

```
tel:411;phone-context=+1314
```

This indicates a call to directory assistance valid only within country code 1 and area code 314 as identified in the required `phone-context` parameter. An example of a global phone number is:

```
tel:+13145551212
```

Global phone numbers always begin with the "+" identifier followed by the country code, 1 in this case, followed by the remaining telephone digits.

A `tel` URL can also contain some characters and information about dialing strings and patterns. For example:

```
tel:#70555-1212;isub=1000
```

In this example, the dialed digit string, interpreted by a PSTN gateway, would be the DTMF digit # then 70 (to cancel call waiting, for example), then the digits 555–1212. Additional parameters include an ISDN subaddress of 1000. This example shows both types of optional visual separators allowed, either "-" or "." as the separator.

Tel URLs can also be embedded in Web pages and can be included in HTML as, for example:

```
Click <A HREF="tel:+1972.555.1212">here</A> to get information
about Dallas.
```

The syntax and parameters of the `tel` URL may be used in the user portion of a `sip` URI. For example, the first `tel` example could be represented as a `sip` URI as follows:

```
sip:411;phone-context=+1314@gateway.example.com
```

The SIP URI adds a domain portion which represents the domain or gateway that will route the request.

4.2.3 Presence and Instant Messaging URLs

The pres URL scheme is defined [25] as a URL scheme that represents a "presentity" or presence agent. The im URL scheme is defined [25] as a URL scheme that represents an "instant inbox" or an instant message client. Both URL schemes do not represent a new protocol but are resolved using DNS SRV resource records, which return another URI that indicates the actual presence or instant messaging protocol. For example, if the presence agent reference by the presence URL:

```
pres:user@example.com
```

supports SIP presence, the DNS SRV query would return a SIP URI, for example:

```
sip:user@example.com
```

This would then allow a presence agent to send a SUBSCRIBE to this SIP URI to obtain the presence agent of this user.

The same procedure would be used for resolving an im URL into a SIP URI for sending a MESSAGE request.

4.3 Tags

A tag is a cryptographically random number with at least 32 bits of randomness, which is added to To and From headers to uniquely identify a dialog. The examples in Chapters 2 and 16 show the use of the tag header parameter. The To header in the initial INVITE will not contain a tag. A caller must include a tag in the From header, although an RFC 2543 UA generally will not do so as it is optional in that specification. Excluding 100 Trying, all responses will have a tag added to the To header. The sending or receiving of a response containing a From tag creates an early dialog. A tag returned in a 200 OK response is then incorporated as a dialog identifier and used in all future requests for this Call-ID. A tag is never copied across calls. Any response generated by a proxy will have a tag added by the proxy. An ACK generated by either a UA or a proxy will always copy the From tag of the response in the ACK request.

If a UAC receives responses containing different tags, this means that the responses are from different UASs, and the INVITE has been forked. It is up to the UAC as to how to deal with this situation. For example, the UAC could establish separate sessions with each of the responding UAS. The dialogs would contain the same From, Call-ID, and CSeq, but would have different tags in the To header. The UAC also could BYE certain legs and establish only one session.

Note that tags are not part of the To or From URI but are part of the header and always placed outside any "<>".

4.4 Message Bodies

Message bodies in SIP may contain various types of information. They may contain SDP information, which can be used to convey media information, QoS, or even security information.

The optional `Content-Disposition` header is used to indicate the intended use of the message body. If not present, the function is assumed to be `session`, which means that the body describes a media session. Besides `session`, the other defined function is `render`, which means that the message body should be presented to the user or otherwise used or displayed. This could be used to pass a small JPEG image file or URI.

The format of the message body is indicated by the `Content-Type` header. If a message contains a message body, the message must include a `Content-Type` header. All UAs must support a `Content-Type` of `application/sdp`. The encoding scheme of the message body is indicated in the `Content-Encoding` header. If not specified, the encoding is assumed to be `text/plain`. The specification of a `Content-Encoding` scheme allows the message body to be compressed.

The `Content-Length` header contains the number of octets in the message body. If there is no message body, the `Content-Length` header should still be included but with a value of 0. Because multiple SIP messages can be sent in a TCP stream, the `Content-Length` count is a reliable way to detect when one message ends and another begins. If a `Content-Length` is not present, the UAC must assume that the message body continues until the end of the UDP datagram, or until the TCP connection is closed, depending on the transport protocol.

Message bodies can have multiple parts if they are encoded using Multipart Internet Mail Extensions (MIME). Message bodies in SIP, however, should be small enough so that they do not exceed the UDP MTU of the network. Proxies may reject requests with large message bodies with a `413 Request Entity Too Large` response, since processing large messages can load a server. Guidelines for SIP handling of message bodies is described in [31].

As mentioned in the previous section, SIP carries message bodies the same way that e-mails carry attachments. It is possible to carry multiple message bodies within a single SIP message. This is done using a multipart MIME body. The `Content-Type` is listed as `multipart/mime`, and a separator is defined, which is used by the parser to separate the message. Any SIP request or response that can contain a message body may carry a multipart MIME body. An example is in SIP-T (see Section 11.2) in which an `INVITE` carries both a SDP message body (`application/sdp`) and an encapsulated ISUP message (`application/isup`). An example multipart MIME is:

```
INVITE sip:refertarget@carol.example.com SIP/2.0
Via: SIP/2.0/UDP referree.example;branch=z9hG4bKffe209934aac
To: sip:refertarget@carol.example.com
```

```
From: <sip:referree@referree.example>;tag=2909034023
Call-ID: 9023940-a34658d
CSeq: 9823409 INVITE
Max-Forwards: 70
Contact: <sip:referree@bob.example.com>
Referred-By: sip:referror@alice.example.com
 ;cid=%3C20398823.2UWQFN309shb3@alice.example.com%3E
Content-Type: multipart/mixed; boundary=-*-boundary-*-
Content-Length: ...

—-*-boundary-*-

Content-Type: application/sdp
Content-Length: ...

v=0
o=referree 2890844526 2890844526 IN IP4 referree.example
s=Session SDP
c=IN IP4 referree.example
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000

—-*-boundary-*-

Content-Type: multipart/signed;
 protocol=”application/pkcs7-signature”;
 micalg=sha1; boundary=dragons39
Content-ID: <20398823.2UWQFN309shb3@alice.example.com>
Content-Length: ...

-another-boundary-
Content-Type: message/sipfrag
Content-Disposition: auth-id; handling=optional

From: sip:referror@alice.example.com
Date: Thu, 21 Feb 2002 13:02:03 GMT
Call-ID: 2203900ef0299349d9209f023a
Refer-To: sip:refertarget@carol.example.com
Referred-By: sip:referror@alice.example.com
 ;cid=%3C20398823.2UWQFN309shb3@alice.example.com%3E

-another-boundary-
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s
 ;handling=required

(S/MIME data goes here)

—-*-boundary-*—-
```

Between each body part is a string, in this example `-*-boundary-*-` which is defined in the `Content-Type` header field.

4.5 Conclusion

This chapter covered the six base methods in RFC 3261 plus the eight extension methods defined in other RFCs. In addition, SIP URIs, URLs, tags, and message bodies have been covered.

4.6 Questions

Q4.1 Explain what happens when the expiration interval in an `Expires` header field in an INVITE expires.

Q4.2 For the three REGISTER requests sent (in this sequence) below, generate appropriate responses to each from the registrar.

```
REGISTER sip:registrar.athens.gr SIP/2.0
Via: SIP/2.0/UDP 201.202.203.204:5060;branch=z9hG4bK231U3
Max-Forwards:70
To: sip:euclid@athens.gr
From: <sip:euclid@athens.gr>;tag=543131
Call-ID: 48erl8132409wqer
CSeq: 1 REGISTER
Contact: <sip:euclid@parthenon.athens.gr>
Content-Length: 0

REGISTER sip:registrar.athens.gr SIP/2.0
Via: SIP/2.0/UDP 201.202.203.204:5060;branch=z9hG4bK3r13
Max-Forwards:70
To: sip:euclid@athens.gr
From: < sip:euclid@athens.gr>;tag=75653
Call-ID: 48erl8132409wqer
CSeq: 2 REGISTER
Contact: <mailto:euclid@geometry.org>
Content-Length: 0

REGISTER sip:registrar.athens.gr SIP/2.0
Via: SIP/2.0/UDP 201.202.203.204:5060;branch=z9hG4bK3313
Max-Forwards:70
To: sip:euclid@athens.gr
From: <sip:euclid@athens.gr>;tag=4d31f31
Call-ID: 48erl8132409wqer
CSeq: 3 REGISTER
Content-Length: 0
```

Q4.3 Is a message body permitted in an ACK? Give an example of this usage.

Q4.4 Explain what happens if a `200 OK` and the CANCEL for the INVITE cross on the wire between a proxy and a UA.

Q4.5 If a SUBSCRIBE is forked by a proxy, and multiple subscriptions are established, how will the watcher know this and keep the subscriptions separate?

Q4.6 Generate a suitable PRACK message in response to the 180 Ringing response here:

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP boyden.harvard.edu
 ;branch=z9hG4bK452352;received=1.2.3.4
To: Clyde <sip:clyde.tombaugh@lowell.edu>;tag=312323
From: Bill <sip:william.pickering@harvard.edu>
 ;tag=877s981
Call-ID: Ldfk37sfa2DF
RSeq: 17314
CSeq: 53 INVITE
Content-Length: 0
```

Q4.7 Explain how PUBLISH can be used to create, refresh, update, and delete event state.

Q4.8 Generate the SUBSCRIBE that could have caused this NOTIFY to be sent:

```
NOTIFY sip:e.hubble@mtwilson.edu SIP/2.0
Via SIP/2.0/UDP room421.caltech.edu:5060
 ;branch=z9hG4bK3841323
Max-Forwards: 70
To: <sip:e.hubble@mtwilson.edu>;tag=8311814
From: <sip:georgehale@caltech.edu>;tag=5363956k
Call-ID: 58kjeGrkre88er
CSeq: 73 NOTIFY
Contact: <sip:georgehale@room421.caltech.edu>
Event: presence
Subscription-State: active;expires=3540
Allow-Events: dialog
Content-Type: application/xml+dialog
Content-Length: ...
```

Q4.9 How is a subscribe-created dialog terminated?

Q4.10 Explain when a 412 response might be received in response to a PUBLISH. What should the presence UA do after receiving this response?

References

[1] Rosenberg, J., H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261, June 2002 .

[2] Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)," draft-ietf-sip-gruu-15 (work in progress), October 2007.

[3] Rosenberg, J., H. Schulzrinne, and P. Kyzivat, "Indicating User Agent Capabilities in the-Session Initiation Protocol (SIP)," RFC 3840, August 2004.

[4] Johnston, A., and O. Levin, "Session Initiation Protocol (SIP) Call Control—Conferencing for User Agents," BCP 119, RFC 4579, August 2006.

[5] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification," RFC 3265, June 2003.

[6] Petrack, S., and L. Conroy, "The PINT Service Protocol: Extensions to SIP and SDP for IP Access to Telephone Call Services," RFC 2848, June 2000.

[7] Rosenberg, J., H. Schulzrinne, and O. Levin, "A Session Initiation Protocol (SIP) Event Package for Conference State," RFC 4575, August 2006.

[8] Camarillo, G., "The Session Initiation Protocol (SIP) Pending Additions Event Package," RFC 5362, October 2008.

[9] Rosenberg, J., H. Schulzrinne, and R. Mahy, "An INVITE-Initiated Dialog Event Package-for the Session Initiation Protocol (SIP)," RFC 4235, November 2005.

[10] Burger, E., and M. Dolly, "A Session Initiation Protocol (SIP) Event Package for Key Press Stimulus (KPML)," RFC 4730, November 2006.

[11] Mahy, R., "A Message Summary and Message Waiting Indication Event Package for the Session Initiation Protocol (SIP)," RFC 3842, August 2004.

[12] Rosenberg, J., "A Presence Event Package for the Session Initiation Protocol (SIP)," RFC 3856, August 2004.

[13] Rosenberg, J., "A Session Initiation Protocol (SIP) Event Package for Registrations," RFC 3680, March 2004.

[14] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method," RFC 3515, April 2003.

[15] Rosenberg, J., "A Watcher Information Event Template-Package for the Session InitiationProtocol (SIP)," RFC 3857, August 2004.

[16] Clark, A., et al., "RTCP-XR Summary," draft-ietf-sipping-rtcp-summary-06 (work in progress), March 2009.

[17] Niemi, A., "Session Initiation Protocol (SIP) Extension for Event State Publication," RFC 3903, October 2004.

[18] Fielding, R., et al., "Hypertext Transfer Protocol—HTTP/1.1," RFC 2616, June 1999.

[19] Sparks, R., "Internet Media Type message/sipfrag," RFC 3420, November 2002.

[20] Levin, O., "Suppression of Session Initiation Protocol (SIP) REFER Method Implicit Subscription," RFC 4488, May 2006.

[21] Sparks, R., and A. Johnston, "Session Initiation Protocol Call Control—Transfer," RFC 5589, June 2009.

[22] Levin, O., and A. Johnston, "Conveying Feature Tags with the Session Initiation Protocol (SIP) REFER Method," RFC 4508, May 2006.

[23] Campbell, B., et al., "Session Initiation Protocol (SIP) Extension for Instant Messaging," RFC 3428, December 2002.

[24] Klyne, G., and D. Atkins, "Common Presence and Instant Messaging (CPIM): Message Format," RFC 3862, August 2004.

[25] Peterson, J., "Address Resolution for Instant Messaging and Presence," RFC 3861, August 2004.

[26] Donovan, S., "The SIP INFO Method," RFC 2976, October 2000.

[27] Burger, E., H. Kaplan, and C. Holmberg, "Session Initiation Protocol (SIP) INFO Method and Package Framework," draft-ietf-sip-info-events-03 (work in progress) January 2009.

[28] Rosenberg, J., and H. Schulzrinne, "Reliability of Provisional Responses in Session Initiation Protocol (SIP)," RFC 3262, June 2002.

[29] Hoffman, P., L. Masinter, and J. Zawinski, "The mailto URL Scheme," RFC 2368, July 1998.

[30] Schulzrinne, H., "The tel URI for Telephone Numbers," RFC 3966, December 2004.

[31] Camarillo, G., "Message Body Handling in the Session Initiation Protocol (SIP)," draft-ietf-sip-body-handling-06 (work in progress), March 2009.

5

SIP Response Messages

This chapter covers the types of SIP response messages. A SIP response is a message generated by a UAS or a SIP server to reply to a request generated by a UAC. A response may contain additional header fields of information needed by the UAC, or it may be a simple acknowledgment to prevent retransmissions of the request by the UAC. Many responses direct the UAC to take specific additional steps. The responses are discussed in terms of structure and classes. Then each request type is discussed and examined in detail.

There are six classes of SIP responses. The first five classes were borrowed from HTTP; the sixth was created for SIP. The classes are shown in Table 5.1.

If a particular SIP response code is not understood by a UAC, it must be interpreted by the class of the response. For example, an unknown `599 Server Unplugged` response must be interpreted by a user agent as a `500 Server Failure` response.

The reason phrase is for human consumption only—the SIP protocol uses only the response code in determining behavior. Thus, a `200 Call Failed` is interpreted the same as `200 OK`. The reason phrases listed here are the suggested ones from the RFC document. They can be used to convey more information, especially in failure class responses—the phrase is likely to be displayed to the user. Some response codes were borrowed from HTTP, sometimes with a slightly different reason phrase. However, not all HTTP response codes are valid in SIP, and some even have a different meaning.

Unless otherwise referenced, the responses described here are defined in RFC 3261 [1].

Table 5.1
SIP Response Classes

Class	Description	Action
1xx	Informational	This indicates the status of the call prior to completion—also known as a provisional response.
2xx	Success	The request has succeeded. If it was for an INVITE, ACK should be sent; otherwise, stop the retransmissions of the request.
3xx	Redirection	The server has returned possible locations. The client should retry the request at another server.
4xx	Client error	The request has failed due to an error by the client. The client may retry the request if it is reformulated according to the response.
5xx	Server failure	The request has failed due to an error by the server. The request may be retried at another server.
6xx	Global failure	The request has failed. The request should not be tried again at this or other servers.

5.1 Informational

The informational class of responses, 1xx, is used to indicate call progress. Informational responses are end-to-end responses and may contain message bodies. The exception to this is the 100 Trying response, which is only a hop-by-hop response and may not contain a message body. Any number of informational responses can be sent by a UAS prior to a final response (2xx, 3xx, 4xx, 5xx, or 6xx class response) being sent. The first informational response received by the UAC confirms receipt of the INVITE, and stops retransmission of the INVITE, as shown in Figure 3.7. For this reason, servers returning 100 Trying responses minimize INVITE retransmissions in the network. Further informational responses have no effect on INVITE retransmissions. A stateful proxy receiving a retransmission of an INVITE will resend the last provisional response sent to date. Informational responses are optional—a UAS can send a final response without first sending an informational response. While final responses to an INVITE receive an ACK to confirm receipt, provisional responses are not acknowledged, except when using the PRACK method described in Section 4.1.13.

All provisional responses with the exception of 100 Trying must contain a Contact URI and echo all Record-Route headers received in the request. However, a RFC 2543 implementation will not do this, as it was not mandated in that document.

5.1.1 100 Trying

This special case response is only a hop-by-hop request. It is never forwarded and may not contain a message body. A forking proxy must send a 100 Trying response, since the extended search being performed may take a significant amount of time. This response can be generated by either a proxy server or a user agent. It only indicates that some kind of action is being taken to process the call—it does not indicate that the user has been located. A `100 Trying` response typically does not contain a `To` tag and hence does not create an early dialog.

5.1.2 180 Ringing

This response is used to indicate that the `INVITE` has been received by the user agent and alerting is taking place. This response is important in the interworking of telephony protocols, and it is typically mapped to messages such as an ISDN progress or ISUP address complete message (ACM) [2]. When the user agent answers immediately, a `200 OK` is sent without a `180 Ringing`; this scenario is called the "fast answer" case in telephony.

A UA normally generates its own ring back tone or remote ringing indication, unless an `Alert-Info` header field is present.

5.1.3 181 Call is Being Forwarded

This response is used to indicate that the call has been handed off to another end point. It is sent when the information may be of use to the caller. Also, because a forwarding operation may result in the call taking longer to be answered, this response gives a status for the caller.

5.1.4 182 Call Queued

This response is used to indicate that the `INVITE` has been received and will be processed in a queue. The reason phrase can be used to indicate the estimated wait time or the number of callers in line, as shown in Figure 5.1.

5.1.5 183 Session Progress

The `183 Session Progress` response indicates that information about the progress of the session (call state) may be present in a message body or media stream. Unlike a `100 Trying` response, a `183` is an end-to-end response and establishes a dialog (must contain a `To` tag and `Contact`). Unlike a `180`, `181`, or `182` response, it does not convey any specific information about the status of the `INVITE`. A typical use of this response is to allow a UAC to hear a ring tone, busy tone, or recorded announcement in calls through a gateway into the PSTN. This is because call progress information is carried in the media stream in the PSTN. A one-way media connection or trunk is established from the calling party's telephone switch to the called party's telephone switch in the PSTN prior to the call being answered. In SIP, the media session is established after the call is answered—after a `200 OK` and `ACK` have been exchanged between the UAC and UAS. If a gateway

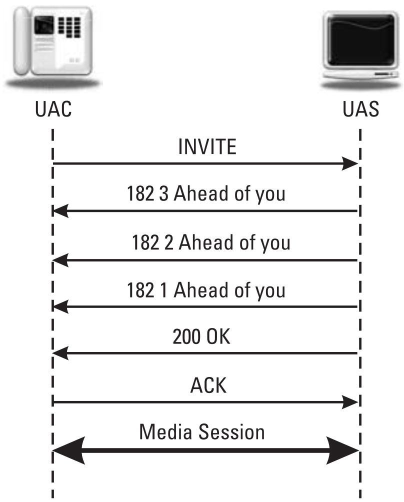


Figure 5.1 Call queuing with a call processing center.

uses a `180 Ringing` response instead, no media path will be established between the UAC and the gateway, and the caller will never hear a ring tone, busy tone, or recorded announcement (e.g., "The number you have dialed has changed, the new number is . . .") since these are all heard in the media path prior to the call being answered. Figure 9.1 shows an example call flow with early media.

5.2 Success

Success class responses indicate that the request has succeeded or has been accepted.

5.2.1 200 OK

The `200 OK` response has two uses in SIP. When used to accept a session invitation, it will contain a message body containing the media properties of the UAS (called party). When used in response to other requests, it indicates a successful completion or receipt of the request. The response stops further retransmissions of the request. In response to an `OPTIONS`, the message body may contain the capabilities of the server. A message body may also be present in a response to a `REGISTER` request. For `200 OK` responses to other methods, a message body is not permitted.

5.2.2 202 Accepted

The `202 Accepted` response [3] indicates that the UAS has received and understood the request, but that the request may not have been authorized or processed by the server. It is commonly used in responses to `SUBSCRIBE` (see Section 4.1.7), `REFER` (see Section 4.1.10), and sometimes `MESSAGE` (see Section 4.1.11) methods.

5.2.3 204 No Notification

The `204 No Notification` response [4] is used in response to a `SUBSCRIBE` request that was successful but no notification associated with the request will be sent. This conditional notification can be suppressed using the `Suppress-If-Match` header field.

5.3 Redirection

Redirection class responses are generally sent by a SIP server acting as a redirect server in response to an `INVITE`, as described in Section 3.5.2. A UAS, however, can also send a redirection class response to implement certain types of call forwarding features. There is no requirement that a UAC receiving a redirection response must retry the request to the specified address. The UAC can be configured to automatically generate a new `INVITE` upon receipt of a redirection class response without requiring user intervention. In addition, proxies may also automatically send an `ACK` to a redirect and proxy the `INVITE` to the new location provided in the `Contact` URI of the redirection. To prevent looping, the server must not return any addresses contained in the request `Via` header field, and the client must check the address returned in the `Contact` header field against all other addresses tried in an earlier call attempt. Note that this type of transaction looping is different from request looping.

5.3.1 300 Multiple Choices

This redirection response contains multiple `Contact` header fields, which indicate that the location service has returned multiple possible locations for the `sip` or `sips` URI in the Request-URI. The order of the `Contact` header fields is assumed to be significant. That is, they should be tried in the order in which they were listed in the response.

5.3.2 301 Moved Permanently

This redirection response contains a Contact header field with the new permanent URI of the called party. The address can be saved and used in future INVITE requests.

5.3.3 302 Moved Temporarily

This redirection response contains a URI that is currently valid but is not permanent. As a result, the Contact header field should not be cached across calls unless an Expires header field is present, in which case the location is valid for the duration of the time specified.

5.3.4 305 Use Proxy

This redirection response contains a URI that points to a proxy server who has authoritative information about the calling party. The caller should resend the request to the proxy for forwarding. This response could be sent by a UAS that is using a proxy for incoming call screening. Because the proxy makes the decisions for the UAS on acceptance of the call, the UAS will only respond to INVITE requests that come from the screening proxy. Any INVITE request received directly would automatically receive this response without user intervention.

5.3.5 380 Alternative Service

This response returns a URI that indicates the type of service the called party would like. An example might be a redirect to a voicemail server.

5.4 Client Error

This class of response is used by a server or UAS to indicate that the request cannot be fulfilled as it was submitted. The specific client error response or the presence of certain header fields should indicate to the UAC the nature of the error and how the request can be reformulated. The UAC should not resubmit the request without modifying it based on the response. The same request, however, can be tried in other locations. A forking proxy receipt of a 4xx response does not terminate the search. Typically, client error responses will require user intervention before a new request can be generated.

5.4.1 400 Bad Request

This response indicates that the request was not understood by the server. An example might be a request that is missing required header fields such as To, From, Call-ID, or CSeq. This response is also used if a UAS receives multiple INVITE requests (not retransmissions) for the same Call-ID.

5.4.2 401 Unauthorized

This response indicates that the request requires the user to perform authentication. This response is generally sent by a user agent, since the 407 Proxy Authentication Required (Section 5.4.8) is sent by a proxy that requires authentication. The exception is a registrar server, which sends a 401 Unauthorized response to a REGISTER message that does not contain the proper credentials. An example of this response is:

```
SIP/2.0 401 Unauthorized
Via: SIP/2.0/UDP proxy.globe.org:5060;branch=z9hG4bK2311ff5d.1
 ;received=192.0.2.1;rport=3213
Via: SIP/2.0/UDP 173.23.43.1:5060;branch=z9hG4bK4545
From: <sip:explorer@geographic.org>;tag=341323
To: <sip:printer@maps-r-us.com>;tag=19424103
From: Copernicus <sip:copernicus@globe.org>;tag=34kdilsp3
Call-ID: 1g23hj45m678a7
CSeq: 1 INVITE
WWW-Authenticate: Digest realm="globe.org",
 nonce="8eff88df84f1cec4341ae6e5a359", qop="auth",
 opaque="", stale=FALSE, algorithm=MD5
Content-Length: 0
```

The presence of the required WWW-Authenticate header field is required to give the calling user agent a chance to respond with the correct credentials. A typical authentication exchange using SIP digest is shown in Figure 14.6. Note that the follow-up INVITE request should use the same Call-ID as the original request; the authentication may fail in some cases if the Call-ID is changed from the initial request to the retried request.

5.4.3 402 Payment Required

This response is a placeholder for future definitions in the SIP protocol. It could be used to negotiate call completion charges.

5.4.4 403 Forbidden

This response is used to deny a request without giving the caller any recourse. It is sent when the server has understood the request, found the request to be correctly formulated, but will not service the request. This response is not used when authorization is required.

5.4.5 404 Not Found

This response indicates that the user identified by the `sip` or `sips` URI in the Request-URI cannot be located by the server, or that the user is not currently signed on with the user agent.

5.4.6 405 Method Not Allowed

This response indicates that the server or user agent has received and understood a request but is not willing to fulfill the request. An example might be a `REGISTER` request sent to a user agent. An `Allow` header field (Section 6.1.2) must be present to inform the UAC as to what methods are acceptable. This is different from the case of an unknown method, in which a `501 Not Implemented` response is returned. Note that a proxy will forward request types it does not understand unless the request is targeted to the proxy server (i.e., the Request-URI is the URI of the proxy server).

5.4.7 406 Not Acceptable

This response indicates that the request cannot be processed due to a requirement in the request message. The `Accept` header field in the request did not contain any options supported by the UAS.

5.4.8 407 Proxy Authentication Required

This request sent by a proxy indicates that the UAC must first authenticate itself with the proxy before the request can be processed. The response should contain information about the type of credentials required by the proxy in a `Proxy-Authenticate` header field. The request can be resubmitted with the proper credentials in a `Proxy-Authorization` header field. Unlike in HTTP, this response may not be used by a proxy to authenticate another proxy.

```
SIP/2.0 407 Proxy Authorization Required
Via: SIP/2.0/UDP discrete.sampling.org:5060;branch=z9hG4bK6563
 ;received=65.64.140.198;rport=17234
From: Shannon <sip:shannon@sampling.org>;tag=59204
To: Schockley <sip:shockley@transistor.com>;tag=142334
Call-ID: adf8gasdd7fld
CSeq: 1 INVITE
Proxy-Authenticate: Digest realm=”sampling.org”, qop=”auth”,
 nonce=”9c8e88df84df1cec4341ae6cbe5a359”,
 opaque=””, stale=FALSE, algorithm=MD5
Content-Length: 0
```

5.4.9 408 Request Timeout

This response is sent when an Expires header field is present in an INVITE request and the specified time period has passed. This response could be sent by a forking proxy or a user agent. The request can be retried at any time by the UAC, perhaps with a longer time period in the Expires header field or no Expires header field at all. Alternatively, a stateful proxy can send this response after the request transaction times out without receiving a final response.

5.4.10 409 Conflict

This response code has been removed from RFC 3261 but is defined in RFC 2543. It indicates that the request cannot be processed due to a conflict in the request. This response is used by registrars to reject a registration with a conflicting action parameter.

5.4.11 410 Gone

This response is similar to the 404 Not Found response but contains the hint that the requested user will not be available at this location in the future. This response could be used by a service provider when a user cancels their service.

5.4.12 411 Length Required

This response code has been removed from RFC 3261 but is defined in RFC 2543. This response can be used by a proxy to reject a request containing a message body but no Content-Length header field. A proxy that takes a UDP request and forwards it as a TCP request could generate this response, since the use of Content-Length is more critical in TCP requests. However, the response code is not very useful since a proxy can easily calculate the length of a message body in a UDP request (it is until the end of the UDP packet) but cannot with a stream-oriented transport such as TCP. In this case, a missing Content-Length header field would cause the message body to go on indefinitely, which would generate a 513 Message Too Large response instead of a 411 Length Required.

5.4.13 412 Conditional Request Failed

This response code was added to deal with conditional SIP publications. A 2xx response to a PUBLISH request will contain an entity tag in a SIP-ETag header field. A subsequent publish to update this information will contain this entity tag in a SIP-If-Match header field. Should the entity tag stored by the event state compositor not match the entity tag in the SIP-If-Match header field, the event state compositor returns a 412 Conditional Request Failed response [5]. A publishing UA receiving this response code knows that the stored entity tag is no longer valid. As a result, a publication without the entity tag must be performed.

5.4.14 413 Request Entity Too Large

This response can be used by a proxy to reject a request with a message body that is too large. A proxy suffering congestion could temporarily generate this response to save processing long requests.

5.4.15 414 Request-URI Too Long

This response indicates that the Request-URI in the request was too long and cannot be processed correctly. There is no maximum length defined for a Request-URI in the SIP standard document.

5.4.16 415 Unsupported Media Type

This response sent by a user agent indicates that the media type contained in the INVITE request is not supported. For example, a request for a video conference to a PSTN gateway that only handles telephone calls will result in this response. The response should contain header fields to help the UAC reformulate the request.

5.4.17 416 Unsupported URI Scheme

The 416 Unsupported URI Scheme response is used when a UAC uses a URI scheme in a Request-URI that the UAS does not understand. For example, if a Request-URI contains a secure SIP (sips) scheme that a proxy does not understand, it would return a 416 response. Since all SIP elements must understand the sip scheme, the request should be retried using a sip uri in the Request-URI.

5.4.18 417 Unknown Resource Priority

A request containing Require: resource-priority and an unknown value for Resource-Priority header field will receive the 417 Unknown Resource Priority response [6]. A 417 response may contain an Accept-Resource-Priority header field listing supported values. The request can be retried either without the Require: resource-priority header field or containing a value chosen from the Accept-Resource-Priority in the Resource-Priority header field.

5.4.19 420 Bad Extension

This response indicates that the extension specified in the Require header field is not supported by the proxy or UA. The response should contain a Supported header field listing the extensions that are supported. The UAC could resubmit the same request without the extension in the Require header field or submit the request to another proxy or UA.

5.4.20 421 Extension Required

The 421 Extension Required response indicates that a server requires an extension to process the request that was not present in a Supported header field in the request. The required extension should be listed in a Required header field in the response. The client should retry the request adding the extension to a Supported header field, or try the request at a different server that may not require the extension.

5.4.21 422 Session Timer Interval Too Small

The 422 Session Timer Interval Too Small response [7] is used to reject a request containing a Session-Expires header field (with too short an interval. The ability to reject short durations is important to prevent excessive re-INVITE or UPDATE traffic. The minimum allowed interval is indicated in the required Min-SE header field. The requestor may retry the request without the Session-Expires header field or with a value less than or equal to the specified minimum.

5.4.22 423 Interval Too Brief

The 423 Interval Too Brief response is returned by a registrar that is rejecting a registration request because the requested expiration time on one or more Contacts is too brief. The response must contain a Min-Expires header field listing the minimum expiration interval that the registrar will accept. A client requesting a too short interval can unnecessarily load a registrar server with registration refresh requests. This response allows a registrar to protect against this.

5.4.23 428 Use Identity Header

The 428 Use Identity Header response [8] is used by a UAS that is requiring the use of enhanced SIP identity. The request should be resent with an Identity header field containing a signature over selected parts of the SIP message.

5.4.24 429 Provide Referror Identity

The 429 Provide Referror Identity response [9] is used to request that a Referred-By header field be resent with a valid Referred-By security token. The security token is carried as an S/MIME message body. The recipient of this error message (the UA that received and accepted the REFER) should relay this request back to the originator of the REFER by including it in a NOTIFY. The sender of the REFER can then generate the Referred-By security token and include it in the REFER, which would then be copied into the triggered request.

5.4.25 430 Flow Failed

The 430 Flow Failed response [10] is part of the SIP outbound NAT traversal extension. It is used by an edge proxy server to indicate that it has lost the flow (keep alive failure or timeout) to the UA in the request. The proxy should then resend the request using a different flow-id if available. This approach allows a UA to register through multiple proxy servers as described in Section 10.11.3 with a call flow in Figure 10.11.

5.4.26 433 Anonymity Disallowed

The 433 Anonymity Disallowed response [11] is used to indicate that a request has failed due to anonymity. Anonymity might be because of an invalid URI or display name in the From header field, or by the presence of a Privacy header field requesting privacy. This header field provides a similar service to the anonymous call rejecting service in the PSTN.

5.4.27 436 Bad Identity-Info Header

The 436 Bad Identity-Info response [8] is used to indicate that the URI cannot be accessed from an Identity-Info header field. The Identity-Info header field URI is used to retrieve the certificate associated with the private key that was used to generate the signature in the Identity header field, used for enhanced SIP identity.

5.4.28 437 Unsupported Certificate

The 437 Unsupported Certificate response [8] is used when the certificate obtained using the Identity-Info header field is unable to be used to verify the signature in the Identity header field. This could be because the certificate is expired, not issued by a trusted certificate authority (CA), or for some other reason.

5.4.29 438 Invalid Identity Header

The `438 Invalid Identity Header` response [8] is used to indicate that the identity signature in the identity header field does not match the message. This could indicate an attack, or that an intermediary server such as a Session Border Controller (SBC) or Application Layer Gateway (ALG) has modified the message since the signature was generated.

5.4.30 439 First Hop Lacks Outbound Support

The `439 First Hop Lacks Outbound Support` response [10] is used by a registrar to indicate to a UA attempting to use the SIP outbound extension that the edge proxy does not support the mechanism, and as a result the mechanism cannot be used.

5.4.31 440 Max-Breadth Exceeded

The `440 Max-Breadth Exceeded` response [12] is used to indicate that a forking proxy operation cannot be carried out due to too many concurrent branches. This is part of an extension to address an amplification vulnerability in forking proxy servers. The full description of the attack and how to protect against it is in [12]. When the `Max-Breadth` count goes to zero, the `440 Max-Breadth Exceeded` response is returned.

5.4.32 470 Consent Needed

The `470 Consent Needed` response [13] is used to reject a request sent to a URI list that had at least one element requiring consent. The elements needing consent will be listed in a `Permission-Missing` header field.

5.4.33 480 Temporarily Unavailable

This response indicates that the request has reached the correct destination, but the called party is not available for some reason. The reason phrase should be modified for this response to give the caller a better understanding of the situation. The response should contain a `Retry-After` header indicating when the request may be able to be fulfilled. For example, this response could be sent when a telephone has its ringer turned off, or a “do not disturb” button has been pressed. This response can also be sent by a redirect server.

5.4.34 481 Dialog/Transaction Does Not Exist

This response indicates that a response referencing an existing call or transaction has been received for which the server has no records or state information.

5.4.35 482 Loop Detected

This response indicates that the request has been looped and has been routed back to a proxy that previously forwarded the request. Each server that forwards a request adds a `Via` header with its address to the top of the request. A `branch` parameter is added to the `Via` header, which is a message digest (hash) of the Request-URI, and the `To`, `From`, `Call-ID`, and `CSeq` number. A second part is added to the `branch` parameter if the request is being forked. The `branch` parameter must be checked to allow a request to be routed back to a proxy, provided that the Request-URI has changed. This could happen with a call forwarding feature. In this case, the `Via` headers would differ by having different `branch` parameters.

5.4.36 483 Too Many Hops

This response indicates that the request has been forwarded the maximum number of times as set by the `Max-Forwards` header in the request. This is indicated by the receipt of a `Max-Forwards: 0` header in a request. In the following example, the UAC included a `Max-Forwards: 4` header in the `REGISTER` request. A proxy receiving this request five hops later generates a `483` response:

```
REGISTER sip:registrar.timbuktu.tu SIP/2.0
Via: SIP/2.0/UDP 201.202.203.204:5060;branch=z9hG4bK45347.1
Via: SIP/2.0/UDP 198.20.2.4:6128;branch=z9hG4bK917a4d4.1
Via: SIP/2.0/UDP 18.56.3.1:5060;branch=z9hG4bK7154.1
Via: SIP/2.0/TCP 101.102.103.104:5060;branch=z9hG4bKa5ff4d3.1
Via: SIP/2.0/UDP 168.4.3.1:5060;branch=z9hG4bK676746
To: sip:explorer@geographic.org
From: <sip:explorer@geographic.org>;tag=341323
Call-ID: 67483010384
CSeq: 1 REGISTER
Max-Forwards: 0
Contact: sip:explorer@national.geographic.org
Content-Length: 0

SIP/2.0 483 Too Many Hops
Via: SIP/2.0/UDP 201.202.203.204:5060;branch=z9hG4bK45347.1
Via: SIP/2.0/UDP 198.20.2.4:6128;branch=z9hG4bK917a4d4.1
Via: SIP/2.0/UDP 18.56.3.1:5060;branch=z9hG4bK7154.1
Via: SIP/2.0/TCP 101.102.103.104:5060;branch=z9hG4bKa5ff4d3.1
Via: SIP/2.0/UDP 168.4.3.1:5060;branch=z9hG4bK676746
To: <sip:explorer@geographic.org>;tag=a5642
From: <sip:explorer@geographic.org>;tag=341323
Call-ID: 67483010384
CSeq: 1 REGISTER
Content-Length: 0
```

5.4.37 484 Address Incomplete

This response indicates that the Request-URI address is not complete. This could be used in an overlap dialing scenario in PSTN interworking where digits are collected and sent until the complete telephone number is assembled by a gateway and routed [14]. Note that the follow-up `INVITE` requests may use the same `Call-ID` as the original request. An example of overlap dialing is shown in Figure 5.2.

5.4.38 485 Ambiguous

This request indicates that the Request-URI was ambiguous and must be clarified in order to be processed. This occurs if the username matches a number of registrations. If the possible matching choices are returned in `Contact` header fields, then this response is similar to the `300 Multiple Choices` response. They are slightly different, however, since the `3xx` response returns equivalent choices for the same user, but the `4xx` response returns alternatives that can be different users. The `3xx` response can be processed without human intervention, but this `4xx` response requires a choice by the caller, which is why it is classified as a client error class response. A server configured to return this response must take user registration privacy into consideration; otherwise a vague or general Request-URI could be used by a rogue UA to try to discover `sip` or `sips` URIs of registered users.

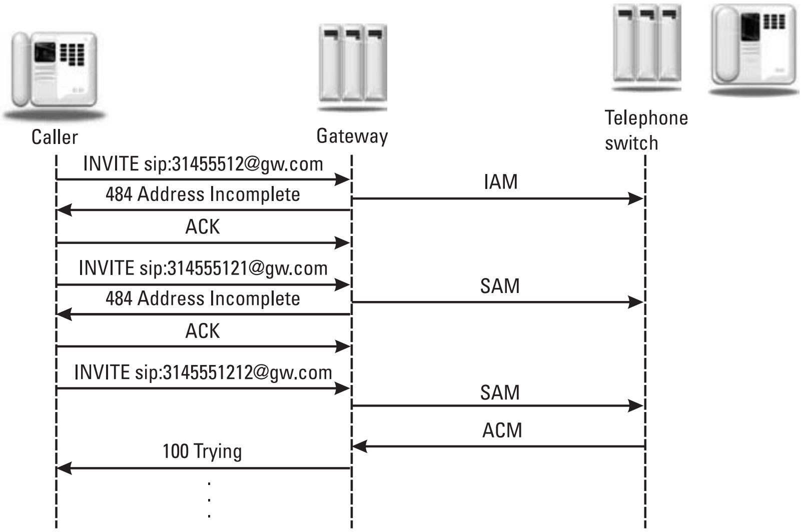


Figure 5.2 Overlap dialing to the PSTN with SIP.

5.4.39 486 Busy Here

This response is used to indicate that the UA cannot accept the call at this location. This is different, however, from the `600 Busy Everywhere` response, which indicates that the request should not be tried elsewhere. In general, a `486 Busy Here` is sent by a UAS unless it knows definitively that the user cannot be contacted. This response is equivalent to the busy tone in the PSTN.

5.4.40 487 Request Terminated

This response can be sent by a UA that has received a `CANCEL` request for a pending `INVITE` request. A `200 OK` is sent to acknowledge the `CANCEL`, and a `487` is sent in response to the `INVITE`.

5.4.41 488 Not Acceptable Here

This response indicates that some aspect of the proposed session is not acceptable and may contain a `Warning` header field indicating the exact reason. This response has a similar meaning to `606 Not Acceptable`, but only applies to one location and may not be true globally as the `606` response indicates.

5.4.42 489 Bad Event

The `489 Bad Event` response [3] is used to reject a subscription request or notification containing an `Event` package that is unknown or not supported by the UAS. The response code is also used to reject a subscription request that does not specify an `Event` package, assuming that the server does not support the PINT protocol (see Section 4.1.7).

5.4.43 491 Request Pending

The `491 Request Pending` response is used to resolve accidental simultaneous re-`INVITE`s by both parties in a dialog. Since both `INVITE`s seek to change the state of the session, they cannot be processed at the same time. While a user agent is awaiting a final response to a re-`INVITE`, any re-`INVITE` request received must be replied to with this response code. This is analogous to the "glare" condition in telephony in which both ends seize a trunk at the same time. The reconsideration algorithm in SIP is for the user agent to generate a delay (randomly selected within a range determined by if the UA sends the initial `INVITE` or not) then retry the re-`INVITE`, assuming that another re-`INVITE` has not been received in the meantime. In this way, one side or the other will "win" the race condition and have the re-`INVITE` processed. An example is shown in Figure 5.3.

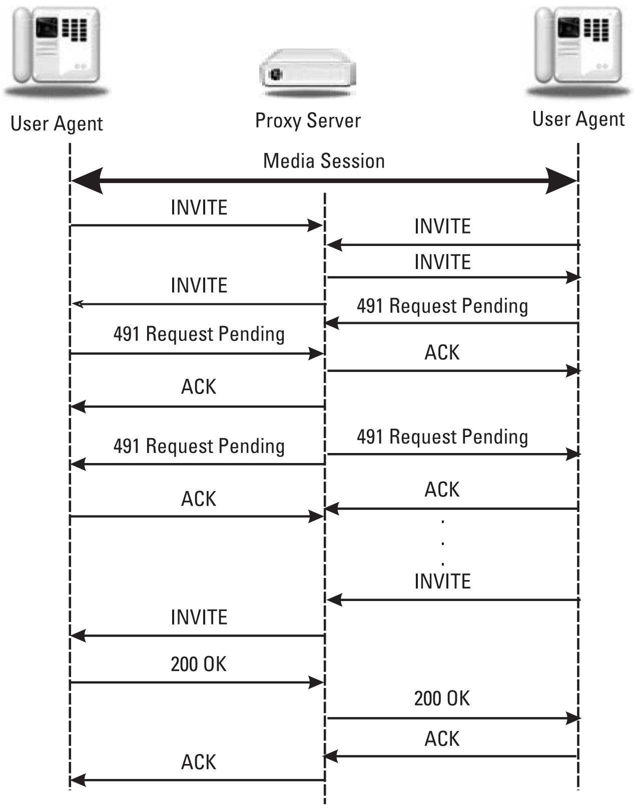


Figure 5.3 Simultaneous re-INVITE resolution example.

5.4.44 493 Request Undecipherable

The `493 Request Undecipherable` response is used when an S/MIME message body cannot be decrypted because the public key is unavailable. If the UAS does not support S/MIME, no message body will be present in the response. If the UAS does support S/MIME, the response will contain a message body containing a public key suitable for the UAC to use for S/MIME encryption. See Section 14.3.4 for more details on S/MIME encryption.

5.4.45 494 Security Agreement Required

The `494 Security Agreement Required` [15] response is used to reject a request containing a `Require: sec-agree` header field as part of the security agreement mechanism.

5.5 Server Error

This class of responses is used to indicate that the request cannot be processed because of an error with the server. The response may contain a `Retry-After` header field if the server anticipates being available within a specific time period. The request can be tried at other locations because there are no errors indicated in the request.

5.5.1 500 Server Internal Error

This server error class response indicates that the server has experienced some kind of error that is preventing it from processing the request. The reason phrase can be used to identify the type of failure. The client can retry the request again at this server after several seconds.

5.5.2 501 Not Implemented

This response indicates that the server is unable to process the request because it is not supported. This response can be used to decline a request containing an unknown method. A proxy, however, will forward a request containing an unknown request method. Thus, a proxy will forward an unknown `SELFDESTRUCT` request, assuming that the UAS will generate this response if the method is not known.

5.5.3 502 Bad Gateway

This response is sent by a proxy that is acting as a gateway to another network, and indicates that some problem in the other network is preventing the request from being processed.

5.5.4 503 Service Unavailable

This response indicates that the requested service is temporarily unavailable. The request can be retried after a few seconds, or after the expiration of the `Retry-After` header field. Instead of generating this response, a loaded server may refuse the connection. This response code is important in that its receipt triggers a new DNS lookup to locate a backup server to obtain the desired service. The set of SIP DNS procedures for locating SIP servers is detailed in [16].

5.5.5 504 Gateway Timeout

This response indicates that the request failed due to a timeout encountered in the other network to which that the gateway connects. It is a server error class response because the call is failing due to a failure of the server in accessing resources outside the SIP network.

5.5.6 505 Version Not Supported

This response indicates that the request has been refused by the server because of the SIP version number of the request. The detailed semantics of this response have not yet been defined because there is only one version of SIP (version 2.0) currently implemented. When additional version numbers are implemented in the future, the mechanisms for dealing with multiple protocol versions will need to be detailed.

5.5.7 513 Message Too Large

The `513 Message Too Large` response is used by a UAS to indicate that the request size was too large for it to process.

5.5.8 580 Preconditions Failure

The `580 Preconditions Failure` response [17] is used to reject an SDP offer in which the required preconditions cannot be met.

5.6 Global Error

This response class indicates that the server knows that the request will fail wherever it is tried. As a result, the request should not be sent to other locations. Only a server that has definitive knowledge of the user identified by the Request-URI in every possible instance should send a global error class response. Otherwise, a client error class response should be sent. A `Retry-After` header field can be used to indicate when the request might be successful.

5.6.1 600 Busy Everywhere

This response is the definitive version of the `486 Busy Here` client error response. If there is a possibility that the call to the specified Request-URI could be answered in other locations, this response should not be sent.

5.6.2 603 Decline

This response has the same effect as the `600 Busy Everywhere` but does not give away any information about the call state of the server. This response could indicate the called party is busy, or simply does not want to accept the call.

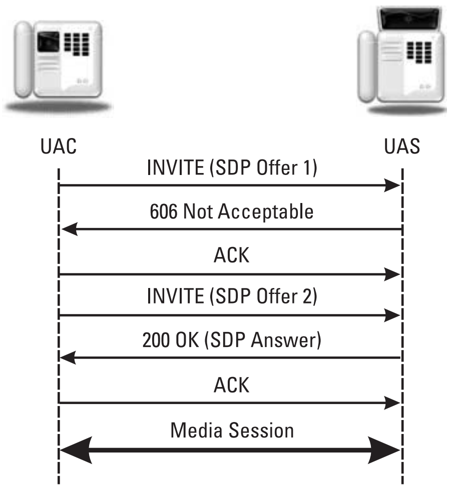


Figure 5.4 Session negotiation with SIP.

5.6.3 604 Does Not Exist Anywhere

This response is similar to the `404 Not Found` response but indicates that the user in the Request-URI cannot be found anywhere. This response should only be sent by a server that has access to all information about the user.

5.6.4 606 Not Acceptable

This response can be used to implement some session negotiation capability in SIP. This response indicates that some aspect of the desired session is not acceptable to the UAS, and as a result, the session cannot be established. The response may contain a `Warning` header field with a numerical code describing exactly what was not acceptable. The request can be retried with different media session information. An example of simple negotiation with SIP is shown in Figure 5.4. If more complicated negotiation capability is required, another protocol should be used.

5.7 Questions

Q5.1 If a UA reboots in the middle of a SIP session and looses all state information, what response is it likely to send to a re-`INVITE` or `BYE` from the other UA?

Q5.2 In terms of the behavior of the UAC originator of the request, explain the difference between a `4xx` and `5xx` response.

Q5.3 Which response would likely be generated to this request?

```
REGISTER sip:registrar.munich.de SIP/2.0
Via: SIP/2.0/UDP 200.201.202.203:5060;branch=z9hG4bKsdus19
Max-Forwards: 70
To: Werner Heisenberg <sip:werner.heisenberg@munich.de>
From: Werner Heisenberg <sip:werner.heisenberg@munich.de>
 ;tag=3431
Call-ID: 7ds376fd4291
CSeq: 1 REGISTER
Contact: <sip:werner.heisenberg@200.201.202.203>;expires=1
Content-Length: 0
```

Q5.4 Give two differences between a `100` response and a `180` response.

Q5.5 What action should be taken by a UAC that receives a `412` response?

Q5.6 How is a `3xx` response different from a `REFER`?

Q5.7 Generate a SIP call flow with two UAs and a proxy where a `408` response is sent.

References

[1] Rosenberg, J., H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261, 2002.

[2] Camarillo, G., et al., "Integrated Services Digital Network (ISDN) User Part (ISUP) to Session Initiation Protocol (SIP) Mapping," RFC 3398, 2002.

[3] Roach, A., "SIP Specific Events," RFC 3265, 2002.

[4] Niemi, A., "An Extension to Session Initiation Protocol (SIP) Events for Conditional Event Notification," draft-ietf-sipcore-subnot-etags-09 (work in progress), April 2009.

[5] Niemi, A., "Session Initiation Protocol (SIP) Extension for Event State Publication," RFC 3903, October 2004.

[6] Schulzrinne, H., and J. Polk, "Communications Resource Priority for the Session Initiation Protocol (SIP)," RFC 4412, February 2006.

[7] Donovan, S., and J. Rosenberg, "Session Timers in the Session Initiation Protocol (SIP)," RFC 4028, April 2005.

[8] Peterson, J., and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)," RFC 4474, August 2006.

[9] Sparks, R., "The SIP Referred-By Mechanism," RFC 3892, September 2004.

[10] Jennings, C., and R. Mahy, "Managing Client Initiated Connections in the Session Initiation Protocol (SIP)," draft-ietf-sip-outbound-20 (work in progress), June 2009.

[11] Rosenberg, J., "Rejecting Anonymous Requests in the Session Initiation Protocol (SIP)," RFC 5079, December 2007.

[12] Sparks, R., et al., "Addressing an Amplification Vulnerability in Session Initiation Protocol (SIP) Forking Proxies," RFC 5393, December 2008.

[13] Rosenberg, J., G. Camarillo, and D. Willis, “A Framework for Consent-Based Communications in the Session Initiation Protocol (SIP),” RFC 5360, October 2008.

[14] Anttalainen, T., *Introduction to Telecommunications Network Engineering,* Norwood, MA: Artech House, 1999.

[15] Arkko, J., et al., “Security Mechanism Agreement for the Session Initiation Protocol (SIP),” RFC 3329, January 2003.

[16] Rosenberg, J., and H. Schulzrinne, “Session Initiation Protocol (SIP): Locating SIP Servers,” RFC 3263, 2002.

[17] Camarillo, G., W. Marshall, and J. Rosenberg, “Integration of Resource Management and Session Initiation Protocol (SIP),” RFC 3312, October 2002.

6

SIP Header Fields

This chapter describes the header fields present in SIP messages. The header fields discussed in this chapter are categorized as request and response, request only, response only, and message body header fields, depending on their usage in SIP. Except as noted, header fields are defined in the SIP specification RFC 3261 [1]. Chapter 7 also lists some special header fields defined for 3GPP IMS and OMA.

SIP header fields in most cases follow the same rules as HTTP header fields [2]. Header fields are defined as `Header: field`, where `Header` is the case-insensitive token (but conventionally lowercase with some capitalization) used to represent the header field name, and `field` is the case-insensitive set of tokens that contain the information. Except when otherwise noted, their order in a message is not important. Header fields can continue over multiple lines as long as the line begins with at least one space or horizontal tab character. Unrecognized header fields are ignored by proxies. Many common SIP header fields have a compact form where the header field name is denoted by a single lower-case character. These header fields are shown in Table 6.1. Header fields can be either end-to-end or hop-by-hop. Hop-by-hop header fields are the only ones that a proxy may insert or, with a few exceptions, modify. Because SIP typically involves end-to-end control, most header fields are end-to-end. The hop-by-hop header fields that may be inserted by a proxy are shown in Table 6.2.

Table 6.1
Compact Forms of SIP Header Fields

Header Field	Compact Form
Accept-Contact	a
Allow-Event	u
Call-ID	i
Contact	m
Content-Encoding	e
Content-Length	l
Content-Type	c
Event	o
From	f
Refer-To	r
Referred-By	b
Reject-Contact	j
Subject	s
To	t
Via	v

6.1 Request and Response Header Fields

This set of header fields can be present in both requests and responses.

6.1.1 Accept

The Accept header field is defined by HTTP [2] and is used to indicate acceptable message Internet media types [3] in the message body. The header field describes media types using the format type/sub-type commonly used in the Internet. If not present, the assumed acceptable message body format is application/sdp. A list of media types can have preferences set using q value parameters. The wildcard "*" can be used to specify all sub-types. Examples are given in Table 6.3.

6.1.2 Accept-Encoding

The Accept-Encoding header field, defined in HTTP [2], is used to specify acceptable message body encoding schemes. Encoding can be used to ensure a SIP message with a large message body fits inside a single UDP datagram. The use of q value parameters can set preferences. If none of the listed schemes are acceptable to the UAC, a 406 Not Acceptable response is returned. If not included, the assumed encoding will be text/plain. Examples include:

Table 6.2
Header Fields That May Be Inserted or Modified by Proxies

Hop-by-Hop Header Fields
Alert-Info
Call-Info
Content-Length
Date
Error-Info
Max-Breadth
Max-Forwards
Organization
Priority
Proxy-Authenticate
Proxy-Authorization
Proxy-Require
Record-Route
Reason
Require
Route
Via
WWW-Authenticate
Security-Client
Security-Verify
Security-Server
Answer-Mode
Priv-Answer-Mode
History-Info
Path
Identity
Identity-Info
P-Asserted-Identity
Reason
Resource-Priority
Auth-Info

Table 6.3
Examples of an `Accept` Header Field

Header Field	Meaning
`Accept: application/sdp`	This is the default assumed even if no `Accept` header field is present.
`Accept: text/*`	Accept all text encoding schemes.
`Accept: application/h.245;q=0.1, application/sdp;q=0.9`	Use SDP if possible, otherwise, use H.245.

```
Accept-Encoding: text/plain

Accept-Encoding: gzip
```

6.1.3 Accept-Language

The Accept-Language header field, defined in HTTP [2], is used to specify preferences of language. The languages specified can be used for reason phrases in responses, informational header fields such as Subject, or in message bodies. The HTTP definition allows the language tag to be made of a primary tag and an optional subtag. This header field could also be used by a proxy to route to a human operator in the correct language. The language tags are registered by IANA, and the primary tag is an ISO-639 language abbreviation. The use of q values allows multiple preferences to be specified. Examples are shown in Table 6.4.

6.1.4 Alert-Info

The Alert-Info header field can be used to provide a "distinctive ring" service. If present in an INVITE, the UAS may use the URI to fetch an alert tone to be used in place of the default alerting tone—that is, it would be rendered to the called party. If present in a 180 Ringing response, the UAC may use the URI to fetch a ring-back tone to be rendered to the calling party. In both uses, the URI is fetched and rendered without user intervention, so careful policy rules are necessary to avoid unwanted sounds and noises being generated.

One use is for a trusted proxy to insert the header field with a local (to the domain of the user agent) URI. This then allows for very simple policy in the user agent in deciding whether or not to render. Another approach is for a URN to be used, which would tell the user agent which service tone to play for the user.

An example is shown here:

```
Alert-Info: <http://www.provider.com/tones/internal_caller.pcm>
```

Table 6.4
Examples of an Accept-Language Header Field

Header Field	Meaning
Accept-Language: fr	French is the only acceptable language.
Accept-Language: en, ea	Acceptable languages include both English and Spanish.
Accept-Language: ea; q=0.5, en ;q=0.9, fr ;q=0.2	Preferred languages are English, Spanish, and French, in that order.

6.1.5 Allow

The Allow header field is used to indicate the methods supported by the UA or proxy server sending the response. The header field must be present in a 405 Method Not Allowed response and should be included in a positive response to an OPTIONS request. Allow is often present in INVITE and 200 OK responses. An example is:

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

6.1.6 Allow-Events

The Allow-Events header field [4] is used to list the event packages that are supported. A UA that supports SIP events will then know that it may send a SUBSCRIBE for that event package. The list of currently defined packages is in Table 4.8. The compact form is u.

Examples are shown here:

```
Allow-Events: dialog

u: conference
```

6.1.7 Answer-Mode

The Answer-Mode header field [5] is used to request an immediate answer (200 OK) to an INVITE. Two values, Manual and Auto, have been defined. The Manual setting is normal behavior while Auto means the request should be accepted immediately without input from the user. A header field parameter require is defined to indicate that if the requested behavior is not permitted, the request should be rejected with a 403 Forbidden response. This extension can be used for a number of features including loopback tests, intercom, and push-to-talk. The Answer-Mode header field can be included in a 200 OK response to an INVITE to indicate how the request was answered. For example, if the require parameter is not present and the call is answered by the user, an Answer-Mode: Manual header field can be included in the 200 OK. The Priv-Answer-Mode is similar but requests a privilege treatment. The SIP option tag answermode is used to indicate support for this extension. Example:

```
Answer-Mode: Auto;require
```

6.1.8 Call-ID

The Call-ID header field is mandatory in all SIP requests and responses. It is part of the dialog used to uniquely identify a call between two user agents. A

Call-ID must be unique across calls, except in the case of a Call-ID in registration requests. All registrations for a user agent should use the same Call-ID. A Call-ID is always created by a user agent and is never modified by a server.

The Call-ID must be a cryptographically random identifier. Some security is provided by the randomness of the Call-ID, because this prevents a third party from *guessing* a Call-ID and presenting false requests. Older UAs generate Call-IDs containing an IP address or host name. However, this is not recommended as it forces topology hiding B2BUAs to rewrite the Call-ID. The compact form of the Call-ID header field is i.

Examples of Call-ID are shown here:

```
Call-ID: 34a5d553192cc35
Call-ID: 44fer23ei4291dekfer34231232
i: 35866383092031257
```

6.1.9 Contact

The Contact header field is used to convey a URI that identifies the resource requested or the request originator, depending on whether it is present in a request or response. Once a Contact header field has been received, the URI can be cached and used for routing future requests within a dialog. For example, a Contact header field in a 200 OK response to an INVITE can allow the acknowledgment ACK message and all future requests during this call to bypass proxies and go directly to the called party. However, the presence of Record-Route header fields in an earlier request or default proxy routing configuration in a user agent may override that behavior. When a Contact URI is used in a Request-URI, all URI parameters are allowed with the exception of the method parameter, which is ignored.

Contact header fields must be present in INVITE requests and 200 OK responses to invitations. In some cases, the Contact URI may not resolve directly to the user agent. For example, a UA behind a firewall ALG will need to use a Contact URI that resolves to the firewall ALG address. Otherwise, the use of the user agent's URI will result in the call failing because of the firewall blocking any direct routed SIP requests. Contact header fields may also be present in 1xx, 2xx, 3xx, and 485 responses. Only in a REGISTER request, a special Contact:*, along with an Expires: 0, header field is used to remove all existing registrations. Examples of Contact header fields in registrations are shown in Table 4.3. Otherwise, wild carding is not allowed. A Contact header field may contain a display name that can be in quotes. If a display name is present, the URI will be enclosed in < >. If any header field parameters are present, the URI will also be enclosed in < > along with any URI parameters, with the header field parameters outside the < >, even if no display name is present.

There are three additional parameters defined for use in `Contact` header fields: `q,` `action`, and `expires`. They are placed at the end of the URI and separated by semicolons. The `q` value parameter is used to indicate relative preference, which is represented by a decimal number in the range 0 to 1. The `q` value is not a probability, and there is no requirement that the `q` values for a given list of `Contact`s add up to 1. (The `action` parameter defined in RFC 2543 has been deprecated and is not used in RFC 3261. It was only used in registration `Contact` header fields, and is used to specify `proxy` or `redirect` operations by the server.) The `expires` parameter indicates how long the URI is valid and is also only used in registrations. The parameter either contains an integer number of seconds or a date in SIP form (see Section 6.1.11). Examples are shown in Table 6.5.

The `Contact` header field may contain a feature tag [6], which can be used to indicate the capabilities of the device identified by the `Contact` URI. For example, the feature tag `isfocus` is used to indicate that the URI in the `Contact` header field is a conference URI, and that the dialog is associates with a focus. A focus is a SIP UA that hosts a particular instance of a conference, called a "bridge" or MCU in other protocols. The presence of the `isfocus` feature tag can be used by a SIP UA that supports advanced conferencing features to invoke certain call control operations [7] or subscribe to the conference package [8]. Section 9.7 has more on SIP conferencing.

Some other common feature tags are listed in Table 6.6. The compact form is `m`.

Table 6.5
Examples of Contact Header Fields

Header Field	**Meaning**
`Contact: sip:bell@telephone.com`	A single SIP URI without a display name.
`Contact: Lentz <h.lentz@petersburg.edu:1234>`	A display name with the URI enclosed in < >; the display name is treated as a token and ignored. Port 1234 is used instead of the default 5060.

`Contact: M. Faraday <faraday@effect.org>,`
`"Faraday" <mailto:faraday@pop.effect.org>`
	Two URIs are listed, the second being a non-SIP URI with a display name enclosed in quotes.

`m: <morse@telegraph.org;transport=tcp>;`
`expires= "Fri, 13, Oct 1998 12:00:00 GMT"`
	The compact form of the header field contains a port number and a URI parameter contained within the < >. An expires header field parameter uses a SIP date enclosed in the quotes.

Table 6.6
Boolean Feature Tags

Feature Tag	Meaning
attendant	Attendant, human or automata
automata	Nonhuman
image	Supports images
message	Supports messaging
text	Supports text media
audio	Supports audio media
video	Supports video media
voicemail	Is a voicemail server
isfocus	Is a focus, a conference server

6.1.10 CSeq

The command sequence CSeq header field is a required header field in every request. The CSeq header field contains a decimal number that increases for each request. Usually, it increases by 1 for each new request, with the exception of CANCEL and ACK requests, which use the CSeq number of the INVITE request to which it refers.

The CSeq count is used by UASs to determine out-of-sequence requests or to differentiate between a new request (different CSeq) or a retransmission (same CSeq). The CSeq header field is used by UACs to match a response to the request it references. For example, a UAC that sends an INVITE request then a CANCEL request can tell by the method in the CSeq of a 200 OK response if it is a response to the invitation or cancellation request. Examples are shown in Table 6.7.

Each user agent maintains its own command sequence number space. For example, consider the case where UA 1 establishes a session to UA 2 and initializes its CSeq count to 1. When user agent 2 initiates a request (such as INVITE or INFO, or even BYE) it will initialize its own CSeq space, totally independent of the CSeq count used by UA 1. The examples of Chapter 16 show this behavior of CSeq.

Table 6.7
CSeq Header Field Examples

Header Field	Meaning
CSeq: 1 INVITE	The command sequence number has been initialized to 1 for this INVITE.
CSeq: 432 REFER	The command sequence number is set to 432 for this REFER request.
CSeq: 6787 INVITE	If this was the first request by the user agent for this dialog, then either the CSeq was initialized to 6787, or the previous request generated for this Call-ID (either an INVITE or another request) would have had a CSeq of 6786 or lower.

6.1.11 Date

The Date header field is used to convey the date when a request or response is sent. The format of a SIP date is based on HTTP dates, but allows only the preferred Internet date standard referenced by RFC 1123 [9]. To keep UA date and time logic simple, SIP only supports the use of the GMT time zone. This allows time entries that are stored in date form rather than second count to be easily converted into delta seconds without requiring knowledge of time zone offsets. Date is included in 200 OK responses to REGISTER requests. This allows UAs to automatically set their date and time.

A Date example is shown here:

```
Date: Fri, 13 Oct 1998 23:29:00 GM
```

6.1.12 Encryption

The Encryption header field was defined in RFC 2543 but is not included in RFC 3261. Instead, encryption using S/MIME is defined as discussed in Chapter 14.

6.1.13 Expires

The Expires header field is used to indicate the time interval in which the request or message contents are valid. When present in an INVITE request, the header field sets a time limit on the completion of the INVITE request. That is, the UAC must receive a final response (non-1xx) within the time period or the INVITE request is automatically canceled with a 408 Request Timeout response. Once the session is established, the value from the Expires header field in the original INVITE has no effect—the Session-Expires header field (Section 6.2.34) must be used for this purpose. When present in a REGISTER request, the header field sets the time limit on the URIs in Contact header fields that do not contain an expires parameter. Table 4.3 shows examples of the Expires header field in registration requests. Expires also is used in SUBSCRIBE requests to indicate the subscription duration. The header field may contain a SIP date or a number of seconds. Examples include:

```
Expires: 60

Expires: Fri, 15 Apr 2000 00:00:00 GMT
```

6.1.14 From

The From header field is a required header field that indicates the originator of the request. It is one of two addresses used to identify the dialog. The From header field contains a URI, but it may not contain the transport, maddr, or ttl URI parameters. A From header field may contain a tag used to identify a particular call. A From header field may contain a display name, in which case the URI is enclosed in < >. If there is both a URI parameter and a tag, then the URI including any parameters must be enclosed in < >. Examples are shown in Table 6.8. A From tag was optional in RFC 2543 but is mandatory to include in RFC 3261.

6.1.15 History Info

The History-Info header field [10] is an extension header field used to capture and convey routing history associated with a SIP request. Information about a request routing can be added whenever a request is retargeted, the Request-URI is rewritten, and so fourth. A Reason header field (Section 6.2.20) is included as a parameter in this header field. Since a request can be forwarded and retargeted multiple times, multiple History-Info header fields can be present (or multiple comma separated History-Info entries). An index parameter is used to keep track of the order of the actions. A common application for History-Info is for voicemail, described in Section 9.4. Below is an example showing three history-info entries, and the escaping of the semicolon (%3B) and equals (%3D) in the Reason header field:

```
History-Info: <sip:UserA@example.com?Reason=SIP%3Bcause%3D302>
 ;index=1.1,
<sip:UserB@example.com?Privacy=history&Reason=SIP%3Bcause%3D486>
 ;index=1.2, <sip:45432@vm.example.com>;index=1.3
```

Table 6.8
Examples of From Header Field

Header Field	Meaning
From: <sip:armstrong@hetrodyne.com> ;tag=3342436	A single SIP URI with a tag.
From: Thomas Edison <sips:edison@electric.com>;tag=532	A secure SIP URI with a display name.
f: "James Bardeen" <sip:555.1313@telephone.com ;transport=tcp>;tag=3a320f03	Using the compact form of the header field, a display name in quotes along with a SIP URI with a parameter inside < >.
From: tel:911	A tel URI without a display name or tag, so no <> is required. Generated by a RFC 2543 UA.

6.1.16 Organization

The Organization header field is used to indicate the organization to which the originator of the message belongs. It can also be inserted by proxies as a message is passed from one organization to another. Like all SIP header fields, it can be used by proxies for making routing decisions and by UAs for making call screening decisions.

An example is:

```
Organization: MCI
```

6.1.17 Path

The Path header field [11] is an optional header field in REGISTER requests. It can be thought of as a Record-Route mechanism for REGISTER requests, which establishes a route set that is valid for the duration of the registration. The Path header field may be inserted by a proxy, which forwards a REGISTER request to a registrar server. The registrar copies the Path header field into the 200 OK response to the REGISTER, which then provides the route set information to the UA that is registering. In a mobile network, the Path header field can be used to discover and inform the UA of the proxies that can be used to populate preloaded Route header fields. The ob URI parameter in a Path header field can be used to indicate to a UA that an edge proxy supports the SIP outbound extension, described in Section 10.11.3.

An example is:

```
Path: <sip:proxy2.another.provider.com;lr;ob>
```

6.1.18 Priv-Answer-Mode

The Priv-Answer-Mode header field is similar to the Answer-Mode header field in that it requests special handling by the recipient of the INVITE. The values Manual and Auto and the require parameter are defined. The difference between Priv-Answer-Mode and Answer-Mode relates to the policy on the UA. For example, an intercom call between an executive and an administrator might use the Answer-Mode header field, and a do-not-disturb setting on the executive's phone could override this feature. However, a building-wide emergency announcement might use the Priv-Answer-Mode header field, which would override the executive's do-not-disturb setting. Example:

```
Priv-Answer-Mode: Auto
```

6.1.19 Record-Route

The Record-Route header field is used to force routing through a proxy for all subsequent requests in a session (dialog) between two UAs. Normally, the presence of a Contact header field allows UAs to send messages directly bypassing the proxy chain used in the initial request (which probably involved database lookups to locate the called party). A proxy inserting its address into a Record-Route header field overrides this and forces future requests to include a Route header field containing the address of the proxy that forces this proxy to be included.

A proxy wishing to implement this inserts the header field containing its own URI, or adds its URI to an already present Record-Route header field. The URI is constructed so that the URI resolves back to the proxy server. The UAS copies the Record-Route header field into the 200 OK response to the request. The header field is forwarded unchanged by proxies back to the UAC. The UAC then stores the Record-Route proxy list plus a Contact header field if present in the 200 OK for use in a Route header field in all subsequent requests. Because Record-Route is bidirectional, messages in the reverse direction will also traverse the same set of proxies. Chapter 16 contains an example of the use of the Record-Route and Route header fields. The lr parameter is new to RFC 3261 and indicates that the proxy server supports "loose routing." Older RFC 2543 compliant proxy servers create Record-Route URIs that instead of the lr parameter often contain the maddr parameter with an address or host that resolves to that proxy server.

Examples are:

```
Record-Route: <sip:proxy1.carrier.com;lr>,
 <sip:firewall33.corporation.com;lr>

Record-Route:<sip:139.23.1.44;lr>
```

6.1.20 Recv-Info

The Recv-Info header field [12] is used to indicate which INFO packages a UA is willing to receive. Defined INFO event packages allow negotiation between UAs of supported applications using INFO. For example:

```
Recv-Info: foo
```

6.1.21 Refer-Sub

The Refer-Sub header field [13] is used to request a particular state for an implicit REFER subscription, or to indicate the state of an existing implicit REFER subscription. When a Refer-Sub: false header field is included in a REFER, the recipient of the REFER is requested not to create an implicit subscription and not to send NOTIFYs about the outcome of the referred operation. If the recipient does this, they return the Refer-Sub: false header field in the 2xx response to the REFER. A value of true or the absence of the Refer-Sub header field in the 2xx response to the REFER means that the implicit subscription has been created. The norefsub option tag indicates that a UA supports this mechanism.

6.1.22 Retry-After

The Retry-After header field is used to indicate when a resource or service may be available again. In 503 Service Unavailable responses, it indicates when the server will be available. In 404 Not Found, 600 Busy Everywhere, and 603 Decline responses, it indicates when the called UA may be available again.

The header field can also be included by proxy and redirect servers in responses if a recent registration was removed with a Retry-After header field indicating when the user may sign on again. The contents of the header field can be either an integer number of seconds or a SIP date. A duration parameter can be used to indicate how long the resource will be available after the time specified. Examples of this header field are shown in Table 6.9.

6.1.23 Subject

The optional Subject header field is used to indicate the subject of the media session. It can be used by UAs for simple call screening. The contents of the header field can also be displayed during alerting to aid the user in deciding whether to accept the call. The compact form of this header field is s. Some examples are:

```
Subject: More good info about SIP

s: Are you awake, yet??
```

Table 6.9

Examples of a Retry-After Header Field

Header Field	Meaning
Retry-After: 3600	Request can be retried again in 1 hour.
Retry-After: Sat, 21 May 2000 08:00:00 GMT	Request can be retried after the date listed.
Retry-After: 3600	Request can be tried after 1 hour.
Retry-After: Mon, 29 Feb 2000 13:30:00 GMT;duration=1800	Request can be retried after the specified date for 30 minutes.

6.1.24 Supported

The Supported header field is used to list one or more options implemented by a UA or server. It is typically included in responses to OPTIONS requests. If no options are implemented, the header field is not included. If a UAC lists an option in a Supported header field, proxies or UASs may use the option during the call. If the option must be used or supported, the Require header field is used instead. Table 6.10 shows the current set of defined feature tags.

An example of the header field is:

```
Supported: rel100
```

Table 6.10

Extension Feature Tags

Tag	Meaning
answermode	Answer-Mode and Priv-Answer-Mode header fields
early-session	Support of early-session content disposition
eventllist	Resourcelist extension
events	SIP Events [4]
from-change	From and To URI changes in a dialog [14]
gruu	Globally Routable User Agent URI [15]
histinfo	History-Info header field [10]
ice	Support for ICE [16]
join	Join call control primitive [17]
multiple-refer	REFER with resource-list [18]
norefsub	REFER without implicit subscription and NOTIFYS [19]
outbound	SIP outbound NAT traversal feature [20]
path	Path header field [21]
precondition	SIP preconditions [22]
pref	Caller prefs [23]
privacy	Privacy mechanisms [24]
recipient-list-invite	INVITE with resource list [25]
recipient-list-message	MESSAGE with resource list [26]
recipient-list-subscribe	SUBSCRIBE with resource list [27]
rel100	Reliable provisional response (PRACK) support [28]
replaces	Replaces call control primitive [29]
resource-priority	Resource-Priority header field [30]
sdp-anat	Alternative address for NAT (deprecated) [31]
sec-agree	SIP security agreement mechanism [32]
tdialog	Target-dialog header field [33]
timer	Session timer feature [34]

6.1.25 Timestamp

The Timestamp header field is used by a UAC to mark the exact time a request was generated in some numerical time format. A UAS must echo the header field in the response to the request and may add another numerical time entry indicating the amount of delay. Unlike the Date header field, the time format is not specified. The most accurate time format should be used, including a decimal point. Examples are shown in Table 6.11. The default value of 500 ms is used for T1.. Timestamp is not commonly used.

6.1.26 To

The To header field is a required header field in every SIP message used to indicate the recipient of the request. Any responses generated by a UA will contain this header field with the addition of a tag. (Note that an RFC 2543 client will typically only generate a tag if more than one Via header field is present in the request.) Any response generated by a proxy must have a tag added to the To header field. A tag added to the header field in a 200 OK response is used throughout the call and incorporated into the dialog. The To header field URI is never used for routing—the Request-URI is used for this purpose. An optional display name can be present in the header field, in which case the SIP URI is enclosed in < >. If the URI contains any parameters or username parameters, the URI must be enclosed in < > even if no display name is present. The compact form of the header field is t. Examples are shown in Table 6.12.

6.1.27 User-Agent

The User-Agent header field is used to convey information about the UA originating the request. Based on the HTTP header field of the same name [2], it can contain manufacturer information, software version, or comments. The field may contain multiple tokens, with the ordering assumed to be from most general to most specific. This information can be used for logging or for generating a specific response for a specific UA. For security reasons, this header field may be suppressed. For example, an attacker probing a UA for vulnerabilities could learn the particular vendor and software load that is susceptible to a particular attack

Table 6.11
Examples of a Timestamp Header Field

Header Field	Meaning
`Timestamp: 235.15`	Client has stamped a start time for the request.
`Timestamp: 235.15 .95`	This header field from the response has the delay time added by the server.

Table 6.12
Examples of a To Header Field

Header Field	Meaning
`To: sip:babage@engine.org;tag=2443a8f7`	A single SIP URI with a tag and without a display name.
`To: Thomas Edison <sips:edison@elec.com>`	A display name is used, so the sips URI is enclosed in < >.
`t: "Jim B." <brattain@bell.org>`	A display name in quotes along with a SIP URI enclosed within < >.

`To: <+1-314-555-1212@carrier.com`
`;user=phone>;tag=8f7f7ad6675`
	Both a URI parameter and tag are used, so URI is enclosed in <>. Note that no line breaks are permitted in a URI.

and reuse that attack against other UAs that have the same software as identified by the User-Agent header field.

Examples include:

```
User-Agent: Acme/v2.2

User-Agent: Carrier/Beta
```

6.1.28 Via

The required Via header field is used to record the SIP route taken by a request and is used to route a response back to the originator. A UA generating a request records its own address in a Via header field. While the ordering of most SIP header fields is not significant, the Via header fields order is significant because it is used to route responses. A proxy forwarding the request adds a Via header field containing its own address to the *top* of the list of Via header fields. A proxy adding a Via header field always includes a branch tag containing a cryptographic hash of the To, From, Call-ID header fields and the Request-URI. A proxy or UA generating a response to a request copies all the Via header fields from the request *in order* into the response, then sends the response to the address specified in the top Via header field. A proxy receiving a response checks the top Via header field to ensure that it matches its own address. If it does not, the response has been misrouted and should be discarded. The top Via header field is then removed, and the response forwarded to the address specified in the next Via header field.

Via header fields contain protocol name, version number, and transport (SIP/2.0/UDP, SIP/2.0/TCP, etc.) and may contain port numbers and parameters such as received, rport, branch, maddr, and ttl. A received tag is added to a Via header field if a UA or proxy receives the request from a different address than that specified in the top Via header field. If an rport tag is included in a Via in a request, a proxy will insert the port the request was received on and use this port for routing the response. This indicates that a NAT or firewall proxy is in the message path. If present, the received and or rport tags are used in response routing. (The hidden parameter, deprecated in RFC 3261, was used to indicate that the Via header field has been encrypted.) A branch parameter is added to Via header fields by UAs and proxies, which is computed as a hash function of the Request-URI, and the To, From, Call-ID, and CSeq number. A second part is added to the branch parameter if the request is being forked as shown in Figure 3.4. The maddr and ttl parameters are used for multicast transport and have a similar meaning as the equivalent SIP URI parameters. The compact form of the header field is v. Examples are given in Table 6.13.

6.2 Request Header Fields

This set of header fields can only be present in a request.

6.2.1 Accept-Contact

The Accept-Contact [23] header field specifies to which URIs the request may be proxied. Some additional parameters are also defined for Contact header fields such as media, duplex, and language. This header field is part of the caller preferences extensions to SIP, which have been defined to give some control to the caller in the way a proxy server processes a call. The compact form is a.

Some examples follow:

Table 6.13
Examples of a Via Header Field

Header Field	Meaning
Via: SIP/2.0/UDP 100.101.102.103 ;branch=z9hG4bK776a	IPv4 address using unicast UDP transport and assumed port of 5060.
Via: SIP/2.0/TCP cube451.office.com:60202 ;branch=z9hG4bK776a	Domain name using TCP transport and port number 60202.
Via: SIP/2.0/UDP 120.121.122.123 ;branch= z9hG4bK56a234f3.1	Proxy added Via header field with branch.
v: SIP/2.0/UDP proxy.garage.org ;branch= z9hG4bK3423423a3.3	Compact form with domain name using UDP; third search location of forking proxy.
Via: SIP/2.0/TCP 192.168.1.2;received=12.4.5.50 ;rport=42212 ;branch=z9hG4bK334	IPv4 address is nonglobally unique. Request has been forwarded through a NAT, which has created a mapping with a different IP address and port (mapped address is 12.4.5.50:42212).

```
Accept-Contact: *;language=en

a: *;media=video
```

6.2.2 Authorization

The `Authorization` header field is used to carry the credentials of a UA in a request to a server. It can be sent in reply to a `401 Unauthorized` response containing challenge information, or it can be sent first without waiting for the challenge if the form of the challenge is known (e.g., if it has been cached from a previous call). The authentication mechanism for HTTP digest is described in Section 14.4.1. Examples are shown in Table 6.14.

6.2.3 Call-Info

The `Call-Info` header field is included in a request by a UAC or proxy to provide a URI with information relating to the session setup. It may be present in an `INVITE`, `OPTIONS`, or `REGISTER` request. The header field parameter `purpose` indicates the purpose of the URI and may have the values `icon`, `info`, `card`, or other IANA registered tokens.

An example follows:

```
Call-Info: <http://www.code.com/my_picture.jpg>;purpose=icon
```

6.2.4 Event

The `Event` header field is used in a `SUBSCRIBE` (see Section 4.1.7) or `NOTIFY` (see Section 4.1.8) methods to indicate which event package is being used by the method. In a `SUBSCRIBE`, it lists the event package to which the client would like to subscribe. In a `NOTIFY`, it lists the event package that the notification contains state information about. Currently defined event packages are listed in Table 4.8. The compact form is `o`.

Table 6.14
Example of an `Authorization` Header Field

Header Field	Meaning

`Authorization: Digest username="Cust1",`
`realm="company.com",`
`nonce="9c8e88df84f1cec4341ae6e5a359",`
`opaque="", uri="sip:user2@company.com",`
`response="e56131d19580cd833064787ecc"`
	This HTTP digest authorization header field contains the credentials of Cust1; the `nonce` was supplied by the SIP server located at the URI specified. The `response` contains the hashed username and password. No `opaque` string is present.

An example follows:

```
Event: dialog
```

```
o: refer
```

6.2.5 Hide

The Hide header field was defined in RFC 2543 but has been deprecated (removed) from RFC 3261. It was intended to be used by UAs or proxies to request that the next hop proxy encrypts the Via header fields to hide message routing path information. Encrypted Via headers were identified with the hidden Via parameter. However, the security provided and the mechanism requiring next hop trust made the value of this header field minimal.

6.2.6 Identity

The Identity header field [35] is part of the enhanced SIP identity extension, described in more detail in Chapter 14. It is inserted by a proxy server in a forwarded request after the request has been authenticated. The header field contains a digital signature over certain parts of the SIP message and the entire message body. The header field is used to certify the identity in the From header field by a proxy in the domain.

6.2.7 Identity-Info

The Identity-Info header field [35] is part of the enhanced SIP identity extension, and is used to convey a URI for the certificates containing the public key of the signing proxy. The alg parameter indicates the algorithm used to generate the signature in the Identity header field. In this example, the certificate is available from the https URI and the algorithm used is RSA with SHA-1:

```
Identity-Info: <https://atlanta.example.com/atlanta.cer>
 ;alg=rsa-sha1
```

6.2.8 In-Reply-To

The In-Reply-To header field is used to indicate the Call-ID that this request references or is returning. For example, a missed call could be returned with a new INVITE and the Call-ID from the missed INVITE copied into the In-Reply-To header field. This allows the UAS to determine that this is not an unsolicited call, which could be used to override call screening logic, for example. Examples of this header field are as follows:

```
In-Reply-To: a8-43-73-ff-43@company.com

In-Reply-To: 12934375@persistance.org, 12934376@persistance.org
```

6.2.9 Info-Package

The Info-Package header field [12] is a mandatory header field in an INFO method used to indicate which INFO package is associated with this message. For example:

```
Info-Package: foo
```

6.2.10 Join

The Join header field [17] is used in an INVITE to request that the dialog (session) be joined with an existing dialog (session). The parameters of the Join header field identify the dialog by the Call-ID, To tag, and From tag in a similar way to the Replaces header field.

If the Join header field references a point-to-point dialog between two user agents, the Join header field is effectively a request to turn the call into a conference call. If the dialog is already part of a conference, the Join header field is a request to be added into the conference. An example call flow is shown in Figure 6.1 in which a two-way call is turned into a conference call.

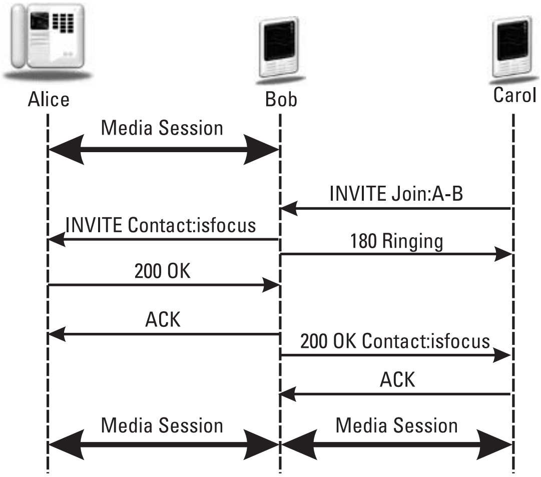


Figure 6.1 Use of Join to create a conference call.

If the dialog referenced in the `Join` header field does not exist, a `481 Call/Dialog Does Not Exist` response is returned. A UA supporting `Join` should indicate this in all requests with a `Supported: join` header field.

In the following example, the dialog:

```
To: <sip:moe@example.org>;tag=42312
From: <sip:larry@server.org>;tag=3443212
Call-ID: 243134123234
```

would match the Join header field:

```
Join: 243134123234;to-tag=42312;from-tag=3443212
```

6.2.11 Priority

The `Priority` header field is used by a UAC to set the urgency of a request. Defined values are `non-urgent, normal, urgent`, and `emergency`. This header field could be used to override screening or by servers in load-shedding mechanisms. Because this header field is set by the UA, it may not be possible for a carrier network to use this field to route emergency traffic, for example. An example is:

```
Priority: emergency
```

6.2.12 Privacy

The `Privacy` header field [24] is used by a UAC to request varying degrees and types of privacy. Currently defined tags include `critical, header, id, session, user`, or `none`.

An example follows:

```
Privacy: header;user;critical
```

6.2.13 Proxy-Authorization

The `Proxy-Authorization` header field is to carry the credentials of a UA in a request to a server. It can be sent in reply to a `407 Proxy Authentication Required` response containing challenge information, or it can be sent first without waiting for the challenge if the form of the challenge is known (e.g., if it has been cached from a previous call). The authentication mechanism for SIP digest is described in Section 14.4.1. A proxy receiving a request containing a `Proxy-Authorization` header field searches for its own realm, and if found it processes the entry. If the credentials are correct, any remaining entries are kept in the request when it is forwarded to the next proxy. An example of this is in Figure 6.2.

Examples are shown in Table 6.15.

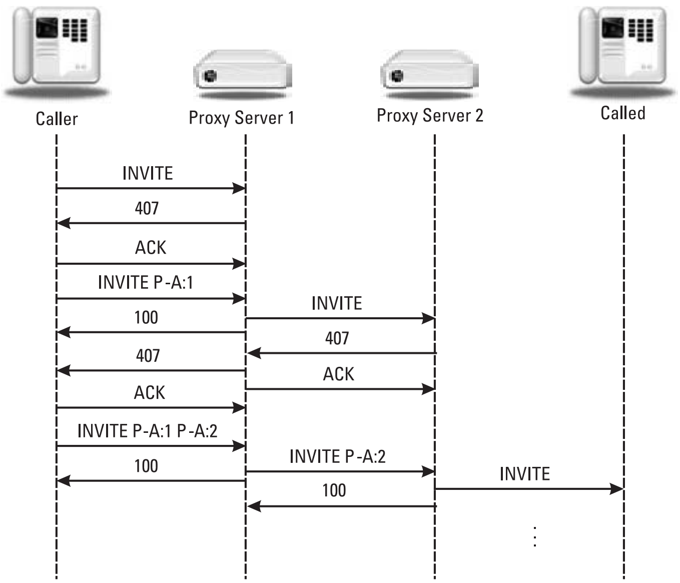


Figure 6.2 Multiproxy authentication example. Note: In this figure, P-A stands for the `Proxy-Authorization` header field.

Table 6.15
Example of a `Proxy-Authorization` Header Field

Header Field	Meaning

`Proxy-Authorization: Digest`
`username="Customer1",`
`realm="company.com",`
`nonce="9c8e88df84f1cec4341ae6e5a359",`
`opaque="", uri="sip:user@company.com",`
`response="e56131d19580cd833064787ecc"`
	This digest authorization header field contains the credentials of Customer1. The `nonce` was supplied by the SIP server located at the URI specified. The `response` contains the hashed user name and password; no `opaque` string is present.

6.2.14 Proxy-Require

The `Proxy-Require` header field is used to list features and extensions that a UA requires a proxy to support in order to process the request. A `420 Bad Extension` response is returned by the proxy listing any unsupported feature in an `Unsupported` header field. Because proxies by default ignore header fields and features they do not understand, the use of a `Proxy-Require` header field is needed for the UAC to be certain that the feature is understood by the proxy. If the support of this option is desired but not required, it is listed in a `Supported` header field instead. An example is:

```
Proxy-Require: timer
```

6.2.15 P-OSP-Auth-Token

The P-OSP-Auth-Token header field [36] is used to transport an Open Settlements Protocol (OSP) token [37] with a SIP INVITE request. A gateway or proxy server receiving a token can verify the token and use this information about accepting the INVITE or rejecting the call. This approach is suitable for a clearinghouse model of VoIP carrier interconnection.

An example is:

```
P-OSP-Auth-Token: 3b8a40c10b4930ff19a85766c15182a34048d9398b834d6
 ;realm="carrier.com"
```

6.2.16 P-Asserted-Identity

The P-Asserted-Identity header field [38] is used between trusted intermediaries (proxies) to assert the identity of a UA that has been authenticated using some means such as those described in Chapter 14. A UA receiving a request from a proxy that it trusts will typically render the value in a P-Asserted-Identity header field to the user as a "Verified Caller ID" as opposed to a From header value which is unverified. A proxy receiving a P-Asserted-Identity from another proxy that it does not trust will remove the header field.

An example is:

```
P-Asserted-Identity: <sip:user@example.com>
```

6.2.17 P-Preferred-Identity

The P-Preferred-Identity header field [38] is used by a UA to tell a trusted intermediary which identity it would prefer to be asserted on its behalf when more than one identity is associated with that UA.

An example is:

```
P-Preferred-Identity: <sip:alternate@example.com>
```

6.2.18 Max-Breadth

The Max-Breadth header field [39] is part of the solution to an amplification attack on forking proxy servers. This header field is inserted by proxy servers and decremented based on the breadth (number of concurrent branches) of a forking operation. When the Max-Breadth count goes to zero, the 440 Max-Breadth Exceeded response is returned. Example:

```
Max-Breadth: 32
```

6.2.19 Max-Forwards

The `Max-Forwards` header field is used to indicate the maximum number of hops that a SIP request may take. The value of the header field is decremented by each proxy that forwards the request. A proxy receiving the header field with a value of zero discards the message and sends a `483 Too Many Hops` response back to the originator.

`Max-Forwards` is a mandatory header field in requests generated by an RFC 3261 compliant UA. However, an RFC 2543 UA generally will not include the header field. The suggested initial value is 70 hops.

An example is:

```
Max-Forwards: 10
```

6.2.20 Reason

The `Reason` header field [40] can be used in `BYE` and `CANCEL` messages to indicate the reason why the session or call attempt is being terminated. It can carry a SIP response code or a Q.850 cause value (from an ISUP REL message, for example).

For example, a forking proxy could include the following header field in a `CANCEL` sent to a leg after one leg has answered the call.

```
Reason: SIP ;cause=200 ;text="Call completed elsewhere"
```

6.2.21 Refer-To

The `Refer-To` header field [41] is a required header field in a `REFER` request, which contains the URI or URL resource that is being referenced. It may contain any type of URI from a `sip` or `sips` to a `tel` URI to an `http` or `mailto` URI. For a `sip` or `sips` URI, the URI may contain a method or escaped header fields. For example, the following `Refer-To` header field (where a line break has been added for display):

```
Refer-To: <sip:UserC@client.anywhere.com?Replaces=
sdjfdjfskdf@there.com%3Bto-tag%3D5f35a3%3Bfrom-tag%3D8675309>
```

contains an escaped `Replaces` header field. The resulting `INVITE` message generated by this `Refer-To` header field would have a Request-URI of

```
sip:UserC@client.anywhere.com
```

and a

```
Replaces: sdjfdjfskdf@there.com;to-tag=5f35a3;from-tag=8675309
```

header field. Note that the characters ";" and "=" are replaced by their hex equivalents %3B and %3D. In the next example, the header field containing a method

```
Refer-To: <sip:UserC@client.anywhere.com?method=SUBSCRIBE>
```

would cause a SUBSCRIBE request to be sent instead of an INVITE, which is the default method if none is present. An example of the Refer-To header field in compact form with an HTTP URL is:

```
r: <http://www.artech-house.com>
```

6.2.22 Referred-By

The Referred-By header field [42] is an optional header field in a REFER request and a request triggered by a REFER. It provides the recipient of a triggered request with information that the request was generated as a result of a REFER and the originator of the REFER. This information can be presented to the user or have policy applied in deciding how the UA should handle the request.

As this header field could be modified or fabricated, a more secure usage involves the addition of a Referred-By security token. The token is carried as a message body whose content id (cid) is indicated in the Referred-By header field. The token is an S/MIME signature over a message/sipfrag, which contains, at a minimum, the From, Date, Call-ID, Refer-To, and Referred-By header fields from the REFER request. An unsigned Referred-By header field may be rejected with a request that the Referred-By security token be included using the 429 Provide Referror Identity response code (see Section 5.4.24). The compact form is b:

```
Referred-By: <sip:user@host.com>

b: <sips:friend@neighbor.org>
```

6.2.23 Reply-To

The Reply-To header field is used to indicate a sip or sips URI, which should be used in replying to this request. Normally, this URI is present in the From header field (the Contact is not used as it is only assumed valid for the duration of the dialog). However, in some cases, the From cannot be populated with this information, so the URI in this header field should be used instead of the From URI.

An example is:

```
Reply-To: <sip:l.tolstoy@stpetersburg.ru>
```

6.2.24 Replaces

The Replaces header field [29] is used in SIP call control applications. A UA in an established dialog receiving another INVITE with a Replaces header field that matches the existing dialog must accept the INVITE, terminate the existing dialog with a BYE, and transfer all resources and state from the existing dialog to the newly established dialog.

If the Replaces header field matches no dialog, the INVITE must be rejected with a 481 Dialog Does Not Exist response.

In addition, Replaces has one application in pending dialogs. A UAC that has sent an INVITE but has not yet received a final response may receive an INVITE containing a Replaces header field that matches the pending INVITE. The UAC must terminate the pending dialog with a CANCEL (and be prepared to send an ACK and BYE if a 200 OK eventually arrives) and accept the new INVITE.

For an INVITE containing both a Require: replaces and Replaces header field, this results in the return of one of the following set of responses:

- 200 (if a match is found);
- 481 (if no match is found);
- 420 (if Replaces is not supported).

Figure 6.3 shows a call flow using Replaces to implement a feature called "call pickup." The early parameter means that the replacement should only be done if the dialog is in an early state; if the dialog has transitioned to a confirmed state, the INVITE should be rejected. Figure 4.9 shows the use of Replaces in an "attended transfer example."

This example Replaces header field:

```
Replaces: 3232904875945;to-tag=34314;from-tag=2343
```

would match the dialog identified by:

```
To: <sip:moe@example.org>;tag=34314
From: <sip:larry@server.org>;tag=2343
Call-ID: 3232904875945
```

6.2.25 Reject-Contact

The Reject-Contact [23] header field specifies the URIs to which the request may not be proxied. Some additional parameters are also defined for Contact header fields such as media, duplex, and language. This header field, along with

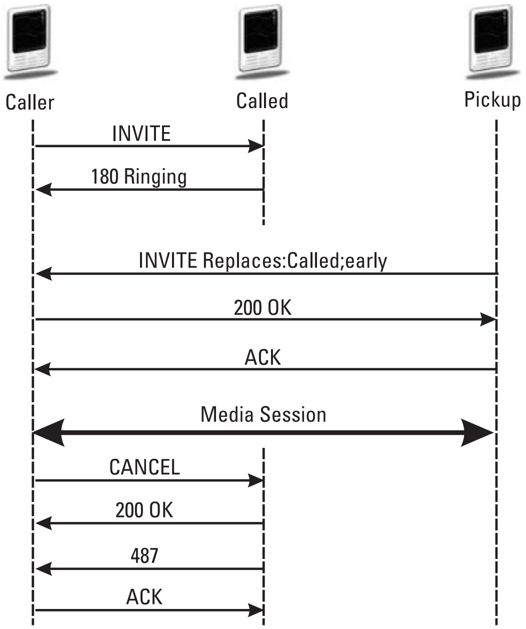


Figure 6.3 Call flow showing call pickup using `Replaces`.

`Accept-Contact` and `Request-Disposition` are part of the SIP caller preferences extensions. The compact form is `j`. Examples include:

```
Reject-Contact: sip:admin@boss.com

j: *;media=video
```

6.2.26 Request-Disposition

The `Request-Disposition` [23] header field can be used to request servers to either proxy, redirect, or initiate serial or parallel (forking) searches. An example is:

```
Request-Disposition: redirect
```

6.2.27 Require

The `Require` header field is used to list features and extensions that a UAC requires a UAS to support in order to process the request. A `420 Bad Extension` response is returned by the UAS listing any unsupported features in an `Unsupported` header field. If support or use of a feature is desirable but not required, the `Supported` header field is used instead. See Table 6.10 for a list of feature tags.

An example is:

```
Require: rel100
```

6.2.28 Resource-Priority

The `Resource-Priority` header field [30] is used to convey resource priority in a SIP request. It has been used to interwork with PSTN pre-emption and priority queuing protocols. The header field contains namespace and a resource priority value, separated by a dot. Multiple values can be included separated by commas. Defined namespaces for `Resource-Priority` are included in Table 6.16. `Resource-Priority` namespaces supported can be listed in an `Accept-Resource-Priority` header field. The `resource-priority` option tag is used to indicate support for this mechanism.

An example is:

```
Resource-Priority: dsn.flash
```

6.2.29 Response-Key

The `Response-Key` header field was defined in RFC 2543 but was deprecated in RFC 3261 along with all PGP-based encryption in favor of S/MIME encryption.

6.2.30 Route

The `Route` header field is used to provide routing information for requests. RFC 3261 introduces two types of routing: strict and loose routing, which have similar meaning as the IP routing modes of the same name. In strict routing, a proxy must use the first URI in the `Route` header field to rewrite the Request-URI,

Table 6.16
`Resource-Priority` Namespaces

Value	Name
`dsn`	Defense switched network
`dsrn`	Defense RED switched network
`q735`	Commercial implementation of DSN multilevel precedence and preemption (MLPP)
`ets`	Emergency telecommunications service
`wps`	Wireless priority service

which is then forwarded. In loose routing, a proxy does not rewrite the Request-URI, but either forwards the request to the first URI in the Route header field or to another loose routing element. In loose routing, the request must route through every server in the Route list (but may also route through other servers) before it may be routed based on the Request-URI. In strict routing, the request must only route through the set of servers in the Route header field with the Request-URI being rewritten at each hop. A proxy or UAC can tell if the next element in the route set supports loose routing by the presence of an lr parameter. An example is:

```
Route: <sip:proxy@example.com;lr>
```

Chapter 16 contains an example of the use of the Record-Route and Route header fields. Examples of Route header fields constructed from the example Record-Route header fields in Section 6.1.19 are:

```
Route: <sip:firewall33.corporation.com;lr>, <sip:proxy1.carrier.com;lr>

Route: <sip:139.23.1.44;lr>
```

6.2.31 RAck

The RAck header field [10] is used within a response to a PRACK request to reliably acknowledge a provisional response that contained an RSeq header field. The RAck header field echoes the CSeq and the RSeq from the provisional response. The reliable sequence number is incremented for each response sent reliably. A call flow is shown in Figure 4.11. An example is:

```
RAck: 8342523 13 INVITE
```

6.2.32 Security-Client

The Security-Client header field [32] is part of the SIP security agreement extension used to negotiate security settings between a UA and a proxy server. The Security-Client header field is used by a UA to declare the mechanisms it supports in a SIP request. UAs and servers compare the security mechanisms in the Security-Client header field with the mechanisms in the Security-Server header field (see Section 6.3.9) and choose the common mechanism with the highest preference value. The SIP option tag sec-agree can be used in Supported, Require, and Proxy-Require header fields. For example:

```
Security-Client: digest
```

6.2.33 Security-Verify

The Security-Verify header field [32] is part of the SIP security agreement extension, used to negotiate security settings between a UA and a proxy server. The Security-Verify header field is used by a UA to echo the mechanisms received in a Security-Server header (see Section 6.3.9) field from a server. If the request is sent with integrity protection, a proxy and user can detect a bid-down attack in the security agreement negotiation. For example:

```
Security-Client: digest
```

6.2.34 Session-Expires

The Session-Expires header field [21] is used to specify the expiration time of the session. To extend the session, either UA can send a re-INVITE or UPDATE with a new Session-Expires header field. At the expiration of the interval in the Session-Expires, either UA may send a BYE and call-stateful proxies may destroy any state information. A proxy may shorten the expiration time by reducing the interval in the header field as it proxies the request. A UAS confirms the session timer by including the Session-Expires header field in the response to the request. A UAS may also shorten the interval by reducing the interval. An example is:

```
Session-Expires: 3600
```

6.2.35 SIP-If-Match

The SIP-If-Match header field [45] is part of the SIP publication mechanism. It is included in a PUBLISH request meant to refresh, modify, or remove previously published state. The header field contains the entity tag of the state information that was returned in a SIP-ETag (Section 6.3.12) header field in a 2xx response to an earlier PUBLISH. If the entity-tag is no longer valid (i.e. the state information has expired, been deleted, replaced, or lost), the server will return a 412 Conditional Request Failed response (Section 5.4.13). See Figure 8.5 for a call flow. Example:

```
SIP-If-Match: 73ikd0Kw3e1D0ds
```

6.2.36 Subscription-State

The Subscription-State header field [3] is a required header field in a NOTIFY request. It indicates the current state of the subscription. Values defined include active, pending, or terminated. Additional parameters include expires, reason, and retry-after. Values defined for the reason parameter include deactivated, giveup, probation, noresource, rejected, and timeout.

An example is:

```
Subscription-State: terminated ;reason=rejected
```

6.2.37 Suppress-If-Match

The Suppress-If-Match header field [46] is an extension for conditional notification. Normally, a SUBSCRIBE sent to refresh a subscription will always generate a NOTIFY, even if no state information has changed since the last notification. If a SIP-ETag was included in a previous notification, the entity tag can be included in the Suppress-If-Match header field and included in a refresh SUBSCRIBE. If the state has not changed, a 204 No Notification response is sent and no NOTIFY will be sent. For example:

```
Suppress-If-Match: Lek31sd
```

6.2.38 Target-Dialog

The Target-Dialog header field [33] is used for authenticating out of dialog SIP requests. A common use case is for the authorization of out-of-dialog REFERs in call transfer scenarios. The header field contains the dialog identifier of the other dialog, which includes the Call-ID, local, and remote tags.

For example:

```
Target-Dialog: 3847349833;to-tag=434232;from-tag=33424212
```

6.2.39 Trigger-Consent

The Trigger-Consent header field provided [43] is used in resource lists requests to trigger consent lookups. For example:

```
Trigger-Consent: sip:alice@atlanta.example.com
```

6.3 Response Header Fields

These header fields are present only in responses.

6.3.1 Accept-Resource-Priority

The Accept-Resource-Priority header field [30] is used to indicate which Resource-Priority namespaces are supported by the UA. It can be included in a `200 OK` to an `OPTIONS` or in a `417 Unknown Resource Priority` response. An example is:

```
Accept-Resource-Priority: ets
```

6.3.2 Authentication-Info

The `Authentication-Info` header field can be inserted in responses when performing mutual authentication using HTTP Digest. In normal HTTP Digest as described in Section 14.4.1, the server challenges the client to provide a shared secret, which the client then provides in a repeat of the request containing an `Authorization` or `WWW-Authenticate` header field. For mutual authentication, the server would then provide an `Authentication-Info` header field containing either a next `nonce` or a response in an `rspauth` parameter. The response auth digest is calculated by the server from the SIP response using the same algorithm as the successful request authentication and same shared secret (client's username and password). In this way, the server proves that it also knows the client's secret, providing mutual authentication. The credentials are carried in the `rspauth` parameter in the header field. Few SIP implementations support this header field.

An example is:

```
Authentication-Info: rspauth="9105jr98li459jgfp"
```

6.3.3 Error-Info

The `Error-Info` header field is used in failure response to convey more information about an error. A UAC receiving the header field in a failure response may fetch and render the URI to the user. The header field can be used to give the client the choice of how the error can be presented to the user. For example, a client with a graphical interface will likely display the reason phrase on the response, which should provide very specific information about the failure. However, an audio-only UA does not have this capability (although a text-to-speech synthesizer could be used to provide this capability). Instead, an audio-only UA could fetch the URI and play the resulting audio stream to the user.

If the URI is a `sip` or `sips` URI, the UA may treat the `Error-Info` as a `Contact` in a redirection response, which would result in a SIP session established to play the recording.

An example is:

```
Error-Info: <sip:recording5@announcementsrus.com>
```

6.3.4 Flow-Timer

The Flow-Timer header field [20] is part of the SIP outbound extension described in Section 10.11.3. The Flow-Timer header field is used by a registrar to tell a UA after how many seconds the server will consider the registration flow dead if no keep alive is sent by the UA. For example:

```
Flow-Timer: 120
```

6.3.5 Min-Expires

The Min-Expires header field is used in a 423 Interval Too Brief response (Section 5.4.22) from a registrar rejecting a REGISTER request in which one or more Contacts have an expiration time that is too short. The header field contains an integer number of seconds that represents the minimum expiration interval that the registrar will accept. A client receiving this header field can update the expiration intervals of the registration request accordingly and resend the REGISTER request.

An example is:

```
Min-Expires: 1200
```

6.3.6 Min-SE

The Min-SE header field [34] is a required header field in a 422 Session Timer Interval Too Small response (Section 5.4.21). The response may also be present in an INVITE or UPDATE containing a Session-Expires header field. It contains an integer number of seconds.

An example is:

```
Min-SE: 480
```

6.3.7 Permission-Missing

The Permission-Missing header field [43] is part of the SIP consent framework. It is used in a 470 Consent Needed response to indicate the URIs to which the relay does not have permission to forward the request. An example of an instant message relay is given in Section 8.5.4. For example:

```
Permission-Missing: sip:voynitch@yale.edu
```

6.3.8 Proxy-Authenticate

The `Proxy-Authenticate` header field is used in a `407 Proxy Authentication Required` authentication challenge by a proxy server to a UAC. It contains the nature of the challenge so that the UAC may formulate credentials in a `Proxy-Authorization` header field in a subsequent request. Examples are shown in Table 6.17.

6.3.9 Security-Server

The `Security-Server` header field [32] is part of the SIP security agreement extension, used to negotiate security settings between a UA and a proxy server. The `Security-Server` header field is used by a server to declare the mechanisms it supports in a `494 Security Agreement Required` header field. UAs and servers compare the security mechanisms in the `Security-Client` header field (see Section 6.2.32) with the mechanisms in the `Security-Server` header field and choose the common mechanism with the highest preference value. The `Security-Server` header field may also be present in a `421 Extension Required` response if the server requires this mechanism. The SIP option tag `sec-agree` can be used in `Supported`, `Require`, and `Proxy-Require` header fields. Example:

```
Security-Server: tls;q=0.5, digest; q=0.4, ipsec-ike;q=0.1
```

6.3.10 Server

The `Server` header field is used to convey information about the UAS generating the response. The use and contents of the header field are similar to the `User-Agent` header field in Section 6.1.27. An example is:

```
Server: Dotcom/B3
```

6.3.11 Service-Route

The `Service-Route` header field [44] can be used in a `2xx` response to a `REGISTER` request. It can be used by a registrar server to provide to the registering UA URIs to include in a preloaded `Route` header field in future requests. The `Service-`

Table 6.17
Example of Proxy-Authenticate Header Field

Header Field	Meaning

`Proxy-Authenticate: Digest`
`realm="example.com",`
`nonce="9c8e88df84f1cec4341ae6e5a359",`
`opaque="", stale=FALSE, algorithm=MD5`
	HTTP digest challenge header field.

Route URIs are only valid for the duration of the registration and should be updated when the registration is refreshed.

An example is:

```
Service-Route: <sip:proxy23.service.provider.com;lr>
```

6.3.12 SIP-ETag

The SIP-ETag header field [45] is part of the SIP publication mechanism. The SIP-ETag header field is returned in a 2xx response to a PUBLISH request. It contains an entity tag uniquely identifying the state information that has been processed. This entity tag can then be used to do conditional publications on this data including refreshing, modifying, and removing, as described in Section 5.4.13. For example:

```
SIP-ETag: 34dw9qFl
```

6.3.13 Unsupported

The Unsupported header field is used to indicate features that are not supported by the server. The header field is used in a 420 Bad Extension response to a request containing an unsupported feature listed in a Require header field. Because multiple features may have been listed in the Require header field, the Unsupported header field indicates all the unsupported features—the rest can be assumed by the UAC to be supported. See Table 6.8 for a list of feature tags.

An example is:

```
Unsupported: rel100
```

6.3.14 Warning

The Warning header field is used in a response to provide more specific information than the response code alone can convey. The header field contains a three-digit warning code, a warning agent that indicates what server inserted the header field, and warning text enclosed in quotes used for display purposes. Warning codes in the 1xx and 2xx range are specific to HTTP [2]. The SIP standard defines 12 new warning codes in the 3xx class. The breakdown of the class is shown in Table 6.18. The complete set of defined warning codes is listed in Table 6.19. Warning is not commonly implemented.

Examples are:

```
Warning: 302 proxy "Incompatible transport protocol"
```

Table 6.18
SIP Warning Codes

Warning Code Range	Error Type
`30x, 31x, 32x`	SDP keywords
`33x`	Network services
`34x, 35x, 36x`	Reserved for future use
`37x`	QoS parameters
`38x`	Reserved
`39x`	Miscellaneous

Table 6.19
SIP Warning Code List

Warning Code	Description
`300`	Incompatible network protocol
`301`	Incompatible network address formats
`302`	Incompatible transport protocol
`303`	Incompatible bandwidth units
`304`	Media type not available
`305`	Incompatible media format
`306`	Attribute not understood
`307`	Session description parameter not understood
`330`	Multicast not available
`331`	Unicast not available
`370`	Insufficient bandwidth
`399`	Miscellaneous warning

```
Warning: 305 room132.hotel.com:5060 "Incompatible media type"
```

6.3.15 WWW-Authenticate

The `WWW-Authenticate` header field is used in a `401 Unauthorized` authentication challenge by a UA or registrar server to a UAC. It contains the nature of the challenge so that the UAC may formulate credentials in a `Proxy-Authorization` header field in a subsequent request. SIP supports HTTP digest authentication mechanisms. Examples are shown in Table 6.20.

6.3.16 RSeq

The `RSeq` header field [28] is used in provisional (`1xx` class) responses to `INVITEs` to request reliable transport. The header field may only be used if the `INVITE` request contained the `Supported: rel100` header field. If present in a provisional

Table 6.20
Example of WWW-Authenticate Header Field

Header Field	Meaning

`WWW-Authenticate: Digest`
`realm="example.com",`
`nonce="9c8e88df84f1cec4341ae6e5a359",`
`opaque="", stale=FALSE, algorithm=MD5`
	HTTP digest challenge.

response, the UAC should acknowledge receipt of the response with a PRACK method, as described in Section 4.1.13. The RSeq header field contains a reliable sequence number that is an integer randomly initialized by the UAS. Each subsequent provisional response sent reliably for this dialog will have a monotonically increasing RSeq number. The UAS will retransmit a reliably sent response until a PRACK is received with a RAck containing the reliable sequence number and CSeq.

An example is:

```
RSeq: 23452
```

6.4 Message Body Header Fields

These header fields contain information about the message body.

6.4.1 Content-Encoding

The Content-Encoding header field is used to indicate that the listed encoding scheme has been applied to the message body. This allows the UAS to determine the decoding scheme necessary to interpret the message body. Multiple listings in this header field indicate that multiple encodings have been used in the sequence in which they are listed. Only encoding schemes listed in an Allow-Encoding header field may be used. The compact form is e. Examples include:

```
Content-Encoding: text/plain

e: gzip
```

6.4.2 Content-Disposition

The Content-Disposition header field is used to describe the function of a message body. Defined values include session, icon, alert, and render. The value session indicates that the message body contains information to describe a media session. The value render indicates that the message body should be displayed or otherwise rendered for the user. If a message body is present in a request or a 2xx response without a Content-Disposition, the function is assumed to be session. For all other response classes with message bodies, the default function is render. An example is:

```
Content-Function: session
```

6.4.3 Content-Language

The Content-Language header field [2] is used to indicate the language of a message body. It contains a language tag, which identifies the language.

```
Content-Language: en
```

6.4.4 Content-Length

The Content-Length is used to indicate the number of octets in the message body. A Content-Length: 0 indicates no message body. As described in Section 2.4.2, this header field is used to separate multiple messages sent within a TCP stream. If not present in a UDP message, the message body is assumed to continue to the end of the datagram. If not present in a TCP message, the message body is assumed to continue until the connection is closed. The Content-Length octet count does not include the CRLF that separates the message header fields from the message body. It does, however, include the CRLF at the end of each line of the message body. An example octet calculation is in Chapter 2. The Content-Length header field is not a required header field to allow dynamically generated message bodies where the Content-Length may not be known a priori. The compact form is l. Examples include:

```
Content-Length: 0

l: 287
```

6.4.5 Content-Type

The Content-Type header field is used to specify the Internet media type [3] in the message body. Media types have the familiar form of type/sub-type. If this header field is not present, application/sdp is assumed. If an Accept header field was present in the request, the response Content-Type must contain a listed type, or a 415 Unsupported Media Type response must be returned. The compact form is c. Specific MIME types that are commonly used are listed in Table 6.21, and Tables 8.4 and 8.9 list common MIME types for presence and instant messaging.

Content indirection [50] can be used to provide a URI in place of an actual MIME message body. An example is:

Table 6.21
Common Content-Types Present in SIP Requests and Responses

Content-Type	Use
`application/sdp`	SDP in `INVITE`, `ACK`, or `UPDATE` requests [47]
`message/sipfrag`	SIP fragment in `NOTIFY` in refer subscription [48]
`application/xml+dialog`	XML dialog [24]
`application/xml+conf`	XML conference info [25]
`text/plain`	Plain text
`text/html`	HTML text
`application/isup`	Encapsulated ISUP in `INVITE`, `BYE`, or `INFO` [49]

```
Content-Type: message/external-body; access-type="URL";
 URL="http://www.example.com/"
```

The compact form is `c`. Examples are:

```
Content-Type: application/sdp

c: text/html
```

6.4.6 MIME-Version

The `MIME-Version` header field is used to indicate the version of MIME protocol used to construct the message body. SIP, like HTTP, is not considered MIME-compliant because parsing and semantics are defined by the SIP standard, not the MIME specification [51]. Version 1.0 is the default value. An example is:

```
MIME-Version: 1.0
```

6.5 Questions

Q6.1 Show a `200 OK` response to an `OPTIONS` that provides the maximum amount of information about the capabilities of the UA.

Q6.2 Generate a call flow that shows the use of the `Reason` header field.

Q6.3 Give three examples of header fields that a proxy might remove from a request before forwarding. Explain why each header field would be removed.

Q6.4 Give two examples of header fields that a proxy must modify when forwarding a request.

Q6.5 Which three header fields can be present in a `REFER` request but not in other methods?

Q6.6 Which header fields would a topology hiding element likely need to modify or remove from a request?

Q6.7 A single header field is modified by a UAC after receiving a `410` response, after which the request receives a `200` response. Which header field was modified?

Q6.8 Which two header fields contain SIP entity tags? How are they typically used?

Q6.9 If a B2BUA between two UAs modified the `Call-ID` header field during a call setup, which header fields might fail to work properly?

Q6.10 Explain which features the `Answer-Mode` header field can be used to implement.

References

[1] Rosenberg, J., H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261, June 2002.

[2] Fielding, R., et al., "Hypertext Transfer Protocol—HTTP/1.1," RFC 2616, June 1999.

[3] Postel, J., "Media Type Registration Procedure," RFC 1590, 1994.

[4] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification," RFC 3265, 2002.

[5] Willis, D., and A. Allen, "Requesting Answering Modes for the Session Initiation Protocol (SIP)," RFC 5373, November 2008.

[6] Rosenberg, J., H. Schulzrinne, and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)," RFC 3840, August 2004.

[7] Johnston, A., and O. Levin, "Session Initiation Protocol (SIP) Call Control—Conferencing for User Agents," RFC 4579, August 2006.

[8] Rosenberg, J., H. Schulzrinne, and O. Levin, "A Session Initiation Protocol (SIP) Event Package for Conference State," RFC 4575, August 2006.

[9] Braden, R., "Requirements for Internet Hosts: Application and Support," RFC 1123, 1989.

[10] Barnes, M., "An Extension to the Session Initiation Protocol (SIP) for Request History Information," RFC 4244, November 2005.

[11] Willis, D., and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts," RFC 3327, December 2002.

[12] Burger, E., H. Kaplan, and C. Holmberg, "Session Initiation Protocol (SIP) INFO Method and Package Framework," draft-ietf-sip-info-events-03 (work in progress), January 2009.

[13] Levin, O., “Suppression of Session Initiation Protocol (SIP) REFER Method Implicit Subscription,” RFC 4488, May 2006.

[14] Elwell, J., “Connected Identity in the Session Initiation Protocol (SIP),” RFC 4916, June 2007.

[15] Rosenberg, J., “Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP),” draft-ietf-sip-gruu-15 (work in progress), October 2007.

[16] Rosenberg, J., “Indicating Support for Interactive Connectivity Establishment (ICE) in the Session Initiation Protocol (SIP),” draft-ietf-sip-ice-option-tag-02 (work in progress), June 2007.

[17] Mahy, R., and D. Petrie, “The Session Initiation Protocol (SIP) Join Header,” RFC 3911, October 2004.

[18] Camarillo, G., et al., “Referring to Multiple Resources in the Session Initiation Protocol (SIP),” RFC 5368, October 2008.

[19] Levin, O., “Suppression of Session Initiation Protocol (SIP) REFER Method Implicit Subscription,” RFC 4488, May 2006.

[20] Jennings, C., and R. Mahy, “Managing Client Initiated Connections in the Session Initiation Protocol (SIP),” draft-ietf-sip-outbound-20 (work in progress), June 2009.

[21] Willis, D., and B. Hoeneisen, “Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts,” RFC 3327, 2002.

[22] Camarillo, G., W. Marshall, and J. Rosenberg, “Integration of Resource Management and Session Initiation Protocol (SIP),” RFC 3312, October 2002.

[23] Rosenberg, J., H. Schulzrinne, and P. Zyzivat, “Caller Preferences for the Session Initiation Protocol (SIP),” RFC 3841, August 2004.

[24] Peterson, J., “A Privacy Mechanism for the Session Initiation Protocol,” RFC 3323, November 2002.

[25] Camarillo, G., and A. Johnston, “Conference Establishment Using Request-Contained Lists in the Session Initiation Protocol (SIP),” RFC 5366, October 2008.

[26] Garcia-Martin, M., and G. Camarillo, “Multiple-Recipient MESSAGE Requests in the Session Initiation Protocol (SIP),” RFC 5365, October 2008.

[27] Camarillo, G., A. Roach, and O. Levin, “Subscriptions to Request-Contained Resource Lists in the Session Initiation Protocol (SIP),” RFC 5367, October 2008.

[28] Rosenberg, J., and H. Schulzrinne, “Reliability of Provisional Responses in Session Initiation Protocol (SIP),” RFC 3262, 2002.

[29] Mahy, R., B. Biggs, and R. Dean, “The Session Initiation Protocol (SIP) Replaces Header,” RFC 3891, September 2004.

[30] Schulzrinne, H., and J. Polk, “Communications Resource Priority for the Session Initiation Protocol (SIP),” RFC 4412, February 2006.

[31] Camarillo, G., and J. Rosenberg, "Usage of the Session Description Protocol (SDP) Alternative Network Address Types (ANAT) Semantics in the Session Initiation Protocol (SIP)," RFC 4092, June 2005.

[32] Arkko, J., et al., "Security Mechanism Agreement for the Session Initiation Protocol (SIP)," RFC 3329, January 2003.

[33] Rosenberg, J., "Request Authorization Through Dialog Identification in the Session Initiation Protocol (SIP)," RFC 4538, June 2006.

[34] Donovan, S., and J. Rosenberg, "Session Timers in the Session Initiation Protocol (SIP)," RFC 4028, April 2005.

[35] Peterson, J., and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)," RFC 4474, August 2006.

[36] Johnston, A., et al., "Session Initiation Protocol Private Extension for an OSP Authorization Token," IETF Internet-Draft, Work in Progress, February 2003.

[37] European Telecommunications Standards Institute, "Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON); Open Settlement Protocol (OSP) for Inter-Domain Pricing, Authorization, and Usage Exchange," Technical Specification 101 321, Version 2.1.0.

[38] Jennings, C., J. Peterson, and M. Watson, "Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity Within Trusted Networks," RFC 3325, 2002.

[39] Sparks, R., et al., "Addressing an Amplification Vulnerability in Session Initiation Protocol (SIP) Forking Proxies," RFC 5393, December 2008.

[40] Schulzrinne, H., D. Oran, and G. Camarillo, "The Reason Header Field for the Session Initiation Protocol (SIP)," RFC 3326, 2002.

[41] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method," RFC 3515, 2003.

[42] Sparks, R., "The SIP Referred-By Mechanism," RFC 3892, September 2004.

[43] Rosenberg, J., G. Camarillo, and D. Willis, "A Framework for Consent-Based Communications in the Session Initiation Protocol (SIP)," RFC 5360, October 2008.

[44] Willis, D., and B. Hoeneisen, "Session Initiation Protocol Extension Header Field for Service Route Discovery During Registration," RFC 3608, October 2003.

[45] Niemi, A., "Session Initiation Protocol (SIP) Extension for Event State Publication," RFC 3903, October 2004.

[46] Niemi, A., "An Extension to Session Initiation Protocol (SIP) Events for Conditional Event Notification," draft-ietf-sipcore-subnot-etags-02 (work in progress), April 2009.

[47] Handley, M., V. Jacobson, and C. Perkins, "SDP: Session Description Protocol," RFC 4566, July 2006.

[48] Sparks, R., "Internet Media Type message/sipfrag," RFC 3420, November 2002.

[49] Zimmerer, E., et al., "MIME Media Types for ISUP and QSIG Objects," RFC 3204, December 2001.

[50] Burger, E., “A Mechanism for Content Indirection in Session Initiation Protocol (SIP) Messages,” RFC 4483, May 2006.

[51] Freed, M., and N. Borenstein, “Multipurpose Internet Mail Extensions (MIME). Part One: Format of Internet Message Bodies,” RFC 2045, 1996.

7

Wireless, Mobility, and IMS

The mobility features of SIP have been discussed in earlier chapters. In this chapter, those aspects will be explored further. The Third Generation Partnership Project (3GPP) [1] has adopted SIP as their call signaling protocol in the Intelligent Multimedia Core Subsystem (IMS). Since then, a number of extensions and usage documents have been authored describing 3GPP's planned use of SIP; these will be discussed in this chapter. The future direction of wireless SIP will be discussed in Section 7.5.

7.1 IP Mobility

There are a number of different types of mobility that will be discussed in this chapter, including terminal mobility, personal mobility, and service mobility [2]. Terminal mobility is the ability of an end device to maintain its connection to the Internet as it moves around and possibly changes its point of connection. Personal mobility is the ability to have a constant address (identifier) across a number of devices. Finally, service mobility is the ability of a user to keep the same services when mobile.

Terminal mobility can be addressed by Mobile IP [3], which has been standardized in the IETF. It allows a terminal to keep the same IP address when roaming as it does in its home network. While roaming, the terminal is reachable by a "care of" address, which is registered in the home network. Packets destined for the roaming terminal are received in the home network, then tunneled to the terminal at the "care of" address, as shown in Figure 7.1. Mobile IP has the advantage of hiding the mobile nature of the terminal from layer 3 protocols and above. These protocols can then be used without any modification.

For example, a TCP connection can be maintained since the terminal appears to have a constant IP address. However, Mobile IP has the disadvantage

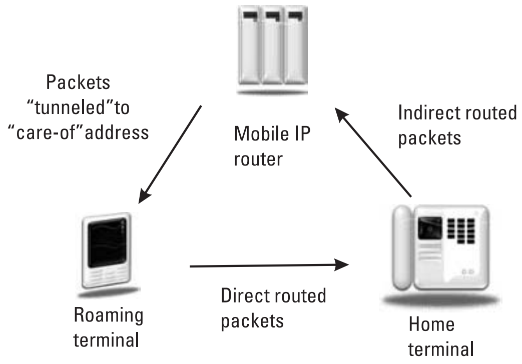


Figure 7.1 Triangular routing of IP packets in Mobile IP.

that incoming packets are not routed directly, and as a result, most efficiently. This results in increased packet latency. While this is not a problem for nonreal-time services such as Web browsing or e-mail, real-time media transport has critical requirements on packet latency. A solution that has been proposed [2] is based on the fact that a protocol such as SIP already has mobility support built in. In addition, SIP is capable of handling some of the terminal mobility aspects at the application layer. This can result in more efficient RTP packet routing and better efficiency (as the additional overhead of IP packet encapsulation required by Mobile IP are avoided).

HIP (Host Identity Protocol) mobility, described in Section 15.5 is another way for mobility to be supported. HIP does not use the triangular routing of Figure 7.1.

The result is that mobile SIP devices are utilizing two different architectures. One is based on the use of Mobile IP and the other utilizes the built-in mobility support in SIP. The next sections will discuss these two approaches. SIP is ideally suited to provide both personal and service mobility.

7.2 SIP Mobility

Personal mobility is the ability to have a constant identifier across a number of devices. A `sip` or `sips` URI has exactly this property and is fundamentally supported by SIP. SIP can also support service mobility (the ability of a user to keep the same services when mobile) although some conventions and extensions have been proposed that provide this in certain architectures.

Basic personal mobility is supported by SIP using the `REGISTER` method, which allows a mobile device to change its IP address and point of connection to the Internet and still be able to receive incoming calls. As discussed in Chapters 2 and 4, registration in SIP temporarily binds a user's AOR (Address of Record)

URI with a `Contact` URI of a particular device. As a device's IP address changes, registration allows this information to be automatically updated in the SIP network. An end device can also move between service providers using multiple layers of registrations, in which a registration is actually performed with a `Contact` as an address of record with another service provider. For example, consider the UA in Figure 7.2, which has temporarily acquired a new SIP URI with a new service provider. (The reasons for doing so could include security, NAT/firewall traversal, or local policy.) The UA then performs a double registration as shown in Figure 7.2. The first registration is with the new service provider, which binds the `Contact` URI of the device with the new service provider's AOR URI. The second `REGISTER` request is routed back to the original service provider and provides the new service provider's AOR as the `Contact` URI. As shown later in the call flow, when a request comes in to the original service provider's network the `INVITE` is redirected to the new service provider who then routes the call to the user.

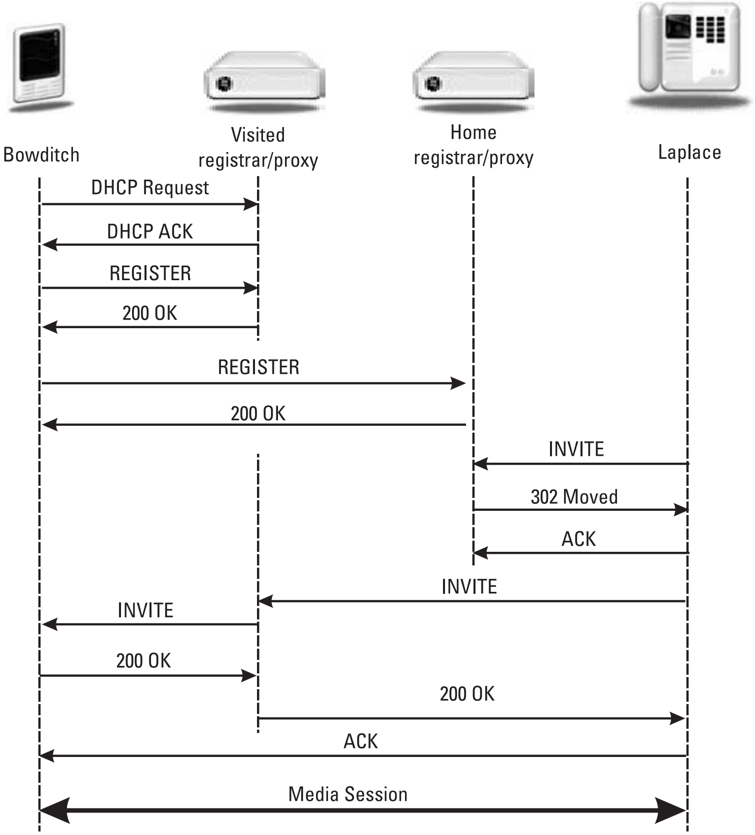


Figure 7.2 Precall mobility using SIP `REGISTER`.

For the first registration message containing the device URI would be:

```
REGISTER sip:registrar.capetown.org SIP/2.0
Via: SIP/2.0/TLS 128.5.2.1:5060;branch=z9hG4bK382112
Max-Forwards: 70
To: Nathaniel Bowditch <sip:bowditch321@capetown.org>
From: Nathaniel Bowditch <sip:bowditch321@capetown.org>
 ;tag=887865
Call-ID: 54-34-19-87-34-ar-gr
CSeq: 3 REGISTER
Contact: <sip:nat@128.5.2.1>
Content-Length: 0
```

and the second registration message with the roaming URI would be:

```
REGISTER sip:registrar.salem.ma.us SIP/2.0
Via: SIP/2.0/TLS 128.5.2.1:5060;branch=z9hG4bK1834
Max-Forwards: 70
To: Nathaniel Bowditch <sip:n.bowditch@salem.ma.us>
From: Nathaniel Bowditch <sip:n.bowditch@salem.ma.us>
 ;tag=344231
Call-ID: 152-45-N-32-23-W3-45-43-12
CSeq: 6421 REGISTER
Contact: <sip:bowditch321@capetown.org>
Content-Length: 0
```

The first `INVITE` that is depicted in Figure 7.2 would be sent to `sip:n.bowditch@salem.ma.us`; the second `INVITE` would be sent to `sip:bowditch321@capetown.org`, which would be forwarded to `sip:nat@128.5.2.1`. It reaches Bowditch and allows the session to be established. Both registrations would need to be periodically refreshed.

A disadvantage of this approach is that SIP does not currently have a means to obtain a local URI. This would have to be done using a non-SIP method such as a Web page signup, which would be coupled with the proper authentication, authorization, and accounting mechanisms.

An optimization of this is for the local registrar to forward the registration information on the roaming UA back to the home registrar. This has been proposed in the IETF [4] but has yet to be adopted or standardized. No changes to SIP messages are required, just a convention adopted by registrars to recognize a roaming registration and take the appropriate action. It is possible that these conventions may become standardized if the authentication and accounting systems needed to properly process such registrations are standardized in the future.

During a session, a mobile device may also change IP address as it switches between one wireless network and another (the Mobile IP protocol is not assumed—it will be discussed in the next section). Basic SIP supports this scenario as well, since a re-`INVITE` in a dialog can be used to update the `Contact` URI and change media information in the SDP. This is shown in the call flow of Figure 7.3. Here, Bowditch detects a new wireless network, uses DHCP to acquire a

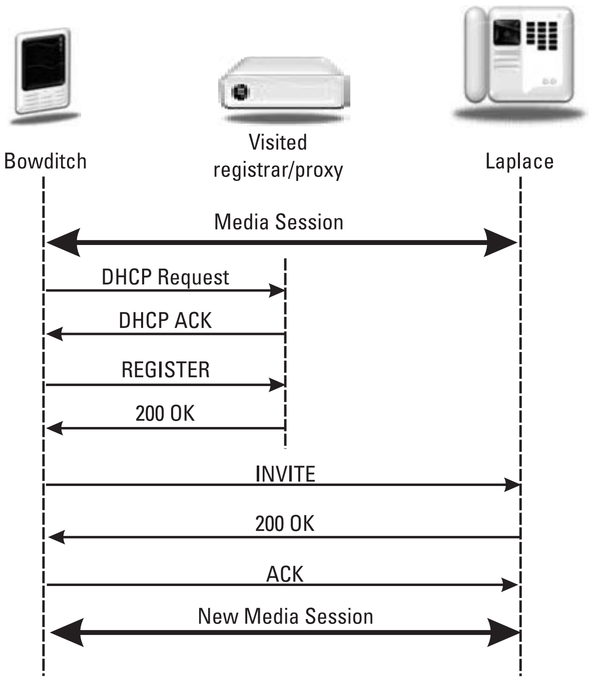


Figure 7.3 Midcall mobility using a re-`INVITE`.

new IP address, then performs a re-`INVITE` to make the signaling and media flow to the new IP address. If the UA momentarily is able to receive media from both networks, the interruption can be almost negligible. If this is not the case, a few media packets may be lost as the media catches up with the signaling, resulting in a slight interruption to the call. The re-`INVITE` would appear as follows:

```
INVITE sip:laplace@client.mathematica.org SIP/2.0
Via: SIP/2.0/UDP 65.32.21.2:5060;branch=z9hG4bK34213
Max-Forwards: 70
To: Marquis de Laplace <sip:laplace@mathematica.org>
 ;tag=90210
From: Nathaniel Bowditch <sip:n.bowditch@salem.ma.us>
 ;tag=4552345
Call-ID: 413830e4leoi34ed4223123343ed21
CSeq: 5 INVITE
Contact: <sip:nat@65.43.21.2>
Content-Type: application/sdp
Content-Length: 143

v=0
o=bowditch 2590844326 2590944533 IN IP4 65.32.21.2
s=Bearing
c=IN IP4 65.32.21.2
t=0 0
m=audio 32852 RTP/AVP 96
a=rtpmap:96 iLBC/8000
```

This contains Bowditch's new IP address in the Via and Contact header fields and SDP media information.

Note that both of the mobility scenarios in Figures 7.2 and 7.3 do not require cooperation between the two wireless networks. As such, this is a useful scenario in which a UA can hand off a call between, for example, a commercial wireless network and a home or office 802.11 wireless network.

For midcall mobility in which the actual route set (set of SIP proxies that the SIP messages must traverse) must change, a re-INVITE cannot be used. For example, if a proxy is necessary for NAT/firewall traversal, then more than just the Contact URI must be changed—a new dialog must be created. The solution to this is to send a new INVITE (which creates a new dialog and a new route set including the new set of proxies) with a Replaces header (Section 6.2.2.4), which identifies the existing session. The call flow is shown in Figure 7.4. It is similar to that in Figure 7.3 except that a BYE is automatically generated to terminate the existing dialog when the INVITE with the Replaces is accepted. In this scenario, the existing dialog between Bowditch and Laplace includes the old visited proxy

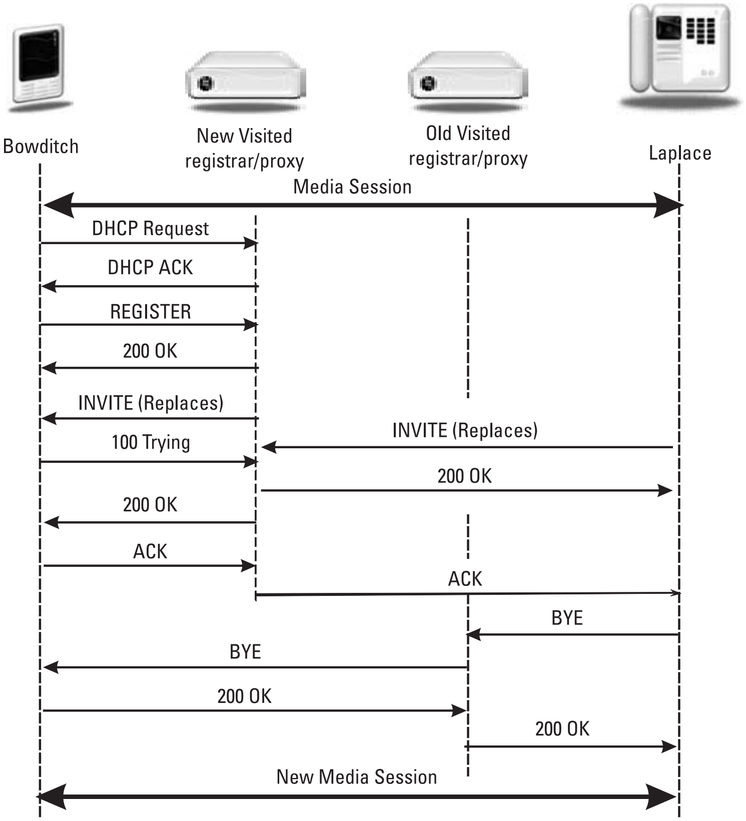


Figure 7.4 Midcall mobility using INVITE with Replaces.

server (the proxy Record-Routed during the initial INVITE). The new dialog using the new wireless network requires the inclusion of the new visited proxy server. As a result, an INVITE with Replaces is sent by Bowditch, which creates a new dialog that includes the new visited proxy server (which Record-Routes) but not the old visited proxy server. When Laplace accepts the INVITE, a BYE is automatically sent to terminate the old dialog that routes through the old visited proxy server that is now no longer involved in the session. The resulting media session is established using Bowditch's new IP address from the SDP in the INVITE.

Services in SIP can be provided in either proxies or in UAs. If the service is resident in the UA, then there are no service mobility problems as the user moves around. However, combining service mobility and personal mobility can be challenging unless each of the user's devices are identically configured with the same services. Also, end-point resident services are only available when the end point is connected to the Internet. A terminating service such as a call forwarding service implemented in an end point will fail if the end point has temporarily lost its Internet connection. For this and other reasons, some services are implemented in the network using SIP proxy servers. For these services, service mobility for a UA means that the same set of proxies are used to route incoming and outgoing requests when mobile.

Due to the nature of the Internet, in general there is no reason why a UA cannot use the same proxies when connected to the Internet at a different point. That is, a UA that is normally in the United States and is configured to use a set of proxies in the United States can still use those proxies when roaming in Europe, for example. Perhaps the SIP hops will have a slightly higher latency due to more router hops and a call setup request may take a second or two longer to complete. However, this has no impact on the quality of the media session as the media always flows directly between the two UAs and does not traverse the SIP proxy servers. As a result, SIP can easily support service mobility over the Internet.

However, there are some cases in which this service mobility approach will not work. For example, if a local proxy server must be traversed in order to facilitate firewall or NAT traversal, or for some other security reason, then a UA may have to use a different first hop proxy when roaming. In this case, service mobility is still possible provided that:

1. The roaming UA is able to discover the necessary local proxy.
2. Both incoming and outgoing requests are routed through the home proxy in addition to any local proxies.

The first requirement is met by the DHCP extension to SIP [5], which allows a UA to learn of a local proxy server at the same time it learns its IP address and other IP configuration information. The second requirement is met using a preloaded Route header field in requests. Normally a Route header is inserted in a request when a proxy requests it using a Record-Route header field. However, it is possible for a configured UA to include a Route header field. If the Route header contains the URI of the home proxy, the request will be routed to the home proxy after the local proxies have been traversed, meeting the requirement for outgoing requests. For incoming requests, the double registration technique will result in both the home and local proxies being traversed by incoming requests. This will result in a call flow similar to Figure 7.2 but with the home proxy server forwarding the INVITE instead of redirecting.

These SIP mobility capabilities are well suited to use over a wireless network such as 802.11 in a home, office, or public space. As roaming agreements allow such wireless "hotspots" to be linked in metropolitan areas, this will provide a wireless service. However, commercial wireless providers plan a specific purpose wireless telephony network using SIP. For some of their business and service requirements, SIP extensions have been developed, which will be discussed in this chapter.

Wireless SIP clients may also make use of voice codecs such as the iLBC [6], which is highly tolerant to packet loss that may be experienced in a heavily loaded 802.11 network.

7.3 IMS and SIP

The 3GPP architecture uses SIP in the IP Multimedia Subsystem (IMS). The main elements of the IMS architecture are listed in Table 7.1. An excellent reference text for IMS is [7].

Table 7.1
IMS Elements

Element	Name
P-CSCF	Proxy call session control function
I-CSCF	Interrogating call session control function
S-CSCF	Serving call session control function
UE	User equipment
MGCF	Media gateway control function
MGW	Media gateway
AS	Application server
MRFC	Media resource function controller
BGCF	Breakout gateway control function
HSS	Home subscriber server

The 3GPP architecture relies on Mobile IP instead of the mobility aspects of SIP described in the previous section. The reasons for doing so are primarily business related rather then technical.

Another requirement of mobility systems is a keep alive signal, which allows end points and proxies to know that a UA still has network connectivity. On an end point to end point basis, this can be done using RTCP (see Section 12.2) reports sent periodically, even when the media is on hold or silence suppression is taking place. However, proxies do not have access to these direct end-to-end reports. Instead, the session timer extension [8] and re-INVITEs can be used for this purpose, or the SIP outbound extension described in Section 10.11.13.

Call Signaling Control Functions or CSCF are SIP proxies that also sometimes behave as a B2BUA under certain circumstances. For example, if a Proxy or P-CSCF looses the radio link to the user equipment (UE) that contains the SIP UA, it can send a BYE on behalf of the UE to tear down the session. The motivation for doing this is for sessions that have a per-minute billing charge, which the out-of-contact UE would otherwise have to pay for but not have the ability to disconnect. To save bandwidth on the wireless connection, a P-CSCF removes Route, Record-Route, Path, Via, Service-Route and other header fields and reinserts them in the opposite direction. To prevent high bandwidth codecs from being used by a UE, a P-CSCF may edit the list of codecs in an SDP offer or answer, preventing the codec from being used. A P-CSCF may change the To and From headers to provide privacy, which is a B2BUA function.

The proxy CSCF provides emergency service, triggers for local services, and normalizes telephone numbers for the rest of the network. The P-CSCF is used as the default outbound proxy server for a UE outside its home network. The Interrogating CSCF queries the HSS to determine the proper service CSCF. The I-CSCF also can do hiding of the S-CSCF network by removing or encrypting Via header fields. The serving CSCF provides the services for the subscriber. It identifies the user's service profile and privileges.

The 3GPP uses IPv6 addresses due to the number of mobile subscribers envisioned and the fact that with Mobile IP, each device may use more than one IP address at a time. SIP RFC 3261 includes full support for IPv6 addresses, and an extension to SDP [9] adds IPv6 support to SDP.

The 3GPP also uses signaling compression [10] to compress SIP messages transmitted over a wireless link. This is primarily done to minimize latency rather than for bandwidth savings. The use of signaling compression with SIP is described in [11], which defines a parameter comp=sigcomp that can be used in Via header fields and as a URI parameter that can be used, for example, in a Contact header field. IMS also makes heavy use of header fields such as Service-Route [13] and Path [14].

The 3GPP uses the adaptive multirate (AMR) [12] codec for the audio encoding.

7.4 IMS Header Fields

Some SIP header fields have been developed based on 3GPP requirements. These so-called P-headers (which stands for proprietary, preliminary, or private) are defined in syntax only in an informational RFC per the old SIP change process [16]. Some are listed in Table 7.2.

In addition, the Open Mobile Alliance (OMA) [21] has registered the P-headers in Table 7.3, which are associated with their Push-to-Talk over Cellular (POC) feature. As part of this feature, a SIP event package is defined [24].

7.5 Conclusion

It is clear that as IP networks become increasingly wireless, SIP will often be utilized over wireless networks. It is well suited for such use for the reasons discussed in this chapter: it has both built-in mobility support when Mobile IP is not used, and can also be used with Mobile IP depending on the wireless network design. Additional work on authentication and roaming will likely be done with SIP as the extensions developed for the 3GPP architecture are too specific to be useful in most other networks.

Table 7.2
3GPP P-Headers

Header Field	Use
`P-Associated-URI`	Lists other URIs associated with the user [15]
`P-Called-Party-ID`	Lists the URI of the called party [15]
`P-Visited-Network-ID`	Identifies the visited network [15]
`P-Access-Network-Info`	Identifies the access network [15]
`P-Charging-Function-Addresses`	Contains charging information [15]
`P-Charging-Vector`	More charging information [15]
`P-User-Database`	Database address of user's profile [17]
`P-Served-User`	Identity of served user [18]
`P-Profile-Key`	Key of profile of the destination URI [19]
`P-Early-Media`	Early media authorization [20]

Table 7.3
OMA P-Headers

Header Field	Use
`P-Answer-State`	Used in PoC for the answering mode of the handset [22]
`P-Refused-URI-List`	Used in PoC to indicate URI-lists related to failures [23]

7.6 Questions

Q7.1 What types of mobility does SIP provide?

Q7.2 Show the call flow where SIP mobility is in use between two networks where SIP requests must traverse a new proxy server.

Q7.3 Which SIP methods and header fields can be used to implement various types of mobility? Give an example of each.

Q7.4 Discuss how SIP mobility and Mobile IP can provide similar and different functions.

References

[1] http://www.3gpp.org.

[2] Schulzrinne, H., and E. Wedlund, "Application-Layer Mobility Using SIP," *Mobility Mobile Computing and Communications Review (MC2R)*, Vol. 4, No. 3, July 2000.

[3] Perkins, C., "IP Mobility Support," RFC 2002, 1996.

[4] Vakil, F., et al., "Supporting Mobility for Multimedia with SIP," IETF Internet-Draft, Work in Progress, December 2000.

[5] Schulzrinne, H., "Dynamic Host Configuration Protocol (DHCP-for-IPv4) Option for Session Initiation Protocol (SIP) Servers," RFC 3361, 2002.

[6] Duric, A., and S. Anderson, "RTP Payload Format for iLBC Speech," RFC 3952, December 2004.

[7] Camarillo, G., and M. Garcia-Martin, *The 3G IP Multimedia Subsystem (IMS): Merging the Internet and the Cellular Worlds,* 3rd ed., New York: John Wiley & Sons, 2008.

[8] Donovan, S., and J. Rosenberg, "Session Timers in the Session Initiation Protocol (SIP),"RFC 4028, April 2005.

[9] Olson, S., G. Camarillo, and A. Roach, "Support for IPv6 in Session Description Protocol (SDP)," RFC 3266, 2002.

[10] Price, R., et al., "Signaling Compression (SigComp)," RFC 3320, 2003.

[11] Camarillo, G., "Compressing the Session Initiation Protocol (SIP)," RFC 3486, February 2003.

[12] Sjoberg, J., et al., "Real-Time Transport Protocol Payload Format and File Storage Format for the Adaptive Multi-Rate (AMR) and Adaptive Multi-Rate Wideband (AMR-WB) Audio Codecs," RFC 3267, June 2002.

[13] Willis, D., and B. Hoeneisen, "Session Initiation Protocol Extension Header Field for Service Route Discovery During Registration," RFC 3608, October 2003.

[14] Willis, D., and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts," RFC 3327, 2003.

[15] Garcia-Martin, M., E. Henrikson, and D. Mills, "Private Header (P-Header) Extensions to the Session Initiation Protocol (SIP) for the 3rd-Generation Partnership Project (3GPP)," RFC 3255, 2003.

[16] Mankin, A., et al., "Change Process for the Session Initiation Protocol (SIP)," RFC 3427, 2002.

[17] Camarillo, G., and G. Blanco, "The Session Initiation Protocol (SIP) P-User-Database Private-Header (P-Header)," RFC 4457, April 2006.

[18] van Elburg, J., "The SIP P-Served-User Private-Header (P-Header) for the 3GPP IP Multimedia (IM) Core Network (CN) Subsystem," RFC 5502, April 2009.

[19] Camarillo, G., and G. Blanco, "The Session Initiation Protocol (SIP) P-Profile-Key Private Header (P-Header)," RFC 5002, August 2007.

[20] Ejzak, R., "Private Header (P-Header) Extension to the Session Initiation Protocol (SIP) for Authorization of Early Media," RFC 5009, September 2007.

[21] http://www.openmobilealliance.org.

[22] Allen, A., J. Holm, and T. Hallin, "The P-Answer-State Header Extension to the Session Initiation Protocol for the Open Mobile Alliance Push to Talk over Cellular," RFC 4964, September 2007.

[23] Hautakorpi, J., and G. Camarillo, "The Session Initiation Protocol (SIP) P-Refused-URI-List Private-Header (P-Header)," RFC 5318, December 2008.

[24] Garcia-Martin, M., "A Session Initiation Protocol (SIP) Event Package and Data Format for Various Settings in Support for the Push-to-Talk over Cellular (PoC) Service," RFC 4354, January 2006.

8

Presence and Instant Messaging

8.1 Introduction

This chapter will cover presence and instant messaging (IM) with SIP. First the history of IM and presence will be covered. Then the SIP events framework will be explored, showing how presence was added to SIP. The set of protocols known as SIMPLE or SIP for Instant Messaging and Presence Leveraging Extensions will be covered. The two different modes of IM will be covered: page mode and session mode using the Message Session Relay Protocol (MSRP). The Jabber presence and instant messaging protocol, also known as the Extensible Messaging and Presence Protocol (XMPP) will also be introduced. Ongoing work to interwork and map presence and instant messages between XMPP and SIMPLE will also be covered.

8.2 History of IM and Presence

Presence is the ability to sense the willingness of another user to communicate. Instant messaging (IM) is a way of exchanging short text messages in near-real time. Presence is often used to determine when another user is available in order to start an instant message exchange. Instant messages are usually sent when the user hits the enter key or when the user clicks a send button. Often, messages are grouped together in a window and shown in sequential order, turning it into a conversation.

A very early presence tool over TCP/IP was the Unix finger command. Finger allowed a user to lookup information about another user, which often included information about the last time the user logged in and the last time mail was read. A very early IM client used on the Internet was known as ICQ (pronounced like "I seek you") [1]. The first version was released in 1996. It provided basic instant messaging between users. America Online's AOL Instant Messenger (AIM) was the first widely used instant messaging and presence application [2]. It was released in 1997 and quickly became popular. It introduced the concept of a "buddy list" or a contact list of other users, which is displayed in a small window. Note that this contact list is stored in the network, allowing the user to have access to the contact list regardless of which computer or device they login from. This user interface is common to nearly all IM systems today. Many other IM clients and systems have been developed, and nearly all are proprietary closed systems. This has resulted in the development of multiheaded clients that present a common user interface and contact list to the user, but log the user into a number of separate systems on the back end.

To address IM and interoperability, the IETF standardized two IM and presence protocols. One was a set of SIP extensions known as SIMPLE (SIP for Instant Messaging Leveraging Extensions) and XMPP (Extensible Messaging and Presence Protocol), which is based on the Jabber open source client. SIMPLE is covered in Section 8.3.4, while XMPP/Jabber is described in Section 8.6. Today, both SIMPLE and XMPP are used to interconnect various closed IM systems. The instant messaging architecture is shown in Figure 8.1 and its elements are in Table 8.1. Both of these systems are built on top of a basic architecture for instant messaging and presence, which is shown in Figures 8.1 and 8.2.

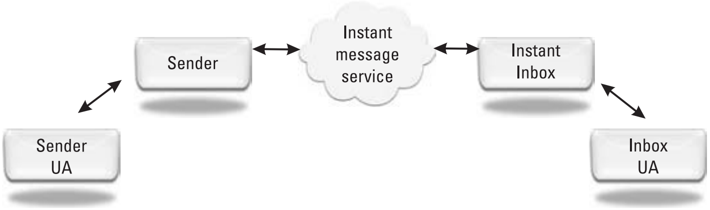


Figure 8.1 IM architecture.

Table 8.1
Instant Messaging Elements

Instant messaging service	Protocol used to transport IM
Sender	Formats message for IM service.
Instant inbox	Receives message from IM service.
Sender user agent	User interface for gathering IM contents from user.
Inbox user agent	User interface for rendering IM to user.

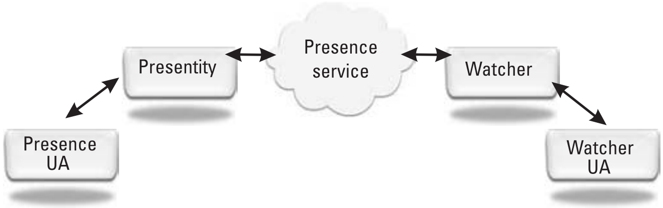


Figure 8.2 Presence architecture.

Table 8.2
Presence Elements

Presence service	Protocol used to transport presence information
Presentity	Publishes presence information to presence service.
Presence user agent	User interface for gathering presence information about user.
Watcher	Requests and receives presence information from presence service.
Watcher user agent	Renders presence information received to the user.

8.3 SIMPLE

In 2001, the IETF chartered a new working group to develop SIP standards and extensions for instant messaging and presence. Over the years, a set of specifications and protocols have been developed, with a few still under development. The standards for presence are summarized in Table 8.3. An overview specification is known as "SIMPLE made simple" [3], which describes how all these protocols work together.

8.4 Presence with SIMPLE

This section will cover presence with SIMPLE. The specifications are summarized in Tables 8.3 and 8.4.

8.4.1 SIP Events Framework

The SIP events framework was defined in RFC 3265 which defined the SUBSCRIBE and NOTIFY methods, as described in Sections 4.1.7 and 4.1.8. SUBSCRIBE is used to establish a dialog and ongoing association between two UAs. In the presence architecture of Figure 8.2, the watcher sends the SUBSCRIBE request to the presentity. If the subscription is authorized, the presentity will send NOTIFYs whenever the state of the presentity changes, and at regular intervals. The basic call flow was shown in Figure 4.5.

Table 8.3
SIMPLE Presence Specifications

Document	Title	Use
RFC 3265	SIP events [4]	Defines SUBSCRIBE and NOTIFY usage with SIP.
RFC 3903	SIP publication [5]	Defines PUBLISH method.
RFC 3863	PIDF [6]	Presence Information Data Format.
RFC 3856	Presence event package [7]	SIP event package used in NOTIFY and PUBLSH methods.
RFC 3857	Watcher info package [8]	SIP event package used to find out who is watching or subscribing to your state.
RFC 3858	Watcher info XML [9]	Body used for watcher info NOTIFYs and PUBLISHes.
RFC 4480	RPID [10]	Rich Presence Information Data format PIDF extensions.
RFC 4482	CIPID [11]	Contact information in presence information data.
RFC 4479	Data model [12]	A data model for presence.
RFC 4662	Resource list extension [13]	Combining subscriptions into a resource list.
RFC 4661	Filtering [14]	XML format for event notification filtering.
RFC 4825	XCAP [15]	XML configuration access protocol.
RFC 4826	Resource list format [16]	XML format for resource lists.
RFC 4827	XCAP usage for presence [17]	Used for manipulating presence document.
RFC 5025	Presence authorization [18]	Presence authorization rules.
RFC 5196	UA capabilities extension [19]	Extensions to PIDF for UA capabilities.
RFC 5261	XML patch framework [20]	
RFC 5262	Partial presence PIDF [21]	
RFC 5263	Partial presence [22]	
RFC 5264	Partial presence publication [23]	
RFC 3861	Pres URI scheme [24]	Use of SRV records for the pres URI scheme.
[25]	Conditional notification	
[26]	XCAP diff	
RFC 5364	SIMPLE made simple	

SIP events allow any number of event packages to be defined. Table 4.8 lists common SIP event packages. For presence, the presence package is used, which also uses the application/xml+presence MIME type. This XML format is used to convey the presence state in NOTIFYs.

8.4.2 Presence Bodies

Presence information is conveyed using SIP message bodies in XML (Extensible Markup Language) format. The basic presence document is known as Presence Information Data Format or PIDF [12] and is shown here:

Table 8.4
SIMPLE Instant Messaging Specifications

Document	Title	Use
RFC 3428	SIP extensions for IM [27]	Defines `MESSAGE` method for page mode IM.
RFC 3994	Message composition [28]	Used for “istyping” indications for IM.
RFC 5365	Multiple recipient IMs [29]	Used to send an IM to a group.
RFC 4975	MSRP [30]	Message Sessions Relay Protocol.
RFC 4976	Relay extensions for MSRP [31]	Relays for MSRP for logging and NAT traversal.
RFC 3861	IM URI scheme [24]	Use of SRV records for the IM URI scheme.
RFC 3862	CPIM [32]	Common profile for IM format.
RFC 5438	IMDN [33]	IM disposition notification.
Draft	MSRP multiparty chat [34]	
Draft	Alt connection with MSRP [35]	

```
<?xml version=”1.0” encoding=”UTF-8”?>
<presence xmlns=”urn:ietf:params:xml:ns:pidf”
  entity=”pres:Sophie.Germain@mathematica.org”>
 <tuple id=”34g45sfde”>
  <status>
   <basic>open</basic>
  </status>
  <contact>sip:sophie@78.34.32.1:51234</contact>
 </tuple>
</presence>
```

The XML document begins with the XML declaration, then the presentity information is contained in the `<presence>` element. The presence information is that of the entity listed in the entity attribute. Presence documents consist of `<tuple>` elements, which contain the `<status>` of the presentity and can be either `open` (available) or `closed` (unavailable). In addition, this example shows `<contact>` information. Presence information is conveyed in a SIP message using the `Content-Type: application/pidf+xml`. Note that this `Content-Type` is also sometimes incorrectly written as `application/cpim-pidf+xml`.

The next example shows the inclusion of both contact information (Contact Information Presence Information Data or CIPID) [11], UA capabilities [19] , and the presence data model [12]:

```
<?xml version=”1.0” encoding=”UTF-8”?>
 <presence xmlns=”urn:ietf:params:xml:ns:pidf”
    xmlns:dm=”urn:ietf:params:xml:ns:pidf:data-model”
    xmlns:cipid=”urn:ietf:params:xml:ns:pidf:cipid”
    xmlns:caps=”urn:ietf:params:xml:ns:pidf:caps”
   entity=”pres:m.c.thomas@brynmawr.edu”>
  <tuple id=”54234g45sfde”>
   <status>
    <basic>open</basic>
   </status>
   <contact>sip:mc@dean.brynmawr.edu</contact>
  <caps:servcaps>
   <caps:audio>true</caps:audio>
   <caps:video>true</caps:video>
  </caps:servcaps>
 </tuple>

 <dm:person id=″1″>
  <cipid:card>http://brynmawr.edu/~m.c.thomas/card.vcd</c:card>
  <cipid:display-name>M. C. Thomas</c:card>
 </dm:person>
</presence>
```

This document defines a default namespace plus three other extension namespaces, `dm`, `cipid`, and `caps`. This example shows that the capabilities of the UA identified by the `<contact>` element include both audio and video, as indicated by the `<audio>` and `<video>` subelements of `<servcaps>`. The data model element `<person>` is used to provide personal information about the contact entity, and includes the contact information including a vcard, display name, homepage, icon, and map. The next example shows Rich Presence Information Data (RIPD) [10]:

```
<?xml version=″1.0″ encoding=″UTF-8″?>
  <presence xmlns=″urn:ietf:params:xml:ns:pidf″
    xmlns:rpid=″urn:ietf:params:xml:ns:pidf:rpid″
    entity=″sip:skovalevsky@su.se″>
  <tuple id=″4sdf432sd″>
   <status>
     <basic>closed</basic>
   </status>
   <rpid:class>IM</rpid:class>
   <contact>im:skovalesky@su.se</contact>
  </tuple>
  <tuple id=″832thr76jk″>
   <status>
     <basic>open</basic>
   </status>
   <rpid:class>voice</rpid:class>
   <contact>tel:+465551212</contact>
  </tuple>
</presence>
```

8.4.3 Resource Lists

Resource lists allow a UA to combine multiple individual subscriptions into a single subscription and receive notifications about multiple individual subscriptions in a single notification message. Resource lists are defined in RFC 4662 [13] and are an extension to the SIP events framework [4]. As such, resource lists can be used with any event package. However, the most common use is for subscriptions to a “buddy list” in presence applications. In this application, a contact list or “buddy list” is represented by a SIP URI and stored in a server known as a resource list server or RLS. For example, if a user's contact list contained 10 SIP URIs, this would normally require 10 separate subscriptions to be maintained, one for each URI in the list. Each of these subscriptions would need to be created, managed, and refreshed, resulting in a lot of overhead traffic messages. With the resource list extension, the list is stored in the RLS, and the presence client creates a single subscription to the RLS over which notifications of the presence of all 10 URIs would be sent. The RLS may need to initiate 10 separate subscriptions, but not the presence UA. An example event list subscription is shown here:

```
SUBSCRIBE sip:beatrix-321223@lakesdistrict.co.uk SIP/2.0
Via: SIP/2.0/TCP cottage43.lakesdistrict.co.uk:5060
 ;branch=z9hG4bKwYb6QREiCL
Max-Forwards: 70
To: <sip:beatrix-321223@lakesdistrict.co.uk>
From: Beatrix Potter <sip:beatrix@lakesdistrict.co.uk>
 ;tag=6733
Call-ID: dkfj39890wssdfj2938d7
CSeq: 23822 SUBSCRIBE
Contact: <sip:beatrix@cottage.lakesdistrict.co.uk;
 transport=tcp>
Ev ent: presence
Expires: 7200
Supported: eventlist
Accept: application/pidf+xml
Accept: application/rlmi+xml
Accept: multipart/related
Content-Length: 0
```

In this subscription, the presence of the `Supported: eventlist` header field indicates that the presence UA supports the extension. The `Accept: application/rlmi+xml` also indicates that it is willing to accept notifications that use the resource list format described in the next section. The `Event: presence` and `Accept: application/pidf+xml` are to indicate that the subscription is for presence information. A NOTIFY sent during this subscription might look like:

```
NOTIFY sip:beatrix@cottage43.lakesdistrict.co.uk SIP/2.0
Via: SIP/2.0/TCP pres.vancouver.example.com
 ;branch=z9hG4bKMgRenTETmm
Max-Forwards: 70
From: <sip:beatrix-321223@lakesdistrict.co.uk>;tag=dkisksk3
To: Beatrix Potter <sip:beatrix@lakesdistrict.co.uk>
 ;tag=673
Call-ID: dkfj39890wssdfj2938d7
CSeq: 997935768 NOTIFY
Contact: <sip:rls34.lakesdistrict.co.uk>
Event: presence
Subscription-State: active;expires=7200
Require: eventlist
Content-Type: multipart/related;type=″application/rlmi+xml″
 ;start=″<38dk2nXYxAE@lakesdistrict.co.uk>″
 ;boundary=″0909e3ksdf893″
Content-Length: 1560
```

```
--0909e3ksdf893
Content-Transfer-Encoding: binary
Content-ID: <38dk2nXYxAE@lakesdistrict.co.uk>
Content-Type: application/rlmi+xml;charset=”UTF-8”

<?xml version=”1.0” encoding=”UTF-8”?>
<list xmlns=”urn:ietf:params:xml:ns:rlmi”
   uri=”sip:beatrix-friends@lakesdistrict.co.uk”
   version=”1” fullState=”true”>
  <resource uri=”sip:hildegard@abbey.org”>
  <name>Hildegard von Bingen</name>
  <instance id=‟juwigmtboe‟ state=‟active‟
       cid=‟K3qrtD83kj2@lakesdistrict.co.uk‟/>
 </resource>
 <resource uri=”sip:juana_ines@cuidaddemexico.mx”>
  <name>Sor Juana Ines de la Crux</name>
  <instance id=”hqzsuxtfyq” state=”active”
       cid=”XdY7yhjxAEw@lakesdistrict.co.uk”/>
 </resource>
 <resource uri=”sip:mmead@amnh.org”>
  <name>Margaret Mead</name>
 </resource>
</list>

--0909e3ksdf893
Content-Transfer-Encoding: binary
Content-ID: <K3qrtD83kj2@lakesdistrict.co.uk>
Content-Type: application/pidf+xml;charset=”UTF-8”

<?xml version=”1.0” encoding=”UTF-8”?>
<presence xmlns=”urn:ietf:params:xml:ns:pidf”
  entity=”sip:hildegard@abbey.org”>
 <tuple id=”93sg89ae”>
  <status>
   <basic>open</basic>
  </status>
  <contact
   priority=”1.0”>sip:hildegard@music.abbey.org</contact>
 </tuple>
</presence>

--0909e3ksdf893
Content-Transfer-Encoding: binary
Content-ID: <XdY7yhjxAEw@lakesdistrict.co.uk>
Content-Type: application/pidf+xml;charset=”UTF-8”

<?xml version=”1.0” encoding=”UTF-8”?>
<presence xmlns=”urn:ietf:params:xml:ns:pidf”
  entity=”sip:juana_ines@cuidaddemexico.mx”>
 <tuple id=”4rslie74”>
  <status>
   <basic>closed</basic>
  </status>
 </tuple>
</presence>

--0909e3ksdf893--
```

The notification contains `Require: eventlist,` which indicates that this is an eventlist notification. The `Content-Type: multipart/related;type="application/rlmi+xml"` indicates that the message body is multipart MIME and contains `application/rlmi+xml` parts, which is the XML format for event lists defined in RFC 4662 [13]. Each part of the multipart MIME is separated by a CRLF and a boundary string, defined to be `0909e3ksdf893` in this example. The `Content-Type: application/pidf+xml;charset="UTF-8"` in each part indicates that each part is a PIDF.

A resource list stored on an RLS is not a static list—the presence user may add or delete users from this list at any time. One way to do this is to use XCAP to manipulate the resource list. The format for this resource list is defined in RFC 4826 [16]. An example document is shown here:

```
<?xml version="1.0" encoding="UTF-8"?>
 <rls-services xmlns="urn:ietf:params:xml:ns:rls-services"
   xmlns:rl="urn:ietf:params:xml:ns:resource-lists"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <service uri="sip:astronomers@lists.org">
   <list name="astronomers">
    <rl:entry uri="sip:janet.taylor@observatory.com"/>
    <rl:entry uri="sip:Maria.Kirch.Winkelmann@comets.r.us"/>
   </list>
   <packages>
    <package>presence</package>
   </packages>
  </service>
 </rls-services>
```

The list is enclosed in a `<rls-services>` element where each `<service>` element defines a resource list. In this example, the event list URI is `sip:marketing@example.com` as defined by the URI attribute to the `<service>` element. The list is included inside the `<list>` element.

Instead of using XCAP to create the list, the resource list subscribe extension [36] allows a SUBSCRIBE to create an event list. An example SUBSCRIBE is shown here:

```
SUBSCRIBE sip:beatrix-321223@lakesdistrict.co.uk SIP/2.0
Via: SIP/2.0/TCP cottage43.lakesdistrict.co.uk
 ;branch=z9hG4bKwYb6QREiCL
Max-Forwards: 70
To: <sip:beatrix-321223@lakesdistrict.co.uk>
From: Beatrix Potter <sip:beatrix@lakesdistrict.co.uk>
 ;tag=26
Call-ID: dkfj39890wssdfj2938d7
CSeq: 23822 SUBSCRIBE
Contact: <sip:beatrix@cottage43.lakesdistrict.co.uk;transport=tcp>
Event: presence
Expires: 7200
Require: recipient-list-subscribe
Supported: eventlist
Accept: application/pidf+xml
```

```
Accept: application/rlmi+xml
Accept: multipart/related
Accept: multipart/signed
Accept: multipart/encrypted
Content-Type: application/resource-lists+xml
Content-Disposition: recipient-list
Content-Length: 337

<?xml version=”1.0” encoding=”UTF-8”?>
<resource-lists xmlns=”urn:ietf:params:xml:ns:resource-lists”
    xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”>
 <list>
  <entry uri=”sip:hildegard@abbey.org” />
  <entry uri=”sip:juana_ines@cuidaddemexico.mx” />
  <entry uri=”sip:mmead@amnh.org” />
 </list>
</resource-lists>
```

The presence of the `Require: recipient-list-subscribe` indicates that this SUBSCRIBE contains a list for the creation of an event list. Note that the `Content-Type: application/resource-lists+xml` is the same format as the list in [16]. However, this extension only allows list creation with the initial SUBSCRIBE. Refresh SUBSCRIBEs cannot modify the list.

An alternative is defined in [37] which allows a SUBSCRIBE request to create or update an event list. Note that this specification is not yet an RFC but is implemented in industry. An example call flow is shown in Figure 8.3.

An example SUBSCRIBE is shown here:

```
SUBSCRIBE sips:manya-friends@warszawa.pl SIP/2.0
Via: SIP/2.0/TLS rs.warszawa.pl
 ;branch=z9hG4bKwYb6QREiCL
```

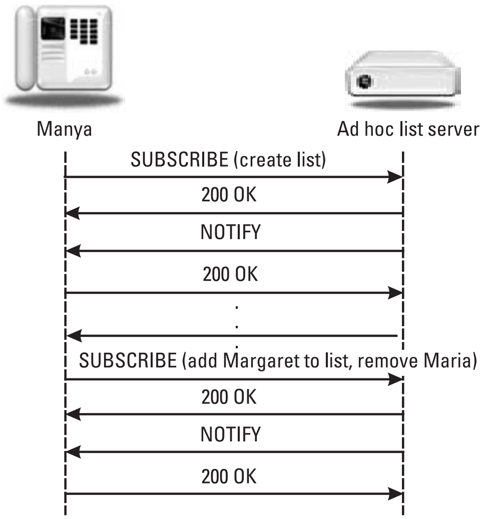


Figure 8.3 Ad hoc list creation and manipulation.

```
To: <sips:manya-friends@warszawa.pl>
From: Manya Sklodowska <sips:manya@warszawa.pl>;tag=22222
Call-ID: 7jh5dD3Fkifr2
CSeq: 3 SUBSCRIBE
Max-Forwards: 70
Event: presence
Require: adhoclist
Accept: application/pidf+xml
Accept: application/rlmi+xml
Contact: <sips:manya@rs.warszawa.pl>
Content-Type: application/adrl+xml
Content-Length: ...

<?xml version="1.0"?>
<adhoclist uri="sip:manya-friends@warszawa"
             name="Manya's Friends">
 <add>
  <resource
    uri="sips:hypatia@greatlibrary.alexandria.eg">
  <resource uri="sips:m.g.mayer@katowice.pl">
  <resource uri="sips:pan.chao@imperial.org">
 </add>
</adhoclist>
```

This subscribe adds three URIs in the `<resource>` element contained in the `<adhoclist>` element. The `Require: adhoclist` and `Content-Type: application/adrl+xml` indicate that this `SUBSCRIBE` is for an RLS that supports adhoc lists. A later refresh `SUBSCRIBE` could be used to add more URIs and delete others. For example:

```
SUBSCRIBE sips:manya-friends@warszawa.pl SIP/2.0
Via: SIP/2.0/TLS rs.warszawa.pl
 ;branch=z9hG4bKwYb6QREiCL
To: <sips:manya-friends@warszawa.pl>;tag=32111df
From: Manya Sklodowska <sips:manya@warszawa.pl>;tag=xz2d
Call-ID: 7jh5dD3Fkifr2
CSeq: 34 SUBSCRIBE
Max-Forwards: 70
Event: presence
Require: adhoclist
Accept: application/pidf+xml
Accept: application/rlmi+xml
Contact: <sips:manya@rs.warszawa.pl>
Content-Type: application/adrl+xml
Content-Length: ...

<?xml version="1.0"?>
<adhoclist uri="sip:manya-friends@warszawa"
               name="Manya's Friends">
<add>
  <resource
   uri="sips:margaret_cavendish@dutchess.newcastle.gb">
</add>
 <delete>
  <resource uri="sips:m.g.mayer@katowice.pl">
 </delete>
</adhoclist>
```

8.4.4 Filtering

An enhancement to SIP events is to allow the use of filters. A filter is an XML document included in a SUBSCRIBE, which contains information about the type and rate of notifications to be sent during the subscriptions. The filters extension is defined in RFC 4660 [38] while the XML filter format is defined in RFC 4661 [14]. Filters can be defined to apply to a single resource, a set of resources, or all resources in the subscription. Filtering can also be applied to specific domains. Filters can be changed during a subscription by including a new filter in a refresh SUBSCRIBE. A refresh SUBSCRIBE without a filter means continued use of the current filter. Filter removal can be done using the remove="true" attribute. Figure 8.4 shows an example of filter creation, manipulation, and removal.

For example, consider the SUBSCRIBE containing a filter here:

```
SUBSCRIBE sip:c.dipisan@authors.org SIP/2.0
Via: SIP/2.0/TCP client.example.com:5060
 ;branch=z9hG4bKxjfdsjfk
To: <sip:c.dipisan@authors.org>
From: <sip:jane.austen@southhampton.uk>;tag:12341111
Call-ID: 232432udfidfjmk342
```

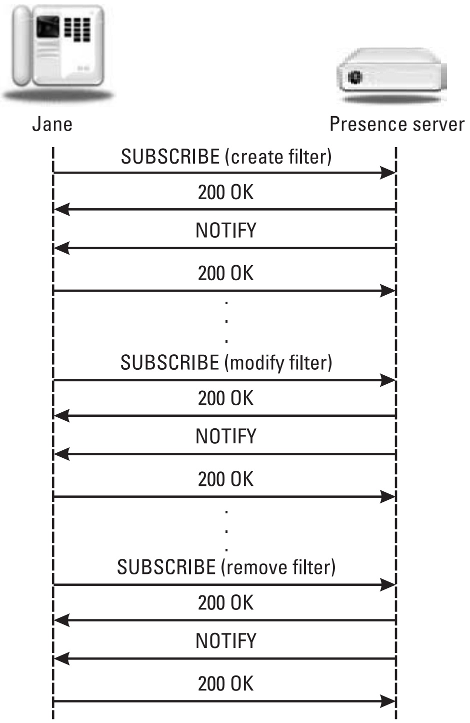

Jane
Presence server
SUBSCRIBE (create filter)
200 OK
NOTIFY
200 OK
SUBSCRIBE (modify filter)
200 OK
NOTIFY
200 OK
SUBSCRIBE (remove filter)
200 OK
NOTIFY
200 OK

Figure 8.4 Example event filter creation, manipulation, and removal.

```
CSeq: 1 SUBSCRIBE
Expires: 3600
Event: Presence
Contact: <sip:jane@chawton.southhampton.uk;transport=tcp>
Content-Type: application/simple-filter+xml
Content-Length: ...

<?xml version=”1.0” encoding=”UTF-8”?>
<filter-set xmlns=”urn:ietf:params:xml:ns:simple-filter”>
 <ns-bindings>
  <ns-binding prefix=”pidf”
         urn=”urn:ietf:params:xml:ns:pidf”/>
  <ns-binding prefix=”rpid”
        urn=”urn:ietf:params:xml:ns:pidf:rpid”/>
 </ns-bindings>
 <filter id=”123” uri=”sip:c.dipisan@authors.org”>
  <what>
   <include type=“xpath“>
    //pidf:tuple/pidf:status[pidf:basic=“open“]/pidf:basic
   </include>
   <include type=“xpath“>
    //pidf:tuple[pidf:status/pidf:basic=“open“]/rpid:class
   </include>

   <include type=“xpath“>
   //pidf:tuple[pidf:status/pidf:basic=“open“]/pidf:contact
   </include>
  </what>
 </filter>
</filter-set>
```

The filter is defined by the `Content-Type: application/simple-filter+xml` message body. The filter is contained in the `<filter-set>` element, which includes a single `<filter>` element. This filter will result in notifications only being sent that have either the value `open` inside the `<status>` element in the PIDF `<basic>, <contact>` elements, or the RPID `<class>` element. All other notifications would be suppressed.

8.4.5 Conditional Event Notifications and ETags

Another optimization involves the use of conditional notifications as defined in [25]. In the normal operations of SIP events, every SUBSCRIBE will generate an automatic NOTIFY. While the initial NOTIFY usually contains useful initial state, the NOTIFYs sent to refresh SUBSCRIBEs often do not contain any new information. NOTIFYs can be sent with the `SIP-ETag` header field, which contains an identifier known as the entity-tag for the current state. A subsequent refresh SUBSCRIBE can then include this identifier in a `Suppress-If-Match` header field. If the entity-tag in the SUBSCRIBE matches the local entity-tag, the automatic NOTIFY will be suppressed and a `204 No Notification` response sent. If the entity-tag does not match, indicating a change in state, the NOTIFY will be generated. This is shown in Figure 8.5.

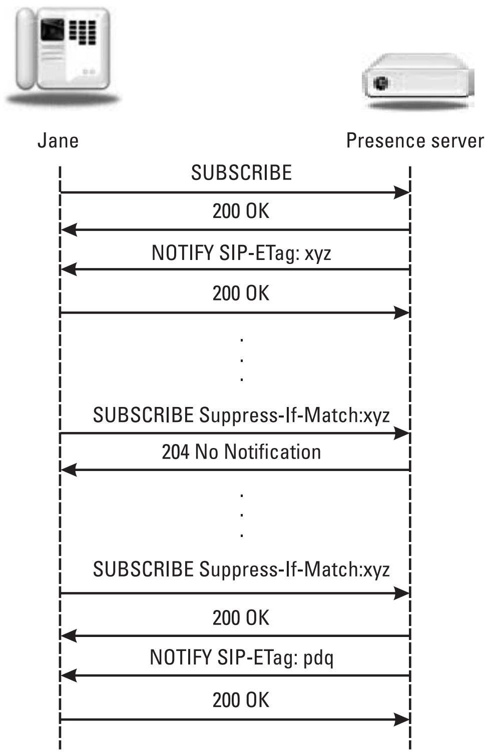


Figure 8.5 Conditional notifications and ETags.

This mechanism can also be used to poll for state. An example is shown in Figure 8.6.

8.4.6 Partial Publication

Another optimization is defined in [23] for the partial publication of presence information. Normally, each time presence information changes, the entire PIDF containing full state must be sent. This extension allows the parts of the PIDF that have changed to be sent. Figure 8.7 shows the basic operation in which first a full publication must be sent. After that, partial publications can be sent. In this flow, the first PUBLISH creates the state information and the second updates it. The third attempts to update it, but the previous state has expired. The full state is then published with the fourth PUBLISH.

For example, consider the partial publication message:

```
PUBLISH sip:murasaki@kyoto.jp SIP/2.0
Via: SIP/2.0/TCP court.kyoto.jp:15332;branch=z9hG4bKfdDsjfk1
To: Murasaki Shikibu <sip:murasaki@kyoto.jp>
```

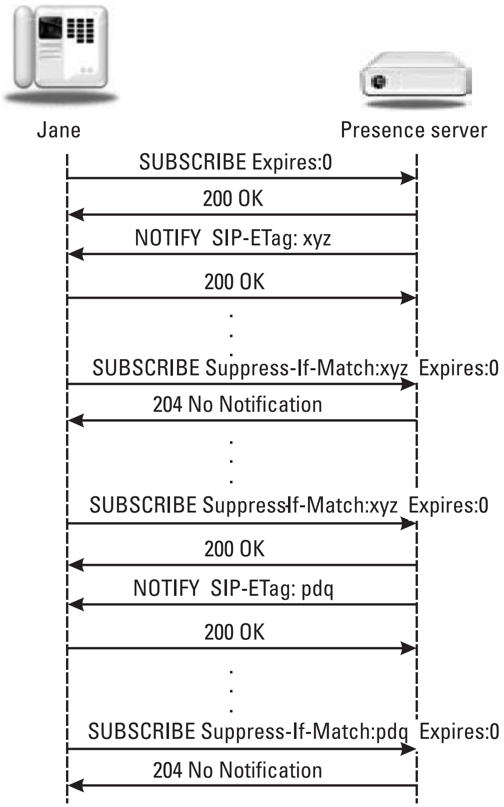


Figure 8.6 Polling for state.

```
From: Murasaki Shikibu <sip:murasaki@kyoto.jp>
 ;tag=v1F23d41111
Call-ID: 8Fd3wfldfa
CSeq: 61 PUBLISH
Event: presence
SIP-If-Match: 34616386238299
Expires: 3600
Content-Type: application/pidf-diff+xml
Content-Length: ...

<?xml version=”1.0” encoding=”UTF-8”?>
<p:pidf-diff xmlns=”urn:ietf:params:xml:ns:pidf”
  xmlns:p=”urn:ietf:params:xml:ns:pidf-diff”
  xmlns:r=”urn:ietf:params:xml:ns:pidf:rpid”
  entity=”pres:murasaki@kyoto.jp”>

<p:add sel=”presence/note” pos=”before”>
<tuple id=”L6erPt47”>
 <status>
 <basic>open</basic>
 </status>
 <contact priority=”0.4”>mailto:genji@tales.jp</contact>
```

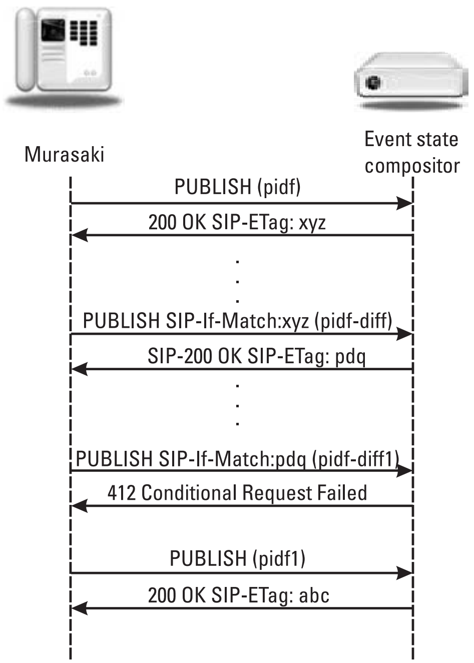


Figure 8.7 Partial publication example.

```
    </tuple>
    </p:add>

    <p:replace
     sel="*/tuple[@id='1d23d0d']/status/basic/text()">open</p:replace>
    <p:remove sel="*/r:person/r:status/r:activities/r:busy"/>

    <p:replace
     sel="*/tuple[@id='Kcg23d1j']/contact/@priority">0.7</p:replace>

    </p:pidf-diff>
```

The `SIP-If-Match` header field containing the entity-tag ensures that this partial presence publication is applied to the correct full presence document, which is identified by the entity-tag. The `Content-Type: application/pidf-diff+xml` message body is used to convey the partial presence document. The `<add>` , `<replace>`, and `<remove>` elements modify the full state presence document in the appropriate way. In each case, the `tuple` identifier is used to select the correct element.

8.4.7 Presence Documents Summary

Table 8.5 lists the various types of presence documents.

Table 8.5
Presence Document Formats

Content-Type	Use	Specification
application/pidf+xml	Basic presence document	RFC 3863
application/pidf-diff+xml	Partial presence document	RFC 5262
application/rlmi+xml	Resource list notification	RFC 4662
application/resource-lists+xml	Resource list creation	RFC 4862
application/adrl+xml	Ad hoc list management	[37]
application/simple-filter+xml	Filter in a SUBSCRIBE	RFC 4661

8.5 Instant Messaging with SIMPLE

Instant messaging with SIP was a very early SIP extension in RFC 3428 [27]. In addition to this simple transport, SIP extensions for "iscomposing" or "istyping" have been standardized. Also, a standard for Instant Message Delivery Notification (IMDN) has been developed. Finally, a session mode instant messaging protocol Message Sessions Relay Protocol (MSRP) has been developed.

8.5.1 Page Mode Instant Messaging

Page mode instant messaging is done using the MESSAGE method. It provides transport of a single message between two SIP endpoints, as described in Section 4.1.11. Typical message body types include text/plain, text/html, and message/cpim [32]. Page mode is suitable for a single message exchange or a series of short messages, similar to paging or SMS on mobile phones. It is not suitable for a long conversation between users or as a channel for transferring files or multimedia clips. For these situations, the session mode should be used.

8.5.2 Common Profile for Instant Messaging

The Common Profile for Instant Messaging (CPIM) [32] was developed as the IETF recognized that multiple internet protocols have been developed for instant messaging and a common format was needed to support interworking between them. A common message format allows for the possibility of end-to-end encryption and signatures to be used even when different IM protocols are used for transport. Both SIMPLE and XMPP (described in Section 8.6) support CPIM and define mapping between their native IM transport and CPIM, which makes mapping between them easier. An example CPIM message carried in a MESSAGE SIP request is shown here:

```
MESSAGE allesandra@bologna.it SIP/2.0
Via: SIP/2.0/TCP lab.rss.org;branch=z9hG4bK7F6sg83dkse
Max-Forwards: 70
From: <sip:florence.nightingale@rss.org>;tag=49583
```

```
To: Alessandra Giliani <allesandra@bologna.it>
Call-ID: 43dKdas88d8V8asd77a
CSeq: 16 MESSAGE
Content-Type: message/cpim
Content-Length: . . .

From: Florence <im:florence.nightingale@rss.org>
To: Alessandra <im:alessandra@bologna.it>
DateTime: 2000-12-13T13:40:00-08:00
Subject: Statistical Tables
Subject:;lang=it Tabelle Statistiche
NS: MyFeatures <mid:MessageFeatures@id.foo.com>
Require: MyFeatures.VitalMessageOption
MyFeatures.VitalMessageOption: Confirmation-requested
MyFeatures.WackyMessageOption: Use-silly-font

Content-Type: text/xml; charset=utf-8
Content-ID: <34do9flsf@rss.org>

<body>
I have the information you requested about the
statistical tables.
</body>
```

The `Content-Type: message/cpim` header field indicates that the message body is a CPIM message. The next lines are the CPIM headers, followed by a blank line, then the actual CPIM message content. The CPIM message content is preceded by the MIME headers `Content-Type` and `Content-ID`. In this case, the actual message is encoded as `text/xml`. The complete set of CPIM header fields is listed in Table 8.6.

8.5.3 Instant Messaging Delivery Notification

The format `message/cpim` was defined so that IM systems standardized by the IETF could interoperate at the message layer. As a result, SIMPLE supports CPIM as does XMPP, allowing interoperability as described in Section 8.6.2. CPIM has also been extended to add delivery notification in [33]. For example, consider the SIP page mode IM here:

Table 8.6
CPIM Header Fields

Header	Meaning
`From`	Sender or originator of IM
`To`	Recipient of IM
`cc`	Courtesy copy
`DateTime`	Date and time IM was sent
`Subject`	Subject of IM
`NS`	Local name space prefix
`Require`	Header or feature that must be implemented

```
MESSAGE sip:florence.nightingale@rss.org
Via: SIP/2.0/TCP library.bologna.it
 ;branch=z9hG4bK6sQpg8dks9e22
Max-Forwards: 70
From: Alessandra Giliani <allesandra@bologna.it>;tag=6312
To: <sip:florence.nightingale@rss.org>
Call-ID: 8765-2555-2103-4723
CSeq: 77 MESSAGE
Content-Type: application/cpim
Content-Length: ...

To: Florence <im:florence.nightingale@rss.org>
From: Alessandra <im:alessandra@bologna.it>
NS: imdn <urn:ietf:params:imdn>
imdn.Message-ID: 384jk3214jW
DateTime: 2006-04-04T12:16:49-05:00
imdn.Disposition-Notification: positive-delivery
 ,negative-delivery
Content-type: text/plain
Content-length: 19

Grazie, Florence!
```

The CPIM wrapper begins after the first `Content-Length` header field and goes to the second `Content-Length` field. The actual message then follows. The `imdn.Disposition-Notification` CPIM field indicates that message disposition is requested for this message, and the two types of notification requested are: `positive-delivery` and `negative-delivery`. Other notification options are `processing` or `display`. An example Internet message delivery notification (IMDN) response is shown here:

```
MESSAGE allesandra@bologna.it SIP/2.0
Via: SIP/2.0/TCP lab.rss.org;branch=z9hG4bK83924
Max-Forwards: 70
From: <sip:florence.nightingale@rss.org>;tag=823123
To: Alessandra Giliani <allesandra@bologna.it>
Call-ID: 8o1eCusjwX99Sfs2M
CSeq: 9321 MESSAGE
Content-Type: application/cpim
Content-Length: ...

From: Florence <im:florence.nightingale@rss.org>
To: Alessandra <im:alessandra@bologna.it>
NS: imdn <urn:ietf:params:imdn>
imdn.Message-ID: 83jk41dlf20fks
Content-type: message/imdn+xml
Content-Disposition: notification
Content-length: ...

<?xml version=″1.0″ encoding=″UTF-8″?>
<imdn xmlns=″urn:ietf:params:xml:ns:imdn″>
 <message-id>384jk3214jW</message-id>
 <datetime>2008-04-04T12:16:49-05:00</datetime>
 <recipient-uri>im:florence.nightingale@rss.org</recipient-uri>
 <delivery-notification>
  <status>
    <delivered/>
    </status>
   </delivery-notification>
  </imdn>
```

This message is sent in the reverse direction to the IM with the IMDN request and contains the CPIM wrapper and the `Content-type: message/imdn+xml`, which conveys the actual XML IMDN message. In this case, the status is `<delivered/>`. Other status values are `<delivered>`, `<failed>`, `<forbidden>`, or `<error>`. An example exchange is shown in Figure 8.8.

8.5.4 Message Composition Indication

Another instant messaging extension is the message composition indication [28]. This extension can provide the familiar "istyping" indication meaning that the other party in an IM conversation is currently composing media. Two states are assumed: `idle` and `active`. Active is the state when composing or typing is taking place. The indications are conveyed using XML objects and transported over the IM channel. For example, if `MESSAGE` is used:

```
MESSAGE sip:winifred@192.168.42.1 SIP/2.0
Via: SIP/2.0/TCP mosses.nybg.org;branch=z9hG4bK2sdfds
Via: SIP/2.0/TCP mosses.nybg.org;branch=z9hG4bK98s
 ;received=73.32.1.2
Via: SIP/2.0/TCP mosses.nybg.org;branch=z9hG4bK76dsgFdksWe10
 ;received=128.56.42.1
Max-Forwards: 68
From: sip:britton@nybg.org;tag=92349583
To: sip:goldring@paleosoc.org
Call-ID: eifk33kfsd2as2df2389sad51kpoef
CSeq: 186 MESSAGE
```

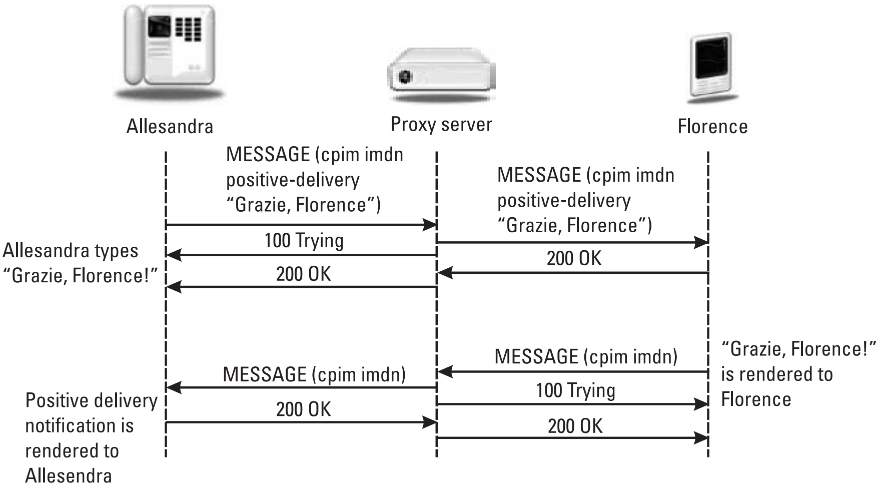

Allesandra
Proxy server
Florence
MESSAGE (cpim imdn positive-delivery "Grazie, Florence")
MESSAGE (cpim imdn positive-delivery "Grazie, Florence")
100 Trying
Allesandra types "Grazie, Florence!"
200 OK
200 OK
MESSAGE (cpim imdn)
"Grazie, Florence!" is rendered to Florence
MESSAGE (cpim imdn)
100 Trying
Positive delivery notification is rendered to Allesendra
200 OK
200 OK

Figure 8.8 Instant Message Delivery Notification example.

```
Content-Type: application/im-iscomposing+xml
Content-Length: ...

<?xml version=″1.0″ encoding=″UTF-8″?>
<isComposing xmlns=″urn:ietf:params:xml:ns:im-iscomposing″
  xmlns:xsi=″http://www.w3.org/2001/XMLSchema-instance″
  xsi:schemaLocation=″urn:ietf:params:xml:ns:im-composing
  iscomposing.xsd″>
 <state>active</state>
 <contenttype>text/plain</contenttype>
 <refresh>90</refresh>
</isComposing>
```

The `Content-Type: application/im-iscomposing+xml` header field indicates that this message is a message composition indication. The composing state is conveyed in the `<isComposing>` element where the `<state>` element has the value `active`. The `<contenttype>` element indicates that the content being composed is text. The receipt of this message could be used to display an indication of "istyping" to the user for a short period of time, after which the indication times out and the state returns to `idle`.

8.5.5 Multiple Recipient Messages

An extension allows an instant message to carry a recipient list [29]. An intermediary server known as a relay or exploder can take such a request and replicate the instant messaging to the recipients in the list. This operation is shown in Figure 8.9.

For example:

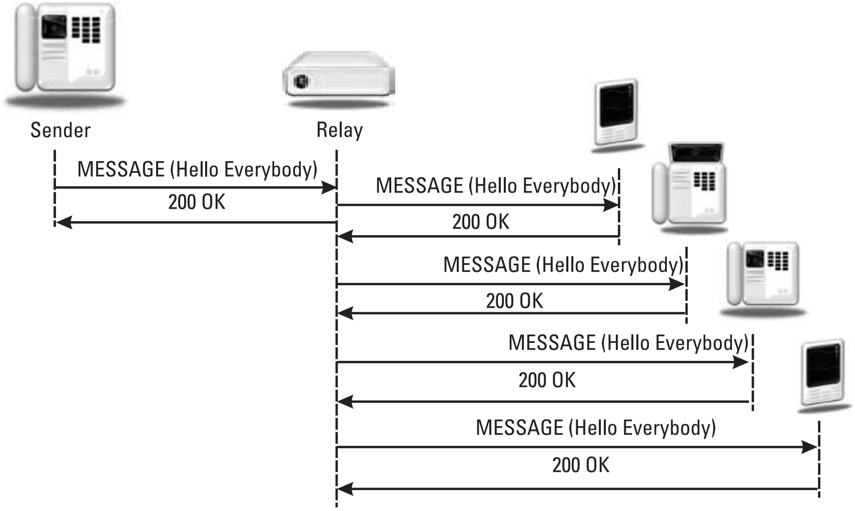

Sender
Relay
MESSAGE (Hello Everybody)
200 OK
MESSAGE (Hello Everybody)
200 OK
MESSAGE (Hello Everybody)
200 OK
MESSAGE (Hello Everybody)
200 OK
MESSAGE (Hello Everybody)
200 OK

Figure 8.9 Multiple recipient messages example.

```
MESSAGE sip:nancy-list@csiro.au SIP/2.0
Via: SIP/2.0/TCP unimelb.csiro.au;branch=z9hG4bKhs8asdLs83
Max-Forwards: 70
To: <sip:nancy-list@csiro.au>
From: Nancy Tyson Burbidge <sip:nancy.burbidge@csiro.au>
 ;tag=Kjd32331
Call-ID: 8uF2sflFSsa
CSeq: 98712 MESSAGE
Require: recipient-list-message
Content-Type: multipart/mixed;boundary=”boundary2”
Content-Length: 501

--boundary2
Content-Type: text/plain

Hello Everybody!

--boundary2
Content-Type: application/resource-lists+xml
Content-Disposition: recipient-list

<?xml version=”1.0” encoding=”UTF-8”?>
<resource-lists xmlns=”urn:ietf:params:xml:ns:resource-lists”
      xmlns:cp=»urn:ietf:params:xml:ns:copycontrol”>
 <list>
 <entry uri=”sip:caroline.atkinson@smh.com.au”
      cp:copyControl=”to” />
 <entry uri=”sip:egould@ramsgate.kent.uk” cp:copyControl=”to”
 <entry uri=”sip:gmolloy@botanicals.org” cp:copyControl=”cc”/>
 <entry uri=”sip:jasminb@clearwater.tx.us” cp:copyControl=”cc”
 </list>
</resource-lists>
 --boundary2--
```

This MESSAGE has a multipart/mixed body which consists of a Content-Type: text/plain instant message followed by a Content-Type: application/resource-lists+xml body, which contains a list. Note that this list format is the same as that used in SUBSCRIBE requests creating resource lists. The copyControl attribute is defined in [26], which allows for the values of to, cc, or bcc that have the same meaning as in e-mail.

8.5.6 Session Mode Instant Messaging

The Message Session Relay Protocol (MSRP) [30] was developed to meet the needs of a session mode IM system. For this case, the use of the SIP MESSAGE method was determined to be suboptimal. As a result, this is a non-SIP protocol. MSRP uses a SIP offer/answer exchange to establish the MSRP session using SDP. MSRP messages are either SEND requests, which carry a message or a REPORT request which reports on the status of a previous message. MSRP allows large messages to be sent by breaking the message into chunks which are sent in separate SEND requests. MSRP uses a stream-based transport such as TCP. As a result,

Table 8.7
MSRP Header Fields

Header	Specification
`To-Path`	RFC 4975
`From-Path`	RFC 4975
`Message-ID`	RFC 4975
`Success-Report`	RFC 4975
`Failure-Report`	RFC 4975
`Byte-Range`	RFC 4975
`Status`	RFC 4975
`Expires`	RFC 4976
`Min-Expires`	RFC 4976
`Max-Expires`	RFC 4976
`Use-Path`	RFC 4976
`WWW-Authenticate`	RFC 4976
`Authorization`	RFC 4976
`Authentication-Info`	RFC 4976

it must provide its own framing. MSRP defines its own URI scheme, which is used when negotiating the session. For example, an SDP offer used to establish an MSRP session:

```
v=0
o=nancy 2890844526 2890844527 IN IP4 unimelb.csiro.au
s=-
c=IN IP4 unimelb.csiro.au
t=0 0
m=message 7344 TCP/MSRP *
a=accept-types:text/plain
a=path:msrp://unimelb.csiro.au:27394/Kdsa2s93i9a;tcp
```

The media line contains the `message` media type while `TCP/MSRP` indicates that MSRP protocol will be used over TCP. An example MSRP exchange is shown in Figure 8.10.

MSRP has defined a number of header fields, listed in Table 8.7, and uses error codes shown in Table 8.8. Table 8.9 lists MSRP message types.

The use and operation of relay with MSRP is defined in [31]. These relays can be used for NAT traversal, logging, and corporate compliance. With MSRP relays, a mechanism for discovery and authentication is defined. An approach to establish MSRP sessions through NATs that uses standard ICE mechanisms is defined in [35]. The usage of MSRP in a multiuser chat is described in [34], which also defines a new MSRP request `NICKNAME` used to set a temporary name, known as a nickname, which can be used as an alias during the multiuser chat session. MSRP multiuser chat uses CPIM messages and the CPIM header to identify the recipient. The recipient can either be the entire chat room or it can

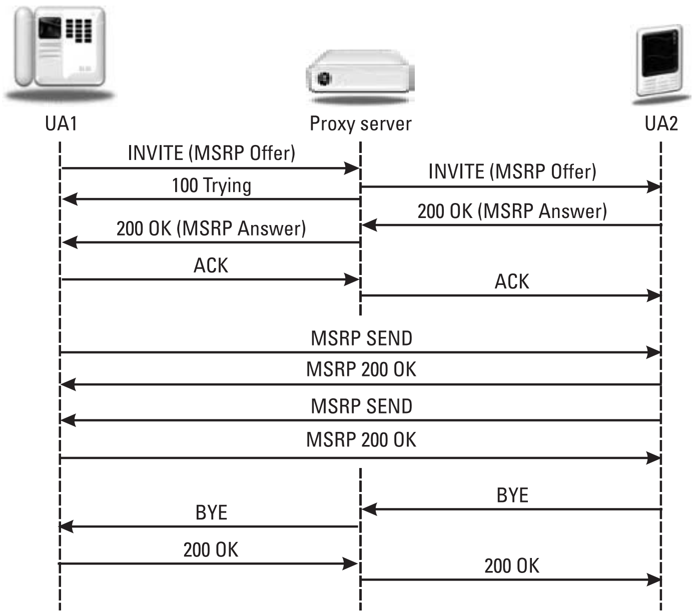


Figure 8.10 MSRP session example.

Table 8.8
MSRP Error Codes

Code	Meaning
`200`	Successful transaction
`400`	Request failed
`401`	Authorization credentials needed
`403`	Request not allowed
`408`	Timeout
`413`	Stop sending
`415`	Unknown media type
`423`	Parameter out of bounds
`481`	Session does not exist
`501`	Method not understood
`506`	Session already in use

be a single participant, in which case it is a private message addressed to a particular nickname.

Table 8.10 lists the common content types in instant messaging.

Table 8.9
MSRP Requests

Request	Specification
SEND	RFC 4975
REPORT	RFC 4975
AUTH	RFC 4976
NICKNAME	[34]

Table 8.10
Common Content-Types in IM

Content Type	Reference
text/plain	RFC 3676
text/html	RFC 2854
message/cpim	RFC 3826
application/resource-lists+xml	RFC 4862
application/im-iscomposing+xml	RFC 3994

8.6 Jabber

Jabber is the XML-based instant messaging and presence protocol invented by Jeremie Miller as an open source software project in 1998. The first public release was in 2000, and the protocol was developed by the not for profit Jabber Software Foundation (JSF), which was formed in 2001 and has been renamed to the XMPP Standards Foundation (XSF) [39].

8.6.1 Standardization as Extensible Messaging and Presence Protocol

Jabber was standardized in the IETF in 2004 as Extensible Messaging and Presence Protocol, or XMPP, a core protocol [40] and extensions for instant messaging and presence [41]. Like SIMPLE, Jabber supports CPIM messages for interoperability. The many extensions to XMPP have not been standardized in the IETF but instead standardized by the XSF as XMP extension protocols or XEPs. Jabber uses XML fragments, known as *stanzas* to communicate between Jabber clients and Jabber servers. Jabber is a true client/server protocol—Jabber servers are required elements in the architecture and use a server-to-server protocol to talk to other servers and a client-to-server protocol to talk to clients. For addresses, Jabber uses a Jabber ID (JID) in the form of `user@domain/resource`. Note that JID is not a URI or a URL.

8.6.2 Interworking with SIMPLE

Currently work is underway in the IETF to standardize the interworking between SIMPLE and Jabber for both presence and IM systems [42]. A gateway must be used to bridge the two protocols. Since XMPP utilizes permanent subscriptions, the gateway must map them to the soft-state subscriptions used in SIMPLE. While an XMPP URI scheme has been defined [43] that SIMPLE can use, XMPP is unable to use URIs so the gateway must map between XMPP or SIP URIs and XMPP JIDs. The basic IM mapping is shown in Table 8.11.

Table 8.12 shows basic presence mapping between XMPP and SIMPLE.

The mapping of group chat between SIMPLE and XMPP is defined in [44].

8.6.3 Jingle

Jingle [45] is an XMPP protocol extension for initiating and managing media sessions between two XMPP entities. Jingle provides a way to perform an offer/answer exchange of media capabilities to negotiate an RTP media session. Jingle was designed to easily interwork with SIP [46] and have NAT traversal using ICE, as described in Chapter 10.

8.6.4 Future Standardization of XMPP

A new working group has formed in the IETF to standardize additional aspects of XMPP [47] and interworking with SIMPLE. Their work will include revisions to the core protocol, optimizations for mobile devices, and end-to-end encryption and security.

8.7 Conclusion

This chapter showed how presence and instant messaging services can be implemented with SIP using a number of extensions. Additionally, Jabber or XMPP can provide these services and can interwork with SIP through gateways.

Table 8.11
Instant Message Mapping Between XMPP and SIMPLE

XMPP	SIMPLE
`<body/>`	Body of `MESSAGE`
`<subject/>`	`Subject`
`<thread/>`	`Call-ID`
`from`	`From`
`to`	`To`
`xml:lang`	`Content-Language`

Table 8.12
Presence Mapping Between XMPP and SIMPLE

XMPP	SIMPLE
`<presence/>` stanza	`Event: presence`
XMPP resource identifier	Tuple '`id`' attribute
`from`	`From`
`id`	`Call-ID`
`to`	`To`
`type`	Basic status
`xml:lang`	`Content-Language`
`<priority/>`	PIDF `priority` for tuple

8.8 Questions

Q8.1 Create an example resource list creating subscription, and show the first notification. Include at least three resources in the list and show at least two different presence states in the notification.

Q8.2 Explain the differences between page mode and session mode instant messaging. Which protocols are typically used for each?

Q8.3 Explain the purpose of CPIM.

Q8.4 Explain why you would not expect to see the SIP header field `Content-Type: message/imdn+xml` as a header field in a SIP MESSAGE method.

Q8.5 Map the following Jabber message into a SIP NOTIFY. Make up any URIs and values you might need:

```
<presence
  from='juliet@example.com/balcony'
  to='romeo@example.net/orchard'
  xml:lang='en'>
 <show>away</show>
 <status>be right back</status>
 <priority>0</priority>
</presence>
```

Q8.6 Create a presence document for the user Bob showing his status as available, his contact URI as sip:bob@pc33.example.com, and his device as capable of both audio and video.

Q8.7 How are entity tags used in SIP?

Q8.8 Explain how partial publication works with an example.

Q8.9 What is a resource list server?

Q8.10 Explain conditional event notification.

References

[1] http://www.icq.com.

[2] http://www.aim.com.

[3] Rosenberg, J., "SIMPLE made Simple: An Overview of the IETF Specifications for Instant Messaging and Presence Using the Session Initiation Protocol (SIP)," draft-ietf-simple-simple-05 (work in progress), March 2009.

[4] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification," RFC 3265, June 2002.

[5] Niemi, A., "Session Initiation Protocol (SIP) Extension for Event State Publication," RFC 3903, October 2004.

[6] Sugano, H., et al., "Presence Information Data Format (PIDF)," RFC 3863, August 2004.

[7] Rosenberg, J., "A Presence Event Package for the Session Initiation Protocol (SIP)," RFC 3856, August 2004.

[8] Rosenberg, J., "A Watcher Information Event Template-Package for the Session Initiation Protocol (SIP)," RFC 3857, August 2004.

[9] Rosenberg, J., "An Extensible Markup Language (XML) Based Format for Watcher Information," RFC 3858, August 2004.

[10] Schulzrinne, H., et al., "RPID: Rich Presence Extensions to the Presence Information Data Format (PIDF)," RFC 4480, July 2006.

[11] Schulzrinne, H., "CIPID: Contact Information for the Presence Information Data Format," RFC 4482, July 2006.

[12] Rosenberg, J., "A Data Model for Presence," RFC 4479, July 2006.

[13] Roach, A., B. Campbell, and J. Rosenberg, "A Session Initiation Protocol (SIP) Event Notification Extension for Resource Lists," RFC 4662, August 2006.

[14] Khartabil, H., et al., "An Extensible Markup Language (XML)-Based Format for Event Notification Filtering," RFC 4661, September 2006.

[15] Rosenberg, J., "The Extensible Markup Language (XML) Configuration Access Protocol (XCAP)," RFC 4825, May 2007.

[16] Rosenberg, J., "Extensible Markup Language (XML) Formats for Representing Resource Lists," RFC 4826, May 2007.

[17] Isomaki, M., and E. Leppanen, "An Extensible Markup Language (XML) Configuration Access Protocol (XCAP) Usage for Manipulating Presence Document Contents," RFC 4827, May 2007.

[18] Rosenberg, J., "Presence Authorization Rules," RFC 5025, December 2007.

[19] Lonnfors, M., and K. Kiss, "Session Initiation Protocol (SIP) User Agent Capability Extension to Presence Information Data Format (PIDF)," RFC 5196, September 2008.

[20] Urpalainen, J., "An Extensible Markup Language (XML) Patch Operations Framework Utilizing XML Path Language (XPath) Selectors," RFC 5261, September 2008.

[21] Lonnfors, M., et al., "Presence Information Data Format (PIDF) Extension for Partial Presence," RFC 5262, September 2008.

[22] Lonnfors, M., et al., "Session Initiation Protocol (SIP) Extension for Partial Notification of Presence Information," RFC 5263, September 2008.

[23] Niemi, A., M. Lonnfors, and E. Leppanen, "Publication of Partial Presence Information," RFC 5264, September 2008.

[24] Peterson, J., "Address Resolution for Instant Messaging and Presence," RFC 3861, August 2004.

[25] Niemi, A., "An Extension to Session Initiation Protocol (SIP) Events for Conditional Event Notification," draft-ietf-sipcore-subnot-etags-20 (work in progress), April 2009.

[26] Garcia-Martin, M., and G. Camarillo, "Extensible Markup Language (XML) Format Extension for Representing Copy Control Attributes in Resource Lists," RFC 5364, October 2008.

[27] Campbell, B., et al., "Session Initiation Protocol (SIP) Extension for Instant Messaging," RFC 3428, December 2002 .

[28] Schulzrinne, H., "Indication of Message Composition for Instant Messaging," RFC 3994, January 2005.

[29] Garcia-Martin, M., and G. Camarillo, "Multiple-Recipient MESSAGE Requests in the Session Initiation Protocol (SIP)," RFC 5365, October 2008.

[30] Campbell, B., R. Mahy, and C. Jennings, "The Message Session Relay Protocol (MSRP)," RFC 4975, September 2007.

[31] Jennings, C., R. Mahy, and A. Roach, "Relay Extensions for the Message Sessions Relay Protocol (MSRP)," RFC 4976, September 2007.

[32] Klyne, G., and D. Atkins, "Common Presence and Instant Messaging (CPIM): Message Format," RFC 3862, August 2004.

[33] Burger, E., and H. Khartabil, "Instant Message Disposition Notification (IMDN)," RFC 5438, February 2009.

[34] Niemi, A., M. Garcia, and G. Sandbakken, "Multi-Party Chat Using the Message Session Relay Protocol (MSRP)," draft-ietf-simple-chat-04 (work in progress), March 2009.

[35] Holmberg, C., and S. Blau, "An Alternative Connection Model for the Message Session Relay Protocol (MSRP)," draft-ietf-simple-msrp-acm-01 (work in progress), August 2009.

[36] Camarillo, G., A. Roach, and O. Levin, "Subscriptions to Request-Contained Resource Lists in the Session Initiation Protocol (SIP)," RFC 5367, October 2008.

[37] Levin, O., "Ad-Hoc Resource Lists Using SUBSCRIBE in SIMPLE," draft-levin-simple-adhoc-list-01.txt (work in progress), November 2003.

[38] Khartabil, H., et al., "Functional Description of Event Notification Filtering," RFC 4660, September 2006.

[39] http://www.xsf.org.

[40] Saint-Andre, P., (ed.), "Extensible Messaging and Presence Protocol (XMPP): Core," RFC 3920, October 2004.

[41] Saint-Andre, P., (ed.), "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence," RFC 3921, October 2004.

[42] Saint-Andre, P., "Basic Messaging and Presence Interworking Between the Extensible Messaging and Presence Protocol (XMPP) and Session Initiation Protocol (SIP) for Instant Messaging and Presence Leveraging Extensions (SIMPLE)," draft-saintandre-xmpp-simple-10 (work in progress), August 2007.

[43] Saint-Andre, P., "Internationalized Resource Identifiers (IRIs) and Uniform Resource Identifiers (URIs) for the Extensible Messaging and Presence Protocol (XMPP)," RFC 4622, July 2006.

[44] Saint-Andre, P., S. Loreto, and F. Forno, "Interworking Between the Session Initiation Protocol (SIP) and the Extensible Messaging and Presence Protocol (XMPP): Multi-Party Text Chat," draft-saintandre-sip-xmpp-groupchat-01 (work in progress), March 2009.

[45] Ludwig, S., et al., "Jingle," XSF XEP 0166, June 2007.

[46] Saint-Andre, P., "Interworking Between the Session Initiation Protocol (SIP) and the Extensible Messaging and Presence Protocol (XMPP): Media Sessions," draft-saintandre-sip-xmpp-media-01.txt (work in progress), March 2009.

[47] http://www.ietf.org/html.charters/xmpp-charter.html.

9

Services in SIP

In this chapter, ways and techniques to implement services using SIP will be covered. Typical services include telephony gateway services, trunking, business services, voicemail, conferencing, fax, and video. Architectures such as application sequencing, service oriented architecture (SOA), and service delivery platforms will also be introduced.

9.1 Gateway Services

A common application of SIP is in PSTN interworking and replacement. For this application, the most important function is that of a gateway to the PSTN. A gateway is an element that converts one protocol to another—a PSTN gateway converts a SIP and RTP session into a PSTN session. As SIP is a signaling protocol, the gateway will map SIP messages to a PSTN signaling protocol such as ISDN (Integrated Services Digital Network) or ISUP (ISDN User Part). The RTP media session is converted by a gateway into a PCM trunk. Gateways are often decomposed using media gateway protocols such as MGCP and H.248. For more details on these PSTN protocols, see Chapter 11.

We have already seen how SIP can handle telephone numbers, either through a telephony URI [1] or SIP URI [2] with a telephone number in the user part. A SIP proxy server can determine when an INVITE needs to be routed to a gateway for termination in the PSTN. Other types of SIP requests, (e.g., presence, instant messaging, and so fourth) do not make sense to route to the PSTN. A proxy can also manage a *dial plan* for a set of UAs. A dial plan maps a dial string (digits dialed on a telephone) into a telephone number suitable for routing on the PSTN. For example, “dial 9 for an outside line” is an example of a very simple dial plan commonly used in business telephone systems. Dial plans also allow private dialing plans within an enterprise. An example of this is when 3 or 4 digit extensions can be used to dial other extensions within the enterprise. A dial string is indicated in a SIP URI when the `user=dialstring` [3] parameter is present.

A gateway receives incoming `INVITE`s and maps the telephone number digits, which are then mapped to the called party number in the PSTN signaling. Common gateway `INVITE` mappings and interworking are defined in [4].

Calls from SIP to the PSTN through a gateway often make use of early media. Early media is RTP media sent prior to the call being answered. In SIP, this means media sent prior to the `200 OK` response. This is usually done in SIP by the gateway sending a `183 Session Progress` response, which creates an early dialog. RTP media is then sent from the gateway to the UA. Often early media carries special ringback tones to recorded announcements and tones. This is shown in Figure 9.1.

In the other direction, a telephone number dialed in the PSTN can be routed and answered by a SIP UA. PSTN routing is done by assigning telephone numbers to telephone switches. The telephone switch then sends the call to the copper loop or PBX trunk that causes the telephone to ring. For a SIP UA to ring, the SIP/PSTN gateway must appear to the PSTN as a local telephone switch. When the call is routed to the gateway, instead of sending the call to the copper loop or PBX trunk, the gateway creates an `INVITE` and routes it to a UA on an IP network. The `INVITE` can be routed to a SIP proxy server to determine the appropriate contact device.

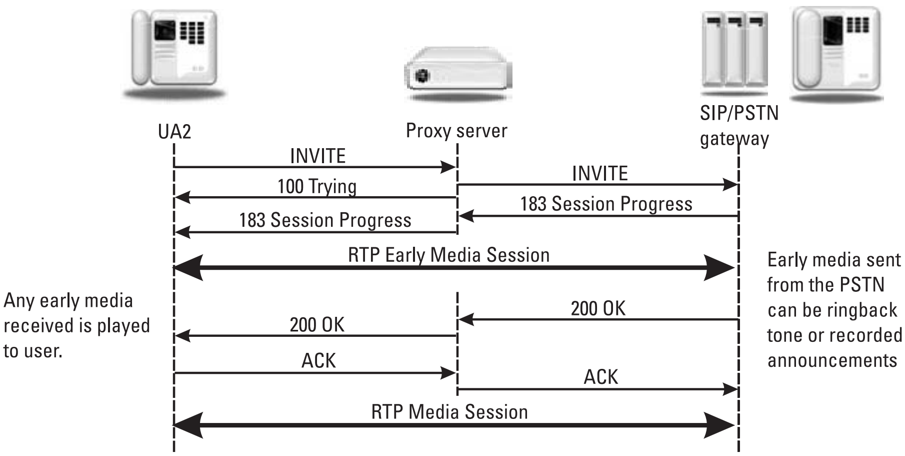


Figure 9.1 Early media in SIP.

9.2 SIP Trunking

In the PSTN, a trunk is a dedicated connection between PSTN switches or between a PSTN switch and a Private Branch Exchange (PBX). A trunk has both signaling and media parts. Trunks in the PSTN originally used one or two pairs of copper wires. With digital trunking and time division multiplexing, the T-1 became the standard trunk in North America with 24 voice circuits multiplexed over two pairs of wires. The E-1 was the European version with 32 voice circuits. With ISDN, the T-1 trunk became a primary rate interface (PRI), which also had 24 circuits. While there is no exact analog in SIP and Internet communications to a trunk, a SIP trunk usually means a connection between SIP telephony devices which uses IP transport and involves RTP media transport. The SIP forum [5] has published a technical recommendation called SIPconnect [6], which describes how SIP and RTP can be configured to provide SIP trunking between a service provider and an enterprise PBX. There are some similarities between SIP trunks and PSTN trunks:

- The interconnection requires configuration on both sides to work.
- VoIP and telephony services can be delivered over it.
- Telephone numbers or E.164 numbers are the identifiers used.

There are some important differences, however:

- SIP trunks do not have an inherent capacity—they are limited only by the bandwidth of the underlying IP transport and the call setup capacity of the SIP servers.
- Media quality can be better than PSTN quality.
- SIP trunks will be extended in the future to provide presence, IM, and multimedia capabilities.

SIPconnect is not a profile of the SIP protocol. Instead, it is a profile of the set of SIP-related protocols. For example, it references the following SIP standards shown in Table 9.1. These references cover SIP as well as identity and security, and RTP media. Figure 9.2 shows a typical use of SIP trunks between an enterprise IP PBX and a service provider.

9.3 SIP Service Examples

The SIP service examples document [7] shows examples of how common PBX, centrex, and business telephony features can be implemented using SIP. The

Table 9.1
SIPconnect Trunking Specifications

Signaling		
	RFC 3261	Core SIP
	RFC 3264	SIP offer/answer protocol
	RFC 3263	SIP DNS usage
	RFC 3265	SIP events
	RFC 3311	SIP UPDATE method
	RFC 3323	Privacy mechanism for SIP
	RFC 3325	SIP network asserted identity
	RFC 3725	Third part call control
	RFC 4028	Session timer
	RFC 2327	SDP
	RFC 2782	DNS SRV
	RFC 3262	PRACK
	RFC 3311	UPDATE
	RFC 3325	P-asserted-identity
	E.164	Telephone numbers
	RFC 3761	ENUM
Media		
	RFC 3550	RTP
	RFC 3581	Symmetric SIP
	RFC 3966	Telephony URI
	RFC 4028	SIP session timer
	RFC 2833	Telephone events for DTMF
	T.38	Fax
	G.168	ITU-T digital echo cancellation
	G.711	ITU-T PCM coded
NAT Traversal		
	RFC 3489	STUN
	RFC 3581	rport
Security		
	RFC 2246	TLS
QoS		
	RFC 2474	DiffServ QoS

features discussed are listed in Table 9.2. For some features, the IETF has developed more detailed best current practice documents such as for call transfer [8], automatic call completion [9], and bridged line appearance/multiple line appearance [10].

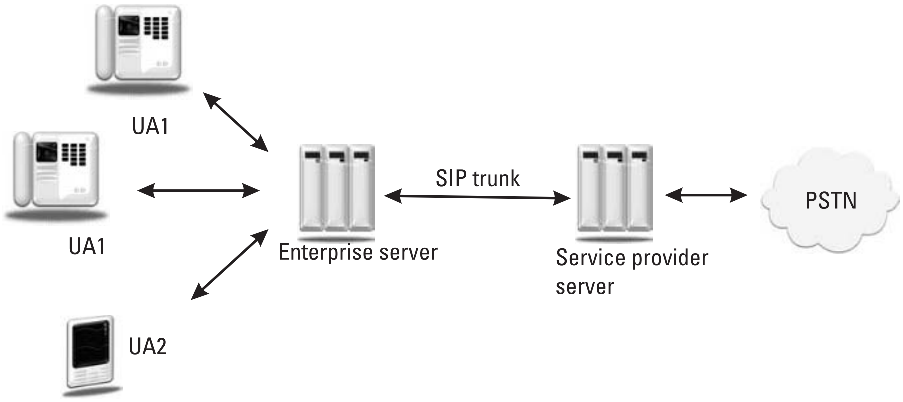


Figure 9.2 SIP trunk between enterprise and service provider.

Table 9.2
SIP Service Examples Features

Call hold
Consultation hold
Music on hold
Transfer—unattended
Transfer—attended
Transfer—instant messaging
Call forwarding unconditional
Call forwarding—busy
Call forwarding—no answer
Three-way conference
Find-me
Call management (incoming call screening)
Call management (outgoing call screening)
Call park
Call pickup
Automatic redial
Click to dial

9.4 Voicemail

Voicemail is a messaging service commonly associated with telephony applications. It can be implemented as a service in the network (such as that provided by mobile phone providers), in a separate device, such as a home answering machine, or incorporated in a telephony device, such as an enterprise PBX or key system. The service involves call forwarding no answer/busy/unavailable to a storage device which plays a customizable greeting and plays a greeting. The user is then alerted by some means that a message is waiting, and can then retrieve the message by dialing into the system. With unified messaging systems, it is becoming increasingly common for a voicemail to be retrievable by other means such as an audio file attachment to an e-mail or a speech recognition system that generates an e-mail or text message of the contents.

In SIP terms, the call forwarding is straightforward, with either a proxy forwarding or endpoint redirection (3xx response) used to send the call to the voicemail server. However, some kind of SIP extension is needed to indicate to the voicemail system which mailbox to use—that is, which greeting to play and where to store the recorded message. There are two main ways to do this. One is to use the Request-URI to signal this information while the other is to use a SIP header field extension. For the Request-URI method, the voicemail URI parameters [11] approach is used. For a voicemail system at `sip:voicemail.example.com,` which is being used to provide voicemail for `sip:alice@example.com`, the Request-URI of the `INVITE` when it is forwarded to the voicemail server could look like:

```
sip:voicemail.example.com;target=sip:alice@example.com;cause=486
```

In this way, the Request-URI carries the mailbox identifier as well as the reason the call is being forwarded to voicemail. This is important if the voicemail system plays different prompts depending on if the user is on the phone or doesn't answer, for example. For the `cause` parameter, the following values are standardized in Table 9.5. The feature is shown in Figure 9.3.

There are two approaches for header field approach. One is a usage of the `History-Info` header field [12] while the other is the unstandardized `Diversion` header field. In this case, the `History-Info` is populated by the proxy server and included when the `INVITE` is forwarded to the voicemail server. The Request-URI is set to the voicemail server URI. Note that the `History-Info` approach pro-

Table 9.3
Voicemail URI Cause Parameter Values

Meaning	Cause Value
Unknown/not available	404
User busy	486
No reply	408
Unconditional	302
Deflection during alerting	487
Deflection immediate response	480
Mobile subscriber not reachable	503

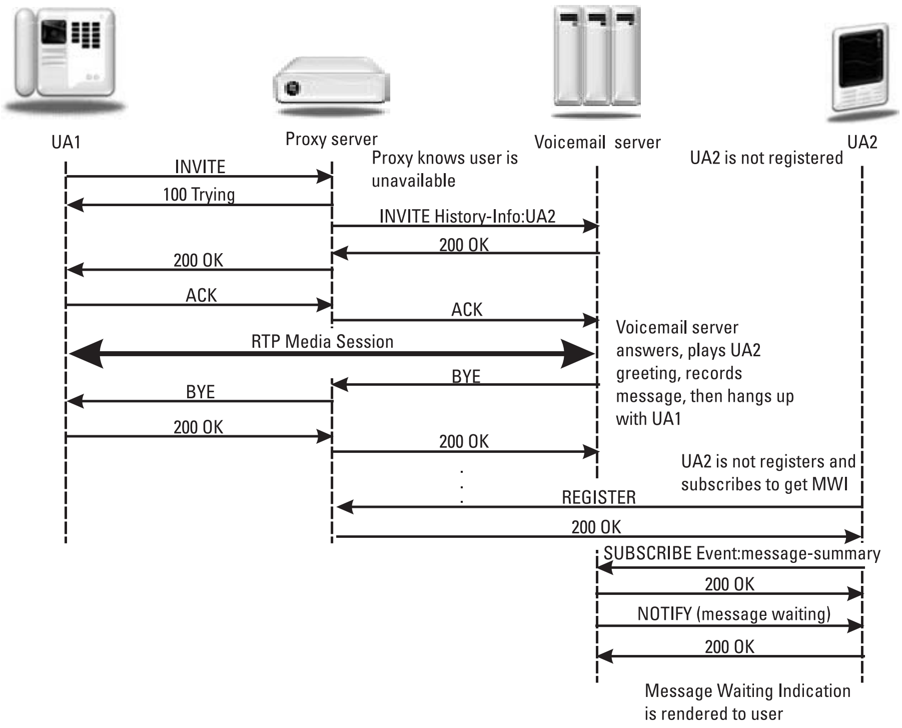

UA1
Proxy server
Voicemail server
UA2
Proxy knows user is unavailable
UA2 is not registered
INVITE
100 Trying
INVITE History-Info:UA2
200 OK
200 OK
ACK
ACK
RTP Media Session
Voicemail server answers, plays UA2 greeting, records message, then hangs up with UA1
BYE
BYE
200 OK
200 OK
UA2 is not registers and subscribes to get MWI
REGISTER
200 OK
SUBSCRIBE Event:message-summary
200 OK
NOTIFY (message waiting)
200 OK
Message Waiting Indication is rendered to user

Figure 9.3 SIP voicemail call flow example.

vides more information than the voicemail URI approach and allows additional policies to be applied.

The other header field approach is the `Diversion` header field [13]. Note that while this header field is widely deployed, it has never been standardized by the IETF.[1] An example of the diversion header field is:

```
Diversion: <sip:alice@example.com>;reason=no-answer
```

This approach directly maps ISDN diversion values, simplifying PSTN interworking. For message notification, SIP events along with the message waiting indicator (MWI) event package can be used.

9.5 SIP Video

Establishing video sessions with SIP is essentially orthogonal to the SIP protocol itself—nothing really needs to be changed, even in the offer/answer SDP ex-

[1] The expired draft can be found by searching on the IETF tools site and other mirror sites, for example, http://tools.ietf.org/id/draft-levy-sip-diversion-08.txt.

change. However, there are some features and services specific to video that often come up when SIP video is discussed.

For example, consider FastUpdate. This is a signaling message sent by a video mixer to a video sender in a multimedia conference. When switching between video sources, the FastUpdate message indicates that the sender must send a key frame (I-frame) so that subsequent P-frames and B-frames have a reference. One method uses an XML object carried in a SIP request, defined in [14]. Another approach is to use RTCP, defined in [15] using the Full Intra Request command.

Another type of SIP video is "dual video" where multiple video streams are sent—typically a main video and a secondary video. The far end has the ability to switch between the two videos. These are also known as "people and content" streams. Some initial work is underway to standardize this feature for SIP [16] and define interworking with H.239 [17]. Current proposals to standardize this feature involve the use of the Binary Floor Control Protocol (BFCP) [18]. SDP media content and label attributes are also proposed for this feature.

Bandwidth management is more critical with video than audio, and being able to allocate bandwidth between both audio and video streams for best performance is important. Another useful tool is the ability to signal the size of the video and preferred frame rate received [19].

9.6 Facsimile

Facsimile or fax is a common PSTN application. Fax is the telecommunications service for sending copies of documents across the PSTN. Fax machines have scanners to read a document, a modem to encode the digital information over the telephone line, and a printer to output received pages. Fax servers are also used today that send and receive faxes over PSTN trunks but offer alternative output including e-mail, PDF, and text-to-speech. Faxes can be implemented in Internet communications in a number of different ways including simple transport to native implementation. Faxes can be transported using G.711 or another noncompressed codec, although delays and latency can cause problems and failures. Sending faxes directly over IP networks can be done using the T.38 [20] fax over IP standard. The MIME type `image/t38` [21] has been defined for the transport of faxes over UDP, using the UDP Transport Layer (UDPTL). While this approach is efficient, it is better to transport faxes over RTP. This allows faxes to utilize various RTP extensions including security, redundancy, and so fourth. Faxes over RTP use the `audio/t38` [22] MIME type. As such, the SIP offer/answer exchange is needed to negotiate a fax session. For example, the following media line could be used, as will be discussed in Chapter 13:

```
m=audio 38202 RTP/AVP 96
a=rtpmap:96 t38/8000
a=fmtp:96 T38FaxVersion=2;T38FaxRateManagement=transferredTCF
```

9.7 Conferencing

SIP conferencing is an important application of SIP. The ability to provide a better experience than today's conventional PSTN teleconferences is an important driver and differentiator for SIP services and endpoints. While SIP conferencing can use the same model as the PSTN, where a centralized mixer/conference bridge accepts media from each participant and mixes the resulting media, there are improvements possible with SIP. Some of the alternative topologies, uses, and applications are described in the SIP conferencing framework document [23]. The resulting SIP extensions and best current practices are described in RFC 4579 known as call control for conferencing [24].

9.7.1 Focus

The centralized point of control for a SIP conference is known as a *focus.* The focus performs SIP authentication and authorization on behalf of all conference participants, and controls the media mixing. Note that this is not exactly the same thing as saying the focus performs media mixing. In some cases, the focus can cause participants in the conference to perform their own mixing if there is a full mesh of media streams among participants in the conference. However, mixing control is done by the focus. The focus also provides information about participants in the conference: their identities and capabilities. This information can be shared with participants in the conference. One SIP method of doing this is using the SIP conference event package [25]. A participant in a conference sends a SUBSCRIBE to the focus, which creates a subscription, resulting in NOTIFYs sent containing information about the conference.

A conference focus uses the isfocus feature tag to indicate that a particular dialog is associated with a conference. This allows a UA to automatically learn that a given point-to-point call is actually part of a conference. This allows behavior such as:

- Use of a conference-specific user interface;
- Automatic subscription for conference events;
- Use of conferencing call control functions.

Some of the conferencing call control functions are described in [24]. The call flows described in this document are listed in Table 9.4.

An example call flow is shown in Figure 9.4 where a conference is created, joined, and two other participants added, using automated SIP approaches.

9.7.2 Mixer

The mixer creates the media combination that allows each party to participate in the group session. For audio, this means mixing in the loudest speakers at any instant. For video, it can mean showing the video of the loudest speaker, or combining all video pictures into a single screen. For text, it can mean sharing all text messages typed by each participant with attribution. The mixer is under the control of the focus, and often it will be a part of the focus.

9.7.3 Non-SIP Conference Control

In addition to the SIP call control means described in [24], the IETF has standardized other conference control protocols. For example, the ability to send media to the group (i.e., act as the presenter) can been controlled as a floor, with only a single floor holder at a time. A floor control protocol provides a method for requesting and granting the floor. The Binary Floor Control Protocol (BFCP) [26] has been developed to enable shared access to a resource within a conference. The Centralized Conference Manipulation Protocol (CCMP) [27] has been defined to create, schedule, and define media types and resources for a SIP conference. CCMP also supports conference control operations during a conference such as muting a participant, and removing or adding a participant. CCMP supports multimedia conferences including voice, video, and text sessions, and can support a full set of multiuser chats. CCMP, along with SIP conference aware UAs and a focus, allows a full standards-based conferencing system to be built.

Table 9.4

Call Control for Conferencing Functions

Joining a conference, dial out and dial in
Creating a conference, manually and automatically
Adding or deleting a participant
Requesting a participant join a conference
Switching UAs during a conference
Transferring a point-to-point session into a conference
Deleting a conference
Discovery of a conference

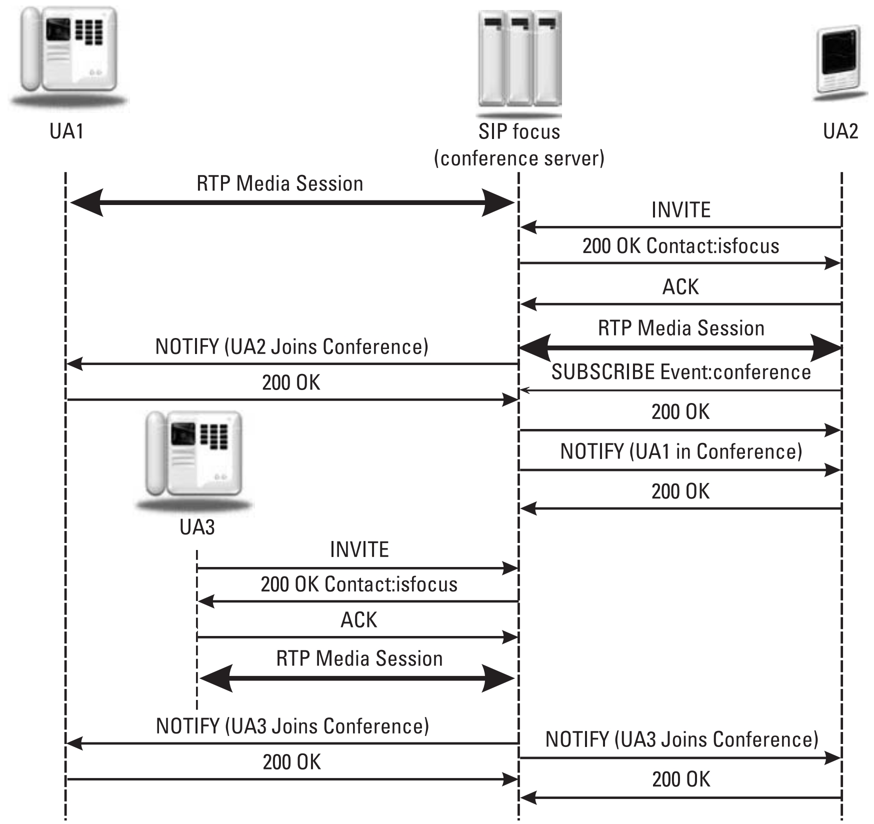


Figure 9.4 SIP conferencing call flow.

9.8 Application Sequencing

Application sequencing is an approach to delivering services using SIP that has been made popular by the IP Multimedia Subsystem (IMS) developed by 3GPP introduced in Chapter 7. Application sequencing uses a proxy server to selectively route a SIP request to multiple application servers, with each application server providing a feature or service. For example, consider Figure 9.5, which shows an example of application sequencing. The SIP session is established between the two SIP UAs, which are in different domains. Each domain has a service controller, which invokes originating or terminating features on behalf of the calling or called party.

The originating service controller sequences through three application servers. The first two are pure SIP elements, while the third receives an RTP stream. For example, the first application server could be a billing application, the second an authentication application, and the third a recording application. After the request has sequenced through all three application servers, the service

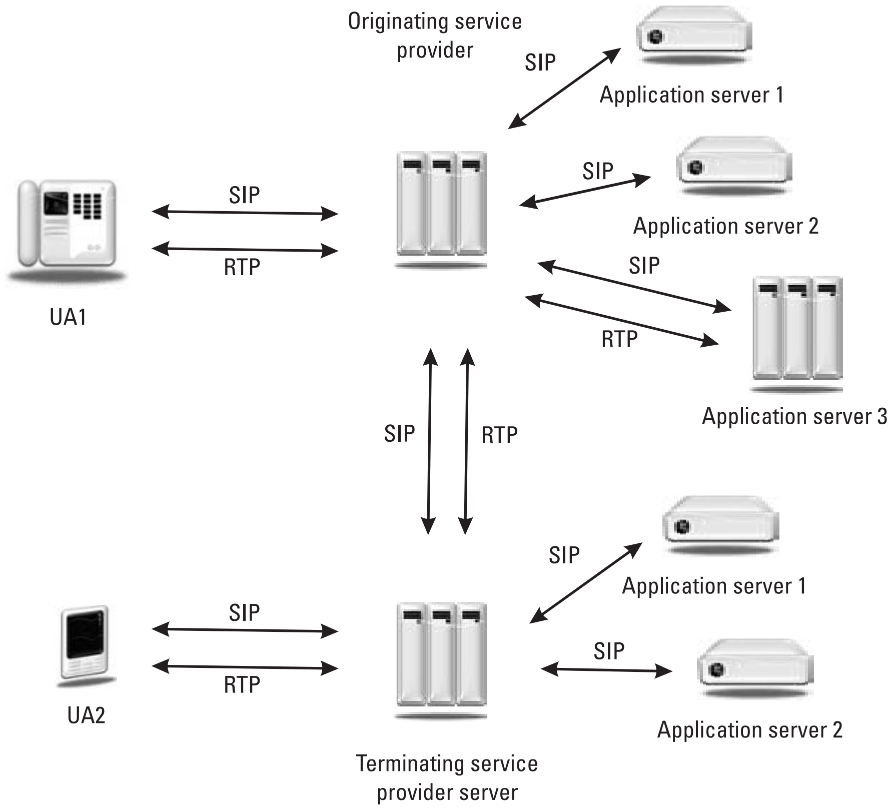


Figure 9.5 Application server sequencing example.

controller forwards the request to the other service controller, which then sequences the request through two application servers to implement features on behalf of the called UA. Finally, the request is forwarded to the UA and the session established. Each application server can choose to either stay out of subsequent dialog signaling (by redirecting or not including a `Record-Route` URI) or to stay in the path and even act as a B2BUA. While this architecture can provide many services in SIP, it cannot provide services that utilize new media types or require a new user interface. For these and many other SIP features, the features must be implemented in the endpoint UAs.

9.9 Other SIP Service Architectures

This section will briefly introduce a number of service architectures that can be supported with SIP including service oriented architecture (SOA), servlets, and, the service delivery platform (SDP).

9.9.1 Service Oriented Architecture

Service oriented architecture (SOA) is architecture for integrating communications into business processes. SOA uses principles and ideas from object-oriented programming. SOA uses a function known as orchestration, which defines how services are linked together. The current preference with SOA is to have a number of roles including the service provider, service broker, and service requestor. A number of companies and enterprises are currently experimenting with integrating SIP into SOA.

9.9.2 Servlets

Servlets are a Java programming method used to provide services in servers. For example, Figure 9.6 shows how a SIP UA can access SIP servlet services through a proxy server. Servlets are not scripts but actual Java function calls. SIP servlets, defined as version 1.1 in [28] provide basic SIP functions. An excellent reference for understanding and implementing SIP servlets is in [29].

9.9.3 Service Delivery Platform

The service delivery platform (SDP) is architecture for developing and deploying network-based services. The major components include a service creation en-

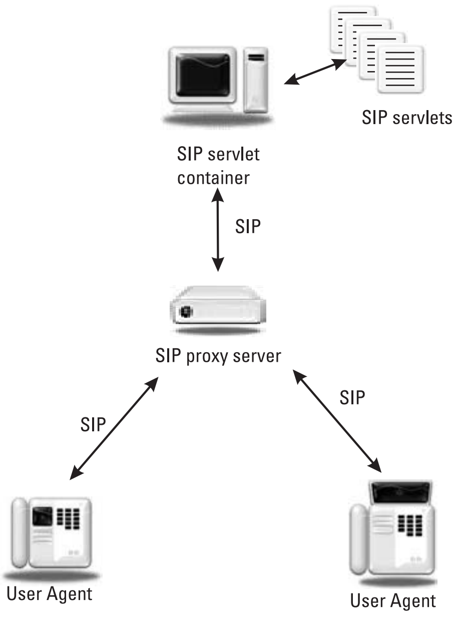


Figure 9.6 SIP services thorough servlets.

vironment, execution environment, media, control, presence/location services, and integration.

9.10 Conclusion

This chapter has discussed a number of common SIP services and features and some common architectures for delivering those services.

9.11 Questions

Q9.1 Compare and contrast SIP trunks and PSTN trunks.

Q9.2 Give the role of a focus in SIP and explain how to identify a focus using SIP signaling messages.

Q9.3 Generate a complete SIP call flow for a call between two UAs with one sequencing application server in between that sequences between two SIP-only application servers (no media).

Q9.4 What are the two methods of setting up a SIP call for sending faxes?

Q9.5 Explain what SIP extensions are needed for voicemail, and why they are needed.

References

[1] Schulzrinne, H., "The tel URI for Telephone Numbers," RFC 3966, December 2004.

[2] Rosenberg, J., H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261, June 2002.

[3] Rosen, B., "Dial String Parameter for the Session Initiation Protocol Uniform Resource Identifier," RFC 4967, July 2007.

[4] Camarillo, G., et al., "Integrated Services Digital Network (ISDN) User Part (ISUP) to Session Initiation Protocol (SIP) Mapping," RFC 3398, December 2002.

[5] http://sipforum.org.

[6] http://sipforum.org/sipconnect.

[7] Johnston, A., et al., "Session Initiation Protocol Service Examples," BCP 144, RFC 5359, October 2008.

[8] Sparks, R., and A. Johnston, "Session Initiation Protocol Call Control—Transfer," RFC 5589, June 2009.

[9] Worley, D., M. Huelsemann, and R. Jesske, "Call Completion for Session Initiation Proto col (SIP)," draft-ietf-bliss-call-completion-03 (work in progress), December 2008.

[10] Johnston, A., M. Soroushnejad, and V. Venkataraman, "Shared Appearances of a SessionInitiation Protocol (SIP) Address of Record (AOR)," draft-ietf-bliss-shared-appearances-03 (work in progress), July 2009.

[11] Jennings, C., F. Audet, and J. Elwell, "Session Initiation Protocol (SIP) URIs for Applications such as Voicemail and Interactive Voice Response (IVR)," RFC 4458, April 2006.

[12] Barnes, M., "An Extension to the Session Initiation Protocol (SIP) for Request History Information," RFC 4244, November 2005.

[13] Levy, S., J. Yang, and B. Byerly, "Diversion Indication in SIP," draft-levy-sip-diversion-08 (work in progress), August 2004.

[14] Levin, O., R. Even, and P. Hagendorf, "XML Schema for Media Control," RFC 5168, March 2008.

[15] Wenger, S., et al., "Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF)," RFC 5104, February 2008.

[16] Even, R., "People and Content Video Streams," draft-even-xcon-pnc-02 (work in progress), March 2007.

[17] International Telecommunication Union, "Role Management and Additional Media Channels," ITU-T Recommendation H.239, July 2003.

[18] Camarillo, G., J. Ott, and K. Drage, "The Binary Floor Control Protocol (BFCP)," RFC 4582, November 2006.

[19] Johansson, I., and K. Jung, "Negotiation of Generic Image Attributes in SDP," draft-ietf-mmusic-image-attributes-02 (work in progress), April 2009.

[20] ITU-T Recommendation T.38, "Procedures for Real-Time Group 3 Facsimile Communication over IP networks," April 2002.

[21] Parsons, G., "Real-Time Facsimile (T.38)—Image/t38 MIME Sub-Type Registration," RFC 3362, August 2002.

[22] Jones, P., and H. Tamura, "Real-Time Facsimile (T.38)—Audio/t38 MIME Sub-Type Registration," RFC 4612, August 2006.

[23] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol (SIP)," RFC 4353, February 2006.

[24] Johnston, A., and O. Levin, "Session Initiation Protocol (SIP) Call Control—Conferencing for User Agents," BCP 119, RFC 4579, August 2006.

[25] Rosenberg, J., H. Schulzrinne, and O. Levin, "A Session Initiation Protocol (SIP) Event Package for Conference State," RFC 4575, August 2006.

[26] Camarillo, G., J. Ott, and K. Drage, "The Binary Floor Control Protocol (BFCP)," RFC 4582, November 2006.

[27] Barnes, M., et al., "Centralized Conferencing Manipulation Protocol," draft-ietf-xcon-ccmp-03 (work in progress), July 2009.

[28] "SIP Servlet v1.1," Java Specification Request JSR-289, August 2008.

[29] Boulton, C., and K. Gronowski, *Understanding SIP Servlets 1.1*, Norwood, MA: Artech House, 2009.

10

Network Address Translation

This chapter introduces Network Address Translation (NAT) and looks at the issues and challenges involved in making SIP and other Internet communications protocols work through them. The motivations for, advantages, and disadvantages of NAT are discussed. NAT classification terminology is introduced. Techniques for NAT traversal such as hole punching and protocols such as STUN, ICE, and TURN are introduced. SIP and SDP extensions enabling NAT traversal are discussed.

First, here is a word about terminology, which can be a bit confusing in this chapter. While NAT usually stands for network address translation, the function of converting or mapping an inside IP address and port to an outside IP address and port, it also has another meaning as a network address translator, the device which performs this function. Which meaning is used can usually be determined by context. In this chapter, I will mainly use the abbreviation for the function, but there are times when it is common terminology to use it for the device. NAT is also sometimes used as a verb, that is, to NAT is to change the IP address and ports of packets on packets as they pass on the wire. This is also known as *NATting*. An address which has been changed by a NAT is also known as a *NATed* address. In this chapter, two other protocols also have an abbreviation used to represent two different things—both STUN and TURN protocols have two different meanings.

10.1 Introduction to NAT

Network address translation (NAT) is used to interconnect IP networks that use different IP address types. For example, in Chapter 1, the concept of private address spaces was introduced. Typically, NAT is used to map an inside address to an outside address. NAT operates at Layer 3 in the Internet protocol stack. Network address and port translation (NAPT) also change the port number in IP packets, operating at Layers 3 and 4. Typically most NAT also perform NAPT, but are still referred to simply as NAT. The reasons for implementing NAT will be discussed in the following sections, but the main reason has to do with the shortage of IPv4 addresses.

The earliest discussion of NAT in the IETF is RFC 1631 [1], which discusses the pros and cons of using NAT. The need for private IP address spaces resulted in the publication of RFC 1918 [2] in 1996 by the IETF, which reserved three IPv4 address blocks. These addresses are not routable on the public Internet—they only have meaning within a private network. RFC 2663 [3] defined NAT for the first time. Recently, serious discussions have started about the standardization of NAT and the architectural implications of NAT been held in the IETF. In 2004, a working group was formed to develop terminology for NAT, define requirements, and define protocols for the testing and probing of NATs. This working group, known obtusely as BEHAVE (Behavioral Engineering for Hindrance Avoidance) [4] has made excellent progress over the past few years and has published a number of documents discussed in this chapter. This chapter will make use of some this new terminology and definitions. The next sections cover the advantages and the problem with NAT as discussed in RFC 2993 [5].

10.2 Advantages of NAT

A number of factors led to the development of NAT, and only a few of them relate to the shortage of IPv4 addresses. Many of them are aspects of management. For example, a network can avoid having to renumber IP addresses when changing internet service providers. The management of IP addresses can be simplified if a network must use a number of small individual address blocks. The use of a single private address range is easier to manage in the end devices, with the complexity of the small address blocks being centralized in the NAT. Networks can also use the basic filtering properties of NAT to provide some firewall security. Any host that does not initiate outbound connections to the Internet is not reachable by a host on the Internet. Hosts that do initiate connections are limited by the filtering rules of the NAT in receiving incoming packets.

For Internet service providers (ISPs), NAT can allow an ISP to conserve the number of IP addresses it needs. The number of IP addresses needed is only the maximum number of concurrent connected users rather than the total number of users. Each user can be assigned a private address and will only be assigned a public address when they connect. NAT also allows an ISP to segment their network for management purposes.

10.3 Disadvantages of NAT

There are many problems with NAT. The biggest problem is that they break the end-to-end model of the Internet. The reachability of any Internet host by any other Internet host was part of the Internet from the beginning, and many assumptions about reachability are included in Internet protocols. NAT also breaks the transitive reachability of hosts (e.g., host A can reach host B, host B can reach host C, but host C cannot reach host A). NAT creates a single point of failure in the Internet where fates are shared. For example, in a connection without NAT between host A and host B, as long as both hosts have Internet connectivity they can exchange packets and communicate. Any of the routers and connections between them can fail and the connection can be maintained, possibly with some lost packets. However, if A and B connect through a NAT, the failure of that NAT will cause the connection to fail with no guarantee that either A or B can re-establish it. For example, many failures experienced during Web browsing are not failures of either the Web browser or the Web server, but of a NAT in between. When the NAT fails, the bindings are lost and the connection stalls. This continues until the user hits the refresh button on the Web browser which closes all the TCP connections and opens new ones, creating new bindings, which allow browsing to continue. The use of NAT also complicates multihoming in a site, since a response must route back through the same NAT that processed the request. NAT also inhibits the implementation of security at the IP layer.

There are cases where NAT will not prevent IP address renumbering. For example, when networks that use the same private address range are combined, the address overlap will require renumbering. NAT with port mapping complicates offering services that use well-known port numbers. For example, Web servers typically use port `80` which can only be allocated once by NAT for a particular public address. If multiple SIP clients are behind the same NAT and share a public IP address, only one of them can be allocated the well-known SIP port of `5060`, which can result in routing failures.

NATs that operate at both layer 3 and layer 4 must understand all transport protocols used through it. While this is not a problem for the common TCP and UDP, it is a major problem for newer transport protocols such as the Datagram Congestion Control Protocol (DCCP), Stream Control Transport Protocol (SCCP), and Host Identity Protocol (HIP). As a result NATs are an impediment to new transport protocols.

Unfortunately, NAT usage is widespread on the Internet today. As a result, all protocols must be aware of NAT operation and be able to overcome their limitations. The next section will discuss how NAT works.

10.4 How NAT Works

NAT operates at the IP Layer 3 as shown in Figure 1.1. NAPT operates at both the IP and transport layers 3 and 4. NAT creates mappings between inside and outside addresses and ports. These mappings are sometimes called bindings. Mappings are temporary, and expire either after a TCP connection is closed with a FIN or after inactivity with UDP transport. There are multiple types of NAT depending on the rules used for mapping and filtering. Older NAT terminology classified NAT in terms of full cone, restricted cone, and symmetric. However, in this chapter, we will use more modern terminology developed by the BEHAVE working group. Some major types of NAT are endpoint address independent, endpoint address dependent, and endpoint address and port dependent. There are also several behavioral options as well.

Table 10.1 lists the parts of the IPv4 and UDP/TCP header field that are modified by NAT. When a request is received on the inside of a NAT and forwarded to the outside, the source address and source port are rewritten. If this is the first packet in a flow, a new mapping will be created to be used for routing responses. If this is not the first packet, an existing mapping will be used. The UDP checksum must also be rewritten. When a response packet is received from the outside of the NAT and forwarded to the inside, the destination address and destination port are rewritten to the inside address and port for this mapping. Again, the UDP checksum must be rewritten.

Note that the fact that NAT operates at Layer 4 with TCP does not mean that the resulting TCP session is no longer end-to-end—the TCP connection is still end-to-end. NAT never maintains a transmission control block or maps sequence numbers. NAT also never performs retransmissions or acknowledgements. As a result, TCP still operates with end-to-end flow control and reliability through NAT.

An example of NAT operation is shown in Figure 10.1. In this example, a SIP client on a host sends a SIP request through a NAT, two routers, and an Ethernet switch. The Ethernet switch operates only at the link and physical layer while the routers operate at the IP, link, and physical layers. These devices only forward the packet without making any changes to the packet. The

Table 10.1
Parts of an IPv4 and UDP/TCP Header Field Modified by NAT

IP header checksum
Source address
Destination address
Source port
Desitnation port
UDP checksum

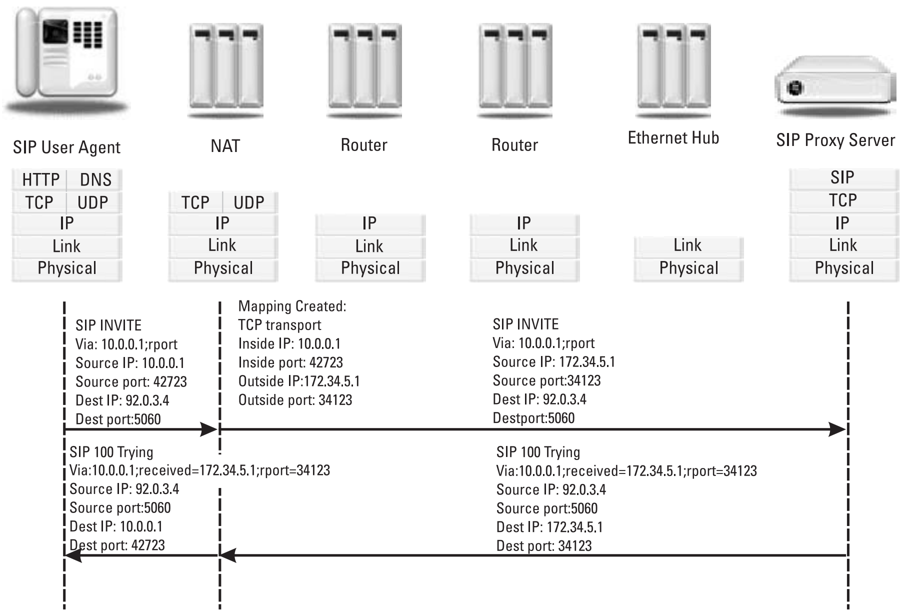


Figure 10.1 NAT example.

NAT, however, creates a new mapping between the source inside private address `10.0.0.1` port `42723` and the outside public address `172.34.5.1` port `34123` and rewrites the IP and transport layers with this information. When the SIP response comes back, the NAT uses the mapping created to rewrite the IP and transport layers with this information. As a result, the SIP server thinks the SIP client it is communicating with it at the address `172.34.5.1:34123` while the SIP client thinks it is using the address `10.0.0.1:42723`. Since we have seen how SIP imbeds IP addresses and ports into SIP and SDP messages, clearly this will be problematic.

10.5 Types of NAT

NAT can be characterized by the type of address and port mapping they do, as well as the type of filtering they employ. To see the different types of NAT, we will use the notation shown in Figure 10.2 from RFC 4787 [6]. This figure shows a host X behind a NAT communicating with two different hosts, host 1 and host 2. Packets sent by X have a source address and port of X:x and a destination address and port of Y:y. The NAT has a number of IP addresses to use for mapping, and they are shown as X1, X2, ... (some NATs only have a single IP address to use for mapping such as a home NAT). Packets sent from device X to host 1 are sent by X to Y1:y1 from X;x. As received by Y1, the packets appear

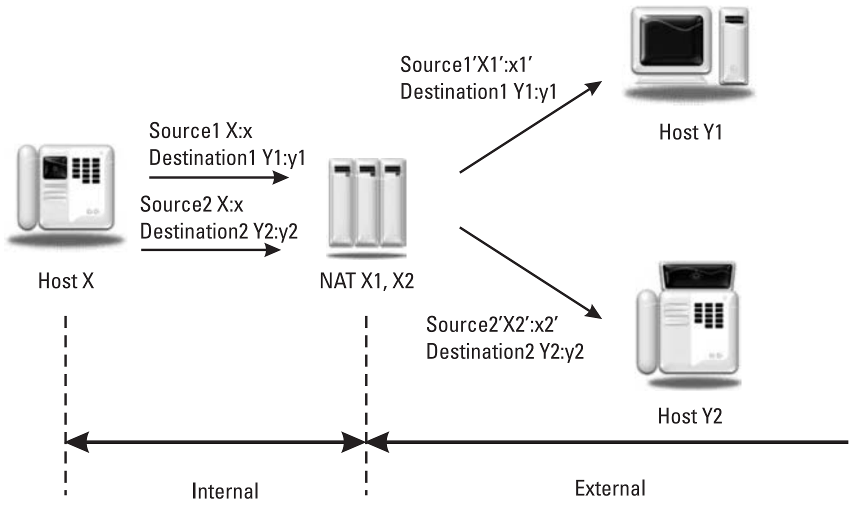


Figure 10.2 NAT mapping classifications.

to have been received from X1':x1'. Packets sent by X to Y2 *from the same source port x* are received by Y2 and appear to have been received from X2':x2'. That the packets sent to Y2 are sent from the same IP address and port as Y2 is essential for this classification method. The resulting relationship between X1':x1' and X2':x2' determines the type of NAT mapping.

Note that NAT classification is not precise, and many NATs change behavior over time and under differing conditions. As a result, trying to predict future NAT behavior based on past behavior is problematic. However, a good understanding of NAT behavior is still essential to understanding how to make SIP and Internet communications traverse NAT.

10.5.1 Endpoint Independent Mapping NAT

Endpoint independent mapping (EIM) NAT is where X1':x1' = X2':x2' for all Y;y (Y1:y1 and Y2:y2 as shown in Figure 10.2). That is, the mapping used is dependent *only* on the source address and port (X;x), independent of the destination address and port. This type of mapping is known as endpoint independent mapping (EIM). It is important to note that x1' does not equal x1, which means that a NAPT can be an endpoint independent NAT. This type of NAT is best for SIP and RTP traversal as we shall see in the next sections.

10.5.2 Address Dependent Mapping NAT

An address dependent mapping (ADM) NAT is where X1':x1' = X2':x2' if and only if Y2 = Y1 in Figure 10.2. That is, the mapping is dependent on the *destina-*

tion IP address of the packet as well as the source address and port. As a result, packets sent by X to different hosts will appear to come from different IP addresses and/or port numbers. This type of NAT is bad for SIP and RTP traversal as we shall see in the following sections.

10.5.3 Address and Port Dependent Mapping NAT

An address and port dependent mapping (ADPM) NAT is where X1':x1' = X2':x2' if and only if Y2:y2 = Y1:y1 as shown in Figure 10.2. That is, the mapping is dependent on both the destination address *and* port as well as the source address and port. As a result, packets sent by X to different ports on the *same* host will appear to come from different IP addresses and/or port numbers. This type of NAT is the most problematic for SIP and RTP traversal.

10.5.4 Hairpinning Support

Hairpinning is shown in Figure 10.3. Any NAT will allow hosts on the inside of the NAT (sometimes described as *behind the same NAT*) to use the private address of the other host—these packets can be routed by the NAT without any modification. A NAT supports hairpinning if an internal host can send packets to another internal host using the external address of the other host. Packets routed this way double back on the NAT giving a shape similar to a hairpin. To route these packets, the NAT must lookup two mappings to determine the destination. In the example, X has a mapping X:x to X1':x1' while Z has a mapping Z;z to X2':x2'. Hairpinning is supported by the NAT if Z receives a packet sent by X from X;x to X2':x2' and vice versa. Support of hairpinning is very beneficial to SIP and RTP traversal. Note that this property is also sometimes called *tromboning* after the shape of the slide of the instrument.

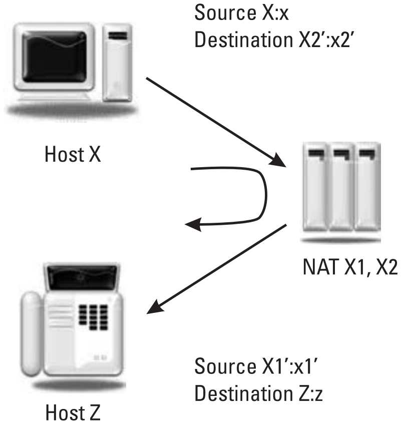


Figure 10.3 Hairpinning support in NAT.

10.5.5 IP Address Pooling Options

NATs that have a number of external or public IP addresses available for mapping have options in the way they allocate from this pool of IP addresses. Having multiple IP addresses is common in large service provider or enterprise NATs and is uncommon in small residential NATs. One pooling policy is known as paired IP address pooling. This policy means that only one external IP address is used for an internal IP address. As a result, all packets sent by this host will appear to come from the same IP address to hosts external to the NAT. This property is very good for SIP and RTP traversal. Another policy is known as arbitrary IP address pooling, which means there could be multiple external IP addresses mapped to an internal IP address. This property is very bad for SIP and RTP transport.

10.5.6 Port Assignment Options

While only NATs that have multiple external IP addresses have address pooling options, every NAT has port assignment options. There are a number of behaviors that are part of implemented NATs. Port preservation means the NAT tries to keep the external port the same as the internal source port. This port assignment approach generally only works if the NAT has a large pool of IP addresses. Otherwise, only one inside host can use a particular port. If a NAT implementing port preservation runs out of a particular port to allocate, it can use one of two strategies. One is to switch to a nonport preservation mode. The other is to do port overloading in which the same port can be used more than once on the inside and outside. Port overloading is very bad for the SIP and RTP transport because it leads to nondeterministic behavior. Another port assignment option is port parity, which preserves the oddness or evenness of ports. This is useful for media transport where the RTP media must be an even port while the RTCP control must use an odd port number. Port contiguity is when the NAT attempts to keep sequential inside ports mapped to sequential outside ports. This can help when an RTCP port is inferred to be one higher than the RTP port. For SIP and RTP traversal, the most important property is that the port assignment mode does not change. Since port preservation always runs the risk of having to switch modes or do port overloading, it is not recommended. Switching port mapping modes makes troubleshooting difficult.

10.5.7 Mapping Refresh

Each time a NAT creates a mapping it uses up memory resources in storing the information. It also uses up addressing resources since the external mapped IP address and port cannot be reused for another endpoint. As a result, a NAT must have behavior to ensure that old mappings are expired and all resources freed up into the pool.

TCP mappings can be created and removed based on TCP signaling. For example, the exchange of SYN messages tells the NAT to create a new mapping for the connection. The exchange of FIN messages tells the NAT the connection is no longer needed and can be safely closed and the mapping state discarded. UDP, however, has no signaling, so the NAT must infer the creation and destruction of a UDP session. Usually, this is done using an inactivity timer. If no packet is received before this timer expires, the connection is considered terminated and the mapping removed. The recommended value is 5 minutes [6] although in practice some NATs use values as short as 30 seconds. Theoretically, a packet from either the inside or outside host can refresh the mapping, although it is usually a good security policy to only allow packets generated internally to refresh the mapping. Otherwise, an outside host could keep the mapping alive by sending refresh packets even after the inside host wants the connection closed.

10.5.8 Filtering Modes

By their basic function, NATs provide filtering functions. If a mapping between an external address and an internal address is not present, packets cannot be sent to that internal host. When a mapping is active, the NAT has options on what additional filtering it can provide. Essentially, these filtering rules control who is permitted to use the mapping. One filtering mode is known as Endpoint Independent Filtering. In this mode, any external endpoint is permitted to send packets to the internal host once the mapping is created. Another mode is called address dependent filtering. This mode allows only external hosts that have received a packet from the internal host to send a packet using the binding. A single packet sent to the external host "opens the latch" and will allow any number of packets to flow in the opposite direction. This filtering mode provides some firewall-like security. Another mode is called address and port dependent filtering. This mode only allows an external endpoint to send packets from the same external IP address and port to which the internal host has sent a packet. Endpoint independent filtering is best for SIP and RTP, while address dependent and address and port dependent filtering make things difficult. This information is summarized in Table 10.2.

Table 10.2
NAT Filtering Mode Summary

Endpoint independent filtering
Address dependent filtering
Address and port dependent filtering

10.6 NAT Mapping Examples

Figure 10.4 shows an example of a NAT. The packet sent from `192.168.0.1:1234` to `2.73.3.2:5678` appears to host 1 to have come from `23.3.2.8:4219`. The packet sent from `192.168.0.1:1234` to `62.3.9.9:5678` appears to host 1 to have also come from `23.3.2.8:4219`. Since these two mapped addresses are the same, this indicates the NAT is an endpoint independent mapping NAT. Since the internal and external port numbers are different, no port preservation is used. Since the even internal port maps to an odd external port, port parity is also not used by this NAT.

Figure 10.5 shows another example of a NAT, which has two IP addresses assigned to it, `23.3.2.8` and `23.3.2.9`. A packet sent from `192.168.0.1:1234` to `2.73.3.2:5678` appears to host 1 to have come from `23.3.2.9:4219`. The packet sent from `192.168.0.1:1234` to `62.3.9.9:5678` appears to host 1 to have come from `23.3.2.9:2194`. Since this is a different mapped address, this indicates an address dependent mapping NAT.

Figure 10.6 shows another example. The packet sent from `192.168.0.1:1234` to `2.73.3.2:5678` appears to host 1 to have come from `23.3.2.8:4219`. The packet sent from `192.168.0.1:1234` to `2.73.3.2:9101` appears to host 1 to have come from `23.3.3.7:6421`. Since the mapping is different despite the destination being the same address, this indicates an address and port dependent mapping NAT.

Figure 10.7 shows another example NAT. In this example, the packet sent from `192.168.0.1:1234` to `2.73.3.2:5678` appears to host 1 to have come from `23.3.2.8:4219`. The packet sent from `192.168.0.1:1235` to `2.73.3.2:5679` appears to host 1 to have come from `23.3.2.8:2170`. Since these two packets are not sent from the same source address and port, these data provide no information about

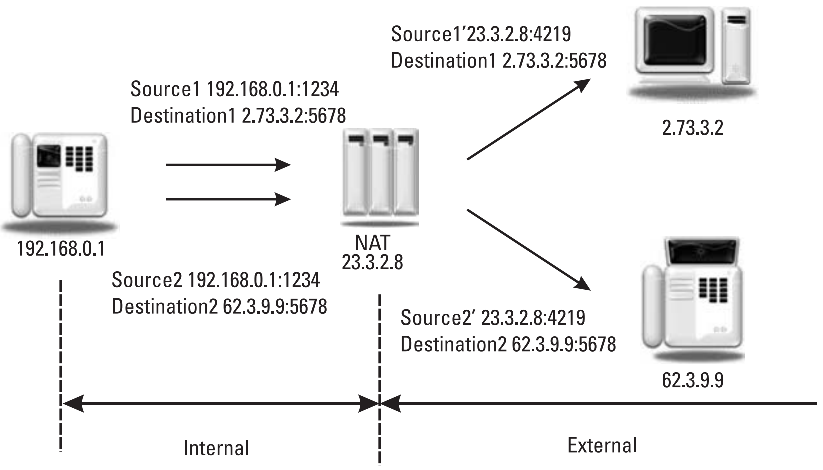


Figure 10.4 Example of endpoint independent mapping NAT.

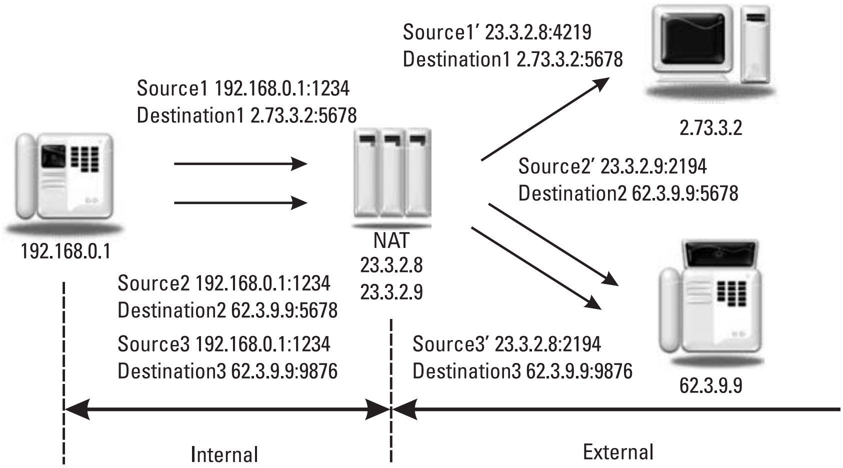


Figure 10.5 Example of address dependent mapping NAT.

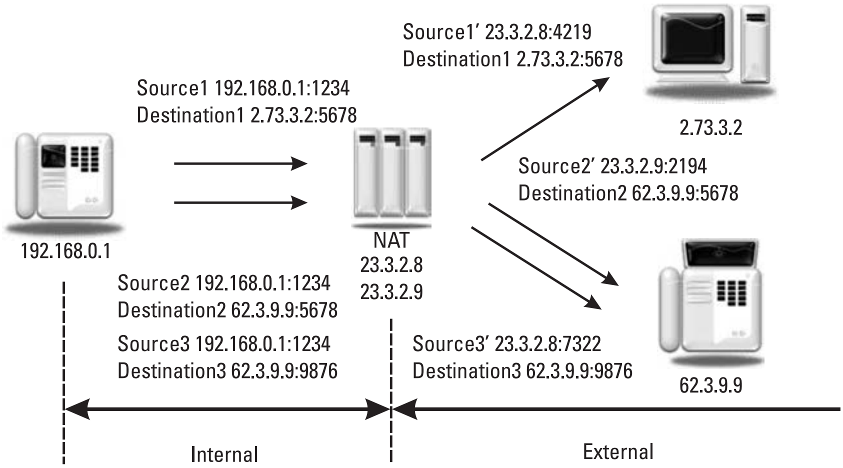


Figure 10.6 Example of address and port dependent mapping NAT.

the type of mapping used by the NAT. However, this indicates paired IP address pooling. No port preservation is used. No port parity or port contiguity is used.

10.7 NATs and SIP

Now that we have an understanding of what NAT is and how it works, we can look at its impact on applications and protocols such as SIP and RTP. Over-

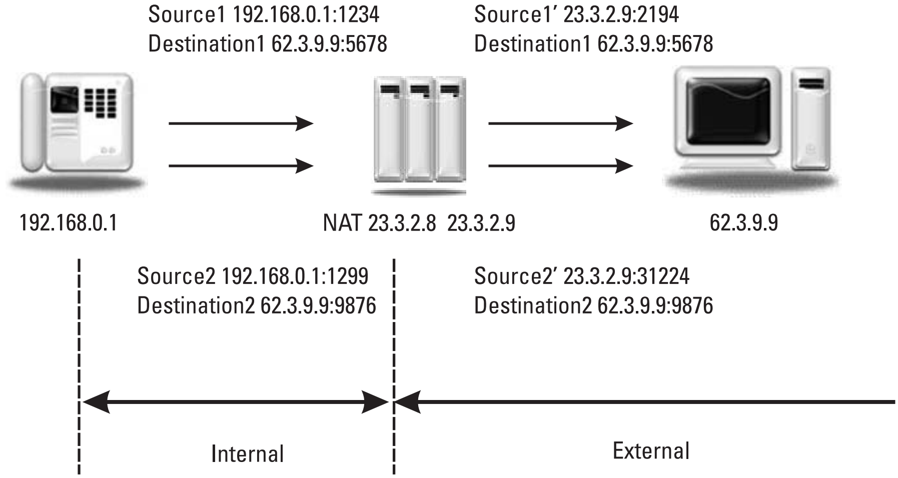


Figure 10.7 Example of paired IP address pooling NAT.

all, NATs work reasonably well with unencrypted client/server protocols such as Web browsing, e-mail, and so forth. However, they can cause problems with IPSec VPNs, which can fail signature checks if the signature includes address and port information. Many NATs have application layer gateways (ALGs) for common applications such as FTP (File Transfer Protocol). However, NAT does cause problems for peer-to-peer protocols such as SIP. NATs also cause problems for protocols that carry imbedded IP addresses and port numbers, such as SIP.

Guidelines for NAT-friendly protocol design were published in 2002 in RFC 3235 [7]. However, this was much too late to be helpful for SIP. The main recommendations were as follows:

- Limit peer-to-peer applications and approaches in favor of client/server applications.
- Don't rely on end-to-end IPSec security.
- Use DNS names not IP addresses.
- Multicast is problematic.
- Avoid session bundles (e.g., one session controlling/establishing another session).
- Use TCP instead of UDP.

Unfortunately, SIP violates most of these recommendations.

Early work on SIP essentially ignored NAT or assumed that IPv6 deployments would cause them to go away. Many properties of SIP assume reflexive routability, which is often not present with NAT. The reality of SIP deployments was that while most servers (proxies, registrars, and so forth) had public IP addresses, most UAs were behind NAT and hence had nonroutable private IP addresses. Since the SIP servers provided rendezvous service, typically the SIP exchange would work while the RTP media session would fail.

Early solutions that were examined included application layer gateways (ALGs) which would effectively make NAT SIP-aware. Another early solution was a simple discovery approach in which a UA would send test packets to determine if it was behind a NAT, and to discover mapped addresses. This protocol was STUN [8], which initially stood for simple traversal of UDP through NAT. A STUN client sent test packets (essentially pings) to a STUN server, which responds with the mapped address and port the packet appeared to be received from. STUN will be described in Section 10.9. Using the mapped addresses discovered using STUN, UAs tried to fix the IP addresses and ports in their SIP messages. Unfortunately, this did not work under all scenarios and with all types of NAT. Instead, a back-to-back user agent (B2BUA) approach has developed in the industry to ensure both SIP and RTP traverse NAT.

To overcome the deficiencies of STUN, the IETF developed the Interactive Communications Establishment (ICE) [9], which runs a series of end-to-end tests using STUN between the two UAs attempting to establish communication. This protocol used an approach known as hole punching which was developed by peer-to-peer gamers to establish gaming connections.

10.8 Properties of a Friendly NAT or How a NAT Should BEHAVE

The basic properties of a NAT, which is friendly towards SIP and RTP, is summarized in Table 10.3. The NAT should have endpoint independent mapping, and paired IP address pooling. The port assignment should not do port preservation or port overloading. Port parity preservation and contiguity is good but not essential. The NAT should use endpoint independent or address dependent filtering. The UDP refresh timer should be 5 minutes.

Table 10.3
How a NAT Should BEHAVE

Endpoint independent mapping
Address independent or address dependent filtering
Pair IP address pooling
Not port preservation
Not port overloading
Port parity preservation is helpful
UDP refresh timer 5 minutes

10.9 STUN Protocol

STUN was first published as RFC 3489 [8] as Simple Traversal of UDP through NAT. It has been significantly updated and revised and published as RFC 5389 with a new acronym expansion, Session Traversal Utilities for NAT [16]. The basic operation is shown in Figure 10.8. The main function of STUN is for a STUN client to request a mapping request from a STUN server. The STUN packet sent by the client traverses any number of NATs before reaching the STUN server. The STUN server returns the mapped address in a response. Note that this mapped address is just the one of the outer NATs —there may be many mappings happening, but only the outermost one is visible to the STUN server. The address must be hidden from ALGs in NATs in this response packet or the NAT might try to fix the address and replace it. This is done by an exclusive OR-ing (XOR), the mapped IP address in the response.

There are four main usages of STUN. The first is for basic mapping discovery. The second is to perform a connectivity check with a server or a peer UA. (This is the usage for ICE.) The third is for media relay usage. The extensions are known as TURN and described in Section 10.14. The fourth is as a keep-alive to refresh NAT mappings for UDP. This is used in the SIP outbound extensions. Another usage of STUN is NAT behavior discovery. This relatively new usage is described in [10]. This creative usage of STUN allows:

- NAT address mapping;
- NAT filtering behavior;
- Discovery of the mapping lifetime;
- Discovery of support for hairpinning;
- Determination of fragmentation handling;

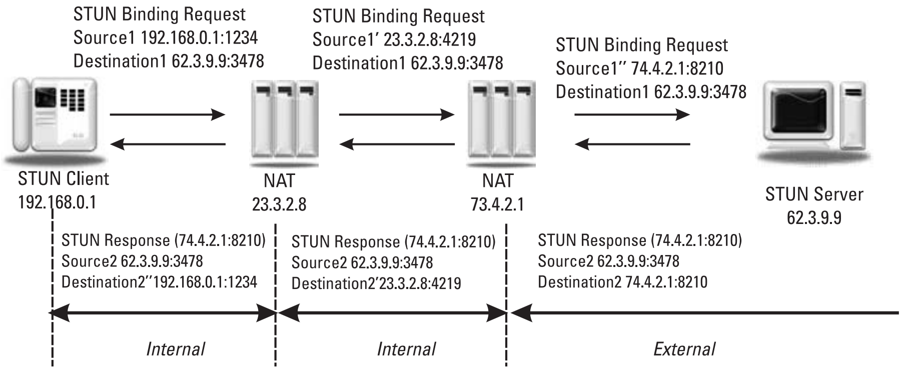


Figure 10.8 STUN: session traversal utilities for NAT.

- Detecting generic ALGs, which rewrite IP addresses.

This new usage defines a number of minor extensions to STUN and will shortly be standardized. Note that this usage of STUN has some limitations in that it can only characterize a NAT *for a particular address at a particular time*. Since a NAT can change its behavior for different addresses and at a later time, care must be taken in using the information derived from these tests.

10.10 UNSAF Requirements

With a number of peer-to-peer protocols such as SIP attempting to fix and work around NAT problems, the Internet Architecture Board (IAB) published RFC 3424 [11] setting the requirements and limitations on IETF solutions to deal with NAT. These approaches are known as Unilateral Self-Address Fixing or UNSAF approaches, the acronym being chosen to highlight the fact that these approaches could, if not done correctly, make the NAT problem worse. Each protocol that tries to work around NATs must clearly scope the problem and describe the exit strategy and transition plan. The approach must discuss specific issues that might make the approach "brittle" or likely to fail. The approach must also discuss known practical issues encountered with real NAT in deployments.

10.11 SIP Problems with NAT

Consider the typical example of a SIP UA behind a NAT trying to communicate with a SIP server outside the NAT. Since SIP has some client/server properties, some SIP operations will work as long as the requests are originated by the UA. Requests originated by the SIP server may be blocked by lack of mappings or filtering rules. One approach used in practice is to use frequent SIP registrations to create the mapping and then reuse the mapping for incoming requests to the UA. However, this approach doesn't quite work without some SIP extensions as we shall see. Media negotiated using SIP offer answer is a big problem as the SDP offer and answer will contain private addresses and ports in the `c=` and `m=` lines. An example SIP message sent from behind a NAT is shown here:

```
INVITE sip:UserB@there.com SIP/2.0
Via: SIP/2.0/UDP 10.1.1.221:5060;branch=z9hG4bKhjh
From: TheBigGuy <sip:UserA@customer.com>;tag=343kdw2
To: TheLittleGuy <sip:UserB@there.com>
Max-Forwards: 70
Call-ID: 123456349fijoewr
CSeq: 1 INVITE
Subject: Wow! It Works...
Contact: <sip:UserA@10.1.1.221>
```

```
Content-Type: application/sdp
Content-Length: ...

v=0
o=UserA 2890844526 2890844526 IN IP4 UserA.customer.coms=-
t=0 0
c=IN IP4 10.1.1.221
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

There are three main problems with this message. The Via header field contains a private IP address (10.1.1.221) and the listening port (5060) may not have an active mapping or filter rule. The Contact URI is unroutable due to the private IP address. Finally, the SDP information in the c= and m= lines will not work behind the NAT.

Via already has a partial solution for the response routing. The use of the received parameter will record the mapped public IP address, which is then used for routing the response. However, this will only work for a port preservation NAT (which is not recommended NAT behavior). A normal NAT that also changes the port will not work. The Contact URI problem is even more serious. If the request is a REGISTER, the AOR binding will not work since the URI is unroutable. If the request is an INVITE, the ACK and all subsequent requests such as BYE will also fail to route.

There are three main solutions to these problems: symmetric SIP, connection reuse, and SIP outbound. Each of these will be discussed in the following sections.

10.11.1 Symmetric SIP

Symmetric SIP operation uses the rport extension defined in RFC 3581 [12]. Just as the received Via parameter saves the mapped address, the rport parameter saves the mapped port number for use in routing responses. A SIP UA indicates support for this extension by including the rport parameter (without the "=" and an address which is not known) in the SIP request. This tells the next hop proxy that the UA will be listening for the response on the same address and port number that the request was sent from, instead of the address and port listed in the Via. When the proxy receives the request, it stores the mapped port in the rport parameter and then uses this for routing any responses back. The use of rport is shown in Figure 10.1.

10.11.2 Connection Reuse

Connection reuse is a method in SIP to reuse existing TCP connections. It is defined in [13]. Once a connection is opened, UAs can reuse the connection for subsequent connections. If a NAT traversal approach such as ICE is used, connection reuse will reduce the number of times it has to be run. Connection reuse is mainly defined between proxy servers. The usage of connection reuse combined with registration is defined for UAs as SIP outbound in the next section.

10.11.3 SIP Outbound

SIP outbound is a mechanism that combines connection reuse, registration through multiple proxies, and keep alives. It solves many of the existing problems with SIP NAT traversal, although it requires both UAs and registrars support the extension. SIP outbound is currently being finalized by the IETF as [14]. The basic configuration is shown in Figure 10.9. UAs have an `instance-id`, which uniquely identifies them. Multiple registrations by the UA through different proxy servers will have different `reg-id`. STUN keep alive messages are used between the UA and the proxy to monitor the flow and keep the NAT mappings active. If the STUN checks reveal the flow has failed (perhaps due to a failure in the UA, the NAT, or the registrar), the UA registers again using the same `instance-id` but a new `flow-id`. This is shown in Figure 10.10.

10.12 Media NAT Traversal Solutions

There are a number of methods used for media traversal of NAT, which will be discussed in the following sections. These solutions involve getting RTP and RTCP negotiated using SIP to work.

10.12.1 Symmetric RTP

Symmetric RTP involves sending RTP from the same port that it is listening for RTP. This results in a UDP connection that the NAT will understand and allow

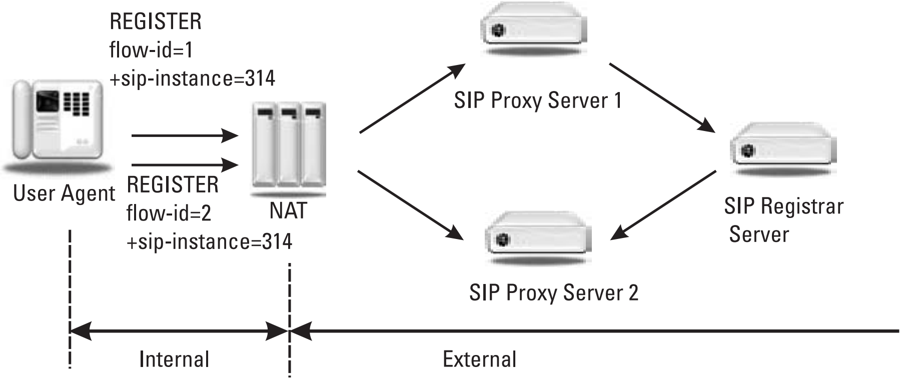


Figure 10.9 SIP outbound.

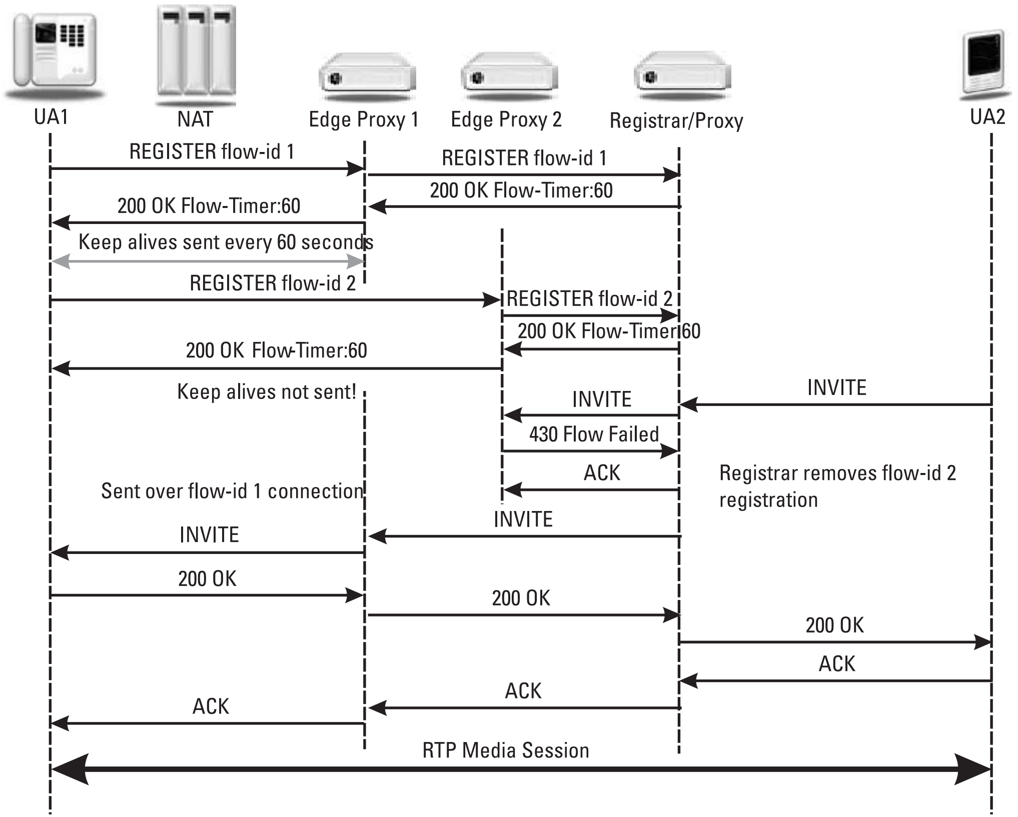


Figure 10.10 SIP outbound call flow.

mapping and filtering to work. However, this is only useful if the media flow is bidirectional and if at least one side can get through initially. Symmetric RTP effectively means ignoring address and port information in the SDP, since it will be different from the mapped addresses and ports.

10.12.2 RTCP Attribute

The normal assumption that the RTCP port is one higher than the RTP port only works through NATs that support port contiguity. For most NATs, the RTCP will not work even when RTP does work. A solution to this is to explicitly signal the RTCP port and address using the `a=rtcp` attribute defined in RFC 3605 [15]. If the mapped RTCP port can be discovered, using STUN, for example, then this approach can work.

10.12.3 Self-Fixing Approach

This approach uses NAT mapping information discovered using STUN, which is then put into the SDP offer or answer to fix the addresses. This works with some NAT such as endpoint independent mapping NATs, but will fail to work for address or port dependent mapping NATs.

The best solution for media NAT traversal is to use hole punching (and a protocol such as ICE) with a backup such as a media relay like TURN as described in the following sections.

10.13 Hole Punching

Hole punching is a probing approach used to discover and actually create NAT mappings and filtering rules. Hole punching will work in many situations that other approaches will fail, although it will not work in every situation—some combinations of NAT properties just will not permit direct exchange of packets between some hosts. Hole punching uses two clients and a rendezvous server. In SIP, the clients are UAs and a proxy server can be the rendezvous server.

In hole punching, two clients simultaneously probe using at least two sets of address, the private address and a discovered public address. The public address could be discovered using a STUN server or with the help of a rendezvous server. The rendezvous server has a public IP address and is reachable by both clients. The rendezvous server helps the two clients exchange address candidate lists. The clients repeatedly try both addresses until one or more work. At that point, they utilize this working address.

Figure 10.11 shows hole punching when the clients are behind different NATs. The private addresses fail since they are not behind the same NAT. The initial test of the public address fails; however, it creates a filtering rule, which allows the test from the other direction to succeed. In this way, the testing has "punched" holes through the NATs and created mappings and filtering rules where none existed prior to hole punching. Note this only works for certain combinations of NAT mapping and filtering rules. For example, this works if both NATs are endpoint independent mapping. Another possibility is that both clients are behind the same NAT. In Figure 10.11, this would mean NAT A and NAT B were the same NAT. Note that this situation is not as easy as it sounds to detect—just because two clients use the same private address range does not mean they are behind the same NAT. In this example, the private addresses work while the public addresses fail, with the resulting connection utilizing the private addresses. Alternatively, there could be multiple levels of NAT; ultimately both clients are behind one NAT. While the private addresses fail (due to the multiple levels of NAT), the public addresses work after an initial failure as long as the common NAT supports hairpinning.

Figure 10.12 shows an example of hole punching where both NATs are endpoint independent mapping with address and port dependent filtering. In this example, host A has two address candidates, `192.168.0.1:1234`, which is host A's private address, and `23.3.2.9:4219`, which is A's public address learned through a STUN server. Host A shares these address candidates with host B using the rendezvous service. Host B has two address candidates, `10.0.1.13:5678` and `2.72.3.2:31212`, which it learns through a STUN server. Host B also shares these addresses with host A through the rendezvous server. At this point, the hole punching begins. Host A sends packets to `10.0.1.13:5678`, but these packets go nowhere as this private address is not routable. Host B sends packets to `192.168.0.1:1234`, but they also go nowhere as they are not routable. Host A sends

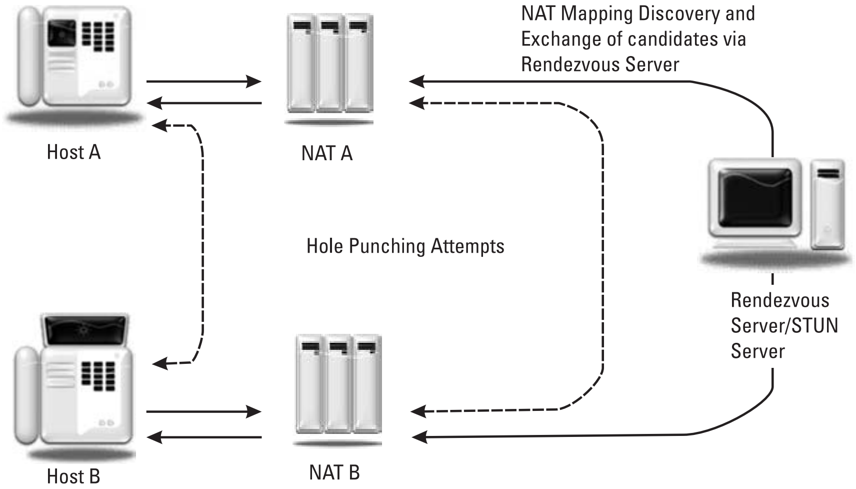

NAT Mapping Discovery and Exchange of candidates via Rendezvous Server
Host A
NAT A
Hole Punching Attempts
Rendezvous Server/STUN Server
Host B
NAT B

Figure 10.11 Hole punching architecture.

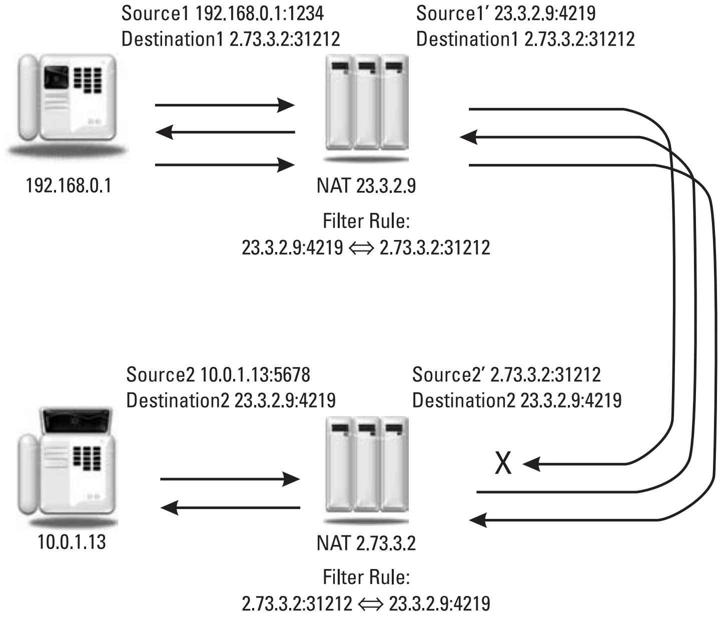


Figure 10.12 Hole punching example.

a packet to `2.73.3.2:3122`. Since NAT A is endpoint independent mapping, the existing mapping is used, so this packet is forwarded to NAT B with a source address of `23.3.2.9:4219`. This creates a filter rule in NAT A that allows packets to be received from `2.73.3.2:31212` to be forwarded to `23.3.2.9:4219`. This packet arrives at NAT B but is dropped by NAT B since there is no filter rule allowing packets from `23.3.2.8:4219` to be received by `2.73.3.2:31212`. Host B then sends a packet from `10.0.1.13:5678` to `23.3.2.9:4219`. This reuses the mapping of `10.0.1.13:5678` to `2.73.3.2:31212` and creates a filter rule that allows packets from `23.3.2.9:4219` to be forwarded to `2.73.3.2:31212`. The packet is forwarded to NAT A. At NAT A, there is an active mapping for `23.73.3.2:31212` and also a filter rule that allows packets to be received from `2.73.3.2:31212`. As a result, the packet is forwarded to host A and the hole punching has succeeded. Note that without the failed packet sent by host A, which created this filter rule, this packet would have been blocked. Now host A can send to host B at `2.73.3.2:31212` using the two mappings and two filter rules in place. Note that hole punching also works if Host B sends a packet first, which fails, then Host A sends a packet, which then succeeds.

Figure 10.13 shows another example of hole punching. In this example, NAT A is endpoint independent mapping NAT with endpoint independent filtering, while NAT B is address and port dependent mapping with address and port dependent filtering. A packet sent from host A `192.168.0.1:1234` creates a mapping of `23.3.2.9:4219`.

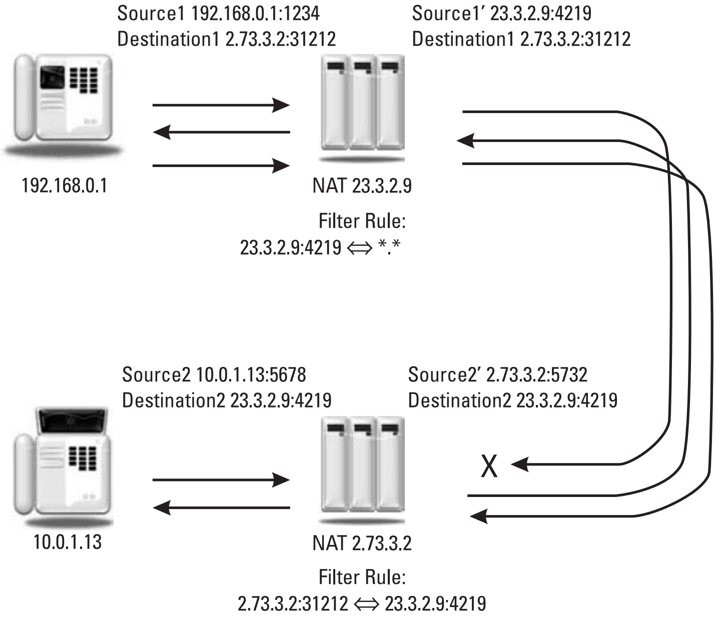


Figure 10.13 Hole punching example.

A packet sent from host B `10.0.1.13:5678` creates a mapping of `2.73.3.2:31212`. A packet sent from host A `192.168.0.1:1234` to `2.73.3.2:31212` creates a filtering rule in NAT A. The packet reaches NAT B but is dropped due to filtering. A packet sent from host B `10.0.1.13:5678` to `23.3.2.9:4219` creates a new mapping to `2.73.4.1:5732` and creates a new filtering rule in NAT B. The packet reaches NAT A. Since NAT A has endpoint independent filtering, the packet is forwarded and host A receives packet. Host A then sends a packet to `2.73.4.1:5732`, which reaches host B due to the two mapping and filtering rules. Hole punching works in this case despite the address and port dependent mapping and filtering in NAT B since NAT A has endpoint independent filtering.

Figure 10.14 shows an example where hole punching fails. NAT A is address and port dependent mapping with address dependent filtering while NAT B is address and port dependent mapping with address dependent filtering. A packet sent from host A `192.168.0.1:1234` creates a mapping of `23.3.2.9:4219`. A packet sent from host B `10.0.1.13:5678` creates a mapping of `2.73.3.2:31212`. A packet sent from A `192.168.0.1:1234` to `2.73.3.2:31212` creates a new mapping of `23.3.3.2:7876` and creates a new filtering rule in NAT A. The packet reaches NAT B but is dropped due to filtering. A packet sent from host B `10.0.1.13:5678` to `23.3.2.9:4219` creates a new mapping of `2.73.4.1:5732` and creates a new filtering rule in NAT B. The packet reaches NAT A but is dropped due to filtering.

Additional packets sent will also fail due to filtering, and hole punching fails for this configuration.

Typically hole punching will fail due to a combination of address or port mapping and address or port dependent filtering, such as that shown in Figure 10.14. When hole punching fails, a media relay must be used that is in the public Internet and reachable by both hosts. TURN is a protocol used by a UA to acquire a media relay transport address to use as a fall back when hole punching fails.

10.14 TURN: Traversal Using Relays Around NAT

TURN is a protocol extension of STUN used for acquiring and configuring a remote relay. TURN has been in development in the IETF for many years. Earlier versions were quite different, and even had a different title: traversal using relay NAT. The current version is [16] and will soon be published as an RFC.

A server operating as a TURN relay uses significant resources on the server. For one thing, each media stream relayed uses up double the bandwidth of the stream (incoming bandwidth + outgoing bandwidth). Also, the relay must process and forward each packet. Media relays also introduce delay (latency) and add extra IP routing hops, which increase the chance of packet loss. As a result, the use of TURN should be minimized for an efficient Internet communication or VoIP system.

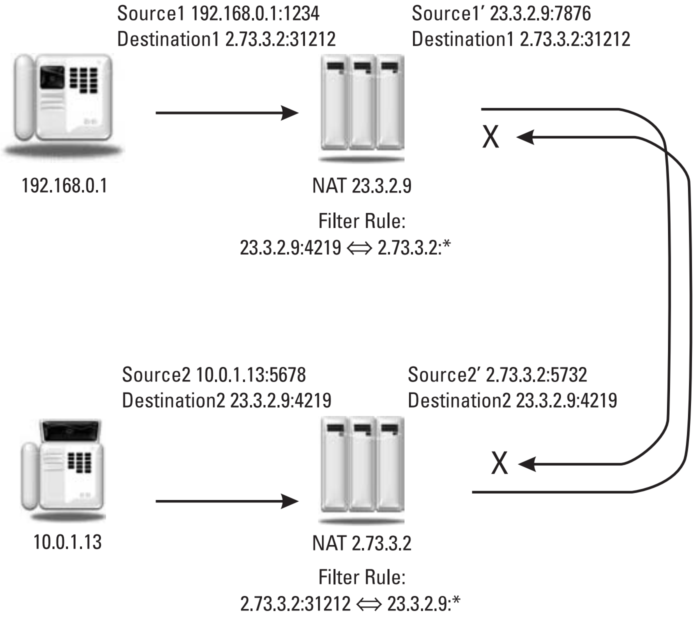


Figure 10.14 Hole punching example.

10.15 ICE: Interactive Connectivity Establishment

Interactive connectivity establishment (ICE) is the solution to the problem of when to use hole punching and when to use a media relay. ICE is an IETF protocol that standardizes hole punching and is an optimal methodology. Users of ICE gather as many transport addresses as they can (the private and public address pairs in the previous section on hole punching are only a minimum). They are listed in the order so the most preferred are tested first. A media relay (TURN) address is included as the lowest priority address. After the candidate addresses are exchanged using a SIP offer answer exchange, both sides begin hole punching and noting successes and failures at the end. Both sides choose the highest priority working transport pairs. In the worst case, this might be the media relay address if the NATs in the path make hole punching fail. The basic call flow is shown in Figure 10.15.

The following is an example set of address candidates in SDP. You can see the two candidate addresses used by Host A in the previous hole punching examples.

```
v=0
o=hosta 2890844526 2890842807 IN IP4 192.168.0.1
s=-
c=IN IP4 23.3.2.9
```

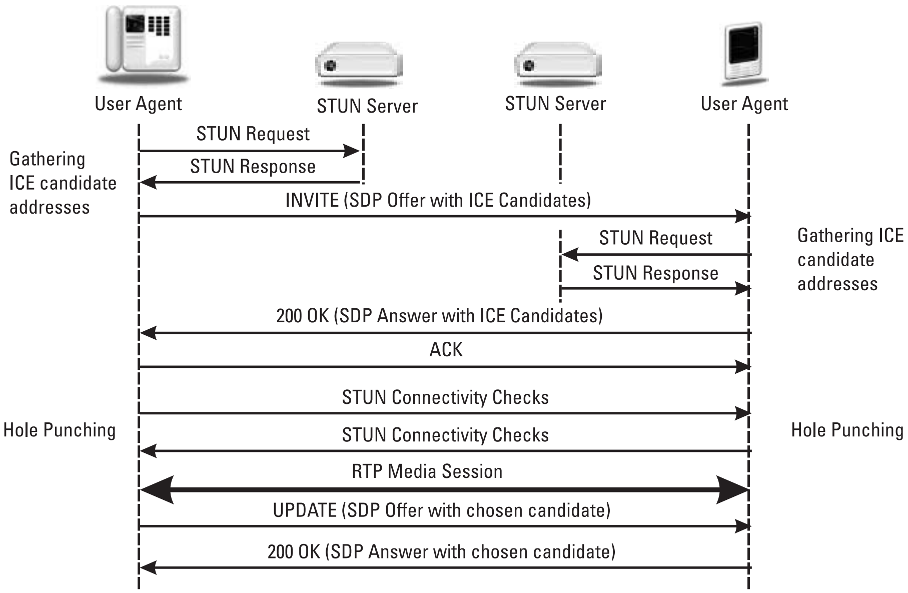

User Agent
STUN Server
STUN Server
User Agent
STUN Request
STUN Response
Gathering ICE candidate addresses
INVITE (SDP Offer with ICE Candidates)
STUN Request
STUN Response
Gathering ICE candidate addresses
200 OK (SDP Answer with ICE Candidates)
ACK
STUN Connectivity Checks
Hole Punching
STUN Connectivity Checks
Hole Punching
RTP Media Session
UPDATE (SDP Offer with chosen candidate)
200 OK (SDP Answer with chosen candidate)

Figure 10.15 ICE call flow.

```
t=0 0
a=ice-pwd:a8fgdfpdd777uzjYhagZg
a=ice-ufrag:88fgdhhY
m=audio 4219 RTP/AVP 0
a=rtpmap:0 PCMU/8000
a=candidate:1 1 UDP 13d0706431 192.168.0.1 1234 typ host
a=candidate:2 1 UDP 69d4498152 23.3.2.9 4219 typ srflx raddr
   192.168.0.1 rport 1234
```

Besides NAT traversal, ICE has other benefits. For example, address candidates can include both IPv4 and IPv6 addresses for dual stack UAs. As such, ICE can help in the transition between IPv4 and IPv6. ICE also includes keep alives to ensure that UDP mappings do not expire through NATs. ICE also provides a level of media authorization. When both UAs use ICE, media will only flow after a successful ICE check exchange. This ensures that both UAs are willing to send and receive media. Compare this to the case without ICE where a UA will start sending media to the address listed in the SDP without any check or verification. For example, a denial of service packet flood could be introduced by sending a high definition video server an `INVITE` and include the address of the target. The target will then receive the video stream without the ability to stop or understand.

10.16 Conclusion

This chapter has looked at the history, justification, and operation of Network Address Translation. The effect of NAT on SIP and RTP has also been discussed. Various approaches to the traversal of SIP and RTP through NAT have been covered including hole punching, relays, STUN, TURN, and ICE. These are summarized in Table 10.4. For more examples of SIP NAT traversal, see the Best Current Practices for SIP NAT Traversal document [17].

Table 10.4
Summary of SIP and RTP NAT Traversal

SIP Symmetric Routing (rport)
Symmetric RTP
Outbound
STUN
ICE
TURN
RTCP port attribute

10.17 Questions

Q10.1 In a few paragraphs, explain how NATs came to be and why they are popular today.

Q10.2 Explain the operation of an address and port dependent mapping NAT that has two IP addresses (19.34.2.1 and 19.34.2.2) assigned to it. Use three examples of UDP packets sent from 192.168.1.101 port 42194 to 204.32.44.21 port 413, 31.32.56.5 port 443, and 204.32.44.21 port 9753. Use port parity preservation in your examples.

Q10.3 Is an endpoint independent mapping NAT or an address dependent mapping NAT more friendly to Internet communications? Why?

Q10.4 Deduce as many properties of the NAT below as you can based on the information in the following tables.

X =	10.0.100.1
X1 =	73.42.4.1
X2 =	73.42.4.8
Y1 =	118.3.4.2
Y2 =	65.65.4.3

Active NAT Mapping Table

10.0.100.1:8080	maps to	73.42.4.1:3420
10.0.100.1:4343	maps to	73.42.4.1:7433
10.0.100.1:8080	maps to	73.42.4.8:3212

Filtering Table

73.42.4.8:3212	⇔	118.3.4.2:*
73.42.4.8:7433	⇔	118.3.4.2:*
73.42.4.1:3420	⇔	65.65.4.3:*

Q10.5 Explain the advantages and disadvantages of a SIP user agent supporting ICE.

Q10.6 For the packets of Question Q10.2, assume that each UDP packet contains a SIP OPTIONS request. Show the Via header field in each of the three 200 OK responses, assuming that the user agent has implemented appropriate SIP NAT traversal extensions.

Q10.7 Consider the hole punching scenario shown next.

NAT B is endpoint independent mapping NAT with endpoint dependent filtering.
NAT A is address and port dependent mapping with endpoint independent filtering.

Packet sent from A `192.168.0.1:1234` to a STUN server at `15.1.2.3` creates a mapping of `23.3.2.8:4219`
Packet sent from B `10.0.1.13:5678` to a STUN server at `15.1.2.3` creates a mapping of `2.73.3.2:31212`

Assume NAT A has only a single IP address. Assume NAT A and B do not implement port preservation.

Show the filter rules created as A and B begin hole punching. Will hole punching succeed?

Q10.8 Repeat Question Q10.7 with everything the same except:

NAT B is address dependent mapping NAT with endpoint dependent filtering.
NAT A is address and port dependent mapping with endpoint dependent filtering.

Show the filter rules created as A and B begin hole punching. Will hole punching succeed?

Q10.9 For a SIP message sent by host A (hostname of `hosta.mappings.org`) to host Y1 in Question Q10.4, show the `Via` header in the response assuming the `rport` extension is used.

Q10.10 Explain the relationship between hole punching and ICE.

References

[1] Egevang, K., and P. Francis, "The IP Network Address Translator (NAT)," RFC 1631, May 1994.

[2] Rekhter, Y., et al., "Address Allocation for Private Internets," BCP 5, RFC 1918, February 1996.

[3] Srisuresh, P., and M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations," RFC 2663, August 1999.

[4] http://www.ietf.org/html.charters/behave-charter.html.

[5] Hain, T., "Architectural Implications of NAT," RFC 2993, November 2000.

[6] Audet, F., and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP," BCP 127, RFC 4787, January 2007.

[7] Senie, D., "Network Address Translator (NAT)-Friendly Application Design Guidelines," RFC 3235, January 2002.

[8] Rosenberg, J., et al., "STUN—Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)," RFC 3489, March 2003.

[9] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols," draft-ietf-mmusic-ice-19 (work in progress), October 2007.

[10] MacDonald, D., and B. Lowekamp, "NAT Behavior Discovery Using STUN," draft-ietf-behave-nat-behavior-discovery-05 (work in progress), November 2008.

[11] Daigle, L., and IAB, "IAB Considerations for Unilateral Self-Address Fixing (UNSAF) Across Network Address Translation," RFC 3424, November 2002.

[12] Rosenberg, J., and H. Schulzrinne, "An Extension to the Session Initiation Protocol (SIP) for Symmetric Response Routing," RFC 3581, August 2003.

[13] Gurbani, V., R. Mahy, and B. Tate, "Connection Reuse in the Session Initiation Protocol (SIP)," draft-ietf-sip-connect-reuse-14 (work in progress), August 2009.

[14] Jennings, C., and R. Mahy, "Managing Client Initiated Connections in the Session Initiation Protocol (SIP)," draft-ietf-sip-outbound-16 (work in progress), October 2008.

[15] Huitema, C., "Real Time Control Protocol (RTCP) Attribute in Session Description Protocol (SDP)," RFC 3605, October 2003.

[16] Rosenberg, J., R. Mahy, and P. Matthews, "Traversal Using Relays Around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)," RFC 5389, October 2008.

[17] Boulton, C., et al., "Best Current Practices for NAT Traversal for Client-Server SIP," Internet Draft draft-ietf-sipping-nat-scenarios-09 (work in progress), September 2008.

11

Related Protocols

This chapter will introduce some related protocols to SIP. Telephony protocols from the PSTN will be discussed such as circuit associated signaling, ISDN, and ISUP. Media gateway control protocols such as MGCP and H.248 will be introduced. Finally, H.323 will be discussed. Note that Jabber and Jingle are covered in Chapter 8.

11.1 PSTN Protocols

Three types of PSTN signaling protocols are mentioned in this text: channel associated signaling (CAS), integrated services digital network (ISDN), and ISDN user part (ISUP). They will be briefly introduced and explained. How these protocols work in the PSTN today are covered in other references [1].

11.1.1 Circuit Associated Signaling

Circuit associated signaling (CAS), also known as channel associated signaling is a legacy technology still used in the PSTN today. The signaling information uses the same audio circuit as the voice path, with digits and characters represented by audio tones. These are the tones that used to be barely discernible at the beginning of some long-distance calls before the ringtone was heard. The tones are called multifrequency (MF) tones. They are somewhat similar to the tones used to signal between a telephone and a central office switch, which are known as dualtone multifrequency (DTMF) tones. Long postdial delay can be introduced because of the time taken to outpulse long strings of digits. Also, CAS was susceptible to fraud, as fraudulent tones could be generated by the caller to make free telephone calls. This type of signaling is still used in trunk circuits between a central office and a corporation's private branch exchange (PBX) switch. DTMF signaling is still commonly used in the PSTN.

11.1.2 ISDN Signaling

Integrated services digital network (ISDN) signaling was developed in the 1980s for all digital telephone connections to the PSTN. The most common types of interfaces are the basic rate interface (BRI) and the primary rate interface (PRI). A BRI can contain two 64-Kbps B-channels for either voice or data and a 16-Kbps D-channel for signaling. BRI was designed as a replacement for conventional telephone lines but requires an ISDN telephone or terminal adapter. PRI was designed for higher volume applications such as PBX trunks. In North America, PRI uses a 1.544 Mbps link called a T-1 or a DS-1, which is divided up into 23 B-channels and one D-channel, with each channel being 64 Kbps. In Europe and much of the rest of the world, it uses a 2.048 Mbps with 30 B-channels and one D channel. The H.323 protocol, described later in this chapter, reuses a subset of the ISDN Q.931 signaling protocol used over the D-channel.

11.1.3 ISUP Signaling

ISDN user part (ISUP) is the part of the signaling system no. 7 (SS7) protocol stack used between telephone switches in the PSTN for call signaling. SS7 is a dedicated packet-switched network used all over the world in the PSTN. This signaling method was developed to overcome some of the delay and security problems with CAS. There are examples of ISUP signaling in the call flow examples of Chapter 17. The adoption of this out-of-band signaling protocol was the first step taken by telecommunications carriers away from circuit-switched networks and towards packet-switched networks. The final step will likely be moving the bearer channels onto a packet-switched network as providers move towards an all-VoIP network using SIP.

11.2 SIP for Telephones

SIP for telephones (SIP-T) is a framework for SIP interworking with the PSTN [2]. It includes two approaches: translation and encapsulation. Translation is the direct mapping between PSTN protocols and SIP. The mapping between common PSTN protocols such as ISUP [3], Q.SIG [4], and others has been defined. Examples of SIP interworking with PSTN protocols including ISDN and CAS are in the SIP PSTN call flows document [5]. In this approach, as much of the information that is common to each protocol are mapped between them, with the remaining values being set to configurable defaults. A SIP call from a PSTN gateway is indistinguishable from a SIP call from a native device, and is handled such by the protocol. However, since not every single parameter in a PSTN signaling message has a counterpart (or has any meaning) in SIP, some information is lost if the call routes back to a PSTN termination point.

Encapsulation is another approach that is only useful for SIP/PSTN gateways. Using this approach, PSTN-to-SIP translation is done to construct the appropriate SIP message, then the PSTN protocol message is encapsulated and included with the SIP message as a message body. If the SIP message is received by another SIP/PSTN gateway, the resulting PSTN signaling message is constructed from both the SIP message *and* the encapsulated PSTN message that was received by the other gateway. This approach offers the possibility of transparency (i.e., no loss of PSTN information as a call is carried across a SIP network). However, this only works in a network in which only one variation of PSTN protocol is used. Unlike Internet protocols, PSTN protocols vary by region and are not compatible without a special type of PSTN switch capable of converting one message format to another. There are many dozens of protocol variants used throughout the world.

Another disadvantage of encapsulation is that the PSTN message bodies must be encrypted if they are transported over the public Internet, or used in a network with native SIP devices. This is because private information can be carried in PSTN messages because PSTN protocols assume a different trust model than an Internet protocol such as SIP. To prevent accidental disclosure of this information, the message bodies must be encrypted by the originating gateway and decrypted by the terminating gateway, which adds significant processing requirements and call setup delay.

Encapsulated PSTN messages are carried as MIME bodies, which have been standardized for both ISUP and QSIG [6].

11.3 Media Gateway Control Protocols

There are a number of protocols used to decompose the operation of a gateway, which are often used in SIP/PSTN gateways. These protocols are known generally as media gateway control protocols. Their relation to a signaling protocol such as SIP is shown in Figure 11.1. Media gateway control protocols are not peer-level signaling protocols—they do not perform the rendezvous and negotiation functions of a signaling protocol such as SIP. Instead, they allow a gateway to be decomposed into a signaling element and a media component. The media component, a media gateway (MG), can provide PSTN trunks or RTP/SRTP connections, which are under the control of the media gateway controller (MGC). The MGC in turn uses a signaling protocol such as SIP or PSTN signaling to setup connections with other elements. Media gateway controllers used to be called *softswitches*. Often, a single or pair of media gateway controllers will control a number of media gateways. Media gateway control protocols are master/slave protocols—the MGC tells the MG what to do—there is no negotiation between them. Some common media gateway control protocols include

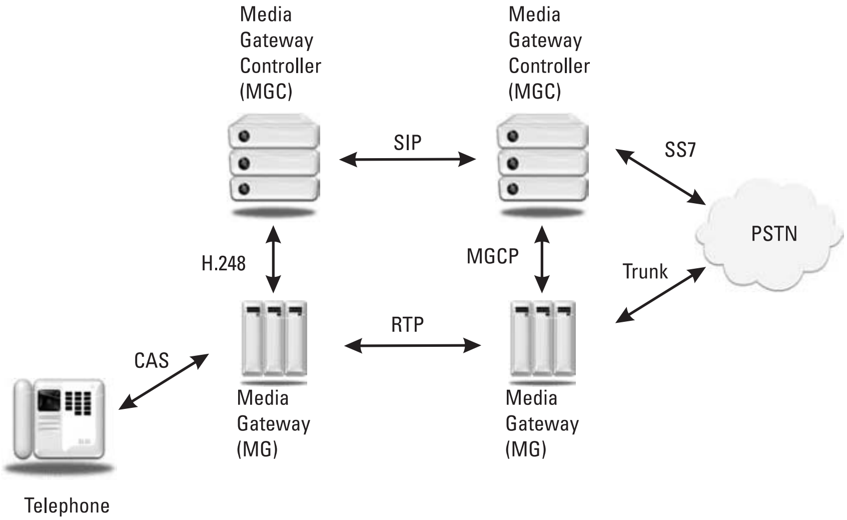


Figure 11.1 SIP and media gateway control protocols.

MGCP [7] and H.248 [8]. H.248 was initially jointly published by the ITU-T as H.248 and in the IETF as MEGACO [9]. The current version is maintained by the ITU-T as H.248.1 version 1 [10].

11.4 H.323

A related Internet communications protocol is the ITU recommendation H.323, entitled "Packet-Based Multimedia Communication." H.323 is introduced as a related protocol to SIP for signaling VoIP and multimedia communication.

11.4.1 Introduction to H.323

H.323 [11] is an umbrella recommendation that covers all aspects of multimedia communication over packet networks. It is part of the H.32x series of protocols that describes multimedia communication over ISDN, broadband (ATM), telephone (PSTN), and packet (IP) networks, as shown in Table 11.1. Originally developed for video conferencing over a single LAN segment, the protocol has been extended to cover the general problem of telephony over the Internet. The first version was approved by the ITU in 1996 and was adopted by early IP telephony networks. Version 2 was adopted in 1998 to fix some of the problems and limitations in version 1. Version 3 was adopted in 1999 and includes modifications and extensions to enable communications over a larger network. Version 4 was adopted in 2000 with some major changes to the protocol. Versions 5 and 6 made very small changes to the protocol.

Table 11.1
TU H.32x Family of Standards

Protocol	Title
H.320	Communication over ISDN networks
H.321	Communication over broadband ISDN (ATM) networks
H.322	Communication over LANs with guaranteed QoS
H.323	Communication over LANs with nonguaranteed QoS (IP)
H.324	Communication over PSTN (V.34 modems)

H.323 references a number of other ITU and IETF protocols to completely specify the environment. Each element of the network is defined and standardized. Figure 11.2 shows the main elements: terminals, gatekeepers, gateways, and multipoint control units (MCUs). Terminals, gateways, and MCUs are network end-devices, often called end points. An end point originates and terminates media streams that could be audio, video, or data, or a combination of all three. At a minimum, all H.323 end points must support basic G.711 PCM audio transmission. Support of video and data are optional. An H.323 gatekeeper is a server that controls a zone, which is the smallest administrative domain in H.323. If a gatekeeper is present, all end points within that zone must register with and defer to the gatekeeper on authorization decisions to place or accept a call. A gatekeeper also provides services to terminals in a zone, such as gateway location, address translation, bandwidth management, feature implementation, and registration. A gatekeeper is not a required element in an H.323 network, but a terminal's capabilities without one are severely limited. A gateway is another optional element in an H.323 network. It interfaces the H.323

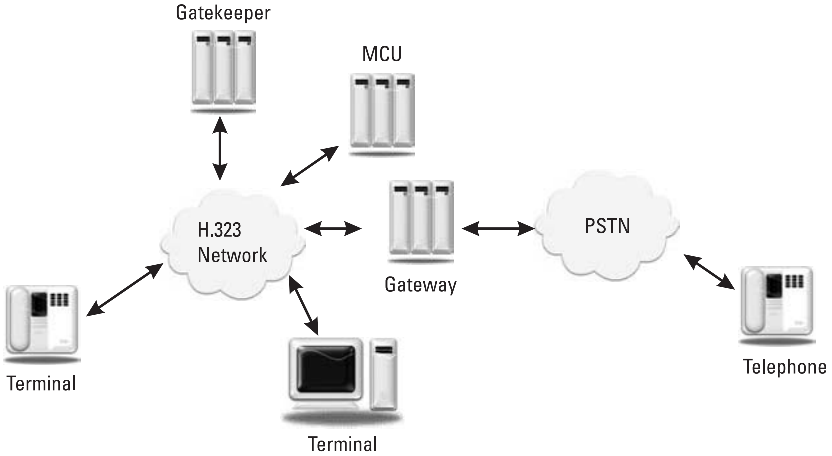


Figure 11.2 Elements of an H.323 network.

network with another protocol network, such as the PSTN. An MCU provides conferencing services for terminals.

Some of the protocols referenced by H.323 are shown in Table 11.2. H.225 is used for registration, admission, and status (RAS), which is used for terminal-to-gatekeeper communication. A modified subset of Q.931 is used for call setup signaling between terminals. (The H.323 usage of Q.931 is not compatible with Q.931 as used in an ISDN network.) H.245 is used for control signaling or media negotiation and capability exchange between terminals. T.120 is used for multipoint graphic communications. H.323 audio codecs are specified in the ITU G.7xx series. Video codecs are specified in the H.26x series. H.323 also references two IETF protocols, RTP and RTCP, for the media transport which are described in Chapter 12. The H.235 recommendation covers privacy and encryption, while H.450 covers supplementary services such as those commonly found in the PSTN (e.g., call forwarding, call hold, and call park).

11.4.2 Example of H.323

Figure 11.3 shows a basic call flow involving two terminals and a gatekeeper. The flow shows the interaction between the various elements and the various protocols used to establish the session. The call begins with an exchange of H.225.0 RAS messages between the calling terminal and the gatekeeper. All RAS messages are transported using UDP. It is assumed that both terminals have already registered with the gatekeeper using the registration request (RRQ) message. The calling terminal sends an admission request (ARQ) message to the gatekeeper containing the address of the called terminal and the type of session desired. The address could be specified as an H.323 alias, E.164 telephone number, e-mail address, or URL [12]. The gatekeeper knows about all calls in the zone it controls; it decides if the user is authorized to make a call and if there is enough bandwidth or other resources available. In this example, there is enough bandwidth, so the gatekeeper allows the call to continue by sending an admission confirmation

Table 11.2
Protocols Referenced by H.323

Protocol	Description
H.225	Registration, admission, and status (RAS) and call signaling
H.245	Control signaling (media control)
T.120	Multipoint graphic communication
G.7xx	Audio codecs
H.26x	Video codecs
RTP	Real-time transport protocol (RFC 3550)
RTCP	RTP control protocol (RFC 3550)
H.235	Privacy and encryption
H.450	Supplementary services

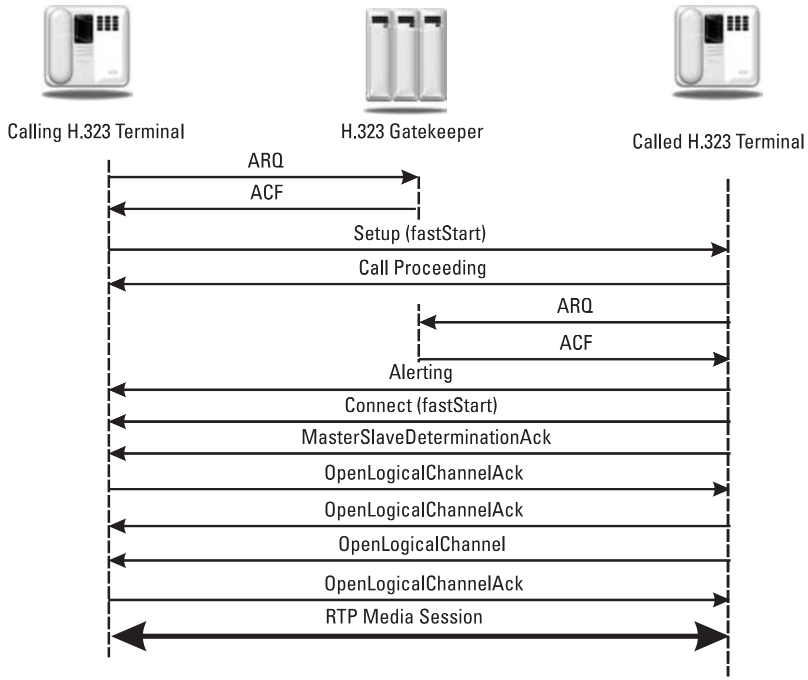


Figure 11.3 H.323 call flow example.

(ACF) message. The ACF indicates to the calling terminal that end-point message routing, or the direct exchange of H.225 call signaling messages with the called terminal, is to be used. Alternatively, the gatekeeper can require gatekeeper routed signaling, where the gatekeeper acts like a proxy and forwards all signaling messages between the terminals. The gatekeeper has also translated the destination in the ARQ into a transport address that was returned in the ACF.

The calling terminal is now able to open a TCP connection to the called terminal using the transport address returned in the ACF and send a Q.931 `Setup` message to the called terminal. The called terminal responds with a `Call Proceeding` response to the calling terminal. The called terminal must also get permission from the gatekeeper before it accepts the call, so an ARQ is sent to the gatekeeper. When it receives the ACF from the gatekeeper, the called terminal begins alerting the user and sends an `Alerting` message to the calling terminal. When the user at the calling terminal answers, a `Connect` message is sent. There is no acknowledgment of messages because all these messages are sent using TCP, which provides reliable transport. These call signaling messages used in H.323 are a subset of the Q.931 protocol that covers ISDN D-channel signaling.

Figure 11.3 shows the use of H.323 FastStart, in which the `Setup` message contains the `TerminalCapabilitySet` information. This saves multiple messages and round trips compared to opening a second TCP connection between the terminals. In H.245 tunneling, a separate H.245 control channel is not opened. Instead, H.245 messages are encapsulated in Q.931 messages in the call signaling channel. This saves overhead in opening and closing a second TCP connection. Now, the terminals begin sending RTP media packets and also RTCP control packets using the IP addresses and port numbers exchanged in the `OpenLogicalChannel` messages.

Figure 11.4 shows a call tear down sequence, which either terminal may initiate. In this example, the called terminal sends an `EndSessionCommand` message in the H.245 control signaling channel. The other terminal responds with an `EndSessionCommand` message in the H.245 control signaling channel, which can now be closed. The called terminal then sends a disengage request (`DRQ`) message and receives a disengage confirmation (`DCF`) message from the gatekeeper. This way, the gatekeeper knows that the resources used in the call have now been freed up. A call detail record (CDR) or other billing record can be written and stored by the gatekeeper. Next, a Q.931 release complete message is sent in the call signaling connection, which can then be closed. Finally, the other terminal sends a `DRQ` to the gatekeeper over UDP and receives a `DCF` response.

The call flows in Figures 11.3 and 11.4 show direct end-point signaling, in which the calling terminal opens TCP connections to the called terminal and exchanges H.225.0 and H.245 messages. In the `ACF` response to the calling terminal, the gatekeeper can require gatekeeper routed signaling, where the call signaling and control signaling channels are opened with the gatekeeper, who then opens the channels with the called terminal. In this way, the gatekeeper stays in

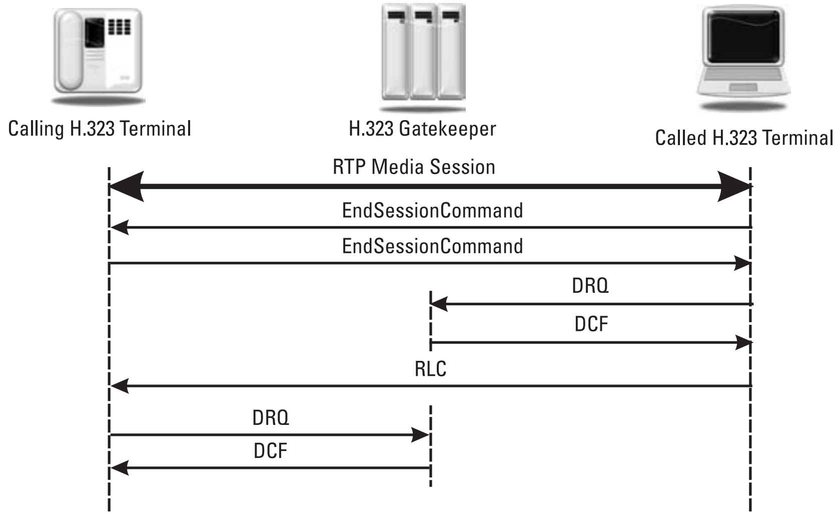


Figure 11.4 H.323 call teardown sequence.

the signaling path and proxies all signaling messages. This allows the gatekeeper to know the exact call state and be able to invoke features.

11.4.3 Versions

There are six versions of H.323, which reflect the evolution of this protocol. H.323 is fully backwards compatible, so gatekeepers and terminals must support flows and mechanisms defined in all previous versions. Version 1 was approved in 1996 and was titled "Visual Telephone Systems over Networks with Non-Guaranteed Quality of Service." Version 2 included alternative call setup schemes to speed up the call setup. Two schemes were added to H.323, called FastStart and H.245 tunneling. Versions 3 and 4 added more features to H.323 and additional annexes. Of interest to Internet devices is the support for H.323 URLs [12], full UDP support instead of TCP, and also the standardization of the use of DNS by H.323 in Annex O. Version 6 is the current version of H.323.

H.323 has carved out two niche areas in current deployed systems—it is widely deployed in small PSTN replacement networks for handling simple phone calls, and it dominates the IP videoconferencing market. Simple PSTN replacement networks that only originate and terminate phone calls without even basic features do not use most of the key advantages of SIP. As a result, they have little incentive to upgrade to SIP until the widespread adoption of SIP eventually makes SIP gateway ports much cheaper than H.323 ports. However, since SIP is earlier in its development cycle than H.323, this is not likely to happen for a number of years. New implementations of these types of systems will likely deploy SIP from the start, seeking to "future proof" the investment, but there is little incentive for deployed systems to upgrade. Also, the availability of commercial SIP to H.323 signaling gateways based on [13] allows both networks to work together and complete calls. The videoconferencing market is dominated by PSTN ISDN devices, which have been deployed since 1990. Since H.323 shares the same heritage, it was designed to easily interwork with these existing systems.

References

[1] Anttalainen, T., *Introduction to Telecommunications Network Engineering,* Norwood, MA: Artech House, 1999.

[2] Vemuri, A., and J. Peterson, "Session Initiation Protocol for Telephones (SIP-T): Context and Architectures," BCP 63, RFC 3372, September 2002.

[3] Camarillo, G., et al., "Integrated Services Digital Network (ISDN) User Part (ISUP) to Session Initiation Protocol (SIP) Mapping," RFC 3398, December 2002.

[4] Elwell, J., et al., "Interworking Between the Session Initiation Protocol (SIP) and QSIG," BCP 117, RFC 4497, May 2006.

[5] Johnston, A., et al., "Session Initiation Protocol (SIP) Public Switched Telephone Network (PSTN) Call Flows," BCP 76, RFC 3666, December 2003.

[6] Zimmerer, E., et al., "MIME Media Types for ISUP and QSIG Objects," RFC 3204, December 2001.

[7] Arango, M., et al., "Media Gateway Control Protocol (MGCP) Version 1.0," RFC 2705, October 1999.

[8] International Telecommunication Union, "Implementors' Guide for Recommendation H.248.1 Version 1 (03/2002) ('Media Gateway Control Protocol')," ITU-T Recommenda tion H.248.1, April 2006.

[9] Cuervo, F., et al., "Megaco Protocol Version 1.0," RFC 3015, November 2000.

[10] Taylor, T., "Reclassification of RFC 3525 to Historic," RFC 5125, February 2008.

[11] "Packet-Based Multimedia Communications Systems," ITU Recommendation H.323, 2006.

[12] Levin, O., "H.323 Uniform Resource Locator (URL) Scheme Registration," RFC 3508, April 2003.

[13] Schulzrinne, H., and C. Agboh, "Session Initiation Protocol (SIP)-H.323 Interworking Requirements," RFC 4123, July 2005.

12

Media Transport

Establishing media sessions is one of the most important applications of SIP in Internet communications. An understanding of the issues relating to media transport of voice, video, DTMF, and text helps motivate the media negotiation capabilities of SIP. In this chapter, the Real-Time Transport Protocol (RTP) will be introduced as the protocol that transports actual media samples. The basic steps in audio and video media encoding and decoding are discussed, along with the effects of common Internet impairments. The RTP header format is covered along with common RTP topologies. The RTP Control Protocol (RTCP) is introduced as a way to monitor call quality. RTP profiles and common codes are discussed—both PSTN codecs and Internet codecs. Common audio and video codecs are discussed. Finally, DTMF transport and conversational text are covered.

12.1 Real-Time Transport Protocol (RTP)

Real-Time Transport Protocol [1] was developed to enable the transport of real-time datagrams containing voice, video, or other information over IP. RTP was not the first VoIP protocol used on the Internet. Network Voice Protocol (NVP) [2] was implemented in 1973 to carry real-time voice communications over the Internet. Early versions of RTP, first implemented in 1992, were used to transport voice over the Internet's multicast backbone (MBONE). Both H.323 and SIP use RTP for media transport, making it the most common standard for Internet communications.

RTP is defined by the IETF proposed standard RFC 3550 (which updates the original RFC 1889). RTP does not provide any quality of service over the IP network—RTP packets are handled the same as all other packets in an IP network. However, RTP allows for the detection of some of the impairments introduced by an IP network, such as:

- Packet loss;
- Variable transport delay;
- Out of sequence packet arrival;
- Asymmetric routing.

Here is how RTP fits into the common media processing steps.

1. *Coding.* The coding step involves analog to digital conversion (A/D), which is implemented by low pass filtering, followed by sampling. The determination of how many bits per sample is specific to a particular codec (coder/decoder) algorithm. The particular codec used is transported by RTP in the payload type field. The sampling rate is carried in the offer/answer exchange in SDP, which negotiates the media session.
2. *Packetization.* The packetization step involves breaking the codec sample data into individual datagrams for transport. The determination of packet size is based on a tradeoff between packetization delay (how many sampling intervals must pass before enough data is ready for the datagram) and transport efficiency (each datagram has the fixed overhead of the RTP header and lower layer headers). Typically, packet sizes are chosen to be small so that packetization time is around 20 ms to 30 ms. Packetization involves adding the RTP header to the codec payload.
3. *Tranport.* RTP, as the name suggests, has a real-time nature, which requires a minimum latency (delay) across the Internet. There is never time to detect a missing packet, signal the loss, and wait for a retransmission. This might be possible for nonreal-time streaming media, but not real-time media. As a result, RTP does not usually use TCP transport but instead uses UDP transport. As a result, datagrams may be lost or may arrive out of sequence. Various fields in the RTP header field allow the detection of this.
4. *Depacketization.* The depacketization step involves removing the RTP header from the codec payload.
5. *Buffering.* The buffering step involves storing or buffering the codec samples before beginning playback. The choice of the buffer size for this step is critical for media quality. Too short a buffer will result in the buffer emptying and gaps in the media playback, while too long a buffer will introduce unpleasant latency. Adapting the size of the play-

back buffer when jitter or delay variation is occurring is best for media quality.

6. *Decoding.* The decoding step involves sending the codec packets to the codec algorithm. The right codec is chosen based on the received payload type in the RTP header.
7. *Playback.* The playback step involves rendering the media to the user as audio, video, or perhaps text (as we shall see in real-time text or text over IP, ToIP).

In terms of media quality, the two most important factors are the packet loss rate and the end-to-end latency. Lost packets mean gaps in the playback stream that the codec algorithm must try to compensate for. Different codecs use different techniques for packet loss concealment (PLC). For example, interpolation can be used to try to predict the missing samples based on received samples either side of the lost ones. A simple replay algorithm can be useful for some media types. Silence or comfort noise insertion can be used to prevent users noticing the dead air of lost samples. Some codecs employ forward error correction (FEC), which allows partial reconstruction of missing packets under low loss conditions. Note that packets aren't really "lost" on the Internet, instead they are discarded by routers in the Internet due to congestion, or discarded by the RTP stack due to out of order arrival or late arrival resulting in missing their playback interval.

The end-to-end latency of real-time communications in general must be kept less than 150 ms. Longer latency than this affects the perceived quality of the call, resulting in users interrupting each other and starting and stopping when both parties speak at the same time. There are many sources of delay in the media path. The codec itself introduces delay as it gathers at least one sample or frame before beginning coding and decoding. The packetization step introduces a delay of around 20 ms, the time it takes to gather a full packet's worth of data before sending it over the Internet. Transport delays are added by the routers and switches that forward and process the IP packets across the Internet. And finally, the buffering delay of the receiver to deal with jitter or delay variation also introduces latency. Some of these sources are shown in Figure 12.1.

Real-Time Transport Protocol (RTP) is an application layer protocol that uses UDP for transport over IP. RTP is not text encoded, but uses a bit-oriented header similar to UDP and IP. RTP version 0 is only used by the vat audio tool for MBONE broadcasts. Version 1 was a pre-RFC implementation and is not in use. The current RTP version 2 packet header has 12 octets. RTP was designed to be very general; most of the headers are only loosely defined in the standard; the details are left to profile documents. The header contains:

- Version (v): This 2-bit field is set to 2, the current version of RTP.

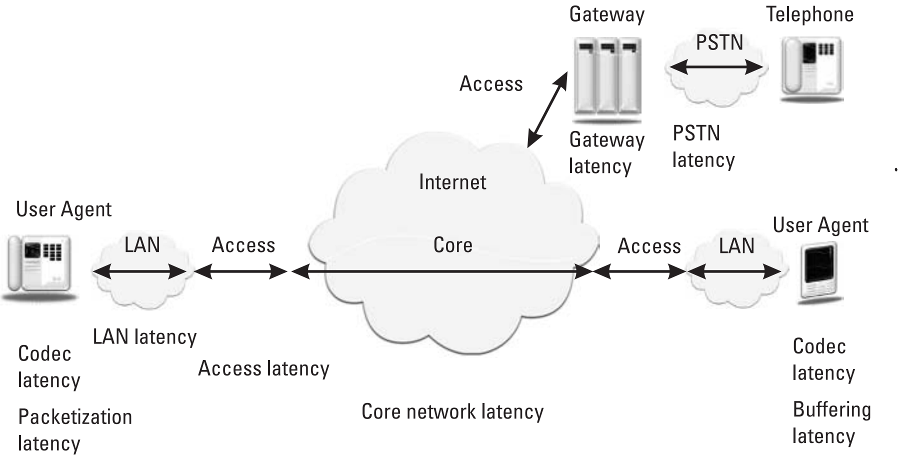


Figure 12.1 Sources of latency and packet loss on the Internet.

- Padding (P): If this bit is set, there are padding octets added to the end of the packet to make the packet a fixed length. This is most commonly used if the media stream is encrypted.
- Extension (X): If this bit is set, there is one additional extension following the header (giving a total header length of 14 octets). Extensions are defined by certain payload types.
- CSRC count (CC): This 4-bit field contains the number of content source identifiers (CSRC) that are present following the header. This field is only used by mixers that take multiple RTP streams and output a single RTP stream.
- Marker (M): This single bit is used to indicate the end of a complete frame in video, or the start of a talk-spurt in silence-suppressed speech.
- Payload type (PT): This 7-bit field defines the codec in use. The value of this field matches the profile number listed in the SDP.
- Sequence Number: This 16-bit field is incremented for each RTP packet sent and is used to detect missing/out of sequence packets.
- Timestamp: This 32-bit field indicates in relative terms the time when the payload was sampled. This field allows the receiver to remove jitter and to play back the packets at the right interval assuming sufficient buffering.
- Synchronization source identifier (SSRCI): This 32-bit field identifies the sender of the RTP packet. At the start of a session, each participant chooses an SSRC number randomly. Should two participants choose the same number, they each choose again until each party is unique.

- Contributing source identifier (CSRC): There can be none or up to 15 instances of this 32-bit field in the header. The number is set by the CSRC count (CC) header field. This field is only present if the RTP packet is being sent by a mixer, which has received RTP packets from a number of sources and sends out combined packets. A nonmulticast conference bridge would utilize this header.

RTP allows detection of a lost packet by a gap in the Sequence Number. Packets received out of sequence can be detected by out-of-sequence Sequence Numbers. Note that RTP allows detection of these transport-related problems but leaves it up to the codec to deal with the problem. For example, a video codec may compensate for the loss of a packet by repeating the last video frame, while an audio codec may play background noise for the interval. Variable delay or jitter can be detected by the Timestamp field. A continuous bit rate codec such as PCM will have a linearly increasing Timestamp. A variable bit rate codec, however, which sends packets at irregular intervals, will have an irregularly increasing Timestamp, which can be used to play back the packets at the correct interval.

RTP media sessions are unidirectional—they define how media is sent from the media source to the media sink. As such, a normal bidirectional media session is actually two RTP sessions, one in each direction.

In a multimedia session established with SIP, the information needed to select codecs and send the RTP packets to the right location is carried in the SDP message body. Under some scenarios, it can be desirable to change codecs during an RTP session. An example of this relates to the transport of dual tone multiple frequency (DTMF) digits. A low bit rate codec that is optimized for transmitting vocal sounds will not transport the superimposed sine waves of a DTMF signal without introducing significant noise, which may cause the DTMF digit receiver to fail to detect the digit. As a result, it is useful to switch to another codec when the sender detects a DTMF tone. Because an RTP packet contains the payload type, it is possible to change codecs on the fly without any signaling information being exchanged between the UAs. On the other hand, switching codecs in general should probably not be done without a SIP signaling exchange (re-INVITE) because the call could fail if one side switches to a codec that the other does not support. The SIP re-INVITE message exchange allows this change in media session parameters to fail without causing the established session to fail.

The use of random numbers for SSRC provides a minimal amount of security against “media spamming” where a literally *uninvited* third party tries to break into a media session by sending RTP packets during an established call. Unless the third party can guess the SSRC of the intended sender, the receiver will detect a change in SSRC number and either ignore the packets or inform the user that something is going on. This behavior for RTP clients, however, is not universally accepted, because in some scenarios (wireless hand-off, announcement server, call center, and so forth) it might be desirable to send media from multiple sources during the progress of a call.

RTP supports encryption of the media through the secure RTP (SRTP) profile discussed in Chapter 14. RTP supports a number of different topologies [3] including unicast (point-to-point) and multicast (point-to-multipoint). They are summarized in Table 12.1. In RTP, a translator is an element that converts the codec or sampling rate of an RTP stream. An RTP mixer is an element that combines multiple RTP streams into a single RTP stream in a media specific way.

At the start of an RTP session, the sender randomly chooses an initial value of the timestamp and SSRC. If both the sender and receiver happen to choose the same SSRC, both sides choose again to ensure each have a different SSRC. Media samples are encoded by the codec. Based on the packetization interval, once a complete frame of media data is available, the RTP header is populated and the packet sent. The sampling instant is used to update the timestamp. The sequence number is updated for each RTP packet sent. The receiver first validates the RTP header, using the sequence number to determine if any packets have been lost or received out of sequence. The timestamp is used to playout the media sample by the codec.

12.2 RTP Control Protocol (RTCP)

The RTP Control Protocol (RTCP) is a related protocol also defined in RFC 3550 that allows participants in an RTP session to send each other quality reports and statistics, and exchange some basic identity information. The five types of RTCP packets are shown in Table 12.2. RTCP has been designed to scale for very large conferences. Because RTCP traffic is all overhead, the bandwidth allocated to these messages remains fixed regardless of the number of participants. That is, the more participants in a conference, the less frequently RTCP packets

Table 12.1
RTP Topologies

Point to point
Point to multipoint using multicast
Point to multipoint using an RTP translator
Point to multipoint using an RTP mixer
Point to multipoint using video switching MCUs
Point to multipoint using RTCP-terminating MCU
Nonsymmetric mixer/translators
Combined topologies

Table 12.2
RTCP Reports

Sender report (SR)
Receiver report (RR)
Source description (SDES)
Bye (BYE)
Application specific (APP)

are sent. For example, in a basic two-participant audio RTP session, the RTP/AVP profile states that RTCP packets are to be sent about every 5 seconds; for four participants, RTCP packets can be sent every 10 seconds. Sender reports (SR) or receiver reports (RR) packets are sent the most frequently, with the other packet types being sent less frequently. The use of reports allows feedback on the quality of the connection including information such as:

- Number of packets sent and received;
- Number of packets lost;
- Packet jitter.

By default, RTCP uses the next highest port from the RTP port, although this can be changed in the offer/answer exchange as discussed in Chapter 13.

12.2.1 RTCP Reports

RTCP is always sent as a compound packet. This means that every RTCP packet starts with a sender report (SR) or receiver report (RR), then any additional packets. As their name suggests, sender reports are sent by media senders while receiver reports are sent by media receivers. Since RTP is unidirectional, a bidirection media session will have two RTP sessions and two RTCP sessions. A source description (SDES) packet is used to exchange information about the sender or receiver. A bye (BYE) packet is used to leave a multicast session. An application specific (APP) packet is used for RTCP extensions. An important RTCP extension is described in the next section.

12.2.2 RTCP Extended Reports

RTCP extended reports (RTCP-XR) [4] defines seven additional report blocks. They were defined due to limitation of the basic SR and RR. For example, the receiver report contains information about the average packet loss rate. However, for call quality, information about burst packet loss is much more important than average packet loss, since a good codec can cope with individual lost packets but not a long sequence of lost packets. In addition, RTCP-XR defines a way to estimate actual voice call quality and exchange this information. Deriving this information from existing receiver reports is not possible. As a result, the definition of RTCP extended reports has driven additional implementation of RTCP.

12.3 Compression

RTP does not provide very efficient transport of media. For example, consider the iLBC codec used in the 12.2 kb/s mode with 20 ms packetization time (ptime) transported over RTP, UDP, IPv4, and Ethernet. The size in octets of each frame of codec data can be calculated using the formula:

$$frame = \frac{bw * ptime}{8}$$

where *frame* is the frame size in octets (8 bits, or a byte), *bw* is the codec bandwidth, and *ptime* is the packetization time. For this example, each frame would contain 38 octets of codec data. RTP has a 12 octet header, UDP adds a 16 octet header, while IPv4 with no options adds 20 octets. Ethernet (IEEE 802.3) adds a 13-octet header and a 3-octet footer. As a result, the header overhead for this example is 60 octets! Overhead makes up over 60% of the total packet size. For normal Internet communications, this is how RTP is utilized. However, for some applications where RTP is to be used over low bit rate or wireless links, compression is performed. Note that saving bandwidth is only one reason to do compression. Compression can also reduce serialization delays when sending packets over very low speed links.

One method of compression is compressed RTP (CRTP) [5]. This method only compresses the RTP header fields, using the fact that many parts of the header are identical in every packet. For example, V, P, X, CC, PT, and SSRC typically do not change once a session has been setup, so they do not need to be sent every packet. Other parts of the header such as sequence numbers and timestamps can be sent as deltas, resulting in saved bandwidth.

Another method that compresses the entire RTP/UDP/IP stack is robust header compression (ROHC) [6]. ROHC can compress 40 octets of overhead into 2 octets. To do this, codebooks are used to encode and decode common elements. Codebooks can be either static—predefined for a given protocol—or dynamic—constructed and used during a given session. ROHC can also be used to compress SIP and the stack below SIP.

Although UDP is normally used, it is possible to transport RTP over a stream transport such as TCP. To do this, a framing method is used, which is defined in [7]. If TCP is used, the retransmissions of TCP must be carefully managed or the latency of the session will increase with every retransmission, resulting in very poor performance. Also, when using TCP for media, the roles of each endpoint must be negotiated. One endpoint will be active, and initiate opening the TCP connection, while the other endpoint will be passive, listening on a port for an open request.

12.4 RTP Audio Video Profiles

The use of profiles enables RTP to be an extremely general media transport protocol. The current audio video profiles defined by RFC 3551 [8] and others are listed in Table 12.3. Four are defined, although only the first one is widely implemented. As secure Internet communications are deployed, the use of the secure audio and video profile (SAVP) is increasing, as described in Chapter 14. The most common profile is the RTP profile for audio and video conferences with minimal control, also known as the RTP/AVP profile. RTP/AVP makes the following specifications for RTP:

- UDP is used for underlying transport.
- RTP port numbers are always even—the corresponding RTCP port number is the next highest port, which is always an odd number.
- No header extensions are used.

Some common audio and video codecs are listed in Tables 12.4 and 12.5. The codecs listed with a payload number in the tables use a static payload number. The RTP/AVP profile document lists details of these codecs, or a reference for the details is provided. Codecs shown with a payload number of dynamic must use a dynamic payload in the range 96–127. Dynamic payloads must be defined dynamically during a session. The minimum payload support is defined as 0 (PCMU) and 5 (DVI4) (although in practice, most only support PCM). The document recommends dynamically assigned UDP port numbers, although ports 5004 and 5005 have been registered for use of the profile and can be used instead. The standard also describes the process of registering new payload types with IANA.

Table 12.3
Defined RTP Profiles

Profile	Name	Specification
RTP profile for audio and video Conferences with minimal control	RTP/AVP	RFC 3551
The secure real-time transport protocol	RTP/SAVP	RFC 3711
RTP audio-visual profile with feedback	RTP/AVPF	RFC 4585
Extended secure RTP profile for RTCP-based feedback	RTP/SAVPF	RFC 5124

Table 12.4
Common RTP/AVP Audio Payload Types

Payload	Codec	Bit Rate
0	PCMU	64 kb/s
3	GSM	13 kb/s
4	G.723	5.3 or 6.3 kb/s
5	DVI4	32 kb/s
8	PCMA	64 kb/s
9	G.722	128 kb/s
18	G.729	8 kb/s
Dynamic	iLBC	13.33 or 15.2 kb/s
Dynamic	AMR	1.8–12.2 kb/s
Dynamic	AMR-WB	6.6–23.85 kb/s
Dynamic	SPEEX	2–44 kb/s
Dynamic	MP3	8–320 kb/s

Table 12.5
Common RTP/AVP Video Payload Types

Payload	Codec	Type
26	JPEG	JPEG video
31	H261	H.261
32	MPV	MPEG-I and MPEG-II
34	H263	H.263
Dynamic	H264	MPEG-4

12.4.1 Audio Codecs

There are two main types of audio codecs—PSTN codecs and Internet codecs. PSTN codecs were developed for the circuit-switched world of the PSTN. They have been designed to minimize bandwidth but were not designed to function over a packet switched network such as the Internet. In particular, their quality rapidly degrades under conditions of packet loss, delay variation (jitter), and other common Internet impairments. Typically, these codecs are only usable when packet loss is less than 1%. Some examples include G.711 (PCM), G.721, G.723, and G.729A. G.711 is also known as pulse coded modulation (PCM) which has two variants, μ-law companding used mainly in the United States and Japan or A-law companding used in the rest of the world. G.711 is uncompressed, with 8 bits samples at 8,000 samples per second resulting in a 64 kb/s data stream. The others implement compression or linear prediction to reduce the bandwidth requirement. However, this compression makes their performance more sensitive to packet loss. For example, a single RTP packet of G.711 lost only affects that sampling interval while a single packet of G.729A can affect audio quality for a number of sampling intervals. These codecs typically require less than 1% packet loss. PSTN codecs are designed based on 8 kHz sampling due a design limitation of the PSTN network which is not present on the Internet. Many of these codecs also have significant intellectual property (IPR) fees and licensing associated with them. Some modern PSTN codecs overcome some of these problems. For example, the adaptive multirate codec (AMR) [9] was developed by the mobile phone industry with packet transport in mind. As a result, it has reasonable performance under packet loss. There is also a wideband version known as AMR-WB. However, AMR still has significant intellectual property and licensing costs.

In contrast, Internet audio codecs were designed with the Internet in mind. They are designed to give good performance even under conditions of packet loss and delay variation. Also, many provide better-than-PSTN quality by ignoring the 8 kHz sampling limitation. Internet codecs that provide this higher quality are often known as wideband codecs. Examples of Internet codecs include the Internet low bit rate codec (iLBC) [10] and SPEEX [11]. Both of these codecs have no intellectual property or licensing costs, and open source implementations can be found on the Internet. These codecs still have reasonable quality even under conditions of up to 10% packet loss.

12.4.2 Video Codecs

Many of the considerations that apply to audio media transport also apply to video transport. However, there are some key differences. For example, the large amount of information present in every video frame, and the frequency of frame updates means that video requires very high bandwidth. Sending uncompressed video is essentially impractical over the Internet. There are two main techniques in video compression. One is intraframe compression, where information in a single frame is compressed. These frames are called I-frames or key frames. For example, a lossy compression technique such as JPEG could be used. A frame will often be transported in multiple RTP packets. In this case, the marker (M) bit is set on the last packet of a frame to indicate to the codec that the frame is ready for processing and rendering. The other compression technique is interframe compression, where successive frames are compared and differences and predictions made. Predicted frames are known as P-frames and are made relative to a key frame or I-frame. Since often only a small amount of the entire screen changes between each frame, this can result in very compressed, relatively static images. For example, in a telepresence video conference, the I-frame would encode the background image and the face of the person while P-frames could carry their moving lips, blinking eyes, and waving hands. In addition bipredicted frames which are based on multiple P-frames can be used to increase compression. For moving objects in an image, motion vectors of macroblocks can be used to achieve excellent compression. As a result, a typical video media stream will consist of combinations of I-frames, P-frames, and B-frames.

Since a frame is typically sent over a number of packets, a single lost packet may cause an entire frame to be discarded by the codec. The effect on the quality of the picture depends on the type of frame lost. If it was an I-frame, the loss will have a major impact on quality, and future P-frames and B-frames will result in an incomplete picture until another I-frame is sent. If the lost frame was a P-frame or B-frame, the impact will be less and for a shorter duration. Video codecs employ a number of loss concealment techniques. For example, some use repetition of previous frames, which can work for stationary or slowly moving images. Spatial and frequency interpolation can be used to try to generate lost frames. Also, sending frames using interleaving can provide protection against burst errors. In this approach, parts of different frames are sent out of sequence.

The most common standard video codecs are the H.26x series. H.261 was a very early codec used for video conferencing. H.262 is essentially the same as MPEG-2 which is used in DVDs and HDTV broadcasts. H.263 is commonly used on the Internet today through the Flash Player plugin used by video sharing sites. Many video systems are moving to H.264, which uses MPEG-4 encoding. H.264 is the recommended codec by YouTube [12]. Motion JPEG is a high quality video codec that only uses I-frames with JPEG compression within each frame. It uses much more bandwidth than H.26x codecs but provides a high quality picture even during fast action and motion sequences. In addition, there are many proprietary video codecs in use over the Internet.

12.5 Conferencing

Audio conferencing and videoconferencing are important applications that utilize media transport. Each of these applications has their own media requirements. The details of SIP conferencing are covered in Section 9.7. Audio conferencing requires an audio mixer: a device which combines multiple RTP audio streams into a single stream. A mixer in RTP synchronizes the input media streams then combines them together. The SSRC of each media stream, which was included (mixed) into the resulting stream, will be copied into the contributing SSRC (CSSRC) field of the header. This allows speaker identification during the conference. A typical mixing strategy uses $N - 1$ mixing—that is, the N loudest speakers will have their media combined and shared, but each speaker will not hear themselves—they get the $N - 1$ mix. Thus for $N = 3$, the mixer will produce four distinct mixes, one with all three speakers that is received by nonspeakers, and three with only two of the speakers. Each speaker will get a version of this mix. An audio mixer is sometimes called a multipoint control unit (MCU).

Video mixing can involve combining multiple video streams into a single stream known as tiling (sometimes called “Hollywood squares” if they are presented in a checkerboard arrangement), or by just selecting a video streaming. If video follows audio is used, the video will switch to the loudest speaker. In other cases, users in a videoconference can select which video stream or streams they view, sometimes from a set of thumbnail images. When video switching is occurring, the new video stream needs an I-frame or key frame to be sent immediately, otherwise, the sequence of P-frames and B-frames being sent will not provide a complete image without the I-frame they reference. This is accomplished using fast update signaling between the video mixer and the video source. One method uses an XML message [13] to convey this signaling. Another method uses a special RTCP message [14] and the audio video profile with feedback (AVPF).

12.6 ToIP—Conversational Text

Conversational text, or text over IP (ToIP), is a bidirectional real-time exchange of text characters. Unlike e-mail where the message is only sent when the user hits send, or instant messaging where the message is sent when the user presses enter or return, conversational text messages are sent character by character, usually in full duplex (i.e., both sides can type at the same time). Devices in the PSTN to accomplish this are known as telecommunications devices for the deaf (TDD). Sometimes they are used only for one direction of the call; a human relay operator receives the conversational text messages from one party and speaks the words to the other party. The PSTN uses many different standards and devices for this communication. T.140 [15] is an International Telecommunications Union (ITU) format for encoding conversational text. RTP has a payload for transporting T.140 information [16] over UDP. This payload can use redundant transmission so that individual lost RTP packets will not result in dropped characters. For conversational text to be truly conversational, the end-to-end latency must be less than 300 ms. An industry group known as the Real-Time Text Taskforce (R3TF) [17] has been formed to help the adoption of this technology to the Internet.

12.7 DTMF Transport

Dual tone multifrequency (DTMF) tones are commonly used on the PSTN for dialing telephone numbers. Although Internet communications do not utilize dialing, DTMF still must be transported and supported for user signaling—for example, when entering a personal identification number (PIN) or password to access voicemail or interactive voice response (IVR) systems. Calling card, telephone banking, and many other systems use DTMF tones for signaling. DTMF, as the name suggests, generates two superimposed sine waves at particular frequencies to send a particular digit (0–9, *, #, or, less commonly, A–F). In the PSTN, DTMF is typically encoded the same way as voice. However, low bit rate codecs, which are optimized for encoding voice, often do not reliably encode DTMF tones. As a result, there is a need to transport DTMF not as sine waves but as actual digits. This is especially appealing for devices such as SIP phones and mobile phones which only need to generate DTMF. A payload known as telephone-events [18] has been defined for transport over RTP. This approach is commonly known in the industry as RFC 2833 tones, where RFC 2833 [19] was the original RFC specification for telephone-events.

The payload contains:

- Event: an octet used to encode the event such as the DTMF key pressed;
- End (E) bit: a bit used to indicate the end of the event;
- Reserved (R) bit: a bit reserved for future use, set to zero and ignored;
- Volume: 6 bits for the level of the tone in dBm0;
- Duration: 16 bits used for a timestamp for the event duration.

When a user presses a DTMF key, or a gateway detects a DTMF tone in band, an RTP telephone-events packet is created and sent. The marker (M) bit in the RTP header is set to indicate that this is the first packet sent. If the key is still being pressed or detected, the duration field will not be valid but should be set to a value higher than the update time. Update RTP telephone-events are sent typically every 50 ms. The RTP timestamp for these update packets will be the same as the first RTP packet but the duration will increase for each. When the key is released or the DTMF tone is no longer detected, a final RTP telephone-event packet is created. The end (E) bit will be set and the duration field will contain the actual tone duration. This final RTP packet will be resent two more times for redundancy. If the DTMF keypress or tone duration is less than the update time, only three RTP telephone events will be sent. The first will have the M bit set, all will have the E bit sent and the duration field will indicate the duration.

12.8 Questions

Q12.1 List the purpose of packet loss concealment. List some methods for packet loss concealment in audio codecs.

Q12.2 Why does RTP usually use UDP transport?

Q12.3 Explain the purpose of the sequence and SSRC fields in an RTP packet.

Q12.4 Calculate the bandwidth required for the SPEEX codec operating at 7.5 kb/s, 25 ms packetization time, assuming transport over UDP, IPv4 (no extensions), and 100BaseT Ethernet.

Q12.5 Explain the differences between RTCP receiver reports and RTCP extended reports.

Q12.6 Describe the three different types of video frames. Explain the need for fast update in a video conferencing system. Which types of frames does motion JPEG use?

Q12.7 How many telephone events packets will be sent if the DTMF key # is pressed and held for 185 ms? Assume the recommended update interval. How many bits in total will be sent, assuming transport over UDP, IPv4 (no extensions), and 1000BaseT Ethernet?

Q12.8 Explain the difference between instant messaging and conversational text.

Q12.9 Describe common audio and video mixing techniques.

Q12.10 Explain the need for the payload type field in the RTP header. Use an example with the iLBC codec and telephone-events to make your point.

References

[1] Schulzrinne, H., et al., "RTP: A Transport Protocol for Real-Time Applications," STD 64, RFC 3550, July 2003.

[2] Cohen, D., "Specifications for the Network Voice Protocol (NVP)," RFC 741, November 1976.

[3] Westerlund, M., and S. Wenger, "RTP Topologies," RFC 5117, January 2008.

[4] Friedman, T., R. Caceres, and A. Clark, "RTP Control Protocol Extended Reports (RTCP XR)," RFC 3611, November 2003.

[5] Casner, S., and V. Jacobson, "Compressing IP/UDP/RTP Headers for Low-Speed Serial Links," RFC 2508, February 1999.

[6] Bormann, C., et al., "Robust Header Compression (ROHC): Framework and Four Profiles: RTP, UDP, ESP, and Uncompressed," RFC 3095, July 2001.

[7] Lazzaro, J., "Framing Real-Time Transport Protocol (RTP) and RTP Control Protocol (RTCP) Packets over Connection-Oriented Transport," RFC 4571, July 2006.

[8] Schulzrinne, H., and S. Casner, "RTP Profile for Audio and Video Conferences with Minimal Control," STD 65, RFC 3551, July 2003.

[9] Sjoberg, J., et al., "Real-Time Transport Protocol (RTP) Payload Format and File Storage Format for the Adaptive Multi-Rate (AMR) and Adaptive Multi-Rate Wideband (AMR-WB) Audio Codecs," RFC 3267, June 2002.

[10] Andersen, S., et al., "Internet Low Bit Rate Codec (iLBC)," RFC 3951, December 2004.

[11] Herlein, G., et al., "RTP Payload Format for the Speex Codec," draft-ietf-avt-rtp-speex-05 (work in progress), February 2008.

[12] YouTube, http://www.youtube.com.

[13] Levin, O., R. Even, and P. Hagendorf, "XML Schema for Media Control," RFC 5168, March 2008.

[14] Wenger, S., et al., "Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF)," RFC 5104, February 2008.

[15] ITU-T Recommendation T.140 (1998)—Text Conversation Protocol for Multimedia Application, with Amendment 1, (2000).

[16] Hellstrom, G., and P. Jones, "RTP Payload for Text Conversation," RFC 4103, June 2005.

[17] http://www.realtimetext.org/.

[18] Schulzrinne, H., and T. Taylor, "RTP Payload for DTMF Digits, Telephony Tones, and Telephony Signals," RFC 4733, December 2006.

[19] Schulzrinne, H., and S. Petrack, "RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals," RFC 2833, May 2000.

13

Negotiating Media Sessions

One of the most important uses of SIP is to negotiate the setup of sessions, as the name suggests. To do this, SIP uses another protocol, Session Description Protocol, to describe the actual parameters of the media session. This includes information such as media type, codec, bit rate, and the IP address and port numbers for the media session. In short, negotiating media sessions is all about exchanging the data necessary to begin the RTP media sessions described in Chapter 12 or SRTP media sessions described next in Chapter 14. This chapter will introduce the Session Description Protocol (SDP) and the Offer/Answer protocol, which is the way SIP uses SDP to negotiate sessions.

13.1 Session Description Protocol (SDP)

The Session Description Protocol, originally defined by RFC 2327 [1], was developed by the IETF MMUSIC working group. It is more of a description syntax than a protocol in that it does not provide a full-range media negotiation capability. The original purpose of SDP was to describe multicast sessions set up over the Internet's multicast backbone, the MBONE. The first application of SDP was by the experimental Session Announcement Protocol (SAP) [2] used to post and retrieve announcements of MBONE sessions. SAP messages carried an SDP message body, and were the template for SIP's use of SDP. Even though it was designed for multicast, SDP has been applied to the more general problem of describing general multimedia sessions established using SIP. SDP is current specified by RFC 4566 [3], which is mostly compatible with RFC 2327.

As seen in the examples of Chapter 2, SDP contains the following information about the media session:

- IP address (IPv4 or IPv6 address or host name);

- RTP profile (usually RTP/AVP although there are others such as RTP/SAVP);
- Port number (used by UDP or TCP for transport);
- Media type (audio, video, interactive whiteboard, and so forth);
- Media encoding scheme (PCM A-Law, MPEG II video, and so forth).

In addition, SDP contains information about the following:

- Subject of the session;
- Start and stop times;
- Contact information about the session.

Like SIP, SDP uses text coding. An SDP message is composed of a series of lines, called fields, whose names are abbreviated by a single lower-case letter, and are in a required order to simplify parsing. The set of SDP fields from RFC 4566 is shown in Table 13.1. The order in this table is the required order in SDP. Optional fields can be skipped, but must be in the correct order if present.

SDP was not designed to be easily extensible, and parsing rules are strict. The only way to extend or add new capabilities to SDP is to define a new attribute type. However, unknown attribute types can be silently ignored. An SDP

Table 13.1
SDP Fields

Field	Name	Mandatory/ Optional
v=	Protocol version number	m
o=	Owner/creator and session identifier	m
s=	Session name	m
i=	Session information	o
u=	Uniform Resource Identifier	o
e=	E-mail address	o
p=	Phone number	o
c=	Connection information	m
b=	Bandwidth information	o
t=	Timer session starts and stops	m
r=	Repeat times	o
z=	Time zone corrections	o
k=	Encryption key (deprecated)	o
a=	Attribute lines	o
m=	Media information	o
a=	Media attributes	o

parser must not ignore an unknown field, a missing mandatory field, or an out-of-sequence line. An example SDP message containing many of the optional fields is shown here:

```
v=0
o=johnston 2890844526 2890844526 IN IP4 43.32.1.5
s=IETF Update
i=This broadcast will cover the latest from the IETF
u=http://www.sipstation.com
e=Alan Johnston alan@avaya.com
p=+1-314-555-3333 (Daytime Only)
c=IN IP4 225.45.3.56/236
b=CT:144
t=2877631875 2879633673
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000
m=video 23422 RTP/AVP 31
a=rtpmap:31 H261/90000
```

The general form of a SDP message is:

```
x=parameter1 parameter2 ... parameterN
```

The line begins with a single lower-case letter, for example, x. There are never any spaces between the letter and the =, and there is exactly one space between each parameter. Each field has a defined number of parameters. Each line ends with a CRLF. The individual fields will now be discussed in detail.

13.1.1 Protocol Version

The v= field contains the SDP version number. Because the current version of SDP is 0, a valid SDP message will always begin with v=0.

13.1.2 Origin

The o= field contains information about the originator of the session and session identifiers. This field is used to uniquely identify the session. The field contains:

```
o=username session-id version network-type address-type address
```

The username parameter contains the originator's login or host or - if none. The session-id parameter is a Network Time Protocol (NTP) [4] timestamp or a random number used to ensure uniqueness. The version is a numeric field that is increased for each change to the session, also recommended to be a NTP timestamp. The network-type is always IN for Internet. The address-type parameter is either `IP4` or `IP6` for IPv4 or IPv6 address either in dotted decimal form or a fully qualified host name.

13.1.3 Session Name and Information

The `s=` field contains a name for the session. It can contain any nonzero number of characters. The optional `i=` field contains information about the session. It can contain any number of characters.

13.1.4 URI

The optional `u=` field contains a uniform resource indicator (URI) with more information about the session.

13.1.5 E-Mail Address and Phone Number

The optional `e=` field contains an e-mail address of the host of the session. If a display name is used, the e-mail address is enclosed in <>. The optional `p=` field contains a phone number. The phone number should be given in globalized format, beginning with a +, then the country code, a space or −, then the local number. Either spaces or - are permitted as spacers in SDP. A comment may be present inside brackets`( )`.

13.1.6 Connection Data

The `c=` field contains information about the media connection. The field contains:

```
c=network-type address-type connection-address
```

The network-type parameter is defined as `IN` for the Internet. The address type is defined as `IP4` for IPv4 addresses and `IP6` for IPv6 addresses. The connection-address is the IP address or host that will be sending the media packets, which could be either multicast or unicast. If multicast, the connection-address field contains:

```
connection-address=base-multicast-address/ttl/number-of-addresses
```

where `ttl` is the time-to-live value, and `number-of-addresses` indicates how many contiguous multicast addresses are included starting with the `base-multicast-address`.

13.1.7 Bandwidth

The optional b= field contains information about the bandwidth required. It is of the form:

```
b=modifier:bandwidth-value
```

The modifier is either CT for conference total or AS for application specific. CT is used for multicast session to specify the total bandwidth that can be used by all participants in the session. AS is used to specify the bandwidth of a single site. The bandwidth-value parameter is the specified number of kilobytes per second.

13.1.8 Time, Repeat Times, and Time Zones

The t= field contains the start time and stop time of the session.

```
t=start-time stop-time
```

The times are specified using NTP [4] timestamps. For a scheduled session, a stop-time of zero indicates that the session goes on indefinitely. A start-time and stop-time of zero for a scheduled session indicates that it is permanent. The optional r= field contains information about the repeat times that can be specified in either in NTP or in days (d), hours (h), or minutes (m). The optional z= field contains information about the time zone offsets. This field is used if a reoccurring session spans a change from daylight savings to standard time, or vice versa.

13.1.9 Encryption Keys

The optional k= field was used to carry encryption keys. However, its use is no longer recommended and was included in RFC 4566 for parser compatibility reasons. Instead, a=crypto or a=key-mgt should be used, whose use is described in Chapter 14.

13.1.10 Media Announcements

The optional m= field contains information about the type of media session. The field contains:

```
m=media port transport format-list
```

The media parameter is either audio, video, text, application, message, image, or control. The port parameter contains the port number. The transport parameter contains the transport protocol or the RTP profile used. The set of defined RTP profiles is in Table 15.3. The `format-list` contains more information about the media. Usually, it contains media payload types defined in RTP audio video profiles. More than one media payload type can be listed, allowing multiple alternative codecs for the media session. For example, the following media field lists three codecs:

```
m=audio 49430 RTP/AVP 0 6 8 99
```

One of these three codecs can be used for the audio media session. If the intention is to establish three audio channels, three separate media fields would be used. For non-RTP media, Internet media types should be listed in the `format-list`. For example,

```
m=application 52341 udp wb
```

could be used to specify the `application/wb` media type. Common SDP media types are listed in Table 13.2.

13.1.11 Attributes

The optional `a=` field contains attributes of the preceding media session. This field can be used to extend SDP to provide more information about the media. If not fully understood by a SDP user, the attribute field can be ignored. There can be one or more attribute fields for each media payload type listed in the media

Table 13.2
Common SDP Media Types

Example	Type	Specificiation
`m=audio 49122 RTP/AVP 0`	Audio media, also used for telephone-events (DTMF)	RFC 3551
`m=video 52134 RTP/SAVP 24`	Video media	RFC 3551
`m=text 11000 RTP/AVP 98`	Real-time text (T.140)	RFC 4103

`m=application 12454 wb udp`
`m=application 3422 TCP/TLS/BFCP *`
`m=application 554 TCP/RTSP rtsp`
	Application media, used for white board (wb), BFCP, RTSP, and others	RFC 4566

`m=message 12763 TCP/MSRP *`	Message media for MSRP	RFC 4975
`m=image 54111 TCP t38`	Fax (T.38) Note: Fax can also use a `m=audio` media type	RFC 3362 RFC 4612
`m=control`	Control media	RFC 2327

field. For the media line example in Section 13.1.9, the following three attribute fields could follow the media field:

```
a=rtpmap:0 PCMU/8000
a=rtpmap:6 DVI4/16000
a=rtpmap:8 PCMA/8000
a=rtpmap:99 iLBC
```

Other attributes are shown in Table 13.3. Full details of the use of these attributes are in the standard document [2]. The details of the iLBC (Internet low bit rate) codec are in [5].

Attributes can be either session level or media level in SDP. Session level means that the attribute is listed before the first media line in the SDP. If this is the case, the attribute applies to all the media lines below it. Media level means it is listed after a media line. In this case, the attribute only applies to this particular media stream. SDP can include both session level and media level attributes. If the same attribute appears as both, the media level attribute overrides the session level attribute for that particular media stream.

Note that the connection data field can also be session level or media level. There are three possibilities:

Table 13.3
SDP Attribute Values Defined in RFC 4566

Attribute	Name
`a=rtpmap:`	RTP/AVP list.
`a=fmtp:`	Format transport.
`a=ptime:`	Length of time in milliseconds for each packet.
`a=maxptime:`	Maximum ptime.
`a=cat:`	Category of the session.
`a=keywds:`	Keywords of session.
`a=tool:`	Name of tool used to create SDP.
`a=orient:`	Orientation for whiteboard sessions.
`a=type:`	Type of conference.
`a=charset:`	Character set used for subject and information fields.
`a=sdplang:`	Language for the session description.
`a=lang:`	Default language for the session.
`a=framerate:`	Maximum video frame rate in frames per second.
`a=quality:`	Suggests quality of encoding.
`a=direction:`	Direction for symmetric media.
`a=inactive`	Inactive mode.
`a=recvonly`	Receive only mode.
`a=sendrecv`	Send and receive mode.
`a=sendonly`	Send only mode.

- A single c= field at the session level. This is the mos t common case.
- A session level c= field and some media level c= fields.
- Each media level field with no session level stream.

The same rules for attributes apply when both session and media level c= fields are present; the media field overrides the session level for that particular media stream.

13.2 SDP Extensions

There are a number of SDP extensions that have been defined. Common ones are summarized in Table 13.4.

The RTCP IP address and port attribute, a=rtcp [6] is covered in Chapter 10. The a=setup and a=connection attributes are used for connection oriented media, such as TCP. Section 8.5.2 shows the use of these attributes in establishing MSRP sessions. Another example is shown below of Binary Floor Control

Table 13.4
Common SDP Extensions

Attribute	Name	Reference
a=rtcp	Port and IP address for RTCP [6]	RFC 3605

a=mid
a=group
	Media session identifier and grouping of media streams [7]	RFC 3388

a=setup
a=connection
	Connection-oriented media using as TCP transport [8]	RFC 4145

a=key-mgt	Key management for MIKEY [9]	RFC 4567
a=crypto	Key management for SRTP [10]	RFC 4568

a=floorctrl
a=confid
a=userid
a=floorid
	Binary Floor Control Protocol (BFCP) information [11]	RFC 4583

a=fingerprint	Connection-oriented media using TLS [12]	RFC 4572
a=label	Media label [13]	RFC 4574

a=accept-types
a=accept-wrapped-types
a=max-size
a=path
	Message Session Relay Protocol (MSRP) information [14]	RFC 4975

a=ice-pwd
a=ice-ufrag
a=ice-lite
a=ice-mismatch
a=ice-options
	Interactive connectivity establishment (ICE) [15]	[15]

a=chatroom	Chat room name for MSRP	[16]

Protocol (BFCP) [11] session establishment, which shows the use of many of these SDP attributes. The first `m=` media line is for a BFCP stream running over TLS over TCP. The `a=connection:new` indicates that a new TCP connection needs to be opened and that this endpoint will do a passive open (the other endpoint will do the active open). The `a=fingerprint` contains a fingerprint of the certificate to be exchanged during the TLS handshake, as described in Section 14.6 the `a=confid` and `a=userid` attributes contain the conference ID and user ID of the user. The `a=floorid` attributes indicate that floor 1 is associated with `a=label:1`, which is associated with the `m=audio` stream while floor 2 is associated with `a=label:2,` which is associated with the `m=video` stream.

```
v=0
o=bob 2808844564 2808844564 IN IP4 130.43.2.1
s=
t=0 0
c=
m=application 54052 TCP/TLS/BFCP *
a=setup:passive
a=connection:new
a=fingerprint:SHA-1 AD:9B:B1:3F:72:18:00:3B:54:02:12:DF:3E:5F:49
:1B:19:E5:DC:AB
a=floorctrl:s-only
a=confid:38921838776
a=userid:bob
a=floorid:1 m-stream:1
a=floorid:2 m-stream:2
m=audio 54026 RTP/AVP 0
a=label:1
m=video 54042 RTP/AVP 31
a=label:2
```

13.3 The Offer Answer Model

The use of SDP with SIP is given in the SDP offer answer RFC 3264 [17]. The default message body type in SIP is `application/sdp`. The calling party lists the media capabilities that they are willing to receive in SDP, usually in either an `INVITE` or in an `ACK`. The called party usually lists their media capabilities in the `200 OK` response to the `INVITE`. More generally, offers or answers may be in `INVITE`s, `PRACK`s, or `UPDATE`s or in reliably sent `18x` or `200` responses to these methods.

Because SDP was developed with scheduled multicast sessions in mind, many of the fields have little or no meaning in the context of dynamic sessions established using SIP. In order to maintain compatibility with the SDP protocol, however, all required fields are included. A typical SIP use of SDP includes the version, origin, subject, time, connection, and one or more media and attribute fields is shown in Table 13.1. The subject and time fields are not used by SIP but are included for compatibility. In the SDP standard, the subject field is a required field and must contain at least one character, suggested to be s=- if there is no subject. The time field is usually set to t=0 0.

SIP uses the connection, media, and attribute fields to set up sessions between UAs. The origin field has limited use with SIP. Usually, the session-id is kept constant throughout a SIP session and the version is incremented each time the SDP is changed. If the SDP being sent is unchanged from that sent previously, the version is kept the same.

Because the type of media session and codec to be used are part of the connection negotiation, SIP can use SDP to specify multiple alternative media types and to selectively accept or decline those media types. The offer answer specification, RFC 3264 [17], recommends that an attribute containing a=rtpmap: be used for each media field. A media stream is declined by setting the port number to zero for the corresponding media field in the SDP response. In the following example, the caller Tesla wants to set up an audio and video call with two possible audio codecs and a video codec in the SDP carried in the initial INVITE:

```
v=0
o=Tesla 2890844526 2890844526 IN IP4 lab.high-voltage.org
s=-
c=IN IP4 100.101.102.103
t=0 0
m=audio 49170 RTP/AVP 0 8
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
m=video 49172 RTP/AVP 32
a=rtpmap:32 MPV/90000
```

The codecs are referenced by the RTP/AVP profile numbers 0, 8, and 32. The called party Marconi answers the call, chooses the second codec for the first media field, and declines the second media field, only wanting a PCM A-Law audio session.

```
v=0
o=Marconi 2890844526 2890844526 IN IP4 tower.radio.org
s=-
c=IN IP4 200.201.202.203
t=0 0
m=audio 60000 RTP/AVP 8
a=rtpmap:8 PCMA/8000
m=video 0 RTP/AVP 32
```

If this audio-only call is not acceptable, then Tesla would send an ACK then a BYE to cancel the call. Otherwise, the audio session would be established and RTP packets exchanged. As this example illustrates, unless the number and order of media fields is maintained, the calling party would not know for certain which media sessions were being accepted and declined by the called party.

The offer/answer rules are summarized in the following sections.

13.3.1 Rules for Generating an Offer

An SDP offer must include all required SDP fields (this includes `v=`, `o=`, `s=`, `c=`, and `t=`). It usually includes a media field (`m=`) but it does not have to. The media lines contain all codecs listed in preference order. The only exception to this is if the endpoint supports a huge number of codecs, the most likely to be accepted or most preferred should be listed. Different media types include audio, video, text, MSRP, BFCP, and so forth.

13.3.2 Rules for Generating an Answer

An SDP answer to an offer must be constructed according to these rules. The answer must have the same number of `m=` lines in the same order as the answer. Individual media streams can be declined by setting the port number to 0. Streams are accepted by sending a nonzero port number. The listed payloads for each media type must be a subset of the payloads listed in the offer. Note that for dynamic payloads, the same dynamic payload number does not need to be used in each direction. Usually, only a single payload is selected. More than one may be selected, but endpoints doing this must be capable of dynamically switching between them without signaling. Since many simple endpoints can only have one codec running at a time, this should be avoided. One common exception is to accept a media codec and also telephone-events (Section 12.7). This allows the codec to be used except when a DTMF key is pressed when a telephone-events payload is used.

13.3.3 Rules for Modifying a Session

Either party can initiate another offer/answer exchange to modify the session. When a session is modified, the following rules must be followed. The origin (`o=`) line version number must either be the same as the last one sent, which indicates that this SDP is identical to the previous exchange, or it may be incremented by one, which indicates new SDP that must be parsed. The offer must include all existing media lines and they must be sent in the same order. Additional media streams can be added to the end of the `m=` line list. An existing media stream can be deleted by setting the port number to 0. This media line must remain in the SDP in this and all future offer/answer exchanges for this session. For an existing media stream, any aspect can be changed.

13.3.4 Special Case—Call Hold

One party in a call can temporarily place the other on hold (i.e., suspending the media packet sending). This is done by sending an `INVITE` with an identical SDP to that of the original `INVITE` but with `a=sendonly` attribute present. The call is made active again by sending another INVITE with the a=sendrecv attribute present. (Note that older RFC 2543 compliant UAs may initiate hold using c=0.0.0.0.) For further examples of SDP use with SIP, see the SDP offer answer examples document [18].

13.4 Static and Dynamic Payloads

The payload type (PT) is used to identify the media codec in the media line of SDP as described in Section 13.1.10. This same payload type is also carried in individual RTP media packets sent during the media session. RFC 3551 defines some *static* payload types. These payloads are considered static because a given payload number defined in the specification always refers to that particular codec. For example, PT 0 for audio always means G.711 PCM codec. The use of a=rtpmap attribute for static payloads is optional, although it is considered good practice to include it. However, static payloads are no longer allocated by the IETF. Instead, all new codecs must make use of *dynamic* payload types. Dynamic payload types are in the range of 96–127. Payloads in this range do not refer to a particular codec; instead the required a=rtpmap attribute must be used to indicate the payload. There are a number of rules associated with the use of dynamic payloads in the SDP offer answer exchange. They are:

- Dynamic payloads must be negotiated with SDP.
- The a=rtpmap attribute is mandatory.
- Dynamic payload numbers cannot be redefined within a session.
- Dynamic payload numbers do not need to be the same in both directions of a bidirectional session.

The last rule means that it is possible that payload 97 means one codec in one direction but another codec in a different direction.

13.5 SIP Offer Answer Exchanges

The main offer answer exchanges with SIP are in the INVITE/200 OK exchange or in the 200 OK/ACK exchange, if the INVITE did not contain an offer. There are other offer/answer modes, summarized in Table 13.5, which is taken from [19]. Support of the specification listed implies that the user agent supports this additional offer/answer exchange mode.

Table 13.5
SIP Offer/Answer Exchange Modes

Offer	Answer	Specification
INVITE	2xx to INVITE	RFC 3261
2xx to INVITE	ACK	RFC 3261
INVITE	Reliable 1xx to INVITE	RFC 3262
Reliable 1xx to INVITE	PRACK	RFC 3262
PRACK	200 to PRACK	RFC 3262
UPDATE	2xx to UPDATE	RFC 3311

Source: [19].

13.6 Conclusion

This chapter has covered the use of SDP in the Offer/Answer Protocol to negotiate the establishment and modification of media sessions. Core SDP and the Offer/Answer Protocol allow basic media sessions to be established. Some SDP extensions are required for more advanced media setup and control.

13.7 Questions

Q13.1 Create an SDP offer for Bob offering audio and video with the following audio codecs: iLBC, GSM and video codecs: MPV, and H.261. Bob wants to receive audio media on port 60322, video on port 60324, and RTCP on port 60326. Bob would prefer a packetization time of 30 ms for audio.

Q13.2 Create an SDP answer for Alice to Bob's offer from the previous question, accepting video but declining audio. You can choose whichever ports and codecs you like.

Q13.3 Find the three syntax errors in this SDP example.

```
v=0
o=alice 289084526 28904529 IP4 231.3.43.1
s=-
c=IN IP4 231.3.43.1
m=audio 49170 RTP/AVP 0 97 98
a=rtpmap:97 iLBC/8000
```

Q13.4 Create an SDP offer by Alice that could have resulted in the following SDP answer.

```
v=0
o=bob 2808844564 2808844564 IN IP4 130.43.2.1
s=-
t=0 0
```

```
m=audio 49174 RTP/AVP 0
c=IN IP4 130.43.2.1
a=rtpmap:0 PCMU/8000
a=recvonly
m=text 49176 RTP/AVP 96
c=IN IP4 130.43.2.2
a=rtpmap:96 t140/1000
```

Q13.5 Indicate the IP address and port number associated with each of the three media streams.

```
v=0
o=Tesla 2890844526 2890844526 IN IP4 lab.high-voltage.org
s=-
c=IN IP4 100.101.102.103
t=0 0
m=audio 49170 RTP/AVP 0 8
c=IN IP4 101.102.103.106
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
m=video 49172 RTP/AVP 34
a=rtpmap:34 H263/90000
m=video 53132 RTP/AVP 26
c=IN IP4 100.102.103.4
a=rtpmap:26 JPEG/90000
```

Q13.6 Describe in words the offer and answer in the SDP below.

Offer:

```
v=0
o=alice 2890844526 2890844526 IN IP4 host.atlanta.example.com
s=
c=IN IP4 host.atlanta.example.com
t=0 0
m=audio 49170 RTP/AVP 0 8 97
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:97 iLBC/8000
m=video 51372 RTP/AVP 31 32
a=rtpmap:31 H261/90000
a=rtpmap:32 MPV/90000
```

Answer:

```
v=0
o=bob 2808844564 2808844564 IN IP4 host.biloxi.example.com
s=
c=IN IP4 host.biloxi.example.com
t=0 0
m=audio 49174 RTP/AVP 0
a=rtpmap:0 PCMU/8000
m=video 49172 RTP/AVP 32
c=IN IP4 otherhost.biloxi.example.com
a=rtpmap:32 MPV/90000
```

Q13.7 Find two errors in the offer/answer exchange:

Offer:

```
v=0
o=Tesla 2890844526 2890844526 IN IP4 lab.high-voltage.org
s=-
c=IN IP4 100.101.102.103
t=0 0
m=audio 49170 RTP/AVP 0 8
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
m=video 49172 RTP/AVP 32
a=rtpmap:32 MPV/90000
```

Answer:

```
v=0
o=Marconi 2890844526 2890844526 IN IP4 tower.radio.org
s=-
c=IN IP4 200.201.202.203
t=0 0
m=audio 60000 RTP/AVP 98
a=rtpmap:98 iLBC/8000
```

Q13.8 Is it permissible to define payload 98 as iLBC codec in one direction and payload 97 as iLBC in the other direction?

Q13.9 A user agent supports RFC 3261 and RFC 3262, but does not support RFC 3311. Which offer/answer modes does this user agent support? Which is likely to be the most commonly used?

Q13.10 For the offer/answer exchange below, generate a new offer answer exchange between Marconi and Tesla where Marconi puts the audio stream on hold.

Offer:

```
v=0
o=Tesla 2890844526 2890844526 IN IP4 lab.high-voltage.org
s=-
c=IN IP4 100.101.102.103
t=0 0
m=audio 49170 RTP/AVP 0 8
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
m=video 49172 RTP/AVP 32
a=rtpmap:32 MPV/90000
```

Answer:

```
v=0
o=Marconi 2890844526 2890844526 IN IP4 tower.radio.org
s=-
c=IN IP4 200.201.202.203
t=0 0
m=audio 60000 RTP/AVP 8
a=rtpmap:8 PCMA/8000
m=video 0 RTP/AVP 32
```

References

[1] Handley, M., and V. Jacobson, "SDP: Session Description Protocol," RFC 2327, April 1998.

[2] Handley, M., C. Perkins, and E. Whelan, "Session Announcement Protocol," RFC 2974, October 2000.

[3] Handley, M., V. Jacobson, and C. Perkins, "SDP: Session Description Protocol," RFC 4566, July 2006.

[4] Mills, D., "Network Time Protocol (Version 3): Specification, Implementation, and Analysis," RFC 1305, March 1992.

[5] Duric, A., and S. Andersen, "Real-Time Transport Protocol (RTP) Payload Format for Internet Low Bit Rate Codec (iLBC) Speech," RFC 3952, December 2004.

[6] Huitema, C., "Real Time Control Protocol (RTCP) Attribute in Session Description Protocol (SDP)," RFC 3605, October 2003.

[7] Camarillo, G., et al., "Grouping of Media Lines in the Session Description Protocol (SDP)," RFC 3388, December 2002.

[8] Yon, D., and G. Camarillo, "TCP-Based Media Transport in the Session Description Protocol (SDP)," RFC 4145, September 2005.

[9] Arkko, J., et al., "Key Management Extensions for Session Description Protocol (SDP) and Real Time Streaming Protocol (RTSP)," RFC 4567, July 2006.

[10] Andreasen, F., M. Baugher, and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams," RFC 4568, July 2006.

[11] Camarillo, G., "Session Description Protocol (SDP) Format for Binary Floor Control Protocol (BFCP) Streams," RFC 4583, November 2006.

[12] Lennox, J., "Connection-Oriented Media Transport over the Transport Layer Security (TLS) Protocol in the Session Description Protocol (SDP)," RFC 4572, July 2006.

[13] Levin, O., and G. Camarillo, "The Session Description Protocol (SDP) Label Attribute," RFC 4574, August 2006.

[14] Campbell, B., R. Mahy, and C. Jennings, "The Message Session Relay Protocol (MSRP)," RFC 4975, September 2007.

[15] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols," draft-ietf-mmusic-ice-19 (work in progress), October 2007.

[16] Niemi, A., M. Garcia-Martin, and G. Sandbakken, "Multi-Party Chat Using the Message Session Relay Protocol (MSRP)," draft-ietf-simple-chat-03 (work in progress), October 2008.

[17] Rosenberg, J., and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)," RFC 3264, June 2002.

[18] Johnston, A., and R. Sparks, "Session Description Protocol (SDP) Offer/Answer Examples," RFC 4317, December 2005.

[19] Sawada, T., and P. Kyzivat, "SIP (Session Initiation Protocol) Usage of the Offer/Answer Model," Internet-Draft, draft-ietf-sipping-sip-offeranswer-10 (work in progress), January 2009.

14

SIP Security

SIP security is a large and complicated topic, and there are entire books on the topic. This chapter will introduce the basics of security and apply them to the SIP protocol and common applications such as establishing secure multimedia sessions. A number of potential attacks on SIP and SIP-related protocols will also be discussed. While the discussion in this chapter will focus on protocol security, it is important to realize that security is an all-encompassing area that needs attention at all levels. For example, besides protocol security, there is physical security, general server security, operating system security, local area network security, password security, and so on. For a detailed analysis and description of SIP security, see [1].

14.1 Basic Security Concepts

Three important security concepts will be briefly introduced here: authentication, confidentiality, and integrity protection. Authentication is the proof of identity of a party in communication. Authentication is usually performed in a protocol by use of a credential: a shared secret that is known by both parties but not known to others. After authentication is performed, authorization policy can be performed. For example, consider a user attempting to subscribe to the presence of another user. Once authentication has been performed and the user knows who it is that is requesting the presence subscription, the subscription can be authorized or denied, and the level of detail of presence information to be shared can be determined and implemented.

Confidentiality is about keeping a communication exchange private. This privacy usually applies to the content of the information exchanged; that is, a third party should not be able to inspect messages, read text, view presence information, or listen in to exchanged media. However, a third party may be able to determine that two users are exchanging IP packets, and hence make some deductions about users exchanging information. Approaches that attempt to conceal and obfuscate the fact that two users are even communicating over IP are much more difficult and will not be discussed in this book.

Integrity protection is the ability to determine that a message in a communication has not been modified or tampered with between the creator and the viewer. Note that integrity is only useful when combined with authentication Knowing that a packet has not been modified since it was sent by an attacker is not a very useful property!

14.1.1 Encryption

Cryptography is the science of encryption. There are excellent historical texts [2] and technical overviews [3]. Here, only a very high-level view will be provided.

Encryption is a common tool used to provide confidentiality over the Internet. Encryption provides confidentiality by turning plain text into ciphertext using a key and a mathematical algorithm, known as a cipher. Typically, while the encryption algorithm may be known, the key is kept secret. The plaintext is the input to the encryption algorithm, which could be a protocol message, text message, or media samples. The ciphertext is the output, and appears to essentially be random numbers, white noise, or gibberish, which provide no information to a third party monitoring the ciphertext. Without knowing the key, it is very difficult to turn the ciphertext back into plaintext. The key length is chosen to be large so that it is difficult for an attacker to simply try every possible key to decrypt the message. This is known as a *brute force* attack. For example, an early encryption algorithm DES, which used a 56 bit key, was considered secure when it was first used on the Internet. Today, however, a brute force attack on this is feasible, as all possible keys can be tried in a relatively short period of time. Most encryption algorithms today use much longer keys in the range of 128 bits to 4,096 bits, making this type of attack currently infeasible. As advances in computing continue, it is likely that longer key lengths will be needed to provide adequate levels of security.

There are two common encryption algorithms used today on the Internet, and they relate to the way in which keys are used to encrypt and decrypt the data. Symmetric key encryption uses the same key to encrypt and decrypt. This is shown in Figure 14.1. Note that the encryption and decryption algorithms are different, but related mathematically. Some common examples of symmetric ciphers include Advanced Encryption Standard (AES) [4], Triple Data Encryption Standard (3DES) [5], Rivest Cipher 4 (RC4) [6], and so fourth. Symmetric key encryption requires a key management protocol to securely distribute the secret key among all parties in a communication. If this key is not kept secret, no confidentiality will be provided. This type of encryption is commonly used

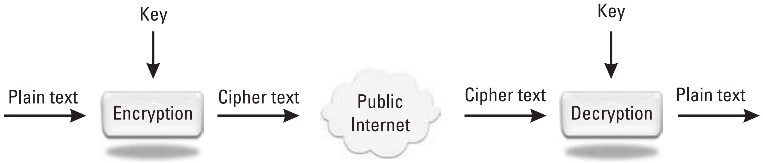


Figure 14.1 Symmetric key cryptography.

today for media encryption such as in the Secure Real-Time Transport Protocol (SRTP), discussed in Section 14.7.2.

14.1.2 Public Key Cryptography

Public key cryptography, also known as asymmetric key cryptography, uses one key to encrypt and a different key to decrypt. This approach uses a pair of keys, a public key, which is freely available and known to others, and a private key, which is kept secret. The private and public keys have a mathematical relationship between them, but it is computationally infeasible to derive the private key from the public key. Consider two users Alice and Bob. Alice wants to send a message to Bob that is encrypted so only Bob can decrypt it. Alice creates the message, then encrypts it using Bob's public key. Bob's public key could be determined by Alice from a directory listing or database, or having it from a previous exchange with Bob. The resulting message can only be decrypted using Bob's private key. Since this key is only known to Bob, only Bob can decrypt the message. This is shown in Figure 14.2. An example of public key encryption is Rivest Shamir Adellmann algorithm (RSA) [7]. This type of encryption is commonly used on the Internet today with SIP as part of transmission layer security (TLS), discussed in Section 14.3.2.

14.1.3 Diffie-Hellman Cryptography

Diffie-Hellman [8] is another type of public key cryptography. It is a very clever scheme that allows two parties to generate the same secret key independently. This is done by exchanging a few parameters between the parties, but this secret key is never transmitted from one party to the other. A third party able to view all the messages exchanged is still not able to generate the same secret key. Diffie-Hellman is defined for Internet protocols in [9] and is used in key agreement protocols such as ZRTP [10], discussed in Section 14.7.5.

14.1.4 Message Authentication

A common first step to provide authentication of messages is to use a message digest function. A message digest is a one-way mathematical function that pro-

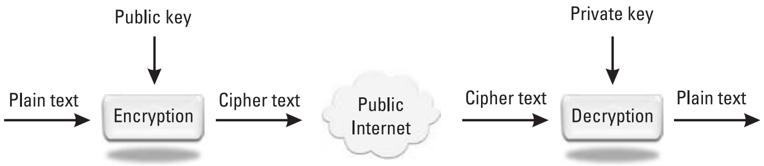


Figure 14.2 Public key cryptography.

duces a fixed length output from a variable length input string. MD5 (message digest 5) and SHA-1 (secure hash 1) are examples of message digest algorithms. There are two important properties of a message digest. The first is that it be very difficult to reverse (i.e., determine the input string based knowing only the message digest output). The other is for it to be difficult to generate two input strings that have the same output message digest. This condition is known as a *hash collision* and is a method used to try to attack message digest algorithms.

A keyed hashed message authentication code (HMAC) is a message digest function followed by encryption. For example, HMAC-SHA-1 uses the SHA-1 message digest followed by encryption, as shown in Figure 14.3. HMACs can provide message authentication—if both the sender and receiver know the secret key, the sender can calculate the HMAC of a message and send it with the message. The receiver can then do its own calculation of the HMAC using the secret key. If the two HMACs match, the message has not been changed or modified since the sender calculated the HMAC. If the key used is the private key of a public key pair, the HMAC can be used as a digital signature. A receiver can verify that only the holder of the private key could have generated and signed this message.

14.1.5 Digital Certificates

A digital certificate is a data object that makes assertions about identity. A common data format is known as X.509 [11]. Certificates utilize a chain of trust in which a certificate authority issues and signs a certificate and the user employs the certificate to prove that a particular public key is associated with an identity. By proving they know the private key associated with the public key of the certificate, the user can prove that they have the identity indicated in the certificate.

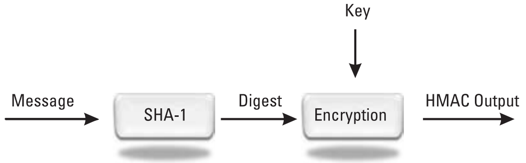


Figure 14.3 HMAC.

There are three important issues to be considered when a certificate is used for authentication:

1. Is the certificate trustworthy? For example, Web browsers have a list of CAs they inherently trust, and administrators can add additional trusted CAs. A browser will only trust a certificate issued and signed by a CA that is trusted. Otherwise, anyone could make up a certificate and use it for authentication.
2. Is the certificate valid? The certificate signature can be validated using the CA public key, which needs to be stored locally in the browser. Also, the certificate can be checked for revocation or replacement by consulting a certificate revocation list (CRL) or another certificate validation protocol. A certificate also must be checked to ensure that it has not expired.
3. What identity assertion does the certificate make? In its simplest form, the certificate will assert an identity, such as a DNS name, or a business name or address. A certificate issued by a trustworthy CA that is valid can only be used for authentication if the identity assertion matches the context in which the certificate is used. For example, if a Web browser opens a secure Web (https) connection for `example.com`, and during the TLS handshake receives a certificate asserting the identity of the Web server as `example.net`, authentication has not succeeded.

While this section has discussed certificates in terms of their common usage in Web browsing, the SIP usage of certificates is similar to the Web usage, and is discussed in Section 14.4.2.

14.2 Threats

A security analysis of a protocol or application is usually done by first compiling a list of threats, then examining how security mechanisms can be used to defend against them. This section will briefly discuss a few of them for SIP. With SIP and Internet communications, two of the biggest threats are denial of service (DOS) and man in the middle (MitM).

Denial of service is an attack by a third party who tries to interrupt service of a protocol or a service. For example, a simple denial of service attack is a packet flood, where an attacker sends a flood of IP packets, which overloads the target. This type of DOS attack is not specific to SIP or Internet communication. When launched from multiple hosts, the attack is called a distributed denial of service attack (DDOS).

A *man in the middle* (MitM) attack is when an attacker is in the middle of communications between two parties and is able to modify any message, delete any message, and inject any message into the communication. A MitM attack is difficult to launch, although it is possible if an attacker has control of a router or wireless hub that packets are routing through. A MitM attacker can introduce false messages, change parts of messages, and introduce new messages into the session. A MitM attacker can usually prevent communication from taking place, and can hijack or misdirect communication.

Theft of service is when an attacker uses resources that would normally be chargeable or have limitations. For example, calls to a PSTN gateway usually involve changes, so an attacker presenting false authentication credentials to place calls to the PSTN is a theft of service attack.

Eavesdropping is an attack that involves monitoring or listening in to a session. This could be the signaling, indicating who is communicating with whom, or it could be the actual conversation itself. As such, this can be a signaling (SIP) or media (RTP) attack. *Impersonation* is pretending to be someone else in a communication. An attacker could pretend to be either a SIP user or a server, depending on the attack. *DNS poisoning* is when an attacker modifies or gives false DNS records for the purpose of disrupting communications. *Credential theft* is when an attacker steals the credentials of a user. These credentials could then be used in an impersonation attack or theft of service attack, for example. *Redirection* or *hijacking* is when an attacker redirects a call or session. *Session disruption* is when an attacker attempts to disrupt a session by injecting false or misleading packets or messages. A *replay attack* is when an attacker stores and then resends valid packets or messages in a session in an attempt to disrupt a session.

Most of these threats apply regardless of whether SIP is used to establish a session or to exchange presence or instant messaging information. The next section will introduce some security protocols that SIP can use to protect against these threats.

14.3 Security Protocols

This section will cover common security protocols including IPSec, TLS, DNSSec, and S/MIME.

14.3.1 IPSec

IPSec or IP security [12] is a protocol that operates at the IP layer of the protocol stack. As a result, it works with any transport protocol above it in the protocol stack, such as TCP and UDP, and protocols such as SIP and RTP run over it without any changes. In general, an IPSec session needs to be established between hosts on the Internet. Sometimes, this session is called a virtual private network or VPN. IPSec can provide just authentication and integrity protection, or confidentiality and authentication. Key management can be a difficult issue with IPSec since it needs symmetric keys in both hosts. IPSec is commonly used between hosts or between gateways where there is significant traffic exchanged. For example, it is commonly used between an enterprise and a service provider, or between enterprise locations. The fact that a single IPSec VPN tunnel can protect both SIP signaling and RTP media for multiple sessions and users is a distinct advantage. However, for general SIP communications, which might go to multiple hosts, establishing multiple VPNs to all hosts prior to sending SIP or RTP is difficult. Also, since IPSec is commonly done in the OS or kernel layer, SIP and other applications often are unaware if they are riding on top of IPSec or if IPSec has been disabled or failed to be setup. As a result, there are advantages for security at layers above the IP layer, as in the next protocol.

14.3.2 TLS

Transmission layer security (TLS) [13] is a security protocol that operates at a shim layer between the application layer and the TCP transport layer. The current version of TLS is 1.1, although versions 1.0 [14] and earlier versions, known as secure sockets layer (SSL), are also sometimes used. TLS provides confidentiality, authentication, and integrity protection. Because it operates above the transport layer, applications, such as SIP, are aware of whether a connection has been successfully secured with TLS or not. TLS is widely used on the Internet today for secure Web browsing. TLS is actually two protocols, one being a handshake protocol used to setup connections, perform authentication, and generate a shared secret. The other protocol is the transport protocol, which uses the shared secret for bulk data encryption. The handshake protocol usually uses digital certificates for authentication while the transport protocol uses a symmetric cipher such as AES or 3DES to encrypt the data. TLS is often used to secure SIP on a hop by hop basis, as will be described in Section 14.4.2. Datagram TLS or DTLS [15] is a version of TLS that uses UDP transport and may be used to secure SIP in the future.

14.3.3 DNSSec

DNS security (DNSSec) [16] is a security protocol, which provides authentication and integrity to DNS queries. DNSSec adds four new resource records listed in Table 14.1. Today, nearly all users of DNS employ it without any security, and must trust DNS servers to give correct answers to queries. In the future when resolvers and servers support DNSSec, a user of DNS will be able to validate responses received. However, DNSSec only works if the entire DNS hierarchy is signed, which represents a significant deployment hurdle. So far, DNSSec is only used in limited environments.

Table 14.1
DNSSec Resource Records

RRSIG	Resource record signature
DNSKEY	DNS public key
DS	Delegation signer
NSEC	Next secure record

14.3.4 Secure MIME

Secure MIME or S/MIME (secure multipart Internet mail exchange) is a security protocol developed to secure e-mail. It provides authentication, integrity, and confidentiality services on an end-to-end basis. S/MIME usually uses digital certificates for keying, and the management of these certificates in endpoints makes deployment difficult.

14.4 SIP Security Model

This section will introduce the security model for SIP. Security begins with authentication, and there are a number ways a SIP message can be authenticated. One way is if it is received over an IPSec or VPN tunnel that has previously been authenticated. Another method is if it is received over a TLS connection that has been properly authenticated. The method by which TLS can provide SIP authentication will be discussed in Section 14.4.2. SIP messages can also include a digital signature, done using S/MIME. In addition, SIP can utilize HTTP digest authentication for a challenge/response authentication mechanism. This approach, discussed in Section 14.4.1, uses a simple shared secret for authentication.

14.4.1 SIP Digest Authentication

SIP digest authentication is based on HTTP digest [17]. A SIP server or UA can challenge a UA to resend a request proving knowledge of a shared secret. The shared secret is never sent in the SIP message, but instead a message digest 5 (MD5) hash is sent instead. This challenge can be done statelessly to prevent denial of service attacks. An example is shown in Figure 14.4. The initial `INVITE` receives a `401 Unauthorized` response, which contains a `WWW-Authenticate` header field. The UA sends an `ACK` to complete the SIP transaction, then resends the `INVITE` with an `Authorization` header field. Usually, the same `Call-ID` and `From` tag is used, but a different branch ID and incremented `CSeq`. The `Authorization` header field contains the credential. If authentication succeeds, the `INVITE` is processed or forwarded. Note that a 401 response is usually sent by a UAS such

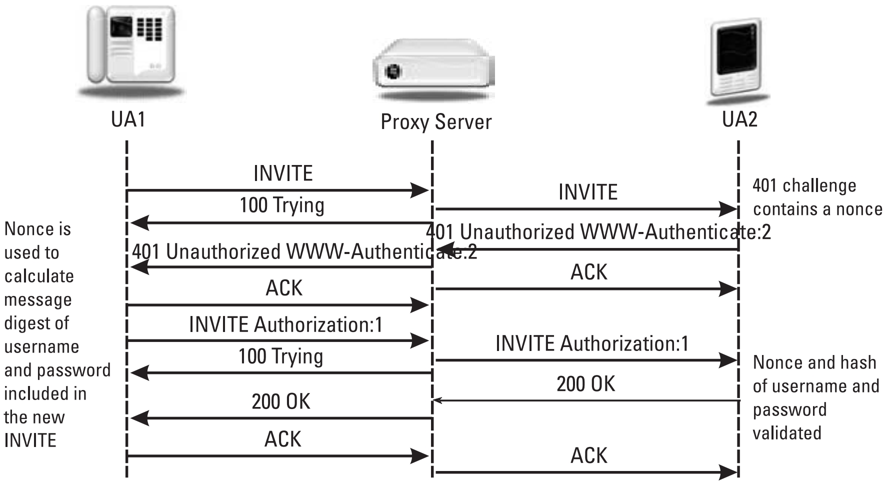

UA1
Proxy Server
UA2
INVITE
100 Trying
INVITE
401 challenge contains a nonce
Nonce is used to calculate message digest of username and password included in the new INVITE
401 Unauthorized WWW-Authenticate:2
401 Unauthorized WWW-Authenticate:2
ACK
ACK
INVITE Authorization:1
100 Trying
INVITE Authorization:1
Nonce and hash of username and password validated
200 OK
200 OK
ACK
ACK

Figure 14.4 SIP authentication using digest.

as another UA or a redirect server or registrar server. Figure 14.5 shows a proxy authentication challenge with a `407` response containing a `Proxy-Authenticate` header field. The resent `INVITE` contains a `Proxy-Authorization` header field.

Although HTTP digest has a mode that provides integrity protection, few SIP UAs and proxies implement this. As a result, requests authenticated using digest are susceptible to hijacking and redirection. For example, a `REGISTER` that has passed a digest authentication challenge could have the `Contact` URI modified by an attacker, resulting in the hijacking of all incoming `INVITE`s. Alternatively, a digest authenticated `INVITE` could have the SDP IP addresses changed so that media is sent to another party.

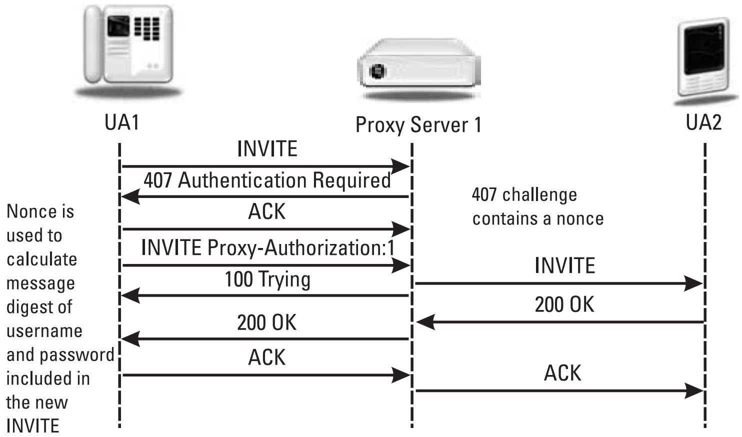

UA1
Proxy Server 1
UA2
INVITE
407 Authentication Required
407 challenge contains a nonce
ACK
Nonce is used to calculate message digest of username and password included in the new INVITE
INVITE Proxy-Authorization:1
INVITE
100 Trying
200 OK
200 OK
ACK
ACK

Figure 14.5 Proxy authentication using digest.

For mutual authentication, the `Authentication-Info` header field can be used, although this header field is rarely supported in UAs. Also, this approach relies on a shared secret in the server. Since a server can support thousands of clients, this approach does not scale well.

Also note that digest can only be used to authenticate requests. Responses cannot be challenged so they cannot be authenticated using digest. Also, `ACK` and `CANCEL` messages cannot be authenticated using digest since they cannot be challenged.

It is perhaps surprising in light of these shortcomings that often SIP digest is the only authentication used in some SIP deployments and environments.

14.4.2 SIP Authentication Using TLS

SIP can use TLS for authentication, confidentially, and integrity protection in a similar way that HTTP uses TLS for secure Web browsing. The registered SIP port for TLS is 5061, although using NAPTR and SRV records, other port numbers can be used. A client opens a TLS connection on this port and can request the server produce a certificate. If the certificate is signed by a certificate authority recognized by the client, the certificate can be used to authenticate the public key of the server. By producing a signature using the associated private key during the TLS handshake, the server can be authenticated. It is also possible for the server to request a certificate from the client. However, unless the client is another proxy server, it is not likely a SIP UA will have a certificate. As a result, TLS typically only provides one-way authentication. A digest challenge by the server can be done over the TLS connection, which provides a level of mutual authentication.

A SIP message can be authenticated if it is received over an authenticated TLS connection. In particular, if the domain of the `From` URI matches the `subjectAltName` (SAN) in the certificate presented during the TLS handshake, the request can be considered authenticated. Figure 14.6 shows how this can be done. TLS also provides integrity protection on a hop-by-hop basis. When a TLS connection is used for routing a SIP response, the response can also be authenticated over this single hop.

However, TLS cannot provide authentication over more than one hop. Consider the examples of Figure 14.4 and 14.5 where there are two SIP hops between the UAs. Each UA can only examine the certificate of the proxy it connects to. On receipt of a request, a UA can verify by examining the `Via` header fields that TLS has been used on every hop, but it must trust these `Via` header fields are accurate, and that each hop proxy has properly authenticated the connection and the certificate.

There are two types of connection reuse with SIP. One is sending a response over the same connection as the request. The other is reusing an existing

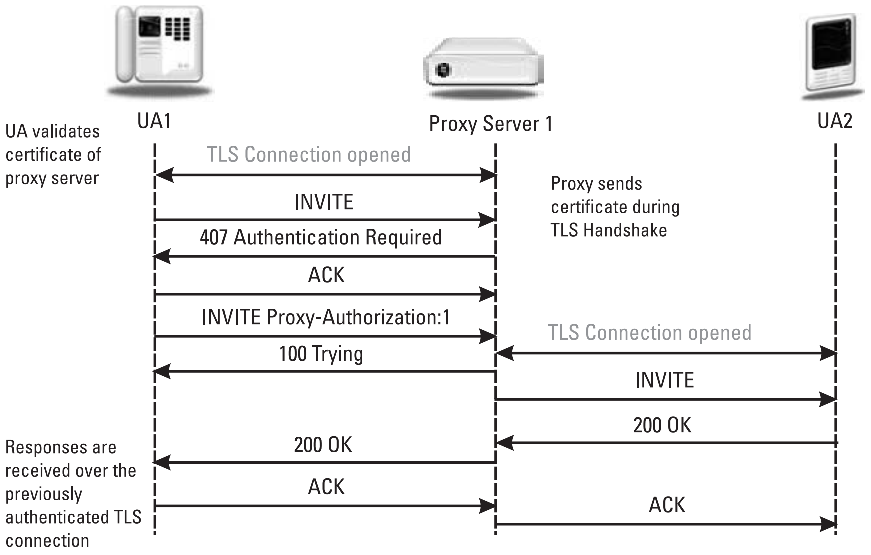

UA validates certificate of proxy server
UA1
Proxy Server 1
UA2
TLS Connection opened
INVITE
407 Authentication Required
ACK
INVITE Proxy-Authorization:1
100 Trying
Proxy sends certificate during TLS Handshake
TLS Connection opened
INVITE
200 OK
200 OK
ACK
ACK
Responses are received over the previously authenticated TLS connection

Figure 14.6 SIP authentication using TLS.

connection for a new SIP transaction or dialog. For example, in Figure 14.6, the TLS connection opened by the client for registration is reused by the server to send an incoming `INVITE`. Two work-in-progress SIP extensions relating to this are [18] and [19].

14.4.3 Secure SIP

Secure SIP or SIPS is a URI scheme for SIP that requires the use of TLS for every hop. As such, a proxy receiving a request with a SIPS Request-URI must either forward the request over a TLS connection or return an error message. This addition to SIP security has unfortunately seen little actual deployment due to some confusion in the specification and lack of support for TLS in some SIP systems and service providers. A working document describing clarifications and corrections in [20] may rectify this in the future.

14.4.4 Identity

Identity is another major issue in Internet communications that will be briefly covered in this section. In SIP, an identity is a SIP or SIPS URI. The identity of the originator of a SIP request will be the `From` header field URI. Note that this is not the same as the `Contact` URI, which may be the identity of the device or endpoint rather than the user. A response to a request in SIP does not carry the identity of the responder, as the `To` header field URI was set by the originator of the request and may not be the actual destination of the request (due to forwarding, redirection, and so forth).

Another issue with SIP identity is that sometimes the From URI set by the initiator may not be the appropriate identity for a request. For example, a SIP user may have a SIP identity (SIP URI) and also a PSTN identity (tel URI). When the request routes over SIP, the SIP identity is the correct one to be presented. However, if the call routes over the PSTN, the PSTN identity is the correct one to assert.

Another issue is that a UA may intentionally or unintentionally set an incorrect identity in a From URI. As such, there is no protocol way for this to be discovered or detected. This could be done within a domain in the following way. A proxy for the example.com domain could challenge all SIP requests sent by UAs in the example.com domain. This would make this proxy a *default outbound proxy* for these UAs. After the authentication challenge, the proxy could compare the authenticated identity (username and password of the digest challenge) with the From header field URI. If they did not match, the request could be rejected with a 400 Bad Request response. The proxy would only forward requests in which they matched. If a proxy in another domain trusted that the proxy in the other domain did this authentication and checking, the From URI could be trusted and treated as an authenticated identity.

A solution for these identity issues led to the development of the P-Asserted-Identity header field. This header field, which may carry one SIP or SIPS URI and one tel URI is inserted by a proxy after authentication, and may be different from the From URI. It is commonly used in PSTN interworking scenarios. However, this approach still requires proxies to trust each other. A better extension is described in the next section.

14.4.5 Enhanced SIP Identity

The enhanced SIP identity [21] extension attempts to improve on the P-Asserted-Identity by providing a cryptographically verifiable SIP identity assertion. The approach was developed specifically for the interdomain interconnection context. A new SIP header field, Identity, is inserted by a proxy server when forwarding a request. The proxy first authenticates the request to make sure it is being sent by the identity in the From header field. If it was, specific parts of the request are signed and the signature included in the Identity header field. An Identity-Info header field is used to provide a link to the certificate used to sign the message. As such, a small number of proxy servers within a domain using a small number of CA-signed certificates can provide verifiable identity assertions for many users in the domain.

When a UA or proxy receives a request containing the Identity and Identity-Info header fields, the UA or proxy first fetches the certificate of the proxy that signed the request. After the certificate is validated, the signature can be checked. If the signature validates, the `From` URI can be treated as an authenticated identity. This exchange is shown in Figure 14.7.

The parts of a SIP message that are protected by the identity signature are listed in Table 14.2.

14.5 SIP Certificate Service

The combination of certificates and TLS provides good security for SIP. However, there is significant difficulty in managing certificates in endpoints. These issues include:

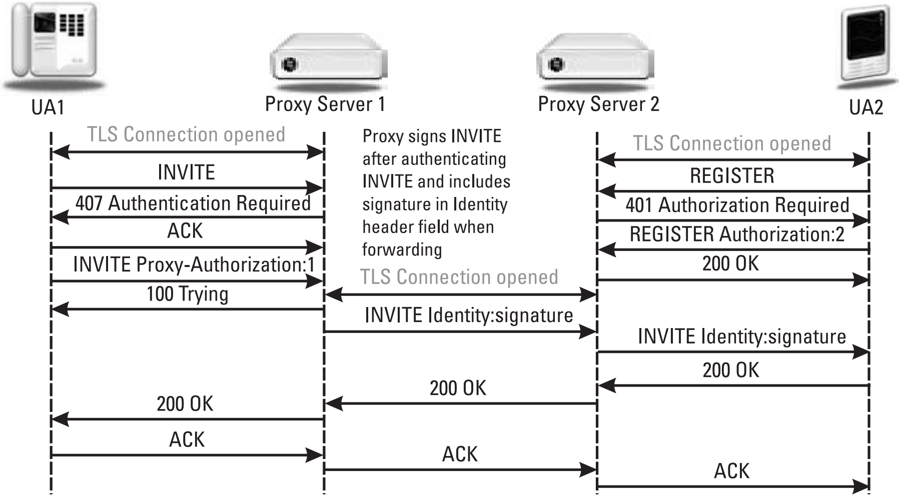

UA1
Proxy Server 1
Proxy Server 2
UA2
TLS Connection opened
INVITE
407 Authentication Required
ACK
INVITE Proxy-Authorization:1
100 Trying
Proxy signs INVITE after authenticating INVITE and includes signature in Identity header field when forwarding
TLS Connection opened
INVITE Identity:signature
TLS Connection opened
REGISTER
401 Authorization Required
REGISTER Authorization:2
200 OK
INVITE Identity:signature
200 OK
200 OK
200 OK
ACK
ACK
ACK

Figure 14.7 Enhanced SIP identity call flow.

Table 14.2
Parts of a SIP Message Protected by Enhanced SIP Identity

`To` URI	Original destination of the request.
`From` URI	Identity of the sender of the request.
`Call-ID`	Uniquely identifies the particular dialog.
`CSeq`	Uniquely identifies the transaction within the dialog.
`Date`	Used to prevent replay attacks.
`Contact` URI	Identifies the device used.
Message body	Identifies the SDP, message, or other end-to-end SIP message content.

1. How to issue certificates in UAs without incurring typical costs of commercial CA-signed certificates. While it is reasonable to have a CA-signed certificate in a few proxies and servers in a domain, having them in every UA is cost prohibitive.
2. How to install and manage certificates in endpoints. Getting a username and password installed on a SIP phone or soft client is easy, but securely importing a certificate is difficult.
3. How users of UA certificates can validate the certificates if they are not commercial CA-signed certificates. Without this validation step, their use is limited.
4. A SIP user of multiple devices will need to synchronize private keys across multiple devices. When keys are reissued, the new private key needs to be distributed in real time to all devices of the user.

An approach to solve this problem is a SIP certificate service for SIP [22]. This approach uses SIP to store, retrieve, and validate certificates. Each of the issues has a solution in this approach.

1. The approach makes so-called self-signed certificates usable. A self-signed certificate does not incur any per year or per user costs. However, the approach relies on enhanced SIP identity, so proxy servers in the domain will need to have a CA-signed certificate.
2. The approach defines how a certificate can be retrieved using SIP. A `SUBSCRIBE` to the SIP certificate server results in a `NOTIFY,` which contains the certificate in a message body. In addition, the subscription allows any changes or updates to the certificate to be applied in real-time, via an additional `NOTIFY`. When used to store private keys, the key can be encrypted by a pass phrase, so that only the user can utilize the private key even if the certificate server has been compromised.
3. A self-signed SIP certificate can be validated not by the certificate itself but by the method the certificate was fetched. In particular, when delivered in a `NOTIFY` that has an `Identity` header field that can be validated, the public key in the certificate can be trusted.
4. Multiple devices utilized by a user can all subscribe to the certificate server of the user. If one device generates a new public/private key pair and hence a new certificate, a `PUBLISH` is sent to the server, which then sends `NOTIFY`s to the other devices. As a result, the devices can all synchronize on the same certificate.

Figure 14.8 shows a call flow showing certificate management UA1 and UA2 are associated with the same user, so they use the same public and private

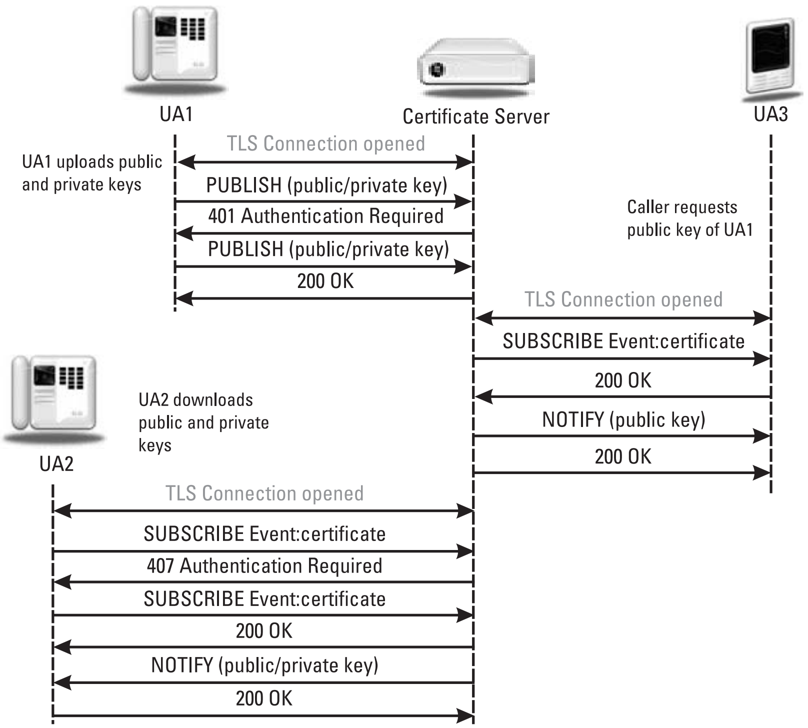

UA1
Certificate Server
UA3
UA1 uploads public and private keys
TLS Connection opened
PUBLISH (public/private key)
401 Authentication Required
PUBLISH (public/private key)
200 OK
Caller requests public key of UA1
TLS Connection opened
SUBSCRIBE Event:certificate
200 OK
NOTIFY (public key)
200 OK
UA2
UA2 downloads public and private keys
TLS Connection opened
SUBSCRIBE Event:certificate
407 Authentication Required
SUBSCRIBE Event:certificate
200 OK
NOTIFY (public/private key)
200 OK

Figure 14.8 SIP certificate service.

keys. UA3 is another user who only accesses the user's public key. An example NOTIFY is shown here:

```
NOTIFY alice@atlanta.example.com SIP/2.0
Subscription-State: active; expires=7200
....
From: <sip:bob@biloxi.example.com>;tag=1234
Identity: "NJguAbpmYXjnlxFmlOkumMI+MZXjB2iV/NW5xsFQqzD/"
Identity-Info: <https://atlanta.example.com/cert>;alg=rsa-sha1
 ....
Event: certificate
Content-Type: application/pkix-cert
Content-Disposition: signal

< certificate data >
```

An example PUBLISH is shown here:

```
PUBLISH sips:alice@atlanta.example.com SIP/2.0
...
Event: credential
Content-Type: multipart/mixed;boundary=boundary
Content-Disposition: signal
```

```
--boundary
Content-ID: 123
Content-Type: application/pkix-cert

< Public certificate for Alice >
--boundary
Content-ID: 456
Content-Type: application/pkcs8

< Private Key for Alice >
--boundary
```

14.6 Media Security

While SIP security relates to authentication, confidentiality, and integrity protection of the signaling information, the ability to establish secure media sessions is a related and equally important issue, which is discussed in this section. While potentially each media type can have unique security issues, they can be analyzed in two groups: non-RTP media and RTP media. To secure non-RTP media, the security challenge is how to establish a security protocol connection using SIP. For RTP media, the challenge is how to signal and key secure RTP or SRTP. These will be discussed in the following sections.

14.6.1 Non-RTP Media

The two most common security protocols for IP are IPSec and TLS as discussed in Sections 14.3.1 and 14.3.2. It is possible to use SIP to establish both of these protocols, although only the approach for TLS has been fully standardized by the IETF. The TLS approach is defined in [23] and is an extension of the approach used to establish connection-oriented media such as TCP using the offer/answer exchange. A partial example SDP offer is:

```
m=image 54111 TCP/TLS t38
c=IN IP4 192.0.2.2
a=setup:passive
a=connection:new
a=fingerprint:SHA-1 4A:AD:B9:B1:3F:82:18:3B:54:02:12:DF:3E:5D:49
:6B:19:E5:7C:AB
```

As defined in [24] the `a=setup` attribute is used to indicate the active/passive role in the exchange, while the `a=connection:new` is used to indicate a new TCP connection should be opened instead of reusing an existing one. The `a=fingerprint` attribute is used to send a SHA-1 hash of the certificate to be used for authentication. This is designed to allow the use of certificates that are not signed by a trusted CA (i.e., a so-called self signed certificate). Note that this fingerprint only has value in authentication if the offer/answer exchange is authenticated and has integrity protection. In SIP terms, this means it is signed using S/MIME and received over a single TLS connection hop, which has been authenticated, or a method such as enhanced SIP identity is used, as described in Section 14.4.5. Without these approaches, a self-signed certificate cannot be used, but a SIP certificate service, for example, could be used.

The approach for establishing IPSec defined in [25] uses SDP to establish an IKE (Internet key exchange) [26] session, which is then used to key IPSec. As with the connection-oriented media approach, the active/passive status must be negotiated and the certificate fingerprint exchanged in the offer/answer. An example SDP offer is:

```
m=application 500 UDP ike-esp
c=IN IP4 192.0.2.10
a=udp-setup:passive
a=fingerprint:SHA-1 4A:AD:B9:B1:3F:82:18:3B:54:02:12:DF:3E:5D:49
:6B:19:E5:7C:AB
```

14.6.2 Secure RTP

Secure RTP or SRTP [27] is a secure profile of RTP that provides authentication, confidentiality, and integrity protection for the media. It is designed to be efficient in terms of bandwidth and to minimize processing and latency. In addition, Secure RTP Control Protocol or SRTCP [27] has been defined. SRTP uses symmetric keys and ciphers, which must be negotiated during the offer/answer exchange with SIP. In addition, SRTP options such as the length of authentication tags and other properties must be negotiated. SRTP uses the AES algorithm for encryption in counter mode (AES-CTR), utilizing 128 bit or newly defined 256 bit keys [28]. The design of AES-CTR generates an encrypted key stream, which is Exclusive to ORed (XOR) with the plaintext media payload. This allows encryption to be done in parallel with codec media processing, reducing vital latency. The media SSRC allows a single SRTP master key to be used in both directions and in multiple media streams. The SRTP master key and master salt is used to generate the session encryption key, the session authorization key, and a session salt key. The SRTP packet header is identical to RTP by design, but the payload is encrypted, and the optional authorization tag covers the payload and the SRTP header. The optional master key index (MKI) is rarely used for Internet communications sessions. The next section will discuss the keying options for SRTP and how they are supported in the SIP offer/answer exchange.

14.6.3 Keying SRTP

A number of protocols have been developed to key SRTP. Common ones are listed in Table 14.3. This list does not include the SDP `k=` field [29], which has

Table 14.3
SRTP Keying Protocols

MIKEY	RFC3830	Multimedia Internet keying
SDES	RFC4568	SDP security descriptions
DTLS-SRTP	[33]	DTLS SRTP key management
ZRTP	[10]	ZRTP media path key management for SRTP

been deprecated in the current version of SDP since it is not able to negotiate all the parameters needed for SRTP. The first key management protocol developed was MIKEY (multimedia internet keying) [30]. However, the use of MIKEY in Internet communications with SIP was difficult, if not impossible, and the resulting deployment has been very limited. When transported with SIP in an offer/answer exchange, MIKEY uses the `a=key-mgt` SDP attribute [31]. A more deployable approach known as SDES (SDP security descriptions) was also standardized, which uses the `a=crypto` SDP attribute [32]. An example is shown below:

```
v=0
o=jdoe 2890844526 2890842807 IN IP4 10.47.16.5
s=SDP Seminar
i=A Seminar on the session description protocol
u=http://www.example.com/seminars/sdp.pdf
e=j.doe@example.com (Jane Doe)
c=IN IP4 161.44.17.12/127
t=2873397496 2873404696
m=video 51372 RTP/SAVP 31
a=crypto:1 AES_CM_128_HMAC_SHA1_80
inline:d0RmdmcmVCspeEc3QGZiNWpVLFJhQX1cfHAwJSoj|2^20|1:32
m=audio 49170 RTP/SAVP 0
a=crypto:1 AES_CM_128_HMAC_SHA1_32
inline:NzB4d1BINUAvLEw6UzF3WSJ+PSdFcGdUJShpX1Zj|2^20|1:32
m=application 32416 udp wb
a=orient:portrait
```

This example shows the specification of AES-CM with 128 bits keys and 80 bit HMAC SHA-1 authentication, a 2^20 key lifetime, and an MKI value of 1 and length of 32 bits. The SRTP master key and salt are included in base64 format. Since this approach effectively involves sending the SRTP key openly in the SDP offer or answer, SIP must provide confidentiality, authorization, and integrity protection for the keying material. While normal security requirements would mean that this required end-to-end S/MIME encryption of the SDP, in practice, many deployments rely instead on hop-by-hop transport over TLS. However, this does expose keying material to each SIP proxy server that forwards the `INVITE` request or `200 OK` response.

Note the use of a secure audio video profile for RTP (SAVP) in the m= line.

Both MIKEY and SDES keying approaches have limitations when used in actual SIP deployments. For example, when SIP forking, redirection, forwarding, and other features are invoked, the resulting security properties are problematic. Many of these issues are described in [34], which introduced new requirements for a media path keying method for SRTP. As a result, two new keying methods have been developed: DTLS-SRTP [33] and ZRTP [10]. Both of these keying methods perform the keying in the media path and thus solve many of the problems discussed in [34]. DTLS-SRTP uses the SIP offer/answer exchange to establish a DTLS connection between the SIP UAs over the same address and port as the media session. A secret from the DTLS handshake is used to derive SRTP keying material. The same methods described in Section 14.7.1 to establish TLS are used to establish the DTLS session. The same limitations of this method also apply to this keying approach—unless the SDP offer/answer exchange has end-to-end integrity protection, the DTLS session cannot be properly authenticated and as a result the SRTP keying is susceptible to an active MitM attack. ZRTP keying is described in Section 14.7.5.

14.6.4 Best Effort Encryption

The offer/answer exchange using SDP was designed to allow many media parameters to be negotiated, as described in Chapter 13. However, it was never designed to allow RTP profiles to be negotiable. As a result, an offered media stream must be declared as either secure (SAVP profile) or nonsecure (AVP profile). If the other UA does not support the chosen RTP profile, the media line must be declined. As a result, a UA offering a secure media call to another UA that does not support secure media will result in a call failure. While this may be a desired outcome in some cases, in many cases, a graceful fallback to a nonsecure media call is more desirable. This is especially true during the transition in current deployments in Internet communication where almost no UAs support secure media, to a future when hopefully all UAs will support secure media. This feature, which was invented by Phil Zimmermann and implemented first in the ZRTP protocol, has become know as best effort encryption. ZRTP inherently supports this mode of operation without any extensions or support from other protocols. For other keying approaches, an SDP extension known as SDP media capabilities negotiation [35] is being developed to allow other keying methods besides ZRTP to support best effort encryption. However, this extension represents a major change to SDP offer/answer negotiation and is very complex. As a result, it is likely to be sometime after this extension is finished before it is deployed in SIP networks.

14.6.5 ZRTP

ZRTP [10] is a self-contained media path keying protocol invented by Phil Zimmermann, the inventor of "pretty good privacy" or PGP encryption protocol [36]. ZRTP does not require an offer/answer exchange using the SAVP profile, and instead does discovery and negotiation in the media path. Early versions of ZRTP were actually RTP header extensions. However, the current version is a separate protocol that is multiplexed over the same transport address as the resulting media stream. ZRTP performs a Diffie-Hellman key agreement to derive the SRTP master secret. ZRTP uses a short authentication string (SAS) rendered to each UA, and compared by human users, to authenticate the Diffie-Hellman exchange and protect against a MitM attack. As a result, ZRTP does not require even an integrity protected signaling channel. Also, ZRTP does not require the same signaling protocol be used between the endpoints. For example, endpoints that support SIP, H.323, and Jingle *without any extensions* can all negotiate secure media sessions if the endpoints all support ZRTP. The ZRTP key agreement exchange is shown in Figure 14.9.

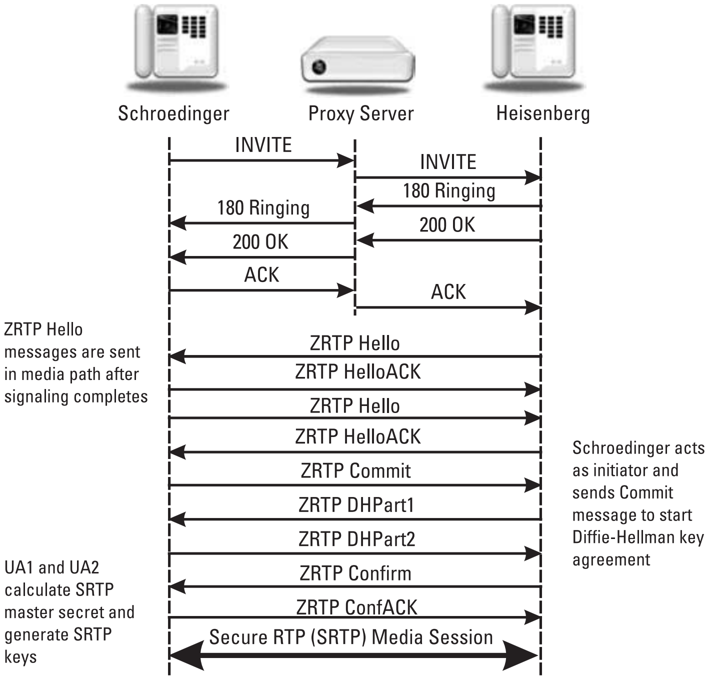

Schroedinger
Proxy Server
Heisenberg
INVITE
INVITE
180 Ringing
180 Ringing
200 OK
200 OK
ACK
ACK
ZRTP Hello messages are sent in media path after signaling completes
ZRTP Hello
ZRTP HelloACK
ZRTP Hello
ZRTP HelloACK
ZRTP Commit
ZRTP DHPart1
ZRTP DHPart2
ZRTP Confirm
ZRTP ConfACK
Secure RTP (SRTP) Media Session
Schroedinger acts as initiator and sends Commit message to start Diffie-Hellman key agreement
UA1 and UA2 calculate SRTP master secret and generate SRTP keys

Figure 14.9 ZRTP key agreement.

14.7 Questions

Q14.1 List the main differences between IPSec and TLS. Give an example of a common usage of each protocol on the Internet today.

Q14.2 Describe how mutual authentication can be achieved between a SIP UA and a SIP proxy. Clearly show how secrets and credentials are being shared.

Q14.3 Which parts of the SIP message shown below are protected by the identity signature? Give an example attack that could be launched for each part that is protected (e.g., if a particular header field is protected, explain what an attacker could do if they modified the header field).

```
INVITE sip:913145551212@siptest.mci.com SIP/2.0
From: sip:alan.johnston@siptest.mci.com;tag=1c31657
To: sip:913145551212@siptest.mci.com
Call-ID: call-1087694972-1
CSeq: 2 INVITE
Contact: <sip:alan.johnston@161.44.17.12>
Content-Type: application/sdp
Content-Length: 306 Accept-Language: en
Allow: INVITE, ACK, CANCEL, BYE, REFER, OPTIONS, NOTIFY,
 REGISTER, SUBSCRIBE
Supported: replaces, join
User-Agent: Pingtel/2.1.11 (VxWorks)
Date: Wed, 16 Apr 2008 01:29:40 GMT
Proxy-Authorization: DIGEST USERNAME=″alanjohnston″, REALM=″MCI″,
 NONCE=″f9d26bd9c4766eacd3101d8d2dc5f38d.1087694984″,
 URI=″sip:913145551212@siptest.mci.com″,
 RESPONSE=″fe1545a816d698b993b83bade8428c0f″
Via: SIP/2.0/UDP 161.44.17.12:5060;branch=dk32234df

v=0
o=jdoe 2890844526 2890842807 IN IP4 161.44.17.12
s=-
i=A Seminar on the session description protocol
u=http://www.example.com/seminars/sdp.pdf
e=j.doe@example.com (Jane Doe)
c=IN IP4 161.44.17.12
t=0 0
m=video 51372 RTP/SAVP 31
a=crypto:1 AES_CM_128_HMAC_SHA1_80
inline:d0RmdmcmVCspeEc3QGZiNWpVLFJhQX1cfHAwJSoj|2^20|1:32
```

Q14.4 Which type of attack on SIP do you think is the most damaging? Give an example of a security mechanism to defend against this attack.

Q14.5 Give an example of parameters that need to be configured in order to use SRTP. (Hint: look at the contents of SDP extensions used to carry SRTP information.)

Q14.6 Explain how best effort encryption works and why it is important.

Q14.7 How does the SIP certificate service simplify key provisioning in UAs?

Q14.8 What are the differences in packet format between RTP and SRTP?

Q14.9 What are the advantages and disadvantages of TLS and IPSec for securing SIP sessions?

Q14.10 Give examples of replay, impersonation, and hijacking attacks on a SIP session.

References

[1] Johnston, A., and D. Piscatello, *Understanding VoIP Security*, Norwood, MA: Artech House, 2006.

[2] Singh, S., *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*, New York: Anchor, 2000.

[3] Schnier, B., *Applied Cryptography: Protocols, Algorithms, and Source Code in C,* 2nd ed., New York: John Wiley & Sons, 1996.

[4] Advanced Encryption Standard (AES), "Federal Information Processing Standard 197," November 2001.

[5] ANSI X3.106, "American National Standard for Information Systems—Data Link Encryption," American National Standards Institute, 1983.

[6] Rivest, R. L., *The RC4 Encryption Algorithm*, RSA Data Security, Inc., 1992.

[7] Rivest, R. L., A. Shamir, and L. M. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, Vol. 21, No. 2, February 1978, pp. 120–126.

[8] Diffie, W., and M. E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, Vol. 22, 1976, pp. 644–684.

[9] Rescorla, E., "Diffie-Hellman Key Agreement Method," RFC 2631, June 1999.

[10] Zimmermann, P., A. Johnston, and J. Callas, "ZRTP: Media Path Key Agreement for Secure RTP," draft-zimmermann-avt-zrtp-15 (work in progress), March 2009.

[11] Housley, R., "Internet X.509 Public Key Infrastructure and CRL Profile," RFC 2459, November 1998.

[12] Kent, S., and R. Atkinson, "Security Architecture for the Internet Protocol," RFC 2401, November 1998.

[13] Dierks, T., and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1," RFC 4346, April 2006.

[14] Dierks, T., and C. Allen, "The TLS Protocol Version 1.0," RFC 2246, January 1999.

[15] Rescorla, E., and N. Modadugu, "Datagram Transport Layer Security," RFC 4347, April 2006.

[16] Arends, R., et al., "DNS Security Introduction and Requirements," RFC 4033, March 2005.

[17] Franks, J., et al, "HTTP Authentication: Basic and Digest Access Authentication," RFC 2617, June 1999.

[18] Jennings, C., and R. Mahy, "Managing Client Initiated Connections in the Session Initiation Protocol (SIP)," draft-ietf-sip-outbound-20 (work in progress), June 2009.

[19] Gurbani, V., R. Mahy, and B. Tate, "Connection Reuse in the Session Initiation Protocol (SIP)," draft-ietf-sip-connect-reuse-13 (work in progress), March 2009.

[20] Audet, F., "The Use of the SIPS URI Scheme in the Session Initiation Protocol (SIP)," draft-ietf-sip-sips-09 (work in progress), November 2008.

[21] Peterson, J., and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)," RFC 4474, August 2006.

[22] Jennings, C., and J. Fischl, "Certificate Management Service for The Session Initiation Protocol (SIP)," draft-ietf-sip-certs-07 (work in progress), November 2008.

[23] Lennox, J., "Connection-Oriented Media Transport over the Transport Layer Security (TLS) Protocol in the Session Description Protocol (SDP)," RFC 4572, July 2006.

[24] Yon, D., and G. Camarillo, "TCP-Based Media Transport in the Session Description Protocol (SDP)," RFC 4145, September 2005.

[25] Saito, M., and D. Wing, "Media Description for IKE in the Session Description Protocol (SDP)," draft-saito-mmusic-sdp-ike-04 (work in progress), March 2009.

[26] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol," RFC 4306, December 2005.

[27] Baugher, M., et al., "The Secure Real-time Transport Protocol (SRTP)," RFC 3711, March 2004.

[28] McGrew, D., "The Use of AES-192 and AES-256 in Secure RTP," draft-ietf-avt-srtp-big-aes-01.txt (work in progress), July 2009.

[29] Handley, M., and V. Jacobson, "SDP: Session Description Protocol," RFC 2327, April 1998.

[30] Arkko, J., et al., "MIKEY: Multimedia Internet KEYing," RFC 3830, August 2004.

[31] Arkko, J., et al., "Key Management Extensions for Session Description Protocol (SDP) and Real Time Streaming Protocol (RTSP)," RFC 4567, July 2006.

[32] Andreasen, F., M. Baugher, and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams," RFC 4568, July 2006.

[33] McGrew, D., and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for Secure Real-Time Transport Protocol (SRTP)," draft-ietf-avt-dtls-srtp-07 (work in progress), February 2009.

[34] Wing, D., et al., “Requirements and Analysis of Media Security Management Protocols,” RFC 5479, April 2009.

[35] Gilman, R., R. Even, and F. Andreasen, “SDP Media Capabilities Negotiation,” draft-ietf-mmusic-sdp-media-capabilities-07 (work in progress), February 2009.

[36] Callas, J., et al., “OpenPGP Message Format,” RFC 4880, November 2007.

15

Peer-to-Peer SIP

Peer-to-peer (P2P) technologies have been becoming an important part of Internet applications. This chapter will introduce P2P technologies and ideas developed from file sharing applications and show how they can be applied to an Internet communications protocol such as SIP. Finally, emerging work in standardizing P2P SIP will be covered including the RELOAD Peer Protocol and Host Identity Protocol (HIP).

15.1 P2P Properties

P2P applications became popular on the Internet with the advent of file sharing applications. Today P2P concepts are used in other applications such as content delivery, distributed computing, application layer multicast, and also Internet communications. The most important properties of P2P systems are:

- Self organizing;
- Decentralized control;
- Direct unmediated communication.

A good overview of P2P architectures and how they relate to common P2P applications such as content delivery networks, distributed computing, and communications is in [1].

P2P systems have these properties to varying degrees. For example, some P2P systems still use a centralized server for authentication; others use centralized servers for NAT and firewall traversal, or use a hierarchy in which peers at the same level communicate but use other protocols (i.e., client-server) between levels in the hierarchy. The next section will discuss some P2P properties of SIP.

15.2 P2P Properties of SIP

SIP [2] by design already has many P2P aspects and properties. For example, the only required element in a SIP network is a UA. UAs can communicate directly with peers: other UAs. In fact, all servers, including proxy, registrar, and redirect servers are optional in the SIP architecture. Compare this to other protocols such as H.323, where a gatekeeper is a required element, or Jabber, where a Jabber server is required–Jabber clients cannot communicate directly with another Jabber client without a server in the middle. Some other fundamental aspects of SIP are also P2P in nature. The dialog identifier—To tag, From tag, and Call-ID—is generated by and under the control of UAs. Servers play no role in its creation. SIP also has P2P call control modes, such as those enabled by the REFER method. Many features which can be implemented by UAs alone using REFER require the use of servers in other telephony architectures. The basic SIP call control model is inherently P2P [3].

However, for all these P2P properties, nearly all SIP deployments use various types of servers. The reason for this is that SIP servers, especially proxy servers, perform very essential roles in discovery, rendezvous, and NAT traversal, as discussed in previous chapters. For example, a SIP UA that uses DHCP and has a dynamic IP address does not have a constant SIP URI that can be used to reach it. While technologies such as dynamic DNS can be used to enable this, sending regular registrations to a SIP proxy is an excellent solution to this. Also, for a SIP UA behind a NAT, the proxy server serves the role of rendezvous server, allowing UAs to discover each other, and exchange candidate addresses for hole punching. Finally, servers must provide fallback for NAT traversal as TURN servers or media relays when hole punching fails.

In addition to proxy servers, many networks also employ B2BUAs as well, which greatly reduce the P2P functionality of SIP. An alternative architecture for SIP, known as simple SIP [4] shows how nearly all features and services can be implemented using P2P approaches and a small set of SIP extensions. This architecture only uses servers for rendezvous functionality. As a result, most SIP exchanges occur over a single SIP hop. Besides simplifying and reducing latency and complexity, this also greatly enhances security. Much of the complexity discussed in the previous chapter on security relate to having proxy servers and B2BUAs in the path between two UAs. For example, if TLS is used in a single hop between UAs, certificates can be used for mutual authentication. In addition, the resulting exchange has end-to-end integrity protection and confidentiality.

If a P2P approach could be developed to replace these discovery, rendezvous, and NAT traversal functions, SIP could be operated in a pure P2P manner, providing all the advantages of P2P systems.

15.3 P2P Overlays

P2P networks form a network known as an *overlay,* which sits on top of the usual Internet connections. When joining the P2P network, the host, known as a node or a peer, becomes a part of the overlay network. Messages are routed between nodes using overlay routing, which sits on top of normal IP routing. The protocol used to route and map messages across the overlay is known as a peer protocol.

P2P systems need an algorithm for distributing data across the overlay. Often, this is done using a distributed hash table or DHT. For an example of a DHT algorithm used to build a P2P overlay, consider the Chord Protocol [5]. Chord uses a ring architecture, where nodes organize themselves in the overlay using their node number *n*. If *m* bits are used for the node number, then there are a maximum of 2^*m* nodes in the overlay. For Chord, typically *m* = 160 is used, which can support an enormous number of nodes. A node in the overlay keeps track of the two closest nodes in the overlay: the *predecessor node*, the closest node with a lower node number; and the *successor node*, the closest node with a higher node number. Chord provides a distributed database function for nodes in the overlay. Each node stores a small amount of information on behalf of other nodes in the overlay. To locate information in the overlay, the search index for the information (confusingly known as a "key" in Chord even though it has nothing to do with encryption) is hashed to determine the node number of the node responsible for storing this information. For example, to find information about a file or service named "voicemail," the message digest of the string "voicemail" would be calculated, and the result would point to the node in the overlay where information about this service or file would be located. Chord uses *m* = 160 and SHA-1 as the hash function, which returns a 160 bit value, so the hash will always return a valid node number in the overlay. Since not every node number in the overlay is occupied by a peer, the actual node responsible will be the successor node to that node location. As a result, a node in the overlay is responsible for storing data about all nodes between its own node number and the node number of the predecessor node. If a new node joins the overlay between the node and its predecessor node, the responsibility will shift, the new node will take over responsibility for the nodes between it and the old predecessor node, and the node will now have the new node as its predecessor.

This distributed database maintains the dynamic mapping between data elements and the Chord node number. The other part of the Chord algorithm is for routing messages between nodes in the overlay. There are two distinct problems:

1. How to find the actual node responsible for a piece of data that is going to be stored or retrieved, given that in most cases there will not be a

node in the overlay with that exact node ID. As a result, the problem is to find the nearest node.

2. Having found that node ID, the problem becomes how to contact that node (i.e., node number to IP address resolution).

For the first problem, each node in Chord maintains a small routing table known as a *finger table*. A finger table is only a small subset of the entire routing table for the overlay. Chord does not attempt to build a complete routing table for the overlay since it would likely be too large and too dynamic to be useful. Instead, each node maintains a finger table with *m* entries. The table starts with the successor node, and the *k*th element in the finger table is the actual node in the overlay closest to the $2^k + n$, where *n* is node's number. For example, consider the tiny Chord overlay in Figure 15.1 where *m* = 8. The chord finger table for node 27 is shown in Table 15.1.

In this example, node 27 wants to find the content whose location in the overlay is 210. The node consults the finger table and finds the closest node is node 159. Node 27 then queries node 159 for the location of node 210. Node 159 consults its finger table and returns node 202. Node 27 then contacts node 202 directly who returns node 215. Node 27 then contacts node 215 who responds indicating that it is responsible for node 210. At this point, node 27 can query node 215 for the desired information. Note that this type of routing is known as iterative routing, in that a number of iterations or steps are performed at the source to get to the destination. The source is always in control and can verify routing integrity. An alternative approach is recursive routing where the request goes around the overlay in hops until it gets to the destination. This is

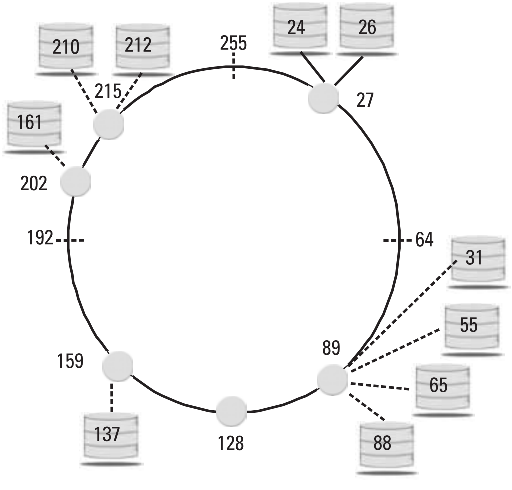


Figure 15.1 Chord routing example.

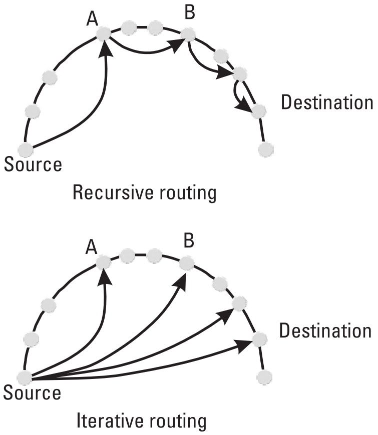
Table 15.1
Chord Finger Table for Node 27 in Figure 15.1

k	Node Closest to	Actual Node
0	$2^0 + 27$	89
1	$2^1 + 27$	89
2	$2^2 + 27$	89
3	$2^3 + 27$	89
4	$2^4 + 27$	89
5	$2^5 + 27$	89
6	$2^6 + 27$	128
7	$2^7 + 27$	159

typically faster than iterative routing and involves less processing on the source. The differences are shown in Figure 15.2.

This routing method in Chord can be shown to allow messages to be routed across the overlay in the order of log(*m*) hops. Since this number only grows logarithmically with the size of the overlay, Chord scales very well. In addition, the queries across the overlay, appearing geometrically as chords on a circle, give Chord its name.

The second aspect of overlay routing is the mapping of Node IDs to IP addresses. Most academic Chord networks assume that all nodes have a public IP address and hence are addressable. They then hash the host IP address (assumed to be unique due to the uniqueness of IP addresses on the Internet) to get the host number. Obviously, for most real Internet applications, the assumption of a public IP address is not realistic, and the IP address cannot be used to generate node IDs. Also, one cannot assume the transitivity of connections across an



Recursive routing

Iterative routing

Figure 15.2 Iterative verses recursive routing on an overlay.

overlay due to NAT. This is one reason why often recursive routing is used on overlays instead of iterative routing.

On real overlays, during the bootstrap process (an example of which is described for the RELOAD protocol in the next section), a node learns its successor and predecessor nodes and establishes connections to them using the bootstrap server for rendezvous. The node then requests the finger tables of the neighbor elements and copies information about the node ID that it would use to populate its own finger table. For the nodes that should be in the new node's finger table, the node uses the other node as a rendezvous server so that it can contact the other nodes, perhaps running ICE for hole punching. Once established, this connection is maintained through keep alives. The node can then request the finger table from this node, and continue in this way until it has a complete finger table and connections established to each of these nodes. As nodes join and leave the overlay, the node will make adjustments to the finger table.

By having nodes in the overlay provide this rendezvous service to other nodes, establishing and maintaining connections, the overlay can work even though nodes are behind NATs.

15.4 RELOAD

The RELOAD (Resource Location and Discovery) Protocol [6] is a P2P protocol being developed in the P2PSIP working group of the IETF. Currently, the protocol is defined in an Internet Draft and still needs significant work before it will be finalized and published as an RFC. Another peer protocol which is not progressing as a standard but is being implemented and deployed is P2PP [7]. Some of the protocol details presented in this section may have changed. Readers interested in peer protocols should check for the latest Internet Drafts and RFCs at the P2PSIP working group charter page [8].

RELOAD attempts to provide a self-organizing overlay network with routing between nodes and service discovery. RELOAD can be used for a number of applications, each of which is defined in separate usage documents. For example, the SIP usage of RELOAD is defined in [9]. In RELOAD, nodes join the overlay by establishing connections to a small number of other nodes in the overlay. Each node in the overlay has a node identifier (Node-ID), which is a 128 bit value used for routing messages across the overlay. Node-IDs are assigned by a centralized enrollment server. The enrollment server also issues credentials (certificates containing the Node-ID) used to authenticate nodes in the overlay. A node joins the overlay using a bootstrap node: a stable node in the overlay with a public IP address that is discoverable based on the name of the overlay. A new node forms new connections to other nodes in the overlay and builds a routing table. The node then actually joins the overlay by inserting itself into the routing tables of surrounding nodes and storing the data associated with that part of the overlay. Information is stored and retrieved in the overlay distributed database using store and fetch requests. The functions of different types of peers in the overlays are discussed in Table 15.2.

RELOAD uses a combination of binary encoding and TLV (tag length value) encoding; messages are routed over TLS or DTLS. Table 15.3 summarizes some of the diverse function of RELOAD that are grouped together in this protocol.

The SIP usage of RELOAD provides two main functions: registration and rendezvous. The registration function is the ability of a UA to store the mapping between their address of record (AOR) and their Node-ID. This allows another UA in the overlay to do a lookup of their AOR URI to determine their Node-ID. The rendezvous function is to use RELOAD to establish a direct connection between two UAs in the overlay so they can exchange SIP messages. Without NAT, this function would not be needed, and instead the registration function could provide a mapping between an AOR and an IP address and port. However, since many peers will be behind NAT, this rendezvous functionality allows them to perform hole punching using ICE in order to establish a direct TLS or DTLS connection between the UAs. At that point, normal SIP messages including INVITE, ACK, and BYE can be exchanged. To complete the NAT traversal, the SIP exchange can provide rendezvous functionality for the media, and hence ICE will be run again. Both the RELOAD ICE and SIP ICE applications need to

Table 15.2
Roles in an Overlay

Peer node	A full member of the overlay that stores information and routes overlay requests.
Client node	Not a full member of the overlay, but it can query the overlay for information.
Joining peer	A peer attempting to join the overlay.
Bootstrap peer	A stable peer with a public IP address used to join the overlay.
Admitting peer	The first peer contacted by joining peer after the bootstrap peer.
Proxy peer	A SIP proxy server that is also a peer in the overlay. It helps route SIP messages to and from the outside of the overlay.
Enrollment server	It issues credentials used by peers for authentication in the overlay.

Table 15.3
RELOAD Protocol Functions

Usage layer	An application that uses RELOAD, such as SIP.
Message transport	Provides end-to-end reliability of overlay messages.
Storage	Provides database functions for overlay users.
Topology plug-in	Supports different topologies based on overlay algorithm.

have a fallback, and that means TURN transport relay addresses. The method of discovering and using TURN servers is still under development. RELOAD can use different overlay protocols. The current version uses a modified version of Chord.

15.5 Host Identity Protocol

RELOAD as a protocol reimplements many layers of the protocol stack. It provides many transport layer and routing layer services, including reliability and fragmentation. An alternative approach to building an overlay is to reuse the existing parts of the IP stack by inserting shim layers. One example of this is the usage of Host Identity Protocol (HIP) [10] over an overlay. The protocol stack of HIP is shown in Figure 15.3. The HIP architecture separates the identifier and locator roles of an IP address. An HIP host has a constant host identity, while its IP address can change in real-time and over time. The host identity is a public private key pair. HIP uses a Host Identity Tag (HIT) as an identifier. A HIT is a self-certifying identity because it is formed as a prefixed, truncated hash of the public key, ORCHID (Overlay Routable Cryptographic Hash Identifier) [11] as an identifier, which is a 128-bit address that has the same format as an IPv6 address, but is not routable using IPv6. To claim a particular HIT, a host uses the private key associated with the public key of the HIT. The use of an ORCHID means that the IPv6 APIs can be used with HIP without any changes.

HIP may one day be run natively as a transport on top of IP. However, today it is tunneled over UDP so that it can traverse NAT.

HIP exchanges messages between hosts by first establishing connections known as HIP associations. An HIP association is secured using an IPSec tunnel, which is keyed using a DH exchange authenticated by the public keys associated with the host HITs. The HIP base exchange is used to signal the HIP

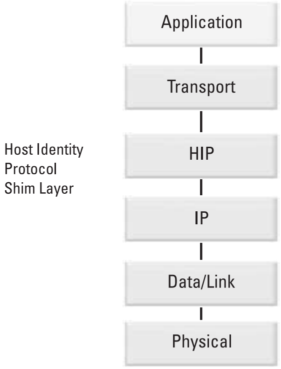


Figure 15.3 Host Identity Protocol stack.

association, and is currently being extended to support ICE hole punching for NAT traversal.

HIP needs a way to map an HIT to an IP address used to establish HIP connections. This can be done with either DNS, a registration method (a HIP rendezvous server), or an overlay routing protocol. This usage of HIP is described in [12]. Currently, the IETF HIP working group is working on taking parts of the RELOAD protocol and using them for discovery and rendezvous for HIP. Adding HIP to RELOAD would have the following advantages:

- Since HIP uses ICE, all other protocols that run on top of HIP, including RELOAD, SIP, and RTP would all get the benefits of ICE without having to implement it. For example, consider two peers who need to establish a RELOAD connection, exchange SIP messages, and then exchange RTP media. The RELOAD messages would be run over HIP and ICE would be run to establish the secure HIP association between the hosts; then RELOAD messages could be exchanged. Next, SIP could be sent over the same HIP association and finally RTP, all without having to run ICE again.
- Minimal changes to SIP and RTP would be required to use HIP. HITs would be carried as IPv6 addresses in SIP URIs and in SDP offer/answer exchanges for RTP media negotiation. This allows direct HIP usage for SIP and RTP without major changes to the protocols. The only change is the way URIs are resolved. In normal SIP, routing is based on the host part, and the user part is only resolved within the domain. In P2PSIP with HIP, the entire user and host part of the URI would need to be resolved. One solution to this would be to only use the domain part of the URI—the user part would essentially be ignored. While both a RELOAD and an HIP approach require this change to SIP, RELOAD requires even more extensive changes.

The complete set of HIP experimental specifications is listed in Table 15.4. For more details of HIP, see [19].

15.6 Conclusion

P2P SIP is an area with quite a lot of activity, both in industry and in standards. P2P has advantages in some applications and scenarios. Protocols such as RELOAD and HIP will likely be used to implement some of these applications and scenarios.

Table 15.4
HIP References

RFC 5201	HIP Base Protocol [10]
RFC 5202	ESP transport of HIP [13]
RFC 5203	HIP registration extension [14]
RFC 5204	HIP rendezvous extension [15]
RFC 5205	HIP DNS extension [16]
RFC 5206	HIP mobility and multihoming [17]
RFC 5207	HIP NAT traversal [18] (specification does not use hole punching and has limited utility)

15.7 Questions

Q15.1 List some P2P properties of SIP.

Q15.2 Explain how an overlay uses a DHT.

Q15.3 For the Chord overlay in Figure 15.4, fill in the missing entries in the finger tables of Tables 15.5 and 15.6.

Q15.4 Explain how HITs are self-certifying.

Q15.5 Why does the use of peer protocol require changes in the way a SIP URI is resolved?

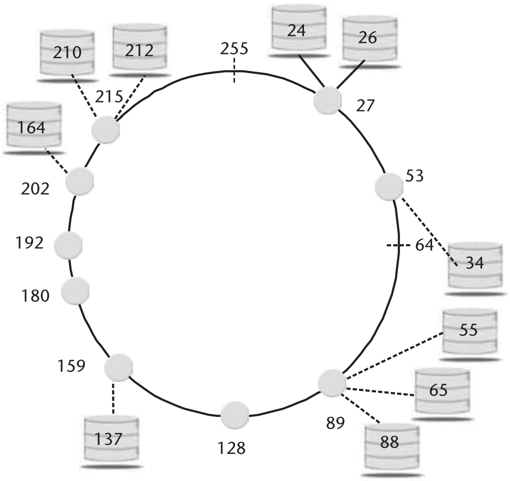


Figure 15.4 Chord overlay for Question Q15.3.

Table 15.5
Chord Finger Table for Node 27 in Figure 15.4

k	Node	Closest to Actual Node
0	$2^0 + 27$	53
1	$2^1 + 27$	53
2	$2^2 + 27$	53
3	$2^3 + 27$	53
4	$2^4 + 27$	?
5	$2^5 + 27$	?
6	$2^6 + 27$	?
7	$2^7 + 27$	159

Table 15.6
Chord Finger Table for Node 180 in Figure 15.4

k	Node	Closest to Actual Node
0	$2^0 + 180$	192
1	$2^1 + 180$	192
2	$2^2 + 180$	?
3	$2^3 + 180$	?
4	$2^4 + 180$	202
5	$2^5 + 180$	?
6	$2^6 + 180$	27
7	$2^7 + 180$	?

References

[1] Camarillo, G., "Peer-to-Peer (P2P) Architectures," draft-iab-p2p-archs-01 (work in progress), April 2009.

[2] Rosenberg, J., H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261, June 2002.

[3] Mahy, R., R. Sparks, J. Rosenberg, D. Petrie, and A. Johnston, "A Call Control and Multi-Party Usage Framework for the Session Initiation Protocol (SIP)," draft-ietf-sipping-cc-framework-11 (work in progress), March 2009.

[4] Sinnreich, H., A. Johnston, and E. Shim, "Simple SIP Usage Scenario for Applications in the Endpoints," draft-sinnreich-sip-tools-07 (work in progress), June 2009.

[5] Stoica, I., et al., "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *ACM SIGCOMM 2001*, San Diego, CA, August 2001, pp. 149-160.

[6] Jennings, C., et al., "Resource Location and Discovery (RELOAD) Base Protocol," draft-ietf-p2psip-base-02 (work in progress), March 2009.

[7] Baset, S., H. Schulzrinne, and M. Matuszewski, "Peer-to-Peer Protocol (P2PP)," draft-baset-p2psip-p2pp-01 (work in progress), November 2007.

[8] http://www.ietf.org/html.charters/p2psip-charter.html.

[9] Jennings, C., et al., "A SIP Usage for RELOAD," draft-ietf-p2psip-sip-01 (work in progress), March 2009.

[10] Moskowitz, R., et al., "Host Identity Protocol," RFC 5201, April 2008.

[11] Nikander, P., J. Laganier, and F. Dupont, "An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers (ORCHID)," RFC 4843, April 2007.

[12] Camarillo, G., et al., "HIP BONE: Host Identity Protocol (HIP) Based Overlay Networking Environment," draft-ietf-hip-bone-01 (work in progress), March 2009.

[13] Jokela, P., R. Moskowitz, and P. Nikander, "Using the Encapsulating Security Payload (ESP) Transport Format with the Host Identity Protocol (HIP)," RFC 5202, April 2008.

[14] Laganier, J., T. Koponen, and L. Eggert, "Host Identity Protocol (HIP) Registration Extension," RFC 5203, April 2008.

[15] Laganier, J., and L. Eggert, "Host Identity Protocol (HIP) Rendezvous Extension," RFC 5204, April 2008.

[16] Nikander, P., and J. Laganier, "Host Identity Protocol (HIP) Domain Name System (DNS) Extensions," RFC 5205, April 2008.

[17] Nikander, P., et al., "End-Host Mobility and Multihoming with the Host Identity Protocol," RFC 5206, April 2008.

[18] Stiemerling, M., J. Quittek, and L. Eggert, "NAT and Firewall Traversal Issues of Host Identity Protocol (HIP) Communication," RFC 5207, April 2008.

[19] Gurtov, A., *Host Identity Protocol (HIP): Towards the Secure Mobile Internet,* Wiley Series on Communications Networking & Distributed Systems, New York: John Wiley & Sons, 2008.

16

Call Flow Examples

In this chapter, many of the concepts and details presented in the preceding chapters will be illustrated with examples. Each example includes a call flow diagram and a discussion of the example, followed by the message details. Each message is labeled in the figure with a message number for easy reference. For more examples of the protocol, refer to the SIP specification [1], the SIP call flows [2, 3] documents, and the SIP service examples document [4].

The purpose of the examples in this chapter is to illustrate aspects of the SIP protocol. The interoperation scenarios with the PSTN are not intended to fully define the interworking or show a complete parameter mapping between the protocols. Likewise, simplifications such as minimal authentication and direct client-to-gateway messaging are used to make the examples more clear.

16.1 SIP Call with Authentication, Proxies, and Record-Route

Figure 16.1 shows a basic SIP call between two SIP UAs involving two proxy servers. Rather than perform a DNS query on the SIP URI of the called party, the calling SIP phone sends the `INVITE` request to a proxy server for address resolution. The proxy server requires authentication to perform this service and replies with a `407 Proxy Authorization Required` response. Using the nonce from the challenge, the caller resends the `INVITE` with the caller's username and password credentials. The proxy checks the credentials, and finding them to be correct, performs the DNS lookup on the Request-URI. The `INVITE` is then forwarded to the proxy server listed in the DNS SRV record that handles the `language.org` domain. That proxy then looks up the Request-URI and locates a registration for the called party. The `INVITE` is forwarded to the destination UAS, a `Record-Route` header having been inserted to ensure that the proxy is present in

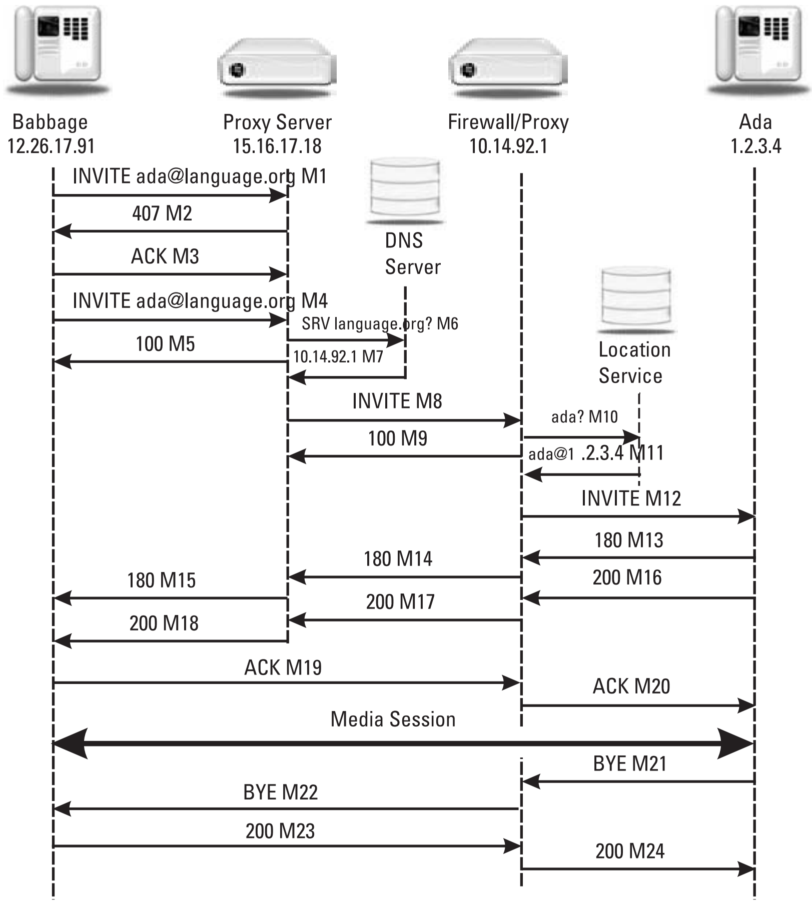


Figure 16.1 SIP-to-SIP call with authentication, proxies, and Record-Route.

all future requests by either party. This is because a directly routed SIP message to Ada would be blocked by the firewall.

The called party receives the `INVITE` request and sends `180 Ringing` and `200 OK` responses, which are routed back to the caller using the `Via` header chain from the initial `INVITE`. The `ACK` sent by the caller includes a `Route` header built from the `Record-Route` header field in the `200 OK` response. This routing skips the first proxy but includes the firewall proxy. The media session begins with the UAs exchanging RTP and RTCP packets.

The call terminates when the called party, Ada, sends a `BYE`, which includes a `Route` header generated from the `Record-Route` header field in the `INVITE`. Note that the `CSeq` for the called user agent is initialized to 1000. The acknowledgment of the `BYE` with a `200 OK` response causes both sides to stop sending media packets.

```
M1   INVITE sip:ada@language.org SIP/2.0                      Request-URI
     Via: SIP/2.0/UDP 12.26.17.91:5060;branch=z9hG4bK454
     Max-Forwards: 70
     From: Charles Babbage <sip:babbage@analyticalsoc.org>;tag=9382
     To: sip:ada@language.org
     Call-ID: f6329a3491e7
     CSeq: 1 INVITE                                           CSeq initialized to 1
     Contact: <sip:babbage@client.analyticalsoc.org>
     Subject: RE: Software
     User-Agent: DifferenceEngine/1
     Content-Type: application/sdp
     Content-Length: 137

     v=0
     o=Babbage 2890844534 2890844534 IN IP4 12.26.17.91
     s=-
     t=0 0
     c=IN IP4 12.26.17.91                                     Babbage’s IP address
     m=audio 49170 RTP/AVP 0                                  Port number
     a=rtpmap:0 PCMU/8000                                     Codec info

M2   SIP/2.0 407 Proxy Authentication Required
     Via: SIP/2.0/UDP 12.26.17.91:5060;branch=z9hG4bK454
     From: Charles Babbage <sip:babbage@analyticalsoc.org>;tag=9382
     To: <sip:ada@language.org>;tag=34q4356g
     Call-ID: f6329a3491e7
     CSeq: 1 INVITE
     Proxy-Authenticate: Digest                               Authentication challenge
       realm=”language.org”,
       nonce=”9c8e88df84f1cec4341ae6e5a359”,
       opaque=””, stale=FALSE, algorithm=MD5

M3   ACK sip:ada@language.org SIP/2.0
     Via: SIP/2.0/UDP 12.26.17.91:5060;branch=z9hG4bK454
     Max-Forwards: 70
     From: Charles Babbage <sip:babbage@analyticalsoc.org>;tag=9382
     To: <sip:ada@language.org>;tag=34q4356g
     Call-ID: f6329a3491e7
     CSeq: 1 ACK                          CSeq not incremented Method set to ACK

M4   INVITE sip:ada@language.org SIP/2.0          INVITE resent with credentials
     Via: SIP/2.0/UDP 12.26.17.91:5060;branch=z9hG4bK221
     Max-Forwards: 70
     From: Charles Babbage <sip:babbage@analyticalsoc.org>;tag=9382
     To: sip:ada@language.org
     Call-ID:f6329a3491e7                                     Call-ID unchanged
     CSeq: 2 INVITE                                           CSeq incremented
     Proxy-Authorization: Digest
       username=”Babbage”,                                    Credentials
       realm=”language.org”,
       nonce=”9c8e88df84f1cec4341ae6e5a359”,
       opaque=””, response=”e56131d19580cd833064787ecc”
     Contact: <sip:babbage@client.analyticalsoc.org>
     Subject: RE: Software
     User-Agent: DifferenceEngine/1
     Content-Type: application/sdp
     Content-Length: 137

     v=0
     o=Babbage 2890844534 2890844534 IN IP4 12.26.17.91
     s=-
     t=0 0
     c=IN IP4 12.26.17.91
     m=audio 49170 RTP/AVP 0
     a=rtpmap:0 PCMU/8000

M5   SIP/2.0 100 Trying                                Credentials accepted
     Via: SIP/2.0/UDP 12.26.17.91:5060;branch=z9hG4bK221
     From: Charles Babbage <sip:babbage@analyticalsoc.org>;tag=9382
     To: sip:ada@language.org
     Call-ID: f6329a3491e7
     CSeq: 2 INVITE

M6   DNS Query:

     SRV lookup on _udp._sip.language.org

M7   DNS Response:

     _sip._udp.language.org. 300 IN SRV 0 100 5060 proxy.languagel.org.
     proxy.language.org. 3600 IN A 10.14.92.1

M8   INVITE sip:ada@language.org SIP/2.0
     Via: SIP/2.0/UDP 15.16.17.18:5060;branch=z9hG4bK3f31049.1
     Via: SIP/2.0/UDP 12.26.17.91:5060;branch=z9hG4bK221
     Max-Forwards: 69
     From: Charles Babbage <sip:babbage@analyticalsoc.org>;tag=9382
     To: sip:ada@language.org
     Call-ID: f6329a3491e7
     CSeq: 2 INVITE
     Contact: <sip:babbage@client.analyticalsoc.org>
     Subject: RE: Software
     User-Agent: DifferenceEngine/1
     Content-Type: application/sdp
     Content-Length: 137

     v=0
     o=Babbage 2890844534 2890844534 IN IP4 12.26.17.91
     s=-
     t=0 0
     c=IN IP4 12.26.17.91
     m=audio 49170 RTP/AVP 0
     a=rtpmap:0 PCMU/8000

M9   SIP/2.0 100 Trying                                      Not Forwarded
     Via:SIP/2.0/UDP 15.16.17.18:5060;branch=z9hG4bK3f31049.1
     Via: SIP/2.0/UDP 12.26.17.91:5060;branch=z9hG4bK221
     From: Charles Babbage <sip:babbage@analyticalsoc.org>;tag=9382
     To: sip:ada@language.org
     Call-ID: f6329a3491e7
     CSeq: 2 INVITE

M10  Location Service Query: ada?

M11  Location Service Response: sip:ada@1.2.3.4

M12  INVITE sip:ada@1.2.3.4 SIP/2.0
```

```
     Via: SIP/2.0/UDP proxy.language.org:5060;branch=z9hG4bK24105.1
     Via: SIP/2.0/UDP 15.16.17.18:5060;branch=z9hG4bK3f31049.1
     Via: SIP/2.0/UDP 12.26.17.91:5060;branch=z9hG4bK221
     Max-Forwards: 68
     From: Charles Babbage <sip:babbage@analyticalsoc.org>;tag=9382
     To: sip:ada@language.org
     Call-ID: f6329a3491e7
     CSeq: 2 INVITE
     Contact: <sip:babbage@client.analyticalsoc.org>
     Subject: RE: Software
     User-Agent: DifferenceEngine/1
     Record-Route: <sip:10.14.92.1;lr>          Record-Route added by proxy
     Content-Type: application/sdp
     Content-Length: 137

     v=0
     o=Babbage 2890844534 2890844534 IN IP4 12.26.17.91
     s=-
     t=0 0
     c=IN IP4 12.26.17.91
     m=audio 49170 RTP/AVP 0
     a=rtpmap:0 PCMU/8000

M13  SIP/2.0 180 Ringing
     Via : SIP/2.0/UDP proxy.language.org:5060;branch=z9hG4bK24105.1
      ;received=10.14.92.1
     Via: SIP/2.0/UDP 15.16.17.18:5060;branch=z9hG4bK3f31049.1
     Via: SIP/2.0/UDP 12.26.17.91:5060;branch=z9hG4bK221
     From: Charles Babbage <sip:babbage@analyticalsoc.org>;tag=9382
     To: <sip:ada@language.org>;tag=65a3547e3          Tag added by called party
     Call-ID: f6329a3491e7
     CSeq: 2 INVITE
     Contact: sip:ada@drawingroom.language.org
     Record-Route: <sip:10.14.92.1;lr>

M14  SIP/2.0 180 Ringing
     Via: SIP/2.0/UDP 15.16.17.18:5060;branch=z9hG4bK3f31049.1
     Via: SIP/2.0/UDP 12.26.17.91:5060;branch=z9hG4bK221
     From: Charles Babbage <sip:babbage@analyticalsoc.org>;tag=9382
     To: <sip:ada@language.org>;tag=65a3547e3
     Call-ID: f6329a3491e7
     CSeq: 2 INVITE
     Contact: sip:ada@drawingroom.language.org
     Record-Route: <sip:10.14.92.1;lr>

M15  SIP/2.0 180 Ringing
     Via: SIP/2.0/UDP 12.26.17.91:5060;branch=z9hG4bK221
     From: Charles Babbage <sip:babbage@analyticalsoc.org>;tag=9382
     To: <sip:ada@language.org>;tag=65a3547e3
     Call-ID: f6329a3491e7
     CSeq: 2 INVITE

M16  SIP/2.0 200 OK                                                Call accepted
     Via: SIP/2.0/UDP 10.14.92.1:5060;branch=z9hG4bK24105.1
     Via: SIP/2.0/UDP 15.16.17.18:5060;branch=z9hG4bK3f31049.1
     Via: SIP/2.0/UDP 12.26.17.91:5060;branch=z9hG4bK221
     From: Charles Babbage <sip:babbage@analyticalsoc.org>;tag=9382
     To: <sip:ada@language.org>;tag=65a3547e3
     Call-ID: f6329a3491e7
     CSeq: 2 INVITE
     Contact: sip:ada@drawingroom.language.org
     Record-Route: <sip:10.14.92.1;lr>
     Content-Type: application/sdp
     Content-Length: 126

     v=0
     o=Ada 2890844536 2890844536 IN IP4 1.2.3.4
     s=-
     t=0 0
     c=IN IP4 1.2.3.4                                     Ada’s IP address
     m=audio 52310 RTP/AVP 0                              Port number
     a=rtpmap:0 PCMU/8000                                 Codec information

M17  SIP/2.0 200 OK
     Via: SIP/2.0/UDP 15.16.17.18:5060;branch=z9hG4bK3f31049.1
     Via: SIP/2.0/UDP 12.26.17.91:5060;branch=z9hG4bK221
     From: Charles Babbage <sip:babbage@analyticalsoc.org>;tag=9382
     To: <sip:ada@language.org>;tag=65a3547e3
     Call-ID: f6329a3491e7
     CSeq: 2 INVITE
     Contact: sip:ada@drawingroom.language.org
     Record-Route: <sip:10.14.92.1;lr>
     Content-Type: application/sdp
     Content-Length: 126

     v=0
     o=Ada 2890844536 2890844536 IN IP4 1.2.3.4
     s=-
     t=0 0
     c=IN IP4 1.2.3.4
     m=audio 52310 RTP/AVP 0
     a=rtpmap:0 PCMU/8000

M18  SIP/2.0 200 OK
     Via: SIP/2.0/UDP 12.26.17.91:5060;branch=z9hG4bK221
     From: Charles Babbage <sip:babbage@analyticalsoc.org>;tag=9382
     To: <sip:ada@language.org>;tag=65a3547e3
     Call-ID: f6329a3491e7
     CSeq: 2 INVITE
     Contact: sip:ada@drawingroom.language.org
     Record-Route: <sip:10.14.92.1;lr>
     Content-Type: application/sdp
     Content-Length: 126

     v=0
     o=Ada 2890844536 2890844536 IN IP4 1.2.3.4
     s=-
     t=0 0
     c=IN IP4 1.2.3.4
     m=audio 52310 RTP/AVP 0
     a=rtpmap:0 PCMU/8000

M19  ACK sip:ada@drawingroom.language.org SIP/2.0             Sent to ALG
     Via: SIP/2.0/UDP 12.26.17.91:5060;branch=z9hG4bK789
     Max-Forwards: 70
     From: Charles Babbage <sip:babbage@analyticalsoc.org>;tag=9382
     To: <sip:ada@language.org>;tag=65a3547e3
     Call-ID: f6329a3491e7
     CSeq: 2 ACK
     Route: <sip:10.14.92.1;lr>

M20  ACK sip:ada@drawingroom.language.org SIP/2.0
     Via: SIP/2.0/UDP 10.14.92.1:5060;branch=z9hG4bK24105.1
     Via: SIP/2.0/UDP 12.26.17.91:5060;branch=z9hG4bK789
     Max-Forwards: 69
     From: Charles Babbage <sip:babbage@analyticalsoc.org>;tag=9382
     To: <sip:ada@language.org>;tag=65a3547e3
     Call-ID: f6329a3491e7
     CSeq: 2 INVITE

M21  BYE sip:babbage@client.analyticalsoc.org SIP/2.0
     Via: SIP/2.0/UDP 1.2.3.4:5060;branch=z9hG4bK543
     Max-Forwards: 70
     From: Ada Lovelace <sip:ada@language.org>;tag=65a3547e3
     To: Charles Babbage <sip:babbage@analyticalsoc.org>;tag=9382
     Call-ID: f6329a3491e7
     CSeq: 1000 BYE                                  CSeq initialized to 1000
     Route: <sip:10.14.92.1;lr>                      From Record-Route header


M22  BYE sip:babbage@client.analyticalsoc.org SIP/2.0
     Via: SIP/2.0/UDP 10.14.92.1:5060;branch=z9hG4bK24105.1
     Via: SIP/2.0/UDP 1.2.3.4:5060;branch=z9hG4bK543
     Max-Forwards: 69
     From: Ada Lovelace <sip:ada@language.org>;tag=65a3547e3
     To: Charles Babbage <sip:babbage@analyticalsoc.org>;tag=9382
     Call-ID: f6329a3491e7
     CSeq: 1000 BYE

M23  SIP/2.0 200 OK
     Via: SIP/2.0/UDP 10.14.92.1:5060;branch= z9hG4bK24105.1
     Via: SIP/2.0/UDP 1.2.3.4:5060;branch=z9hG4bK543
     From: Ada Lovelace <sip:ada@language.org>;tag=65a3547e3
     To: Charles Babbage <sip:babbage@analyticalsoc.org>;tag=9382
     Call-ID: f6329a3491e7
     CSeq: 1000 BYE

M24  SIP/2.0 200 OK
     Via: SIP/2.0/UDP 1.2.3.4:5060;branch=z9hG4bK543
     From: Ada Lovelace <sip:ada@language.org>;tag=65a3547e3
     To: Charles Babbage <sip:babbage@analyticalsoc.org>;tag=9382
     Call-ID: f6329a3491e7
     CSeq: 1000 BYE
```

16.2 SIP Call with Stateless and Stateful Proxies with Called Party Busy

Figure 16.2 shows an example of a SIP with a stateless proxy server and a stateful proxy server. The call is not completed because the called party is busy. The called UA initially sends a `180 Ringing` response but then sends a `600 Busy Everywhere` response containing a `Retry-After` header to indicate that the call is being rejected. The stateful proxy returns a `100 Trying` response to the `IN-`

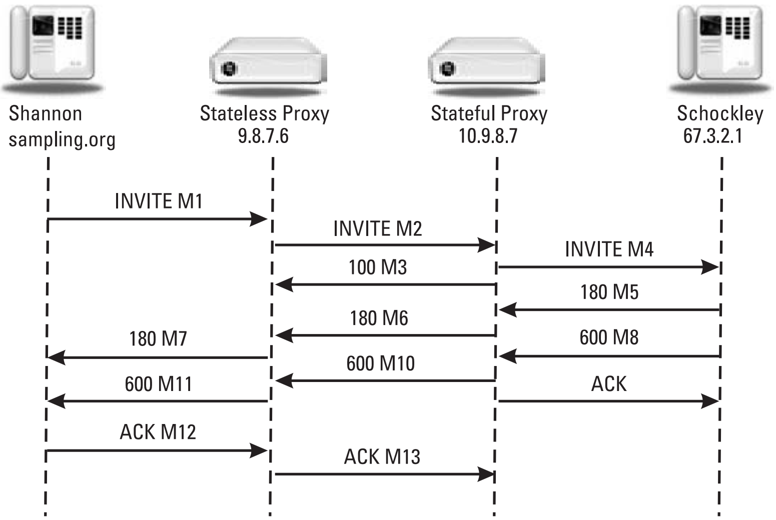


Figure 16.2 SIP call example with stateless and stateful proxies and a busy called party.

VITE, and also acknowledges the 600 Busy Everywhere response with an ACK. The stateless proxy does not send a 100 Trying and forwards the 600 Busy Everywhere and the ACK sent by the caller UA. Also note that the initial INVITE does not contain a message body.

```
M1     INVITE sip:schockley@transistor.org SIP/2.0
       Via: SIP/2.0/UDP discrete.sampling.org:5060;branch=z9hG4bK5654
       Max-Forwards: 70
       From: Shannon <sip:shannon@sampling.org>;tag=cgdf4
       To: Schockley <sip:shockley@transistor.com>
       Call-ID: 83727119273913
       CSeq: 1 INVITE
       Contact: <sip:Shannon@discrete.sampling.org>
       Date: Sat, 8 Jul 2000 08:23:00 GMT                Optional date header
       Content-Length: 0                          Optional content-length header

M2     INVITE sip:schockley@transistor.org SIP/2.0 X           Stateless proxy
       Via: SIP/2.0/UDP 9.8.7.6:5060;branch=z9hG4bK1.1   does not send 100
       Via: SIP/2.0/UDP discrete.sampling.org:5060;branch=z9hG4bK5654
       Max-Forwards: 69
       From: Shannon <sip:shannon@sampling.org>
       To: Schockley <sip:shockley@transistor.com>
       Call-ID: 83727119273913
       CSeq: 1 INVITE
       Contact: <sip:Shannon@discrete.sampling.org>
       Date: Sat, 8 Jul 2000 08:23:00 GMT
       Content-Length: 0

M3     SIP/2.0 100 Trying                          Stateful proxy does send 100
       Via: SIP/2.0/UDP 9.8.7.6:5060;branch=z9hG4bK1.1XX X
       Via: SIP/2.0/UDP discrete.sampling.org:5060;branch=z9hG4bK5654
       From: Shannon <sip:shannon@sampling.org>;tag=cgdf4
```

```
     To: Schockley <sip:shockley@transistor.com>
     Call-ID: 83727119273913
     CSeq: 1 INVITE
     Content-Length: 0
     M4 INVITE sip:schockley@transistor.org SIP/2.0
     Via: SIP/2.0/UDP 10.9.8.7:52103;branch=z9hG4bkff7d.1
     Via: SIP/2.0/UDP 9.8.7.6:5060;branch=z9hG4bK1.1
     Via: SIP/2.0/UDP discrete.sampling.org:5060;branch=z9hG4bK5654
     Max-Forwards: 68
     From: Shannon <sip:shannon@sampling.org>;tag=cgdf4
     To: Schockley <sip:shockley@transistor.com>
     Call-ID: 83727119273913
     CSeq: 1 INVITE
     Contact: <sip:Shannon@discrete.sampling.org>
     Date: Sat, 8 Jul 2000 08:23:00 GMT
     Content-Length: 0

M5   SIP/2.0 180 Ringing
     Via: SIP/2.0/UDP 10.9.8.7:52103;branch=z9hG4bKff7d.1
     Via: SIP/2.0/UDP 9.8.7.6:5060;branch=z9hG4bK1.1
     Via: SIP/2.0/UDP discrete.sampling.org:5060;branch=z9hG4bK5654
     From: Shannon <sip:shannon@sampling.org>;tag=cgdf4
     To: Schockley <sip:shockley@transistor.com>;tag=1
     Call-ID: 83727119273913
     CSeq: 1 INVITE
     Contact: <sip:shockley@4.5.6.7>
     Content-Length: 0

M6   SIP/2.0 180 Ringing
     Via: SIP/2.0/UDP 9.8.7.6:5060;branch=z9hG4bK1.1
     Via: SIP/2.0/UDP discrete.sampling.org:5060;branch=z9hG4bK5654
     From: Shannon <sip:shannon@sampling.org>;tag=cgdf4
     To: Schockley <sip:shockley@transistor.com>;tag=1
     Call-ID: 83727119273913
     CSeq: 1 INVITE
     Contact: <sip:shockley@4.5.6.7>
     Content-Length: 0

M7   SIP/2.0 180 Ringing
     Via: SIP/2.0/UDP discrete.sampling.org:5060;branch=z9hG4bK5654
     From: Shannon <sip:shannon@sampling.org>;tag=cgdf4
     To: Schockley <sip:shockley@transistor.com>;tag=1
     Call-ID: 83727119273913
     CSeq: 1 INVITE
     Contact: <sip:shockley@4.5.6.7>
     Content-Length: 0

M8   SIP/2.0 600 Busy Everywhere                          Schockley is busy
     Via: SIP/2.0/UDP 10.9.8.7:52103;branch=z9hG4bKff7d.1
     Via: SIP/2.0/UDP 9.8.7.6:5060;branch=z9hG4bK1.1
     Via: SIP/2.0/UDP discrete.sampling.org:5060;branch=z9hG4bK5654
     From: Shannon <sip:shannon@sampling.org>;tag=cgdf4
     To: Schockley <sip:shockley@transistor.com>;tag=1
     Call-ID: 83727119273913
     CSeq: 1 INVITE
     Retry-After: Sun, 9 Jul 2000 11:59:00 GMT
     Content-Length: 0
```

```
M9   ACK sip:schockley@transistor.com SIP/2.0               Stateful proxy does ACK
     Via: SIP/2.0/UDP 10.9.8.7:52103;branch=z9hG4bK5f7e.1
     Max-Forwards: 70
     From: Shannon <sip:shannon@sampling.org>;tag=cgdf4
     To: Schockley <sip:shockley@transistor.com>;tag=1
     Call-ID: 83727119273913
     CSeq: 1 ACK
     Content-Length: 0

M10  SIP/2.0 600 Busy Everywhere
     Via: SIP/2.0/UDP 9.8.7.6:5060;branch=z9hG4bK1.1
     Via: SIP/2.0/UDP discrete.sampling.org:5060;branch=z9hG4bK5654
     From: Shannon <sip:shannon@sampling.org>;tag=cgdf4
     To: Schockley <sip:shockley@transistor.com>;tag=1
     Call-ID: 83727119273913
     CSeq: 1 INVITE
     Retry-After: Sun, 9 Jul 2000 11:59:00 GMT
     Content-Length: 0
     Call Flow Examples 163

M11  SIP/2.0 600 Busy Everywhere          Stateless proxy does not ACK response
     Via: SIP/2.0/UDP discrete.sampling.org:5060;branch=z9hG4bK5654
     From: Shannon <sip:shannon@sampling.org>;tag=cgdf4
     To: Schockley <sip:shockley@transistor.com>;tag=1
     Call-ID: 83727119273913
     CSeq: 1 INVITE
     Retry-After: Sun, 9 Jul 2000 11:59:00 GMT
     Content-Length: 0

M12  ACK sip:schockley@transistor.com SIP/2.0
     Via: SIP/2.0/UDP discrete.sampling.org:5060;branch=z9hG4bK5654
     Max-Forwards: 70
     From: Shannon <sip:shannon@sampling.org>;tag=cgdf4
     To: Schockley <sip:shockley@transistor.com>;tag=1
     Call-ID: 83727119273913
     CSeq: 1 ACK
     Content-Length: 0

M13  ACK sip:schockley@transistor.com SIP/2.0
     Via: SIP/2.0/UDP 9.8.7.6:5060;branch=z9hG4bK5.1
     Via: SIP/2.0/UDP discrete.sampling.org:5060;branch=z9hG4bK5654
     Max-Forwards: 69
     From: Shannon <sip:shannon@sampling.org>;tag=cgdf4
     To: Schockley <sip:shockley@transistor.com>;tag=1
     Call-ID: 83727119273913
     CSeq: 1 ACK
     Content-Length: 0
```

16.3 SIP to PSTN Call Through Gateways

In the example shown in Figure 16.3, the calling SIP phone places a telephone call to the PSTN through a PSTN gateway. The SIP phone collects the dialed digits and puts them into a SIP URI used in the Request-URI and the `To` header. The caller may have dialed either the globalized phone number 1-202-555-1313 or they may have just dialed a local number 555-1313, and the SIP phone added

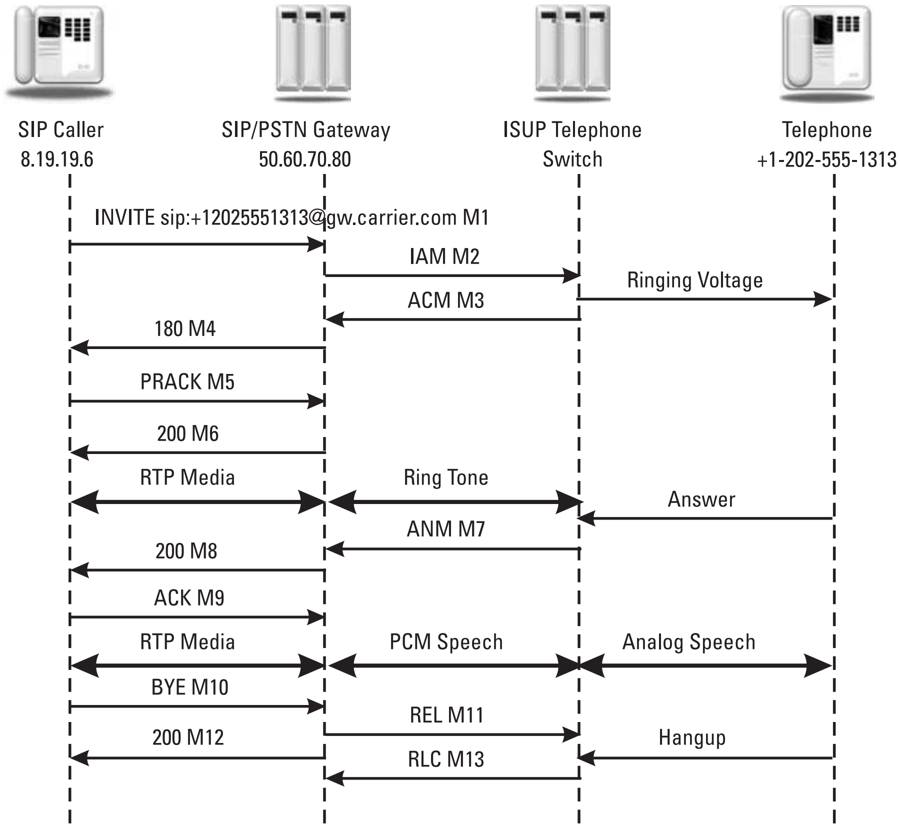


Figure 16.3 SIP to PSTN call flow through a gateway.

the assumed country code and area code to produce the globalized URI using the built-in dial plan. The SIP phone has been preconfigured with the IP address of the PSTN gateway, so it is able to send the `INVITE` directly to `gw.carrier.com`. The gateway initiates the call into the PSTN by selecting an SS7 ISUP trunk to the next telephone switch in the PSTN. The dialed digits from the `INVITE` are mapped into the ISUP IAM. The ISUP address complete message (ACM) is sent back by the PSTN to indicate that the trunk has been seized. Progress tones are generated in the one-way audio path established in the PSTN. In this example, ringtone is generated by the far end telephone switch. The gateway maps the ACM to the `183 Session Progress` response containing an SDP indicating the RTP port that the gateway will use to bridge the audio from the PSTN. Upon reception of the `183`, the caller's UAC begins receiving the RTP packets sent from the gateway and presents the audio to the caller so they know that the call is progressing in the PSTN.

The call completes when the called party answers the telephone, which causes the telephone switch to send an answer message (ANM) to the gateway.

The gateway then cuts the PSTN audio connection through in both directions and sends a `200 OK` response to the caller. Because the RTP media path is already established, the gateway echoes the SDP in the `183` but causes no changes to the RTP connection. The UAC sends an `ACK` to complete the SIP signaling exchange. Because there is no equivalent message in ISUP, the gateway absorbs the `ACK`.

The call terminates when the caller sends the `BYE` to the gateway. The gateway maps the `BYE` to the ISUP release message (REL). The gateway sends the `200 OK` to the `BYE` and receives an RLC from the PSTN. These two messages have no dependency on each other; if, for some reason, either the SIP or PSTN network does not respond properly, one does not want resources held in the other network as a result.

```
M1    INVITE sip:+12025551313@gw.carrier.com;user=phone SIP/2.0
      Via: SIP/2.0/UDP 8.19.19.06:5060;branch=z9hG4bK4545
      Max-Forwards: 70
      From: <sip:filo.farnsworth@television.tv>;tag=12
      To: <sip:+12025551313@gw.carrier.com;user=phone>
      Call-ID: 49235243082018498
      CSeq: 1 INVITE
      Supported: 100rel
      Contact: sip:filo.farnsworth@studio.television.tv
      Content-Type: application/sdp
      Content-Length: 154

      v=0
      o=FF 2890844535 2890844535 IN IP4 8.19.19.06
      s=-
      t=0 0
      c=IN IP4 8.19.19.06
      m=audio 5004 RTP/AVP 0 8         Two alternative codecs, PCM μ-Law or PCM A-Law
      a=rtpmap:0 PCMU/8000
      a=rtpmap:8 PCMA/8000


M2    IAM
      CdPN=202-555-1313, NPI=E.164,
      NOA=National                       Gateway maps telephone into called party number


M3    ACM


M4    SIP/2.0 183 Session Progress
      Via: SIP/2.0/UDP 8.19.19.06:5060;branch=z9hG4bK4545
      From: <sip:filo.farnsworth@television.tv>;tag=12
      To: <sip:+12025551313@gw.carrier.com;user=phone>;tag=37        Tag and
      Call-ID: 49235243082018498                                      brackets
      CSeq: 1 INVITE
      RSeq: 08071
      Contact: <sip:50.60.70.80>
      Content-Type: application/sdp
      Content-Length: 139

      v=0
      o=Port1723 2890844535 2890844535 IN IP4 50.60.70.80
      s=-
      t=0 0
```

```
     c=IN IP4 50.60.70.80
     m=audio 62002 RTP/AVP 0                      Gateway selects μ-Law codec
     a=rtpmap:0 PCMU/8000

M5   PRACK sip:50.60.70.80 SIP/2.0
     Via: SIP/2.0/UDP 8.19.19.06:5060;branch=z9hG4bK454
     Max-Forwards: 70
     From: <sip:filo.farnsworth@television.tv>;tag=37
     To: <sip:+12025551313@gw.carrier.com;user=phone>;tag=12
     Call-ID: 49235243082018498
     CSeq: 2 PRACK
     Contact: sip:filo.farnsworth@studio.television.tv
     RAck: 08071 1 INVITE
     Content-Length: 0

M6   SIP/2.0 200 OK
     Via: SIP/2.0/UDP 8.19.19.06:5060;branch=z9hG4bK454
     From: <sip:filo.farnsworth@television.tv>;tag=37
     To: <sip:+12025551313@gw.carrier.com;user=phone>;tag=12
     Call-ID: 49235243082018498
     CSeq: 2 PRACK

M7   ANM

M8   SIP/2.0 200 OK
     Via: SIP/2.0/UDP 8.19.19.06:5060;branch=z9hG4bK4545
     From: <sip:filo.farnsworth@television.tv>;tag=12
     To: <sip:+12025551313@gw.carrier.com;user=phone>;tag=37
     Call-ID: 49235243082018498
     CSeq: 1 INVITE
     Contact: <sip:50.60.70.80>
     Content-Type: application/sdp
     Content-Length: 139

     v=0
     o=Port1723 2890844535 2890844535 IN IP4 50.60.70.80
     s=-
     t=0 0
     c=IN IP4 50.60.70.80
     m=audio 62002 RTP/AVP 0
     a=rtpmap:0 PCMU/8000

M9   ACK sip:50.60.70.80 SIP/2.0
     Via: SIP/2.0/UDP 8.19.19.06:5060;branch=z9hG4bKfgrw
     Max-Forwards: 70
     From: <sip:filo.farnsworth@television.tv>;tag=12
     To: <sip:+12025551313@gw.carrier.com;user=phone>;tag=37
     Call-ID: 49235243082018498
     CSeq: 1 ACK

M10  BYE sip:50.60.70.80 SIP/2.0
     Via: SIP/2.0/UDP 8.19.19.06:5060;branch=z9hG4bK321
     Max-Forwards: 70
     From: <sip:filo.farnsworth@television.tv>;tag=12
     To: <sip:+12025551313@gw.carrier.com;user=phone>;tag=37
     Call-ID: 49235243082018498
     CSeq: 3 BYE                                           CSeq incremented
```

```
M11   REL CauseCode=16 Normal Clearing


M12   SIP/2.0 200 OK
      Via: SIP/2.0/UDP 8.19.19.06:5060;branch=z9hG4bK321
      From: <sip:filo.farnsworth@television.tv>;tag=12
      To: <sip:+12025551313@gw.carrier.com;user=phone>;tag=37
      Call-ID: 49235243082018498
      CSeq: 3 BYE


M13   RLC
```

16.4 PSTN to SIP Call Through a Gateway

Figure 16.4 shows a call originating from a telephone in the PSTN that terminates on a SIP phone in the Internet. The compact form of SIP is used throughout the example. Note that there is no compact form for `CSeq` or `Max-Forwards`.

```
M1    Setup
      CdPN=6512345, NPI=E.164
      NOA=International                          Dialed telephone number
      CgPN=4567890, NPI=E.164,
      NOA=International                             PSTN caller's number
```

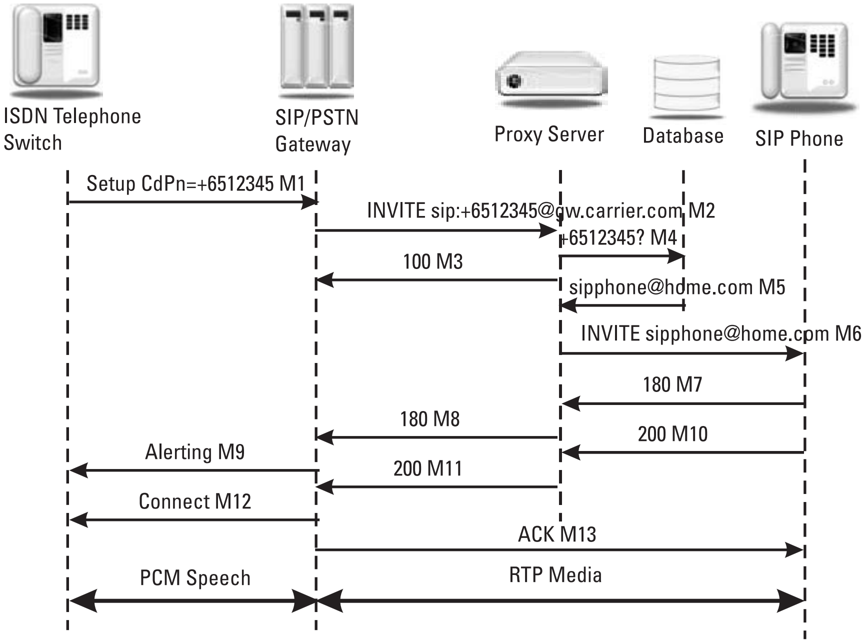


Figure 16.4 PSTN to SIP call flow through a gateway.

```
M2   INVITE sip:+6512345@incoming.com;user=phone SIP/2.0
     v: SIP/2.0/UDP 65.3.4.1:5060;branch=z9hG4bK343             Compact form of
     Max-Forwards: 70                                        headers includes tag
     f: <sip:+45.67890@incoming.com;user=phone>;tag=6a589b1
     t: <sip:+65.12345@incoming.com;user=phone>
     i: a3-65-99-1d
     CSeq: 1 INVITE
     m: 65.3.4.1
     c: application/sdp
     l: 126

     v=0
     o=- 2890844535 2890844535 IN IP4 65.3.4.1
     s=-
     t=0 0
     c=IN IP4 65.3.4.1
     m=audio 62432 RTP/AVP 0
     a=rtpmap:0 PCMU/8000

M3   SIP/2.0 100 Trying
     v: SIP/2.0/UDP 65.3.4.1:5060;branch=z9hG4bK343
     f: <sip:+45.67890@incoming.com;user=phone>;tag=6a589b1
     t: sip:+65.12345@incoming.com;user=phone
     i: a3-65-99-1d
     CSeq: 1 INVITE

M4   Service Query: +65-12345

M5   Location Service Response:
     sip:user@home.com                                    Number maps to SIP URI

M6   INVITE sip:user@home.com SIP/2.0
     v: SIP/2.0/UDP 176.5.8.2:5060;branch=z9hG4bK942834822.1
     v: SIP/2.0/UDP 65.3.4.1:5060;branch=z9hG4bK343
     Max-Forwards: 69
     f: <sip:+45.67890@incoming.com;user=phone>;tag=6a589b1
     t: sip:+65.12345@incoming.com;user=phone
     i: a3-65-99-1d
     CSeq: 1 INVITE
     m: 65.3.4.1
     c: application/sdp
     l: 126

     v=0
     o=- 2890844535 2890844535 IN IP4 65.3.4.1
     s=-
     t=0 0
     c=IN IP4 65.3.4.1
     m=audio 62432 RTP/AVP 0
     a=rtpmap:0 PCMU/8000

M7   SIP/2.0 180 Ringing
     v: SIP/2.0/UDP 176.5.8.2:5060;branch=z9hG4bK942834822.1
     v: SIP/2.0/UDP 65.3.4.1:5060;branch=z9hG4bK343
     f: <sip:+45.67890@incoming.com;user=phone>;tag=6a589b1
     t: <sip:+65.12345@incoming.com;user=phone>;tag=8657
     i: a3-65-99-1d
     m: sip:user@client.home.com
     CSeq: 1 INVITE
```

```
M8   SIP/2.0 180 Ringing
     v: SIP/2.0/UDP 65.3.4.1:5060;branch=z9hG4bK343
     f: <sip:+45.67890@incoming.com;user=phone>;tag=6a589b1
     t: <sip:+65.12345@incoming.com;user=phone>;tag=8657
     i: a3-65-99-1d
     CSeq: 1 INVITE

M9   Alerting

M10  SIP/2.0 200 OK
     v: SIP/2.0/UDP 176.5.8.2:5060;branch=z9hG4bK942834822.1
     v: SIP/2.0/UDP 65.3.4.1:5060;branch=z9hG4bK343
     f: <sip:+45.67890@incoming.com;user=phone>;tag=6a589b1
     t: <sip:+65.12345@incoming.com;user=phone>;tag=8657
     i: a3-65-99-1d
     CSeq: 1 INVITE
     m: sip:user@client.home.com
     c: application/sdp
     l: 125

     v=0
     o=- 2890844565 2890844565 IN IP4 7.8.9.10
     s=-
     t=0 0
     c=IN IP4 7.8.9.10
     m=audio 5004 RTP/AVP 0
     a=rtpmap:0 PCMU/8000

M11  SIP/2.0 200 OK
     v: SIP/2.0/UDP 65.3.4.1:5060;branch=z9hG4bK343
     f: <sip:+45.67890@incoming.com;user=phone>;tag=6a589b1
     t: <sip:+65.12345@incoming.com;user=phone>;tag=8657
     i: a3-65-99-1d
     CSeq: 1 INVITE
     M: sip:user@home.com
     c: application/sdp
     l: 125

     v=0
     o=- 2890844565 2890844565 IN IP4 7.8.9.10
     s=-
     t=0 0
     c=IN IP4 7.8.9.10
     m=audio 5004 RTP/AVP 0
     a=rtpmap:0 PCMU/8000

M12  Connect

M13  ACK sip:user@home.com SIP/2.0
     v: SIP/2.0/UDP 65.3.4.1:5060;branch=z9hG4bK453
     Max-Forwards: 70
     f: <sip+45.67890@incoming.com;user=phone>;tag=6a589b1
     t: <sip+65.12345@incoming.com;user=phone>;tag=8657
     i: a3-65-99-1d
     CSeq: 1 ACK
```

16.5 Parallel Search

In this example the caller receives multiple possible locations for the called party from a redirect server. Instead of trying the locations one at a time, the UA implements a parallel search for the called party by simultaneously sending the INVITE to three different locations, as shown in Figure 16.5. The SIP specification gives an example of this behavior in a proxy server, which is called a forking proxy.

In this example the first location responds with a 404 Not Found response. The second location responds with a 180 Ringing response, while the third location returns a 180 Ringing then a 200 OK response. The caller then sends an ACK to the third location to establish the call. Because one successful response has been received, a CANCEL is sent to the second location to terminate the search.

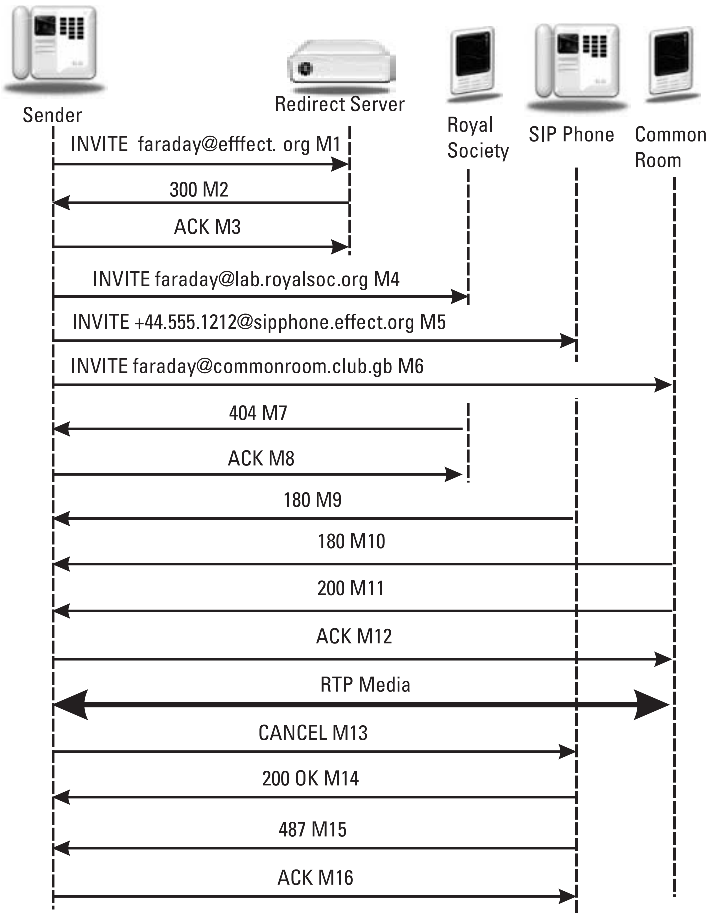


Figure 16.5 Parallel search example call flow.

The second location sends a `200 OK` to the `CANCEL` and a `487 Request Terminated` to the `INVITE`. This example shows some customized reason phrases in messages M7, M10, and M11. This example also shows a `Call-ID` with an IP address which is not recommended.

```
M1    INVITE sip:faraday@effect.org SIP/2.0
      Via: SIP/2.0/UDP kings.cambridge.edu.uk:60000;branch=z9hG4bK3    Port
       ;received=7.9.18.12                                       60000 is used
      Max-Forwards: 70
      From: J.C. Maxwell <sip:james.maxwell@kings.cambridge.edu.uk>;
       tag=4
      To: <sip:faraday@effect.org>
      Call-ID: mNjdwWjkBfWrd@7.9.18.12
      CSeq: 54 INVITE                                   CSeq initialized to 54
      Contact:<sip:james.maxwell@kings.cambridge.edu.uk>
      Content-Type: application/sdp
      Content-Length: 129

      v=0
      o=max 2890844521 2890844521 IN IP4 7.9.18.12
      s=-
      t=0 0
      c=IN IP4 7.9.18.12
      m=audio 32166 RTP/AVP 96
      a=rtpmap:96 iLBC/8000


M2    SIP/2.0 300 Multiple locations        Redirect server returns three locations
      Via: SIP/2.0/UDP kings.cambridge.edu.uk:60000;branch=z9hG4bK3
       ;received=7.9.18.12
      From: J.C. Maxwell <sip:james.maxwell@kings.cambridge.edu.uk>
       ;tag=4
      To:<sip:faraday@effect.org>;tag=1024
      Call-ID: mNjdwWjkBfWrd@7.9.18.12
      CSeq: 54 INVITE
      Contact: <sip:faraday@lab.royalsoc.gb>
      Contact: <sip:+44.555.1212@sip-phone.effect.org;user=phone>
      Contact: <sip:michael.faraday@commonroom.club.gb>


M3    ACK sip:faraday@effect.org
      Via: SIP/2.0/UDP kings.cambridge.edu.uk:60000;branch=z9hG4bK3
      Max-Forwards: 70
      From: J.C. Maxwell <sip:james.maxwell@kings.cambridge.edu.uk>
       ;tag=4
      To: <sip:faraday@effect.org>;tag=1024
      Call-ID: mNjdwWjkBfWrd@7.9.18.12
      CSeq: 54 INVITE


M4    INVITE sip:faraday@lab.royalsoc.gb SIP/2.0
      Via: SIP/2.0/UDP kings.cambridge.edu.uk:60000;branch=z9hG4bK1
      Max-Forwards: 70
      From: J.C. Maxwell <sip:james.maxwell@kings.cambridge.edu.uk>
       ;tag=5
      To: <sip:faraday@effect.org>                           Tag is not copied
      Call-ID: mNjdwWjkBfWrd@7.9.18.12                       Call-ID unchanged
      CSeq: 55 INVITE                                        CSeq incremented
      Contact: <sip:james.maxwell@kings.cambridge.edu.uk>
      Content-Type: application/sdp
```

```
     Content-Length: 129

     v=0
     o=max 2890844521 2890844521 IN IP4 7.9.18.12
     s=-
     t=0 0
     c=IN IP4 7.9.18.12
     m=audio 32166 RTP/AVP 96
     a=rtpmap:96 iLBC/8000

M5   INVITE sip:+44.555.1212@sip-phone.effect.org;user=phone SIP/2.0
     Via: SIP/2.0/UDP kings.cambridge.edu.uk:60000;branch=z9hG4bK2
     Max-Forwards: 70
     From: J.C. Maxwell <sip:james.maxwell@kings.cambridge.edu.uk>
      ;tag=5
     To: <sip:faraday@effect.org>
     Call-ID: mNjdwWjkBfWrd@7.9.18.12
     CSeq: 55 INVITE
     Contact: <sip:james.maxwell@kings.cambridge.edu.uk>
     Content-Type: application/sdp
     Content-Length: 129

     v=0
     o=max 2890844521 2890844521 IN IP4 7.9.18.12
     s=-
     t=0 0
     c=IN IP4 7.9.18.12
     m=audio 32166 RTP/AVP 96
     a=rtpmap:96 iLBC/8000

M6   INVITE sip:faraday@commonroom.club.gb SIP/2.0
     Via: SIP/2.0/UDP kings.cambridge.edu.uk:60000;branch=z9hG4bK3
     Max-Forwards: 70
     From: J.C. Maxwell <sip:james.maxwell@kings.cambridge.edu.uk>
      ;tag=5
     To: <sip:faraday@effect.org>
     Call-ID: mNjdwWjkBfWrd@7.9.18.12
     CSeq: 55 INVITE
     Contact: <sip:james.maxwell@kings.cambridge.edu.uk>
     Content-Type: application/sdp
     Content-Length: 129

     v=0
     o=max 2890844521 2890844521 IN IP4 7.9.18.12
     s=-
     t=0 0
     c=IN IP4 7.9.18.12
     m=audio 32166 RTP/AVP 96
     a=rtpmap:96 iLBC/8000

M7   SIP/2.0 404 The member you have requested is not available
     Via: SIP/2.0/UDP kings.cambridge.edu.uk:60000;branch=z9hG4bK1
      ;received=7.9.18.12
     From: J.C. Maxwell <sip:james.maxwell@kings.cambridge.edu.uk>
      ;tag=5
     To: <sip:faraday@effect.org>;tag=f6
     Call-ID: mNjdwWjkBfWrd@7.9.18.12
     CSeq: 55 INVITE
```

```
M8    ACK sip:faraday@lab.royalsoc.gb SIP/2.0
      Via: SIP/2.0/UDP kings.cambridge.edu.uk:60000z9hG4bK1
      Max-Forwards: 70
      From: J.C. Maxwell <sip:james.maxwell@kings.cambridge.edu.uk>
       ;tag=5
      To: <sip:faraday@effect.org>;tag=f6
      Call-ID: mNjdwWjkBfWrd@7.9.18.12
      CSeq: 55 ACK

M9    SIP/2.0 180 Ringing
      Via: SIP/2.0/UDP kings.cambridge.edu.uk:60000;branch=z9hG4bK2
       ;received=7.9.18.12
      From: J.C. Maxwell <sip:james.maxwell@kings.cambridge.edu.uk>
       ;tag=5
      To: <sip:faraday@effect.org>;tag=6321
      Call-ID: mNjdwWjkBfWrd@7.9.18.12
      Contact: <sip:+44.555.1212@sip-phone.effect.org>
      CSeq: 55 INVITE

M10   SIP/2.0 180 Please wait while we locate Mr. Faraday
      Via: SIP/2.0/UDP kings.cambridge.edu.uk:60000;branch=z9hG4bK3
       ;received=7.9.18.12
      From: J.C. Maxwell <sip:james.maxwell@kings.cambridge.edu.uk>
       ;tag=5
      To: <sip:faraday@effect.org>;tag=531
      Call-ID: mNjdwWjkBfWrd@7.9.18.12
      Contact: <sip:faraday@commonroom.club.gb>
      CSeq: 55 INVITE

M11   SIP/2.0 200 Mr. Faraday at your service?
      Via: SIP/2.0/UDP kings.cambridge.edu.uk:60000;branch=z9hG4bK3
       ;received=7.9.18.12
      From: J.C. Maxwell <sip:james.maxwell@kings.cambridge.edu.uk>
       ;tag=5
      To: <sip:faraday@effect.org>;tag=531
      Call-ID: mNjdwWjkBfWrd@7.9.18.12
      CSeq: 55 INVITE
      User-Agent: PDV/v4
      Contact: <sip:faraday@commonroom.club.gb>
      Content-Type: application/sdp
      Content-Length: 131

      v=0
      o=max 2890844521 2890844521 IN IP4 6.22.17.89
      t=0 0
      c=IN IP4 6.22.17.89
      m=audio 43782 RTP/AVP 4
      a=rtpmap:4 DVI/8000

M12   ACK sip:faraday@commonroom.club.gb;user=ip SIP/2.0
      Via: SIP/2.0/UDP kings.cambridge.edu.uk:60000;branch=z9hG4bK3
      Max-Forwards: 70
      From: J.C. Maxwell <sip:james.maxwell@kings.cambridge.edu.uk>
       ;tag=5
      To: <sip:faraday@effect.org>;tag=531
      Call-ID: mNjdwWjkBfWrd@7.9.18.12
      CSeq: 55 ACK

M13   CANCEL sip:+44.555.1212@sip-phone.effect.org;user=phone SIP/2.0
```

```
     Via: SIP/2.0/UDP kings.cambridge.edu.uk:60000;branch=z9hG4bK2    Cancels search
     Max-Forwards: 70
     From: J.C. Maxwell <sip:james.maxwell@kings.cambridge.edu.uk>
     To: <sip:faraday@effect.org>;tag=6321
     Call-ID: mNjdwWjkBfWrd@7.9.18.12
     CSeq: 55 CANCEL                                           CSeq not incremented
                                                               Method set to CANCEL

M14  SIP/2.0 200 OK                                            CANCEL acknowledged
     Via: SIP/2.0/UDP kings.cambridge.edu.uk:60000;branch=z9hG4bK2
      ;received=7.9.18.12
     From: J.C. Maxwell <sip:james.maxwell@kings.cambridge.edu.uk>
      ;tag=5
     To: <sip:faraday@effect.org>;tag=6321
     Call-ID: mNjdwWjkBfWrd@7.9.18.12
     CSeq: 55 CANCEL

M15  SIP/2.0 487 Request Terminated                       Final response to INVITE
     Via: SIP/2.0/UDP kings.cambridge.edu.uk:60000;branch=z9hG4bK2
      ;received=7.9.18.12
     From: J.C. Maxwell <sip:james.maxwell@kings.cambridge.edu.uk>
      ;tag=5
     To: <sip:faraday@effect.org>;tag=6321
     Call-ID: mNjdwWjkBfWrd@7.9.18.12
     CSeq: 55 INVITE

M16  ACK SIP/2.0
     Via: SIP/2.0/UDP kings.cambridge.edu.uk:60000;branch=z9hG4bK2
     From: J.C. Maxwell <sip:james.maxwell@kings.cambridge.edu.uk>
      ;tag=5
     To: <sip:faraday@effect.org>;tag=6321
     Call-ID: mNjdwWjkBfWrd@7.9.18.12
     CSeq: 55 ACK
```

16.6 Call Setup with Two Proxies

This section contains the complete message flow shown in Figure 2.2.

```
M1   INVITE sip:werner.heisenberg@munich.de SIP/2.0
     Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bKmp17a
     Max-Forwards: 70
     To: Heisenberg <sip:werner.heisenberg@munich.de>
     From: E. Schroedinger <sip:schroed5244@aol.com>;tag=42
     Call-ID: 4827311-391-32934
     CSeq: 1 INVITE
     Subject: Where are you exactly?
     Contact: <sip:schroed5244@pc33.aol.com>
     Content-Type: application/sdp
     Content-Length: 159

     v=0
     o=schroed5244 2890844526 2890844526 IN IP4 100.101.102.103
     s=Phone Call
     t=0 0
     c=IN IP4 100.101.102.103
```

```
      m=audio 49170 RTP/AVP 0
      a=rtpmap:0 PCMU/8000

M2    INVITE sip:werner.heisenberg@200.201.202.203 SIP/2.0
      Via: SIP/2.0/UDP proxy.munich.de:5060;branch=z9hG4bK83842.1
      Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bKmp17a
      Max-Forwards: 69
      To: Heisenberg <sip:werner.heisenberg@munich.de>
      From: E. Schroedinger <sip:schroed5244@aol.com>;tag=42
      Call-ID: 4827311-391-32934
      CSeq: 1 INVITE
      Contact: <sip:schroed5244@pc33.aol.com>
      Content-Type: application/sdp
      Content-Length: 159

      v=0
      o=schroed5244 2890844526 2890844526 IN IP4 100.101.102.103
      s=Phone Call
      c=IN IP4 100.101.102.103
      t=0 0
      m=audio 49172 RTP/AVP 0
      a=rtpmap:0 PCMU/8000

M3    SIP/2.0 180 Ringing
      Via: SIP/2.0/UDP proxy.munich.de:5060;branch=z9hG4bK83842.1
       ;received=100.101.102.105
      Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bKmp17a
      To: Heisenberg <sip:werner.heisenberg@munich.de>;tag=314159
      From: E. Schroedinger <sip:schroed5244@aol.com>;tag=42
      Call-ID: 4827311-391-32934
      CSeq: 1 INVITE
      Contact: <sip:werner.heisenberg@200.201.202.203>
      Content-Length: 0

M4    SIP/2.0 180 Ringing
      Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bKmp17a
      To: Heisenberg <sip:werner.heisenberg@munich.de>;tag=314159
      From: E. Schroedinger <sip:schroed5244@aol.com>;tag=42
      Call-ID: 4827311-391-32934
      CSeq: 1 INVITE
      Contact: <sip:werner.heisenberg@200.201.202.203>
      Content-Length: 0

M5    SIP/2.0 200 OK
      Via: SIP/2.0/UDP proxy.munich.de:5060;branch=z9hG4bK83842.1
       ;received=100.101.102.105
      Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bKmp17a
      To: Heisenberg7 <sip:werner.heisenberg@munich.de>;tag=314159
      From: E. Schroedinger <sip:schroed5244@aol.com>;tag=42
      Call-ID: 4827311-391-32934
      CSeq: 1 INVITE
      Contact: <sip:werner.heisenberg@200.201.202.203>
      Content-Type: application/sdp
      Content-Length: 159

      v=0
      o=heisenberg 2890844526 2890844526 IN IP4 200.201.202.203
      s=Phone Call
      c=IN IP4 200.201.202.203
```

```
     t=0 0
     m=audio 49172 RTP/AVP 0
     a=rtpmap:0 PCMU/8000

M6   SIP/2.0 200 OK
     Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bKmp17a
     To: Heisenberg <sip:werner.heisenberg@munich.de>;tag=314159
     From: E. Schroedinger <sip:schroed5244@aol.com>;tag=42
     Call-ID: 4827311-391-32934
     CSeq: 1 INVITE
     Contact: sip:werner.heisenberg@200.201.202.203
     Content-Type: application/sdp
     Content-Length: 159

     v=0
     o=heisenberg 2890844526 2890844526 IN IP4 200.201.202.203
     c=IN IP4 200.201.202.203
     t=0 0
     m=audio 49170 RTP/AVP 0
     a=rtpmap:0 PCMU/8000

M7   ACK sip:werner.heisenberg@200.201.202.203 SIP/2.0
     Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bKka42
     Max-Forwards: 70
     To: Heisenberg <sip:werner.heisenberg@munich.de>;tag=314159
     From: E. Schroedinger <sip:schroed5244@aol.com>;tag=42
     Call-ID: 4827311-391-32934
     CSeq: 1 ACK
     Content-Length: 0

M8   BYE sip:schroed5244@pc33.aol.com SIP/2.0
     Via: SIP/2.0/UDP 200.201.202.203:5060;branch=z9hG4bK4332
     Max-Forwards: 70
     To: E. Schroedinger <sip:schroed5244@aol.com>;tag=42
     From: Heisenberg <sip:werner.heisenberg@munich.de>;tag=314159
     Call-ID: 4827311-391-32934
     CSeq: 2000 BYE
     Content-Length: 0

M9   SIP/2.0 200 OK
     Via: SIP/2.0/UDP 200.201.202.203:5060;branch=z9hG4bK4332
     To: E. Schroedinger <sip:schroed5244@aol.com>;tag=42
     From: Heisenberg <sip:werner.heisenberg@munich.de>;tag=314159
     Call-ID: 4827311-391-32934
     CSeq: 2000 BYE
     Content-Length: 0
```

16.7 SIP Presence and Instant Message Example

This section contains the call flow details of Figure 2.4.

```
M1   SUBSCRIBE sip:poisson@probability.org SIP/2.0
     Via SIP/2.0/TCP lecturehall21.academy.ru:5060
      ;branch=z9hG4bK348471123
     Max-Forwards: 70
     To: M. Poisson <sip:poisson@probability.org>
```

```
      From: P. L. Chebychev <sip:chebychev@academy.ru>;tag=21171
      Call-ID: 58dkfj34924lk34452k592520
      CSeq: 3412 SUBSCRIBE
      Allow-Events: presence
      Allow: ACK, INVITE, CANCEL, BYE, NOTIFY, SUBSCRIBE, MESSAGE
      Contact: <sip:pafnuty@lecturehall21.academy.ru;transport=tcp>
      Event: presence
      Content-Length: 0

M2    SIP/2.0 200 OK
      Via SIP/2.0/TCP lecturehall21.academy.ru:5060
       ;branch=z9hG4bK348471123;received=19.34.3.1
      To: M. Poisson <sip:poisson@probability.org>;tag=25140
      From: P. L. Chebychev <sip:chebychev@academy.ru>;tag=21171
      Call-ID: 58dkfj34924lk34452k592520
      CSeq: 3412 SUBSCRIBE
      Allow-Events: presence
      Allow: ACK, INVITE, CANCEL, BYE, NOTIFY, SUBSCRIBE, MESSAGE
      Contact: <sip:s.possion@dist.probability.org;transport=tcp>
      Event: presence
      Expires: 3600
      Content-Length: 0

M3    NOTIFY sip:pafnuty@lecturehall21.academy.ru SIP/2.0
      Via SIP/2.0/TCP dist.probablilty.org:5060
       ;branch=z9hG4bK4321
      Max-Forwards: 70
      To: P. L. Chebychev <sip:chebychev@academy.ru>;tag=21171
      From: M. Poisson <sip:poisson@probability.org>;tag=25140
      Call-ID: 58dkfj34924lk34452k592520
      CSeq: 1026 NOTIFY
      Allow: ACK, INVITE, CANCEL, BYE, NOTIFY, SUBSCRIBE, MESSAGE
      Allow-Events: dialog
      Contact: <sip:s.possion@dist.probability.org;transport=tcp>
      Subscription-State: active;expires=3600
      Event: presence
      Content-Type: application/pidf+xml
      Content-Length: 244

      <?xml version=”1.0” encoding=”UTF-8”?>
      <presence xmlns=”urn:ietf:params:xml:ns:pidf”
       entity=”sip:poisson@probability.org”>
       <tuple id=”452426775”>
       <status>
       <basic>closed</basic>
       </status>
       </tuple>
      </presence>

M4    SIP/2.0 200 OK
      Via SIP/2.0/TCP dist.probablilty.org:5060
       ;branch=z9hG4bK4321;received=24.32.1.3
      To: P. L. Chebychev <sip:chebychev@academy.ru>;tag=21171
      From: M. Poisson <sip:poisson@probability.org>;tag=25140
      Call-ID: 58dkfj34924lk34452k592520
      CSeq: 1026 NOTIFY
      Content-Length: 0
```

```
M5   NOTIFY sip:pafnuty@lecturehall21.academy.ru SIP/2.0
     Via SIP/2.0/TCP dist.probablilty.org:5060
      ;branch=z9hG4bK334241
     Max-Forwards: 70
     To: P. L. Chebychev <sip:chebychev@academy.ru>;tag=21171
     From: M. Poisson <sip:poisson@probability.org>;tag=25140
     Call-ID: 58dkfj34924lk34452k592520
     CSeq: 1027 NOTIFY
     Allow: ACK, INVITE, CANCEL, BYE, NOTIFY, SUBSCRIBE, MESSAGE
     Allow-Events: presence
     Contact: <sip:s.possion@dist.probability.org;transport=tcp>
     Subscription-State: active;expires=1800
     Event: presence
     Content-Type: application/pidf+xml
     Content-Length: 325

     <?xml version=”1.0” encoding=”UTF-8”?>
     <presence xmlns=”urn:ietf:params:xml:ns:pidf”
      entity=”sip:poisson@probability.org”>
      <tuple id=”452426775”>
      <status>
      <basic>open</basic>
      </status>
      <contact>sip:s.possion@dist.probability.org;transport=tcp
      </contact>
      </tuple>
     </presence>


M6   SIP/2.0 200 OK
     Via SIP/2.0/TCP dist.probablilty.org:5060
      ;branch=z9hG4bK334241;received=24.32.1.3
     To: P. L. Chebychev <sip:chebychev@academy.ru>;tag=21171
     From: M. Poisson <sip:poisson@probability.org>;tag=25140
     Call-ID: 58dkfj34924lk34452k592520
     CSeq: 1027 NOTIFY
     Content-Length: 0


M7   MESSAGE sip:s.possion@dist.probability.org SIP/2.0
     Via SIP/2.0/TCP lecturehall21.academy.ru:5060
      ;branch=z9hG4bK3gtr2
     Max-Forwards: 70
     To: M. Poisson <sip:s.possion@dist.probability.org>
     From: P. L. Chebychev <sip:chebychev@academy.ru>;tag=4542
     Call-ID: 9dkei93vjq1ei3
     CSeq: 15 MESSAGE
     Allow: ACK, INVITE, CANCEL, BYE, NOTIFY, SUBSCRIBE, MESSAGE
     Content-Type: text/plain
     Content-Length: 9

     Hi There!


M8   SIP/2.0 200 OK
     Via SIP/2.0/TCP lecturehall21.academy.ru:5060
      ;branch=z9hG4bK3gtr2;received=19.34.3.1
     To: M. Poisson <sip:s.possion@dist.probability.org>;tag=2321
     From: P. L. Chebychev <sip:chebychev@academy.ru>;tag=4542
     Call-ID: 9dkei93vjq1ei3
     CSeq: 15 MESSAGE
     Content-Length: 0
```

```
M9    MESSAGE sip:chebychev@academy.ru SIP/2.0
      Via SIP/2.0/TCP dist.probablilty.org:5060
       ;branch=z9hG4bK4526245
      Max-Forwards: 70
      To: P. L. Chebychev <sip:chebychev@academy.ru>
      From: M. Poisson <sip:s.possion@dist.probability.org>;tag=14083
      Call-ID: lk34452k592520
      CSeq: 2321 MESSAGE
      Allow: ACK, INVITE, CANCEL, BYE, NOTIFY, SUBSCRIBE, MESSAGE
      Content-Type: text/plain
      Content-Length: 30

      Well, hello there to you, too!
```

```
M10   SIP/2.0 200 OK
      Via SIP/2.0/TCP dist.probablilty.org:5060
       ;branch=z9hG4bK4526245;received=24.32.1.3
      To: P. L. Chebychev <sip:chebychev@academy.ru>;tag=mc3bg5q77wms
      From: M. Poisson <sip:s.possion@dist.probability.org>;tag=14083
      Call-ID: lk34452k592520
      CSeq: 2321 MESSAGE
      Content-Length: 0
```

References

[1] Rosenberg, J., H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, “SIP: Session Initiation Protocol,” RFC 3261, June 2002.

[2] Johnston, A., et al., “Session Initiation Protocol (SIP) Basic Call Flow Examples,” BCP 75, RFC 3665, December 2003.

[3] Johnston, A., et al., “Session Initiation Protocol (SIP) Public Switched Telephone Network (PSTN) Call Flows,” BCP 76, RFC 3666, December 2003.

[4] Johnston, A., et al., “Session Initiation Protocol Service Examples,” BCP 144, RFC 5359, October 2008.

17

Future Directions

SIP is still an evolving protocol. New extensions and applications are being developed every day. Also, the SIP ecosystem continues to grow within the service provider and vendor communities. This chapter will discuss some future areas of work in SIP-related working groups in the IETF. Instead of attempting to list and discuss a snapshot of current activity in the IETF, the reader should gather the information directly from the IETF. Table 17.1 lists the most important SIP-related working groups. Note that the closed SIP [1] and SIPPING [2] working groups are not listed as they have been replaced by the SIPCORE [3] and DISPATCH [4] working groups, respectively. The charter page for each working group (given in References) lists the deliverables of the group along with RFCs (finished documents) and Internet Drafts (works in progress). Only Internet Drafts that have been adopted as official work group items are listed on these Web pages—these are the documents most likely to become RFCs in the near future. The Web page also contains information about joining the working group e-mail list, which discusses the listed set of Internet Drafts. Finally, one can search the IETF Internet Draft archives for documents relating to SIP at http://www.ietf.org. However, be warned: There are many, many documents and most will likely never be published as an RFC—always consult someone familiar with the working group activity before assuming that an Internet Draft not listed on a working group charter page is likely to ever become an Internet standard.

The following sections will discuss some active topics of standardization and development in SIP-related working groups including: bug fixes and clarification of RFC 3261, additional extensions to SIP, more work on SIP identity, interdomain SIP, emergency calling, P2P and HIP, security, and better feature interoperability.

Table 17.1
SIP-Related IETF Working Groups

Working Group		Area
SIPCORE	Session Initiation Protocol Core	Maintenance and development of core SIP
DISPATCH	Dispatch	Examine proposals for new SIP work
SIMPLE	SIP for Instant Messaging and Presence Leveraging Extensions	Presence and IM extensions for SIP
BLISS	Basic Level of Interoperability for SIP Services	Feature interoperability for SIP
ECRIT	Emergency Context Resolution with Internet Technologies	Emergency calling
XCON	Centralized Conferencing	Conferencing
BEHAVE	Behavior Engineering for Hindrance Avoidance	NAT behaviors and traversal protocols
P2PSIP	Peer-to-Peer SIP	Peer protocol and SIP usage of P2P overlays
ENUM	E.164 Telephone Number Mapping	Telephone number to URI resolution
SPEERMINT	Session Peering for Multimedia Interconnect	SIP Peering best current practices
DRINKS	Data for Reachability of Inter/tra-Network SIP	Provisioning for SIP interconnection
MMUSIC	Multiparty Multimedia Session Control	Session description protocol and extensions

17.1 Bug Fixes and Clarifications

In the new SIPCORE working group in the IETF, there will be continuing discussion about bug fixes and clarifications to the base specification RFC 3261. There is a database of bugs and issues with the specification [5]. Some of these will find their way into Internet Drafts that will eventually update RFC 3261. In addition there is discussion about progressing SIP from proposed standard to draft standard in the IETF standards ladder. This would require extensive documentation of existing interoperability and also a major rewrite of the base specification. However, work towards this goal is underway. For the status on this and other changes to the core SIP protocol, visit the IETF SIPCORE working group's charter page [3].

17.2 More Extensions

There continues to be a steady stream of SIP extensions being proposed to the IETF. Some extensions add new functionality and features to the protocol, reflecting the continued deployment of the protocol. For example, the use of SIP in contact centers (call centers) will require more SIP extensions to be developed. Fixes and clarifications to the core SIP protocol are done in SIPCORE, while the DISPATCH working group is used to help plan future SIP extension working groups.

17.3 Better Identity

Identity continues to be an active area of discussion. Most deployed systems use `P-Asserted-Identity`, which provides only the same level of identity assurance as the PSTN. Enhanced SIP identity [6] provides a better identity assurance, but has been very slow to deploy. One possible reason is the integrity protection of the SDP message body used to negotiate the media session. Many service providers and intermediaries modify the SDP information for "media steering" (NAT traversal) or media quality monitoring. Some proposals to allow these intermediaries to perform this function without compromising security have been made. In addition, enhanced SIP identity is mainly only useful for e-mail style identities—when telephone numbers (E.164 numbers) are used, the properties are greatly reduced. This future work might be done in a working group organized in DISPATCH.

17.4 Interdomain SIP

The usage of SIP between multiple service providers, between multiple enterprises, and between a service provider and a domain is on the rise. A number of IETF working groups are tackling various aspects of the problem space. For example, E.164 Telephone Number Mapping (ENUM) [7] is working on the resolution of telephone numbers to URIs for lookup and routing. The Session Peering for Multimedia Interconnect (SPEERMINT) [8] working group is working on policies and best current practices for SIP peering. The Data for Reachability of Inter/tra-Network SIP (DRINKS) [9] working group is working on provisioning issues relating to SIP peering.

17.5 Making Features Work Better

Many telephony features have been defined for SIP including those discussed in Chapter 9. However, due to the flexibility and "building block" nature of SIP, some features can be implemented in more than one way. As a result, features in clients and servers can be implemented using different SIP standards, but will not be able to interoperate. The Basic Level of Interoperability of SIP Services (BLISS) working group [10] has been formed by the IETF to address this issue. The approach of the working group is described in the problem statement draft [11]. In addition, BLISS is filling in the gaps for feature provisioning and activation.

17.6 Emergency Calling

Emergency calling is a very important application for Internet communications and SIP. The Emergency Context Resolution with Internet Technologies (ECRIT) working group [12] has been working on various aspects of this problem, along with location-based solutions from the Geographic Location (GEOPRIV) [13] working group. Eventually, public service answering points (PSAPs) will be enabled to receive VoIP and multimedia calls.

17.7 More SIP Trunking

Today's SIP trunking seeks to replicate PSTN trunking by providing voice-only services and features. As of the writing of this book, SIPconnect version 1.1 [14] is currently under development by the SIP Forum. This work should be completed in 2009. Work is expected to begin on a SIPconnect 2.0 recommendation which will likely include multimedia and presence capabilities.

17.8 P2P and HIP

As both P2PSIP and Host Identity Protocol (HIP) are developed, they are likely to be implemented together in order to utilize the benefits of both. The use of P2P approaches will be driven by market forces and competition, and the desire for highest reliability and scalability.

17.9 Improved NAT Traversal

With the finalization of hole punching protocols such as ICE [15] and a better understanding of NAT behavior and operation, NAT traversal for SIP should improve dramatically in the future. SIP extensions such as outbound [16] should also improve the reliability of UA connections through NATs.

17.10 Security Deployment

Many current SIP deployments use minimal security or no media security at all. As deployable methods to securely key SRTP are used, hopefully secure media will become the norm. Making secure media deployable in an incremental way means utilizing best effort encryption, which was discussed in Chapter 14. Best effort encryption can be done using either ZRTP [17] or SDP capability negotiation [18].

17.11 Better Interoperability

SIP needs to continually improve its interoperability. Some kind of certification might even be implemented by the SIP Forum in the future. Currently, regularly attending SIPit SIP interoperability test events [19] is still the best way to ensure your product or service will interoperate with others.

References

[1] http://www.ietf.org/html.charters/OLD/sip-charter.html.

[2] http://www.ietf.org/html.charters/OLD/sipping-charter.html.

[3] http://www.ietf.org/html.charters/sipcore-charter.html.

[4] http://www.ietf.org/html.charters/dispatch-charter.html.

[5] http://bugs.sipit.net/.

[6] Peterson, J., and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)," RFC 4474, August 2006.

[7] http://www.ietf.org/html.charters/enum-charter.html.

[8] http://www.ietf.org/html.charters/speermint-charter.html.

[9] http://www.ietf.org/html.charters/drinks-charter.html.

[10] http://www.ietf.org/html.charters/bliss-charter.html.

[11] Rosenberg, J., "Basic Level of Interoperability for Session Initiation Protocol (SIP) Services (BLISS) Problem Statement," draft-ietf-bliss-problem-statement-04 (work in progress), March 2009.

[12] http://www.ietf.org/html.charters/ecrit-charter.html

[13] http://www.ietf.org/html.charters/geopriv-charter.html

[14] http://www.sipforum.org/sipconnect

[15] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols," draft-ietf-mmusic-ice-19 (work in progress), October 2007.

[16] Jennings, C., and R. Mahy, "Managing Client Initiated Connections in the Session Initiation Protocol (SIP)," draft-ietf-sip-outbound-16 (work in progress), October 2008.

[17] Zimmermann, P., A. Johnston, and J. Callas, "ZRTP: Media Path Key Agreement for Secure RTP," draft-zimmermann-avt-zrtp-15 (work in progress), March 2009.

[18] Gilman, R., R. Even, and F. Andreasen, "SDP Media Capabilities Negotiation," draft-ietf-mmusic-sdp-media-capabilities-07 (work in progress), February 2009.

[19] http://www.sipit.net/.

Appendix
Introduction to ABNF and XML

Augmented Backus-Naur Form (ABNF) and Extensible Markup Language (XML) are used to represent nearly all the protocols discussed in this book. However, syntax is explained in this book by example rather than using ABNF rules or XML schemas or DTDs. This appendix provides an introduction to help when reading the actual RFC specifications.

A.1 ABNF Rules

Augmented Backus-Naur Form is a computer science metasyntax used to define many Internet protocols including SIP. It defines how text messages are parsed, and was initially defined in RFC 822; the latest version is defined in RFC 5234 [1]. ABNF uses a 7-bit ASCII character set and defines rules for matching character strings.

For example:

```
Message = Request / Response
```

Defines a rule named `Message` in terms of two other rules `Request` and `Response`. The "`/`" indicates an alternative, meaning that a message can be either a request or a response. A basic rule in ABNF has the form:

```
Name = elements; Comment CRLF
```

where `Name` is the name of the rule, and the `elements` follow after the equal sign. `Comment`s begin after a semicolon (`;`) and continue until the end of the line, terminated with a carriage return line feed (*CRLF*). Elements can be simple strings. For example:

```
Element = "test"
Element2 = "Test"
Element3 = "TEST"
```

Rules defined using literal strings are actually case-insensitive. As a result `Element`, `Element2`, and `Element3` are all equivalent. Terminals, or individual ASCII characters, in ABNF can be expressed using a percent sign and are often encoded in hexadecimal. For example:

```
value1 = %x61 ; a
value2 = %x65 ; A
```

Rule `value1` matches the lowercase a, while `value2` matches uppercase A. Value ranges can be defined using a dash:

```
Digit = %x30-39 ; Digits "0" through "9"
```

Concatenation in ABNF is done by listing rules together:

```
Element5 = value1 value2 ; aA
```

Groupings in ABNF are enclosed by parenthesis `( )` and are treated as a single element. Optional rules are enclosed by square brackets `[]`. Rules can be invoked numerous times as shown in Table A.1.

The precedence rules of ABNF are given in Table A.2. For example:

`"a" / "b" "c"` matches "`a`" or "`bc`" but not "`ac`"
`("a" / "b") "c`" matches "`ac`" or "`bc`" but not "`a`"
`*("a" "b") "c"` matches "`ababc`" but not "`aaabc`"

Table A.3 has an example ABNF, which is a simplified version of the host rule in SIP.

Table A.1
ABNF Examples of Repetition

`2*3Rule`	`Rule` appears between two and three times.
`*4Rule`	`Rule` can appear up to four times.
`3Rule`	`Rule` must appear three times.
`*Rule`	`Rule` can appear any number of times including zero times.

Table A.2
ABNF Order of Precedence

Strings, names formation
Comment
Value range
Repetition
Grouping, optional
Concatenation
Alternative

Table A.3
ABNF Example for Host

```
host             = hostname / IPv4address / IPv6ref
hostname         = *( domainlabel “.” ) toplabel [ “.”
domainlabel      = alphanum/alphanum *( alphanum / “-” ) alphanum
toplabel         = ALPHA / ALPHA *( alphanum / “-” ) alphanum
IPv4address      = 1*3DIGIT “.” 1*3DIGIT “.” 1*3DIGIT “.” 1*3DIGIT
IPv6ref          = “[” IPv6address “]”
IPv6address      = hexpart [ “:”IPv4address]
hexpart          = hexseq / hexseq “::”[ hexseq ] / “::”[ hexseq ]
hexseq           = hex4 *( „:“ hex4)
hex4             = 1*4HEXDIG
alphanum         = ALPHA / DIGIT
ALPHA            = %x41-5A / %x61-7A ; A-Z / a-z
DIGIT            = %x30-39 ; 0-9
HEXDIG           = DIGIT / “A” / “B” / “C” / “D” / “E” / “F”
```

This rule allows a host to be either a domain name or an IPv4 address or IPv6 address. For example, `ese.wustl.edu` matches this rule but `ese..edu` does not. `Example.com3` matches this rule but `example.3com` does not (this is due to a rule in DNS that top level domain names may not begin with a digit and enforced by the rule `toplabel)`. For IPv4 addresses, any four sets of three digits separated by a “.” will match the rule `IPv4address`.

A.2 Introduction to XML

XML [2] is a simplification of the Standardized Generalized Markup Language (SGML). It is very similar to the Hypertext Markup Language (HTML) used to represent documents on the World Wide Web (WWW). While SIP does not use XML encoding, many bodies used with SIP do. XML is standardized by the World Wide Web Consortium (W3C). Elements in XML are known as *tags* or *elements* and are enclosed in <>. Here is an example tag, which contains a single value:

```
<tag>value</tag>
```

For every element opened in XML (`<tag>` in the above example), the tag must be closed (`</tag>`). An XML document is said to be *well-formed* if every opened tag is also closed. The value, which is enclosed by the open and closed tags, is the value associated with that element. In addition to values, elements can also have attributes inside the <>. For example:

```
<tag attribute=″another value″>value</tag>
```

is the same as the previous element but with the addition of the information in the attribute. Attribute values can be enclosed in either double quotes (“) or single quotes (‘). Elements can also be opened and closed at the same time:

```
<tag attribute=’information’ />
```

This element has no value but does have the single attribute, which is enclosed in single quotes. Elements can also be enclosed in other elements:

```
<address>
   <number>402</number>
   <street>Wildwood Ave</street>
</address>
```

In this example, the `<number>` and `<street>` elements are subelements inside the `<address>` element.

XML documents can be validated by another document, which indicates what elements, information, and attributes may be present. Two common methods of defining XML documents are XML schema and a document type definition (DTD). Both schema and DTD are XML documents. A complete XML document begins with an XML declaration, which indicates the current version of XML (1.0) and the encoding (commonly UTF-8):

```
<?xml version=″1.0″ encoding=″UTF-8″>
```

Table A.4 shows some common entity values in XML. Comments begin with `<!--` and end with `-->`. XML documents usually use the file extension .xml. XML is commonly used for encoding information in SIP message bodies. A key advantage of XML over, for example, ABNF, is that a general purpose XML parser can be used to parse and validate an XML document. XML documents

Table A.4
Entity Values in XML

Value	Meaning
&amp	ampersand
&lt;	less than
&gt;	greater than
&apos	apostrophe
&quot;	quotation mark
&#x20	escaped ASCII space

can also be used to write IETF Internet Drafts using the XML document format [3] and the XML2RFC tool [4].

Namespaces are an XML extension used to manage XML extensions while avoiding name collisions. For example, it is common for elements to define a namespace using the `xmlns` attribute. Additional namespace attributes can also be defined. If a particular namespace is not understood, it can be ignored by the XML parser. For example,

```
<presence xmlns=″urn:ietf:params:xml:ns:pidf″
 xmlns:dm=″urn:ietf:params:xml:ns:pidf:data-model″
 xmlns:cipid=″urn:ietf:params:xml:ns:pidf:cipid″
 xmlns:caps=″urn:ietf:params:xml:ns:pidf:caps″
   entity=″pres:someone@example.com″>
  <tuple id=″34g45sfde″>
   <status>
    <basic>open</basic>
   </status>
   <contact>sip:someone@pc29.example.com</contact>
   <caps:servcaps>
    <caps:audio>true</caps:audio>
    <caps:video>true</caps:video>
   </caps:servcaps>
  </tuple>
</presence>
```

This presence element has a default namespace, which is the IETF URN for PIDF, and three other namespaces of `dm`, `cipid`, and `caps`. The subelements `tuple`, `status`, and `contacts` are all defined in the default namespace while the elements `servcaps`, `audio`, and `video` are defined in the `caps` namespace.

Note that the line breaks and indentation tabs often shown with XML are optional, but are a good idea to help with the readability of XML. XML elements and values are case sensitive.

References

[1] Crocker, D., and P. Overell, “Augmented BNF for Syntax Specifications: ABNF,” STD 68, RFC 5234, January 2008.

[2] World Wide Web Consortium, "Extensible Markup Language (XML) 1.0," W3C XML, February 1998.

[3] Rose, M., "Writing I-Ds and RFCs Using XML," RFC 2629, June 1999.

[4] http://xml.resource.org.

About the Author

Alan B. Johnston is an engineer in Avaya's CTO office and has been involved with SIP and VoIP since the mid-1990s. While at MCI, he served as an architect of the first enterprise SIP VoIP product in the United States. He is also an adjunct professor of electrical systems engineering at Washington University in St. Louis. Dr. Johnston is a coauthor of the SIP protocol specification RFC 3261 and editor of the basic and PSTN call flows BCP (Best Current Practices) documents, RFC 3665 and RFC 3666, and several other RFCs. His recent areas of work include the SIP Service Examples, Peer-to-Peer (P2P) SIP, and security, where he has couthored the ZRTP protocol. He is also the cochair of the IETF (Internet Engineering Task Force) Centralized Conferencing Working Group, and is a former member of the board of directors of the international SIP Forum. Dr. Johnston has been awarded several U.S. and European patents in VoIP technology. He holds a bachelor's of engineering degree with honours from the University of Melbourne, Australia, and a Ph.D. in electrical engineering from Lehigh University, Bethlehem, Pennsylvania.

Index

Accept-Contact header field, 149–50
Accept-Encoding header field, 134–36
Accept header field, 134
Accept-Language header field, 136
Accept-Resource-Priority header field, 163–64
ACK method, 78–80
  defined, 78
  header fields, 80
  hop-by-hop, 79, 80
Acknowledgment of messages, 65–66
Ada, 344
Adaptive multirate (AMR) codec, 185
Address and port dependent mapping (ADPM) NAT, 241, 245
Address dependent mapping (ADM) NAT, 240–41, 245
Advanced Encryption Standard (AES), 308
Advanced Encryption Standard (AES) for encryption in counter mode (AES-CTR), 323
Alert-Info header field, 136
Allow-Events header field, 137
Allow header field, 137
American Registry for Internet Numbers (ARIN), 3–4
Answer-Mode header field, 137
Application layer, 9
Application layer gateways (ALGs), 246
Application sequencing, 229–30
Audio codecs, 282–83
Audio conferencing, 284
Audio video profile with feedback (AVPF), 285
Augmented Backus-Naur Form (ABNF), 11, 375–77
  basic rule, 375–76
  character set, 375
  defined, 375
  example for host, 377
  examples of repetition, 376
  groupings, 376
  order of precedence, 377
  percent sign, 376
Authentication
  certificates for, 311
  defined, 307
  digest, 314–16
  message, 309–10
  mutual, 316
  performance of, 307
  proxy, 315
  with TLS, 316–17
Authentication-Info header field, 164
Authorization header field, 150

Back-to-back user agents (B2BUAs), 53–54
  defined, 53
  as device part, 53–54
  uses, 53
  See also User agents (UAs)
Backus Naur Format (BNF), 11
Bandwidth management, 226
Basic Level of Interoperability of SIP Services (BLISS), 371
Best effort encryption, 325

Binary Floor Control Protocol (BFCP), 228, 296–97
Brute force attack, 308
Buffering, 274–75
Bug fixes, 370
BYE method, 78, 79
Bytes, 2

Call agents, 55
Call cancellation, race condition in, 82
Call flows, *xxiv,* 343–68
  call setup with two proxies, 363–65
  conferencing, 229
  examples, 343–68
  H.323, 269
  ICE, 258
  identity, 319
  MESSAGE method, 95
  parallel search, 359–63
  PRACK method, 98
  PSTN to SIP call through gateway, 356–58
  PUBLISH method, 89
  SIP call with authentication, proxies, and record-route, 343–49
  SIP call with stateless and stateful proxies with called party busy, 349–52
  SIP outbound, 252
  SIP presence and instant message, 365–68
  SIP to PSTN call through gateways, 352–56
  SUBSCRIBE method, 85
  voicemail, 225
Call-ID header field, 137–38
Call-Info header field, 150
Call Signaling Control Functions (CSCF), 185
Call state information, 35
CANCEL method, 81–82
  defined, 81
  header fields, 83
  in hop-to-hop request, 82
  See also Request messages
Centralized Conference Manipulation Protocol (CCMP), 228
Certificates, 310–11
  for authentication, 311
  defined, 310
  TLS with, 319
Certificate service, 319–22
Chord
  finger table, 334, 335
  routing, 334
Circuit associated signaling (CAS), 263
Client error responses, 116–28
  400 Bad Request, 116–17
  401 Unauthorized, 117
  402 Payment Required, 117
  403 Forbidden, 117
  404 Not Found, 118
  405 Method Not Allowed, 118
  406 Not Acceptable, 118
  407 Proxy Authentication Required, 118
  408 Request Timeout, 119
  409 Conflict, 119
  410 Gone, 119
  411 Length Required, 119
  412 Conditional Request Failed, 119–20
  413 Request Entity Too Large, 120
  414 Request-URI Too Long, 120
  415 Unsupported Media Type, 120
  416 Unsupported URI Scheme, 120
  417 Unknown Resource Priority, 120
  420 Bad Extension, 121
  421 Extension Required, 121
  422 Session Timer Interval Too Small, 121
  423 Interval Too Brief, 121
  428 Use Identity Header, 121
  429 Provide Referror Identity, 122
  430 Flow Failed, 122
  433 Anonymity Disallowed, 122
  436 Bad Identity-Info Header, 122
  437 Unsupported Certificate, 122
  438 Invalid Identity Header, 123
  439 First Hop Lacks Outbound Support, 123
  440 Max-Breadth Exceeded, 123
  470 Consent Needed, 123
  480 Temporarily Unavailable, 123
  481 Dialog/Transaction Does Not Exist, 123
  482 Loop Detection, 124
  483 Too Many Hops, 124
  484 Address Incomplete, 125
  485 Ambiguous, 125
  486 Busy Here, 126
  487 Request Terminated, 126

488 Not Acceptable Here, 126
489 Bad Event, 126
491 Request Pending, 126
493 Request Undecipherable, 127
494 Security Agreement Required, 127
See also Response messages
Coding, 274
Common Presence and Instant Message Presence Information Data Format (CPIM PIDF), 41, 42
Common Profile for Instant Messaging (CPIM), 205–6
header fields, 206–8
wrapper, 207
Compressed RTP (CRTP), 280
Compression, 280–81
Conditional notifications, 201–2
Conferencing, 227–29, 284–85
audio, 284
call control functions, 228
call flow, 229
defined, 227
focus, 227–28
mixer, 228
non-SIP control, 228
video, 284–85
See also Services
Confidentiality, 307–8
Connection reuse, 250–51
Contact header field, 35, 138–39
Content-Disposition header field, 169–70
Content-Encoding header field, 169
Content-Language header field, 170
Content-Length header field, 45, 170
Content-Type header field, 170–71
Credential theft, 312
Cryptography
defined, 308
Diffie-Hellman, 309
public key, 309
symmetric key, 309
CSeq header field, 140

Data for Reachability of Inter/tra-Network SIP (DRINKS), 371
Datagram Congestion Control Protocol (DCCP), 9, 237
Data/link layer, 2
Date header field, 141
Decoding, 275
Denial of service (Dos), 311
Depacketization, 274
Diffie-Hellman cryptography, 309
Digest authentication, 314–16
Digital certificates, 310–11
DISPATCH, 369
Distributed denial of service (DDOS), 311
DNS security (DNSSec), 313–14
Domain Name Service (DNS), *xxxiii,* 1, 13–17
address resource records, 15
defined, 13
lookups, 32, 33, 48
naming authority pointer resource records (NAPTR), 16
records, 13–14
resolvers, 16–17
resource records, 14–15
servers, 13
service resource records (SRV), 15–16
Draft archives, 369
Dual tone multifrequency (DTMF), 277, 285–86
encoding, 286
keypresses, 286
tones, 285, 286
Dynamic Host Configuration Protocol (DHCP), 4, 9
Dynamic payloads, 300

E.164 Telephone Number Mapping (ENUM), 371
Eavesdropping, 312
Emergency calling, 372
Emergency Context Resolution with Internet Technologies (ECRIT), 372
Encryption
best effort, 325
defined, 308
Encryption header field, 141
Endpoint independent mapping (EIM) NAT, 240
Enhanced identity, 318–19
Error-Info header field, 164
ETags, 202
Ethernet, 2
Event header field, 150–51
Events
framework, 191–92

Events (continued)
- notifications, conditional, 201–2
- packages, 8, 192

Expires header field, 75, 141

Facsimile, 226–27
Filtering, 200–201
- defined, 200
- example, 200–201
- NAT modes, 243
- notifications from, 201

Finger tables, 334
Flow-Timer header field, 165
Forking proxy server, 59, 60
From header field, 141–42
Future directions, 369–73

Gateways, 54–55
- application layer (ALGs), 246
- defined, 54
- illustrated, 55
- media, (MG), 55
- proxy servers versus, 57
- PSTN, 353
- PSTN to SIP call through, 356–58
- services, 219–20
- SIP to PSTN call through gateways, 352–56

Global error responses, 129–30
- 600 Busy Everywhere, 129
- 603 Decline, 129
- 604 Does Not Exist Anywhere, 130
- 606 Not Acceptable, 130
- *See also* Response messages

H.26x series, 284
H.245, 270
H.323, 266–71
- audio codecs, 268
- call flow example, 269
- call signaling messages in, 269
- call teardown sequence, 270
- defined, 266
- in deployed systems, 271
- example, 268–71
- introduction to, 266–68
- network elements, 267
- protocol references, 267, 268
- versions, 271

Hairpinning support, 241
Hash collisions, 310
Hashed message authentication code (HMAC), 310
Header fields, 133–71
- Accept, 134
- Accept-Contact, 149–50
- Accept-Encoding, 134–36
- Accept-Language, 136
- Accept-Resource-Priority, 163–64
- Alert-Info, 136
- Allow, 137
- Allow-Events, 137
- Answer-Mode, 137
- Authentication-Info, 164
- Authorization, 150
- Call-ID, 137–38
- Call-Info, 150
- compact forms, 134
- Contact, 35, 138–39
- Content-Disposition, 169–70
- Content-Encoding, 169
- Content-Language, 170
- Content-Length, 45, 170
- Content-Type, 170–71
- CSeq, 140
- Date, 141
- Encryption, 141
- Error-Info, 164
- Event, 150–51
- Expires, 75, 141
- Flow-Timer, 165
- From, 141–42
- Hide, 151
- History Info, 142
- Identity, 151
- Identity-Info, 151
- Info-Package, 152
- In-Reply-To, 151–52
- Join, 152–53
- Max-Breadth, 155
- Max-Forwards, 156
- message body, 169–71
- MIME-Version, 171
- Min-Expires, 165
- Min-SE, 165
- Organization, 143
- P-Asserted Identity, 155
- Path, 143
- Permission-Missing, 165
- P-OSP-Auth-Token, 155
- P-Preferred Identity, 155

Priority, 153
Privacy, 153
Priv-Answer-Mode, 143
Proxy-Authenticate, 166
Proxy-Authorization, 153–54
Proxy-Require, 154
RAck, 161
Reason, 156
Record-Route, 144
Recv-Info, 144
Referred-By, 94, 157
Referred-To, 94
Refer-Sub, 144–45
Refer-To, 156–57
Reject-Contact, 158–59
Replaces, 158
Reply-To, 157–58
request, 149–63
request and response, 134–49
Request-Disposition, 159
Require, 159–60
Resource-Priority, 160
response, 163–69
Response-Key, 160
Retry-After, 145
Route, 160–61, 184
RSeq, 168–69
rules, 133
Security-Client, 161
Security-Server, 166
Security-Verify, 162
Server, 166
Service-Route, 166–67
Session-Expires, 162
SIP-ETag, 167
SIP-If-Match, 162, 204
Subject, 145
Subscription-State, 162–63
Supported, 146
Suppress-If-Match, 163
Target-Dialog, 163
Timestamp, 147
To, 147
Trigger-Consent, 163
Unsupported, 167
User-Agent, 147–48
Via, 34, 148–49
Warning, 167–68
WWW-Authenticate, 168

Hide header field, 151
Hijacking, 312
History Info header field, 142
Hole punching, 253–57
architecture, 254
defined, 253
examples, 255–56, 257
failure, 257
process, 253
Host Identity Protocol (HIP), 237, 338–39
advantages, 339
defined, 338
future directions, 372
message exchange, 338
stack, 338
Hypertext Markup Language (HTML), 377
Hypertext Transport Protocol (HTTP), 20
digest, 315
use of, 20

Identity, 317–18
better, 371
call flow, 319
enhanced, 318–19
as issue, 317–18
solutions, 318
Identity header field, 151
Identity-Info header field, 151
I-frames, 283
INFO method, 96–97
base specification, 97
defined, 96
example, 96–97
header fields, 97
See also Request messages
Info-Package header field, 152
Informational responses, 112–14
100 Trying, 112–13
180 Ringing, 113
181 Call is Being Forward, 113
182 Call Queued, 113
183 Session Progress, 113–14
See also Response messages
In-Reply-To header field, 151–52
Instant messaging (IM)
architecture, 190
call flow example, 365–68
delivery notification, 206–8
elements, 190
history of, 189–91
page mode, 205

Instant messaging (continued)
  SIMPLE, 205–13
  SIMPLE specifications, 193
Integrated Services Digital Network (ISDN), 54, 264
Integrity protection, 308
Intelligent Multimedia Core Subsystem (IMS), 177
  elements, 184
  header fields, 186
  SIP and, 184–85
Interactive Communications Establishment (ICE), 247, 258–59
  benefits, 259
  call flow, 258
  defined, 258
  in IP transition, 259
Interdomain SIP, 371
International SIP Forum, 19
Internet
  audio codecs, 283
  multicast backbone (MBONE), 10, 273, 275, 289
  names, 11
Internet Architecture Board (IAB), 19
Internet Assigned Names Association (IANA), 4, 19
Internet Control Message Protocol (ICMP), defined, 10
Internet Engineering Steering Group (IESG), 19
Internet Engineering Task Force (IETF), *xxvi, xxvii,* 1, 18–20
  defined, 18
  IAB, 19
  MMUSIC, 20
  working groups, 370
Internet Group Management Protocol (IGM), 10
Internet message delivery notification (IMDN), 207–8
Internet multimedia protocol stack, 2–11
  application layer, 9
  data/link layer, 2
  multicast, 10–11
  network layer, 3–4
  physical layer, 2
  transport layer, 4–9
  utility applications, 9–10
Internet Protocol (IP), 1, 3
  mobility, 177–78
  packets, 4
  version 4 (IPv4), 3
  version 6 (IPv6), 3
  See also IP addresses
Internet Research Task Force (IRTF), 19
Internet service providers (ISPs), 236
Interoperability, 373
`INVITE` method, 73–76
  defined, 73
  example request, 75
  header fields, 75–76
  reliability example, 68
  triggered, 93–94
  without SDP offer, 74
IP addresses, 3–4, 239
  assignment, 3
  pooling options, 242
  private, 4
IPSec, 312–13, 323
IPTV, 11
ISDN User Part (ISUP), 17–18, 54, 264

Jabber, 213–14
  defined, 213
  ID (JID), 213
  interworking with SIMPLE, 214
Jingle, 214, 326
`Join` header field, 152–53

Man in the middle (MitM) attack, 312
Mapping
  address and port dependent (ADPM), 241
  address dependent (ADM), 240–41
  instant message, 214
  NAT, examples, 244–45
  presence, 215
  refresh, 242–43
  TCP, 243
Marconi's media information, 28–29
`Max-Breadth` header field, 155
`Max-Forwards` header field, 156
Media gateway controllers (MGC), 55, 265
Media gateway control protocols (MGCP), 265–66
Media gateways (MG), 55
Media security, 322–26
  best effort encryption, 325
  keying SRTP, 323–25
  non-RTP media, 322–23
  secure RTP (SRTP), 323

ZRTP, 326
See also Security
Media support, 273–86
Message bodies
header fields, 169–71
parts, 105
request messages, 105–6
Message composition indication, 208–9
Message digest 5 (MD5), 314
`MESSAGE` method, 94–96
call flow, 95
defined, 94
response, 95
See also Request messages
Messages
acknowledgment of, 65–66
authentication, 309–10
multiple recipient, 209–10
request, 73–107
response, 111–30
Message Session Relay Protocol (MSRP), 96, 210–13
defined, 210
error codes, 212
header fields, 211
requests, 213
session example, 212
sessions, 210, 211
Message transport, 43–47
SCTP, 46–47
TCP, 45–46
TLS, 46
UDP, 43–44
Midcall mobility, 181, 182
MIKEY (multimedia Internet keying), 324, 325
`MIME-Version` header field, 171
`Min-Expires` header field, 165
`Min-SE` header field, 165
Mobility
capabilities, 184
IP, 177–78
midcall, 181, 182
personal, 178–84
precall, 179
service, 183
Multicast, 10–11
support, 68–69
transport, 10–11
Multicast backbone (MBONE), 10, 273, 275
defined, 10
sessions, 289
Multihoming, 47
Multipart Internet Mail Extensions (MIME), 105
Multiple recipient messages, 209–10
Multipoint control units (MCU), 284
Mutual authentication, 316

Namespaces, 379
Naming authority pointer resource records
(`NAPTR`), 16
Network address and port translation (NAPT), 235–36, 238
Network Address Translation (NAT), *xxiv,* 235–59
address and port dependent mapping (ADPM), 241, 245
address dependent mapping (ADM), 240–41, 245
advantages, 236
behavior, 247
defined, 235
disadvantages, 237
endpoint independent mapping (EIM), 240
examples, 239, 244–45
filtering modes, 243
friendly, 247
function, 4
functioning of, 238–39
hairpinning support, 241
hole punching, 253–57
improved traversal, 372
introduction to, 235–36
IP addresses, 239
IP address pooling, 242, 246
mapping classifications, 240
mapping refresh, 242–43
media traversal solutions, 251–57
port assignment options, 242
protocol design guidelines, 246
SIP and, 245–47
SIP problems with, 249–51
traversal, 235
types of, 239–43
Network layer, 3–4
`NOTIFY` method, 87–88
defined, 87
example request, 87–88

NOTIFY method (continued)
- header fields, 88
- *See also* Request messages

Offer answer exchanges, 300–301
Offer answer model, 297–300
- call hold and, 299–300
- defined, 297–98
- rules for generating answer, 299
- rules for generating offer, 299
- rules for modifying sessions, 299
- *See also* Session Description Protocol (SDP)

Open Mobile Alliance (OMA), 186
Open Settlement Protocol (OSP), 155
OPTIONS method, 82–84
- defined, 82–83
- header fields, 84
- request generation, 83

Organization, this book, xxiii–xxiv
Organization header field, 143

Packetization, 274
Page mode instant messaging, 205
Parallel search, 359–63
Partial publication, 202–4
P-Asserted Identity header field, 155
Path header field, 143
Payload type (PT), 300
Peer-to-peer (P2P), *xxiv,* 331–40
- future directions, 372
- overlays, 333–36
- properties, 331
- properties of SIP, 332

Permission-Missing header field, 165
Personal mobility, 178–84
P-frames, 283
Physical layer, 2
PINT, 20
Playback, 275
Polling, for state, 203
Port assignment, 242
Port numbers, 7, 45
P-OSP-Auth-Token header field, 155
P-Preferred Identity header field, 155
PRACK method, 97–99
- call flow example, 98
- defined, 97
- example exchange, 98–99
- generation, 97
- header fields, 99
- *See also* Request messages

Precall mobility, 179
Presence
- agents (PAs), 52–53
- architecture, 191
- call flow example, 365–68
- defined, 189
- documents summary, 204–5
- elements, 191
- history, 189–91
- server, 39, 40
- with SIMPLE, 191–205
- UAs, 91

Presence information, 38
- collection, 53
- defined, 38

Priority header field, 153
Privacy header field, 153
Priv-Answer-Mode header field, 143
Private IP addresses, 4
Proxy-Authenticate header field, 166
Proxy authentication, 315
Proxy-Authorization header field, 153–54
Proxy-Require header field, 154
Proxy servers
- access, 57
- defined, 56
- forking, 59, 60
- SIP call with, 31–36
- stateful, 58, 349–52
- stateless, 57, 349–52
- user agents/gateways versus, 57
- *See also* SIP servers

PSTN and Internet Interworking (PINT), 86
Publication, partial, 202–4
Public key cryptography, 309
Public service answering points (PSAPs), 372
Public Switched Telephone Network (PSTN), 54–55
- codecs, 282
- protocols, 263–64
- signaling, 230

PUBLISH method, 88–91
- call flow example, 89
- defined, 88
- example request, 88–89
- header fields, 90
- *See also* Request messages

RAck header field, 161

Real-Time Text Taskforce (R3TF), 285
Real-Time Transport Protocol (RTP), *xxiv, xxviii,* 9, 273–78
- audio video profiles, 281–84
- client behavior, 277–78
- in common media processing steps, 274–75
- compressed (CRTP), 280
- defined, 273
- encryption support, 278
- header, 275–77
- impairment detection, 274
- implementation, 273
- lost packet detection, 277
- media sessions, 277
- port, 253
- secure (SRTP), 323
- sessions, 278
- symmetric, 251–53
- topologies, 278
- UDP use, 275

`Reason` header field, 156
`Record-Route` header field, 144
`Recv-Info` header field, 144
Redirection, 312
Redirect servers, 45
- defined, 61
- example, 61
- *See also* SIP servers

`REFER` method, 91–94
- acceptance, 94
- for attended transfer feature, 93
- defined, 91
- header fields, 95
- message example, 92
- with Web page push, 92
- *See also* Request messages

`Referred-By` header field, 94, 157
`Referred-To` header field, 94
`Refer-Sub` header field, 144–45
`Refer-To` header field, 156–57
Regional internet registries (RIR), 3
`REGISTER` method, 76–78
Registrar servers, 63–64
Registration
- as additive process, 38
- defined, 36
- example, 36–38
- illustrated, 37
- third-party, 77
- user agent, 52
- wireless phone, 37

`Reject-Contact` header field, 158–59
Reliability, 66–68
- example, 67
- mechanisms, 66

RELOAD (Resource Location and Discovery) protocol, *xxiv,* 336–38
- binary encoding/TLV combination, 337
- defined, 336
- functions, 337
- HIP advantages, 339
- overlay protocols, 338
- SIP usage, 337
- uses, 336

Rendezvous, 35
`Replaces` header field, 158
Replay attack, 312
`Reply-To` header field, 157–58
Request and response header fields, 134–49
`Request-Disposition` header field, 159
Request header fields, 149–63
Request messages, 73–107
- `ACK,` 78–80
- `BYE,` 78, 79
- `CANCEL,` 81–82
- `INFO,` 96–97
- `INVITE,` 73–76
- `MESSAGE,` 94–96
- message bodies, 105–6
- methods, 73–100
- `NOTIFY,` 87–88
- `OPTIONS,` 82–84
- `PRACK,` 97–99
- `PUBLISH,` 88–91
- `REFER,` 91–94
- `REGISTER,` 76–78
- `SUBSCRIBE,` 84–87
- tags, 104
- `UPDATE,` 99–100
- URI schemes, 101–2
- URL schemes, 102–4

`Require` header field, 159–60
Resource lists, 194–99
- ad hoc creation and manipulation, 198
- defined, 194
- extension, 195
- stored on RLS, 197

`Resource-Priority` header field, 160
Response header fields, 163–69

Response-Key header field, 160
Response messages, 111–30
  classes, 27, 111, 112
  client error, 116–27
  defined, 111
  global error, 129–30
  informational, 112–14
  server error, 128–29
  success, 114–16
Responses, 27
Retry-After header field, 145
RFC 822, 11
RFC 2327, 289
RFC 2543, 68
RFC 3235, 246
RFC 3261, 68
RFC 3264, 297
RFC 3489, 248
RFC 3550, 273
RFC 3551, 300
RFC 4662, 194
Rich Presence Information Data (RIPD), 194
Rivest Cipher 4 (RC4), 308
Rivest Shamir Adellmann (RSA) algorithm, 309
Robust header compression (ROHC), 280
Roundtrip time (RTT), 67
Route header field, 160–61, 184
RSeq header field, 168–69
RTP Control Protocol (RTCP), 278–80
  defined, 278
  extended reports (RTCP-XR), 279–80
  packets, 278, 279
  port, 253
  reports, 279

Secure audio and video profiles (SAVP), 281
Secure MIME (S/MIME), 314
Secure RTP (SRTP), *xxiv,* 323
  defined, 323
  keying, 323–25
Secure SIP, 317
Secure sockets layer (SSL), 313
Security, 307–26
  authentication and, 307, 314–17
  basic concepts, 307–11
  confidentiality and, 307–8
  deployment, 372–73
  digital certificates and, 310–11
  encryption and, 308–9
  media, 322–26
  model, 314–19
  protocols, 312–14
  public key cryptography and, 309
  threats, 311–12
Security-Client header field, 161
Security-Server header field, 166
Security-Verify header field, 162
Self-fixing approach, 253
Server error responses, 128–29
  500 Server Internal Error, 128
  501 Not Implemented, 128
  502 Bad Gateway, 128
  503 Service Unavailable, 128
  504 Gateway Timeout, 128–29
  505 Version Not Supported, 129
  513 Message Too Large, 129
  580 Preconditions Failure, 129
  See also Response messages
Server header field, 166
Service delivery platform (SDP), 230, 231–32
Service mobility, 183
Service oriented architecture (SOA), 230, 231
Service resource records (SRV), 15–16
Service-Route header field, 166–67
Services, 219–32
  application sequencing, 229–30
  architectures, 230–32
  certificate, 319–22
  conferencing, 227–29
  examples, 221–23
  facsimile, 226–27
  gateway, 219–20
  video, 225–26
  voicemail, 223–25
Servlets, 230, 231
Session Description Protocol (SDP), *xxviii,* 289–96
  attributes, 294–96
  bandwidth, 293
  common media types, 294
  connection data, 292
  contents, 29
  defined, 289
  e-mail address and phone number, 292
  encryption keys, 293
  extensions, 296–97
  fields, 290
  media announcements, 293–94

media session information, 289–90
offer answer model, 297–300
origin, 291–92
protocol version, 291
session name and information, 292
in SIP session establishment, 26–27
time, repeat times, time zones, 293
URI, 292
`Session-Expires` header field, 162
Session Initiation Protocol Investigation (SIPPING), 20, 369
Session Initiation Protocol (SIP)
call with proxy server, 31–36
defined, 1
evolution, 369
as global open standard, 17–18
history, 20–21
interdomain, 371
Internet and, 1–21
introduction to, 23–48
peer-to-peer (P2P), 331–40
popularity, 20
presence and instant message example, 38–43
presence information functions, 2
registration example, 36–38
security, 307–26
services, 219–32
session establishment example, 23–31
signaling functions, 1–2
simplicity, *xxviii*
symmetric, 250
for telephones, 264–65
as text-encoded protocol, 24
trunking, 221
See also SIP clients; SIP servers
Session Peering for Multimedia Interconnect (SPEERMINT), 371
Session timer extension, 60
Signaling protocols, 1–2
SIMPLE, *xxiii,* 20, 191
defined, 191
IM specifications, 193
IM with, 205–13
Jabber interworking with, 214
presence specifications, 192
presence with, 191–205
XMPP and, 214, 215
Simple Mail Transport Protocol (SMTP), 20–21
host/system independent name, 31
text encoding scheme, 21
Simple Traversal of UDP through NAT. *See* STUN
SIP clients, 30
SIPconnect, 221, 222
SIPCORE, 20, 369
`SIP-ETag` header field, 167
`SIP-If-Match` header field, 162, 204
SIP outbound, 251
call flow, 252
defined, 251
illustrated, 251
SIP servers, 29, 56–64
defined, 56
proxy, 56–61
redirect, 45, 61–63
registrar, 63–64
SPIRITS, 20
Standardized Generalized Markup Language (SGML), 377
Stateful proxies, 58, 349–52
Stateless proxies, 57, 349–52
Static payloads, 300
Stream Control Transmission Protocol (SCTP), 8–9, 237
message level delineation, 46
multihoming support, 47
transport, 46–47
STUN, 248–49
clients, 247
deficiencies, 247, 249
defined, 248
usages, 248–49
`Subject` header field, 145
`SUBSCRIBE` method, 84–87
defined, 84
example call flow, 85
header fields, 87
See also Request messages
`Subscription-State` header field, 162–63
Success messages, 114–16
`200 OK`, 114
`202 Accepted`, 115
`204 No Notification`, 225
`300 Multiple Choices`, 115
`301 Moved Permanently`, 116
`302 Moved Temporarily`, 116
`305 Use Proxy`, 116
`380 Alternative Service`, 116

Success messages (continued)
See also Response messages
Supported header field, 146
Suppress-If-Match header field, 163
Symmetric RTP, 251–53
Symmetric SIP, 250

Tags
extension feature, 146
request messages, 104
Target-Dialog header field, 163
Telephones, SIP for, 264–65
Telephony Gateway Registration Protocol (TGREP), 55
Telephony Routing over IP (TRIP) protocol, 55
Text over IP (ToIP), 285
Theft of service, 312
Third Generation Partnership Project (3GPP), *xxv,* 177
adaptive multirate (AMR) codec, 185
headers, 186
IPv6 addresses, 185
signaling compression, 185
Threats, 311–12
Timestamp header field, 147
To header field, 147
Transaction stateful proxy, 58
Transmission Control Protocol (TCP), 1, 5–7
congestion control, 7
connections, 46
defined, 5
flow control, 5–6, 7
handshake example, 6
header field, 7
mappings, 243, 245
reliability example, 6
segments, 5
sequence numbers, 5
transmission illustration, 45
transport, 45–46
Transmission Layer Security (TLS), 8, 313
authentication with, 316–17
transport, 46
Transport layer, 4–9
DCCP, 9
port numbers, 7
SCTP, 8–9
TCP, 5–7
TLS, 8
UDP, 7–8
Transport protocol selection, 47–48
Trigger-Consent header field, 163
Triple Data Encryption Standard (3DES), 308
Trunking, 221
between enterprise and service provider, 223
future directions, 372
SIPconnect specifications, 222
TURN protocol, 257

Uniform Resource Indicators (URIs), 12, 64–65
categories, 32
device, 32
embedded, 103
example, 64
instant messaging, 104
parameters, 65
presence, 104
request messages, 101–2, 102–3
SIP, 101
in SIP session establishment, 25
telephone, 102–3
in telephone number encoding, 65
user, 32
Uniform Resource Locators (URLs), *xxiii,* 11
qualifiers, 12
schemes, 12, 13
Uniform Resource Names (URNs), 12–13
UNSAF requirements, 249
Unsupported header field, 167
UPDATE method, 99–100
defined, 99
example, 100
header fields, 101
See also Request messages
User-Agent header field, 147–48
User agents (UAs), 51–52
back-to-back (B2BUAs), 53–54
client (UAC), 52
defined, 51
functions, 51–52
presence, 91
proxy servers versus, 57
registration, 52
server (UAS), 52
User Datagram Protocol (UDP), 1, 7–8
congestion control and, 44

datagrams, 44
multicast, support, 68
refresh timer, 247
transmission illustration, 44
transport, 43–44
Utility applications, 9–10

Via header fields, 34, 148–49
Video, 225–26
Video codecs, 283–84
Videoconferencing, 284–85
Video mixing, 285
Voicemail, 223–25
call flow, 225
defined, 223
URI cause parameter values, 224
See also Services

Warning header field, 167–68
World Wide Web Consortium (W3C), 377
World Wide Web (WWW), 377
WWW-Authenticate header field, 168

XML (Extensible Markup Language), 377–79
defined, 377
documents, 378
elements opened in, 378
for encoding information in SIP message bodies, 378
entity values in, 379
extensions, 379
format, 192
introduction to, 377–79
parser, 379
XMPP, 214
instant message mapping and, 214
presence mapping and, 215

ZRTP protocol
defined, 325, 326
implementation, 325
key agreement, 326