Reliable, Scalable and Interoperable Internet Telephony

Kundan Narendra Singh

Submitted in partial fulfillment of the

requirements for the degree

of Doctor of Philosophy

in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2006

ABSTRACT

Reliable, Scalable and Interoperable Internet Telephony

Kundan Narendra Singh

The public switched telephone network (PSTN) provides ubiquitous availability and very high scalability of more than a million busy hour call attempts per switch. If large carriers are to adopt Internet telephony, then Internet telephony servers should offer at least similar quantifiable guarantees for scalability and reliability using metrics such as call setup latency, server call handling capacity, busy hour call arrivals, mean-time between failures and mean-time to recover. This thesis presents a reliable, scalable and interoperable Internet telephony architecture for user registration, call routing, conferencing and unified messaging using commodity hardware. The results extend beyond Internet telephony to encompass multimedia communication in general.

The architecture presented in this thesis deals with two aspects: at least PSTN-grade reliability and scalability of the Internet telephony servers, and interoperable Internet telephony services such as conferencing and voice mail using existing protocols. We describe the architecture and implementation of our Session Initiation Protocol (SIP)-based enterprise Internet telephony architecture known as Columbia InterNet Extensible Multimedia Architecture (CINEMA). It consists of a SIP registration and proxy server, a multi-party conferencing server, a gateway for interworking SIP with ITU's H.323, an interactive voice response system and a multimedia mail server. CINEMA provides a distributed interoperable architecture for collaboration using synchronous communications like multimedia conferencing, instant messaging, shared web-browsing, and asynchronous communications like discussion forum, shared files, voice and video mails. It allows seamless integration with various communication means like telephone, IP phone, web and electronic mail.

We present two techniques for providing scalability and reliability in SIP: server redundancy and a novel peer-to-peer architecture. For the former, we use DNS-based load sharing

among multiple distributed servers that use backend SQL databases to maintain user records. Our two-stage architecture scales linearly with the number of servers. For the latter, we propose a peer-to-peer Internet telephony architecture that supports basic user registration and call setup as well as advanced services such as offline message delivery, voice mail and multi-party conferencing using SIP. It interworks with server-based SIP infrastructures.

Contents

List of Figures **ix**

List of Tables **xv**

Acknowledgments **xvi**

Chapter 1 Introduction **1**

1.1 Scalability and Reliability 3

1.2 Peer-to-Peer IP Telephony 6

1.3 Internet Telephony Interoperability 8

1.4 Original Contributions 9

1.4.1 Failover and Load Sharing in SIP Telephony 10

1.4.2 Peer-to-peer Internet Telephony using SIP (P2P-SIP) 10

1.4.3 Enterprise Internet Telephony and Multi-platform Collaboration 11

1.5 Overview of the Thesis 15

Chapter 2 Background: Session Initiation Protocol (SIP) **17**

I Server Redundancy **26**

Chapter 3 Failover and Load Sharing in SIP-based IP Telephony **27**

3.1 Introduction 27

3.2 Related Work 28

3.3 Availability: Failover 30
3.3.1 Client-based Failover 30
3.3.2 DNS-based Failover 30
3.3.3 Failover based on Database Replication 31
3.3.4 Failover using IP Address Takeover 32
3.3.5 Reliable Server Pooling 33
3.3.6 Implementation 34
3.3.7 Analysis 36
3.4 Scalability: Load Sharing 39
3.4.1 Network Address Translation 39
3.4.2 Multiple Servers with the Same IP Address 39
3.4.3 DNS-based Load Sharing 40
3.4.4 Identifier-based Load Sharing 41
3.4.5 Two-stage Reliable and Scalable Architecture 42
3.5 Performance Evaluation 44
3.5.1 Test Setup 44
3.5.2 Analysis 46
3.5.3 Non-uniform Call Distribution 50
3.5.4 Performance of Stateful Proxy 51
3.5.5 Effect of DNS Lookups 52
3.5.6 Other SIPstone Tests 54
3.6 Server Architecture 55
3.6.1 Processing Steps 56
3.6.2 Stateless Proxy 57
3.6.3 Stateful Proxy 59
3.6.4 The Best Architecture 63
3.6.5 Effect on Load Sharing Performance 63
3.7 Conclusions 64

II Peer-to-peer IP Telephony **66**

Chapter 4 Overview of Peer-to-Peer Internet Telephony using SIP **67**

4.1 Introduction 67

4.2 Related Work 70

4.2.1 Skype and Related Systems 70

4.2.2 P2P-SIP Telephony 71

4.2.3 IP Telephony vs. File Sharing 72

4.2.4 Robustness and Scalability 73

4.3 Design Requirements 74

4.4 SIP-using-P2P and P2P-over-SIP 77

Chapter 5 SIP-using-P2P: Using an External DHT as a SIP Location Service **80**

5.1 Introduction 80

5.2 Background: DHT API 81

5.3 Data and Service Models 82

5.4 Logical Operations 84

5.5 Deployment Scenarios 88

5.5.1 P2P Client 88

5.5.2 P2P Proxy 92

5.5.3 P2P Client Adaptor 93

5.6 Security and Trust 94

5.7 Implementation Issues 95

5.8 Advanced Services 99

5.8.1 Offline Messages 99

5.8.2 Presence 100

5.9 Evaluation 100

Chapter 6 P2P-over-SIP: DHT Maintenance using SIP **104**

6.1 Introduction 104

6.2 Background and Design Alternatives 105

6.3 Architecture Overview . . . 108
6.3.1 SIP Layer . . . 109
6.3.2 Node Startup and Peer Discovery . . . 110
6.3.3 User Registration . . . 112
6.3.4 Node Shutdown or Failure . . . 113
6.3.5 User Location and Call Setup . . . 115
6.4 Details of the DHT Module . . . 116
6.4.1 Initialization . . . 117
6.4.2 Peer Discovery . . . 118
6.4.3 Joining the DHT . . . 120
6.4.4 Stabilization . . . 123
6.4.5 Node Shutdown (Graceful Termination) . . . 127
6.4.6 Node Failure and Failover . . . 129
6.5 User Registration . . . 130
6.5.1 Registration Handling . . . 133
6.5.2 Node Shutdown (Graceful Termination) . . . 135
6.5.3 Node Failure and Failover . . . 136
6.6 Call Setup and Message Proxy . . . 136
6.6.1 Multimedia Call Setup and Instant Messages . . . 137
6.7 Advanced Services . . . 138
6.7.1 Offline Messages . . . 139
6.7.2 Multi-party Conferencing . . . 141
6.7.3 Device Independence . . . 141
6.7.4 Presence and Event Notification . . . 142
6.7.5 Adaptor for Existing SIP Phones . . . 145
6.7.6 NAT and Firewall Traversal . . . 145
6.8 Inter-domain Operation: Multiple DHTs . . . 146
6.9 Security . . . 151
6.10 Performance Evaluation . . . 160

6.11 Conclusions 162

III Enterprise IP Telephony 165

Chapter 7 Background: Conferencing, Streaming and Voice Dialogs 166

7.1 Multi-party Conferencing 166

7.1.1 Conferencing Models 168

7.1.2 Requirements for Centralized Conferencing 172

7.2 VoiceXML: Interactive Voice Response 173

7.3 RTSP: Media Streaming 175

Chapter 8 Related Work: Internet Telephony and Multimedia Collaboration 177

8.1 Interworking Between SIP and H.323 179

8.2 Unified Messaging using SIP and RTSP 179

8.3 Centralized Conferencing using SIP 180

8.4 Integrating VoiceXML with SIP Services 181

Chapter 9 Multi-platform Collaboration in CINEMA 182

9.1 Introduction 182

9.2 Requirements 183

9.3 Architecture Overview 185

9.3.1 Web Interface 186

9.3.2 Personal Calendar and Address Book 187

9.3.3 Events and Event-groups 188

9.4 Synchronous Collaboration 188

9.4.1 Audio Mixing 189

9.4.2 Video Forwarding 193

9.4.3 Instant Messaging 194

9.4.4 Shared Web Browsing 194

9.4.5 Screen Sharing 195

9.4.6 Conference Control 195
9.4.7 Dial-in vs Dial-out Conferences 196
9.5 Asynchronous Collaboration 196
9.5.1 File Sharing 197
9.5.2 Discussion Forum 198
9.5.3 Conference Event Recording 198
9.5.4 Unified Messaging and Multimedia Mail 200
9.5.5 Notifications and Announcements 207
9.6 Additional Services 209
9.6.1 Presence 209
9.6.2 Interactive Voice Response (IVR) 211
9.6.3 Interaction among Email, Telephone and IM 218
9.7 Conclusions 221

Chapter 10 Scalable Centralized Conferencing 225

10.1 Introduction 225
10.2 Scalability 226
10.2.1 Requirements 226
10.2.2 Performance Evaluation 228
10.2.3 Cascaded Conference Servers 238
10.2.4 Distributing Conferences 242
10.2.5 Handling Overload: Graceful Denial and Admission Control 243
10.3 Reliability 243
10.3.1 Reactive Failover 244
10.3.2 Proactive Redundancy 245
10.4 Conclusions 246

Chapter 11 Interworking Between SIP/SDP and H.323 248

11.1 Background and Requirements 249
11.1.1 Protocol Overview 249

11.1.2 Translation Requirements 250
11.2 Architecture for User Registration 255
11.2.1 IWF Contains SIP Proxy and Registrar 256
11.2.2 IWF Contains an H.323 Gatekeeper 259
11.2.3 IWF is Independent of Proxy or Gatekeeper 261
11.3 Signaling Address Translation 263
11.4 Connection Establishment 264
11.4.1 Using H.323v2 Fast Connect 265
11.4.2 Call Translation Without using Fast Connect 265
11.5 Calculating a Common Subset of Media Capabilities 272
11.6 Translating Advanced Services 276
11.6.1 Multi-party Conferencing 276
11.6.2 Call Transfer 278
11.7 Conclusion 279

Chapter 12 Conclusions and Future Directions 281
12.1 Summary of the Problems and Contributions 281
12.2 Connecting Themes 283
12.3 Server-based vs. Peer-to-peer Internet Telephony 284
12.4 Implications of this Research 286
12.5 Future Directions 287

Appendix A Design and Implementation of the Columbia SIP Library 289
A.1 Background 289
A.2 User Agent Library 297

Appendix B Two-way Replication in MySQL 302

Appendix C Data Format for SIP-using-P2P 305

Appendix D Implementation Details of SIP-H.323 Interworking Function 311
D.1 Implementation Requirements 311

D.2 Signaling Address Translation 320
D.2.1 Converting SIP Addresses to H.323 Addresses 321
D.2.2 Converting H.323 Addresses to SIP Addresses 323
D.3 Detailed Description of IWF Behavior 324
D.3.1 SIP-originated Requests 324
D.3.2 H.323-Originated Requests 328

Appendix E Glossary **332**

Appendix F Bibliography **336**

List of Figures

1.1 An example SIP call 3
1.2 SIP network architecture 5
1.3 P2P-SIP deployment architectures 7
1.4 CINEMA architecture 12

2.1 Example SIP Message with SDP 19
2.2 Example SIP call routing 21
2.3 Example CPL script: call routing based on time-of-day 24

3.1 Client-based failover 30
3.2 DNS-based failover 30
3.3 Failover based on database replication 31
3.4 When the primary server fails 32
3.5 When the master database fails 32
3.6 co-located database and proxy 32
3.7 Reliable server pooling for SIP 33
3.8 Failover in CINEMA 35
3.9 Call setup latency on failover 37
3.10 User unavailability on failure 37
3.11 DNS-based 40
3.12 Identifier-based load sharing 40
3.13 Two-stage reliable and scalable architecture 42

3.14 Example test setup for S3P3 45
3.15 Example message flow for S2P2: in the first stage INVITE goes via S2, whereas ACK and BYE via S1, but in the second stage all the requests go via P2 based on the consistent hash of the destination user identifier. 45
3.16 Server throughput in S_nP_m configuration (n first stage and m second stage servers. The results show that the performance increases linearly with the number of servers, i.e., s2p2 is twice and s3p3 is thrice that of s1p1 and s0p1 performance.) 47
3.17 Theoretical and experimental capacity for configuration S_nP_m 48
3.18 Effect of user identifier distribution among second stage servers for S2P22. Uniform distribution gives the best performance, i.e., success rate is close to 100% until the peak performance (1800 CPS), whereas for non-uniform distribution the success rate reduces as soon as one of the server is overloaded (at 1500 CPS). . . 50
3.19 Performance of S_nP_m with stateful proxy in second stage. The results show that the performance increases linearly with the number of servers, i.e., s2p2 is twice and s3p3 is thrice that of s1p1 and s0p1 performance. 51
3.20 Stateful proxy message flow 52
3.21 Performance for S_nP_m with registration server in second stage. The results show that the performance increases linearly with the number of servers, i.e., s2p2 is twice and s3p3 is thrice that of s1p1 and s0p1 performance. 54
3.22 REGISTER message flow 55
3.23 Processing steps in a SIP server. The potentially blocking operations either due to I/O, events or locks are marked with B 56
3.24 Performance of software architectures relative to event-based on different hardware. For example, the performance of stateless proxy on 4xP hardware in the thread pool architecture is approximately three times that in the event-based architecture on the same hardware. 60

3.25 Two-stage thread pool software architecture: the example consists of four threads, numbered 0 to 3, in the thread pool. Any available thread receives the message, parses it and based on the hash of the SIP Call-ID value in the message, forwards the message to the appropriate thread. In the example, the hash is 1, thus both SIP INVITE request and 200 OK response go to the thread number 1. 61

4.1 Client-server vs peer-to-peer distributed systems 68
4.2 Design A: all servers store all user records on registration 75
4.3 Design B: search for the server on call setup 75
4.4 Option 1: Only servers in DHT 76
4.5 Option 2: Complete P2P overlay 76
4.6 Option 3: Intermediate model 76
4.7 Difference between SIP-using-P2P and P2P-over-SIP architectures 77

5.1 Logical operations in a SIP server 82
5.2 Data model vs service model 83
5.3 P2P-SIP: SIP-using-P2P architecture 88

6.1 Example Chord network 106
6.2 No REGISTER 107
6.3 With REGISTER 107
6.4 Block diagram of a P2P-SIP node 108
6.5 Node startup and outgoing registration 111
6.6 Incoming registration 113
6.7 Failure of a super-node in the DHT 114
6.8 User location and call setup 115
6.9 Example Chord network with 4 nodes 118
6.10 After node 7 joins the network 118
6.11 Offline message storage 139
6.12 Inter-domain P2P-SIP 148

7.1 Types of media distribution model 170
7.2 Example sipvxml scenario 175

9.1 SIP-based collaborative work environment 185
9.2 Personal calendar 187
9.3 Audio mixing 190
9.4 Possible optimization in decode-mix-encode sequence 192
9.5 Example SIP MESSAGE for instant messaging 194
9.6 File sharing 197
9.7 Web-based discussion forum 199
9.8 Web interface for conference recording 200
9.9 Forwarding the call to voicemail 201
9.10 CPL script for forwarding a call to voicemail 203
9.11 Voice messages user interface 205
9.12 SIP-based presence 209
9.13 Web-based presence 211
9.14 Operation of sipvxml 212
9.15 Method 1: Joining a conference in blind transfer mode 215
9.16 Method 2: Joining a conference using bridged mode 216
9.17 Email-by-phone architecture 218
9.18 Email notification to phone 218
9.19 Example translation used in email to phone system 219
9.20 SIP-CGI for IM to email translation 222

10.1 Physical configuration 229
10.2 Logical configuration 229
10.3 Server performance with increasing number of participants in a single conference 230
10.4 Server performance with increasing number of four-party conferences 231
10.5 Speaker-to-listener delay for first and last participant to receive packets from the mixer 233

10.6 Effect of packetization interval on performance 234
10.7 Server performance on 360 MHz Sun/SPARC as the number of participants in a single conference increases 236
10.8 Relative audio codec performance in terms of CPU speed on various platforms for processing 20 ms audio. The y-axis provides numbers in Kilo cycles (1024 cycles). For example, GSM encoder took about 300 Kilo cycles on a 900 MHz Sparc, which means $\frac{300\times1024}{900\times1048576}s \approx 325\mu s$. 238
10.9 Tree-based cascaded servers 239
10.10 Full mesh cascaded servers 239
10.11 Performance of two cascaded conference servers for a single conference 242

11.1 H.323 call without fast-connect 251
11.2 Architecture for user registration in SIP-H.323 interworking 256
11.3 Initialization of SIP and H.323 terminals, and the IWF when IWF contains SIP proxy and registrar. The registration may get stored on two independent gatekeepers in the H.323 cloud. 258
11.4 Address translation from SIP to H.323 258
11.5 Address translation from H.323 to SIP 259
11.6 Address translation from SIP to H.323 when IWF contains an H.323 GK 260
11.7 Address translation from H.323 to SIP when IWF contains an H.323 GK 260
11.8 Call setup from SIP UA to H.323 terminal with FastConnect 265
11.9 Call setup from H.323 terminal to SIP UA with FastConnect 266
11.10 Call from SIP terminal to H.323 terminal without Fast Connect 267
11.11 Call from H.323 to SIP terminal without Fast Connect 268
11.12 Call from H.323 to SIP with conversion between OLC and SDP 271
11.13 Call from SIP to H.323 with conversion between OLC and SDP 272
11.14 Ad-hoc conferencing among SIP and H.323 endpoints 277
11.15 Different conferencing architectures 278
11.16 An example of call transfer mapping 279

A.1 Canonicalization, authentication and routing for a call 290
A.2 SQL vs FastSQL 293
A.3 Software design modules 294
A.4 Software library and applications 295
A.5 SIP transaction and client branches 298
A.6 call control state machine 299
A.7 Outgoing registration state machine 300

List of Tables

1.1 Factors contributing to IP telephony scaling 3

3.1 Performance (CPS) of stateless proxy for Proxy 200 test 58
3.2 Performance (CPS) for stateful proxy for Proxy 200 test 63

4.1 Different applications of P2P 73

5.1 Notations used in this chapter 81

7.1 Types of conferences; M is the number of active senders and N the total number of participants 169

10.1 Effect of various parameters on the server performance 227
10.2 Comparison of various audio codecs: time taken for encoding and decoding of 20 ms of audio on Pentium 4, 3 GHz CPU running Linux 2.6.9 in our test-bed: E means encoder, and D means decoder. G.711 and G.722 are ITU-T's, and DVI is Intel/IMA's 237

D.1 Support for Q.931 messages 312
D.2 Mapping between SIP status codes and reason fields 314
D.3 Support for H.245 messages. 317
D.4 Audio capability mapping 319
D.5 Video capability mapping. 320

Acknowledgments

I acknowledge with great pleasure the contributions of Professor Henning Schulzrinne, without whom this work would not have been possible. Professor Schulzrinne has provided guidance, knowledge, advice, and direction whenever it was needed. He provided opportunities for new research and introduced me to other researchers in the field, which have proved invaluable. I would also like to thank him for improving my writing skills, allowing me to mentor various project students and providing support for travels to various conferences.

I thank the rest of my thesis committee members: Dr. Gail Kaiser, Dr. Vishal Misra, Dr. Dan Rubenstein, and Dr. Milind Buddhikot. Their valuable feedback helped me improve my dissertation and shape the direction of my research.

The work presented in this thesis was financially supported by SIPQuest, which has also provided direction and scope for the work. In particular, I would like to acknowledge Yi Qin, Lei Wei and Sibon Barman for insightful discussions on SIP-based enterprise IP telephony architecture. The SIP-H.323 translator and unified messaging research were initially supported by Sylantro Inc.

I have enjoyed the privilege of working with my fellow graduate students, particularly the other members of the Internet Real-Time research group. They have provided spirited discussions, comments, and feedback, both in research group meetings and in one-on-one communications. In particular, Jonathan Lennox, Wenyu Jiang, Xiaotao Wu and Sankaran Narayanan contributed in the core design and implementation of CINEMA (Columbia InterNet Extensible Multimedia Architecture) and deserve special credit. Jonathan Lennox is the primary architect of our SIP server, sipd, and helped with the SIPstone measurement tools. Wenyu Jiang did the hard work in interconnecting our PBX (Private Branch eXchange) with the gateway. Xiaotao Wu

implemented the multimedia collaboration client, sipc, and helped in integrating the peer-to-peer mode. Sankaran Narayanan implemented the SIPstone tool and the efficient database interaction in sipd.

A number of other students have contributed to various components in the architecture as follows. Salman Baset implemented the event notification web interface. Omer Boyaci added the DNS NAPTR (naming authority pointer) record lookup in sipd. Michael Castleman implemented the anonymizer. Joseph Gagliano helped in implementing email notification to phone. Tarun Kapoor installed the initial MySQL database for testing. Gaurav Khandpur implemented the initial web-based discussion forum. Ali Khwaja integrated the high-quality codec (G.722) and sampling rate conversion in the conference server. Anshul Kundaje added RADIUS (Remote Authentication Dial in User Service) accounting to sipd. Jisoo Lee added experimental location detection for emergency calls in sipd. Xu Li added the ENUM (E.164 Numbering) support to sipd. Li Liao helped in TLS (Transport Layer Security) configuration. Chin-hong Lin and Agung Suyono implemented the conference recording feature. Daniel Liu and Naho Ogasawara implemented the initial email-by-phone system using Java Servlets. Gautam Nair helped in the initial implementation of the conference mixer. Ajay Nambi implemented some VoiceXML browser enhancements, voice-mail access and conference joining scripts. Sankaran Narayanan added TLS and IPv6 support. Eva Nautiyal and Manica Piputbundit helped in the implementation of phone announcement service and integration of TRIP (Telephony Routing over IP) to sipd. Timo Ohtonen helped in incorporating IPv6 support. Anurag Pant integrated the text-to-speech support in our media server. Mark Pimentel helped in implementing the conference timeline display. Joe Rosen implemented automatic gain control in the server. Jeffrey Schnurmacher designed the initial web interface layout. Naoya Seta implemented the convertor between instant messages and voice calls. Huitao Sheng helped with the initial performance measurement of the load sharing architecture. Madhuri Shinde enhanced the user interface of our P2P-SIP monitoring tool. Theodore Summe helped in adding the address book access via telephone in our test bed. Priyanka Upadhayay experimented with integrating speech recognition to our media server. Pimrampai Vannacharoen enhanced the email-by-phone system by using Tcl and integrated with the rest of our test bed. Visda Vokhshoori and Sean Mandel helped in initial VoiceXML browser

implementation. Sean West enhanced the auto-attendant application. Huwei Zhang contributed in file-sharing and conference load balancing. Thanks are due to Enlai Chu of Cisco, Keane Chin of SIP Communications Inc., and Sarmistha Dutta of Columbia University for their help with the Cisco gateway and departmental PBX. Thus, a number of people have helped in incremental development of the test bed which I have used in my thesis.

I am infinitely indebted to my family without whom this work would not have completed. In particular, my father Narendrakumar Singh and mother Nimmi Singh, who have consistently encouraged me to continue and finish my doctorate degree. My mother's untimely death in 2005 left me in utter shock and dismay. Words are not enough to thank my close friends, Knarig Arabshian and Shailendra Yede, for their moral support in difficult times like this, and for motivating me in my research.

To my parents

Chapter 1

Introduction

Internet telephony is defined as the transport of telephone calls over the Internet. Internet telephone calls can originate from traditional phone sets through gateways, PCs using software or embedded devices ("Ethernet phones"). Most of the interest in Internet telephony is motivated by cost savings and ease of developing and integrating new services. Internet telephony integrates a variety of services provided by the current Internet and PSTN (public switched telephone network) infrastructure. Internet telephony employs a variety of protocols, including RTP (Real-time Transport Protocol [1, 2]) for transport of multimedia data and SIP (Session Initiation Protocol [3, 4]) for signaling, i.e., establishing and controlling sessions. To ensure wide acceptance of SIP among carriers, SIP servers should demonstrate service availability and scalability at least as good as PSTN.

Decades of engineering and research have gone into providing the high availability and scalability of the public switched telephone network (PSTN). For example, PSTN switches have a "5 nines" reliability requirement, i.e., are available for 99.999% time or all but 5 minutes of a year. The performance of the PSTN is measured using metrics such as call setup delay (or post-dial delay) [5], busy hour call arrivals [6] (a measure of throughput), mean-time between failures (MTBF) and mean-time to recover (MTTR). The first two are concerned with performance, whereas the last two are metrics for quantifying availability. Traditionally, telephony service is perceived as more reliable than the Internet-based services such as web and email. We believe that Internet telephony [4] will fail to completely replace classical PSTN unless it provides at

least similar quantifiable guarantees.

SIP is a signaling protocol for IP telephony, multimedia conferencing, instant messaging and presence. SIP-based Internet telephony (SIP telephony) has been proposed as an alternative to the classical PSTN and offers a number of advantages over circuit switched telephony [4]. SIP signaling servers, like PSTN switches, help in call establishment and user location. However, unlike closed network-centric PSTN systems, SIP provides more control to the end users. SIP uses a number of other Internet services such as Domain Name Service (DNS) and non-guaranteed (best effort) IP packet delivery. Secondly, SIP servers that run on commodity hardware running Unix or Windows cannot assume strict operating system performance guarantees. SIP servers are closer to web servers than to PSTN switches, because of their request-response nature, text-based message format and programmable call routing behavior. However, unlike HTTP [7] which presumes a reliable underlying transport, SIP can use UDP for transport with application level retransmission for reliability and maintains transaction state at the proxy server. This makes the SIP reliability and scalability problem different from classical telephony or web.

The SIP proxy servers are light-weight compared to PSTN switches because they only route call signaling messages without maintaining any per-call state. The SIP proxy server of a *domain* is responsible for forwarding the incoming requests destined for the logical address of the form *user@domain* to the current transport address of the device used by this logical entity, and forwarding the responses back to the request sender. Consider the example shown in Fig. 1.1. When a user, Bob, starts his SIP phone, it registers his unique identifier *bob@home.com* to the SIP server in the *home.com* domain. The server maintains the mapping between his identifier and his phone's IP address. When another user, Alice, calls *sip:bob@home.com*, her phone does a DNS (Domain Name Service) lookup for *home.com* and sends the SIP call initiation message to the resolved server IP address. The server *proxies* the call to Bob's currently registered phone. Once Bob picks up the handset, the audio packets are sent directly between the two phones without going through the server. Further details [4, 8, 3] of the call are omitted for brevity.

SIP is designed to integrate with other Internet services, such as email, web, voice mail, instant messaging, multi-party conferencing and multimedia collaboration. We have implemented a SIP-based software suite called Columbia InterNet Extensible Multimedia Architecture

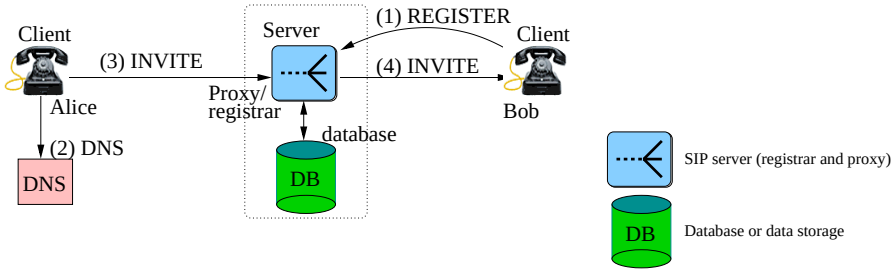


Figure 1.1: An example SIP call

(CINEMA) for Internet telephony and used it within the Computer Science department at Columbia University, integrating it with the existing PBX (Private Branch eXchange) infrastructure. The architecture provides inter-operability with the PSTN, programmable Internet telephony services, IP-based voice mail, integration with web and email for unified messaging, multi-party multimedia conferencing, and inter-operability with existing multimedia tools. The setup allowed us to extend our PBX capacity and eventually replace it, while keeping our existing phone numbers. This test-bed provides an environment where we can add new services and features, for example, accessing emails from a regular telephone, network appliance control, and support for instant messaging and presence. We believe that our setup can be readily used by other organizations.

1.1 Scalability and Reliability

In the example of Fig. 1.1, the SIP server, which includes the proxy and registration functions, forms the core of the Internet telephony infrastructure. For a scalable and reliable Internet telephony system, the core must be scalable and reliable. There are two components in providing high capacity reliable IP telephony services: network and servers as shown in Table 1.1.

Network	**Server**
1. Location of server in the network. 2. Backbone network capacity. 3. Load balancing among multiple distributed servers	1. Hardware configuration. 2. Non blocking I/O. 3. Server profiling (where it spends time). 4. Throughput as a function of load.

Table 1.1: Factors contributing to IP telephony scaling

Scalability determines how well a solution to some problem works when the size of the problem increases. For Internet telephony systems, scalability determines the performance as the load increases. SIPstone [9] defines various metrics to measure the SIP server performance such as registrations per second (RPS) and calls per second (CPS). It describes a series of tests with a pre-configured workload to simulate the activities of multiple users initiating SIP calls. The tests measure the performance of user registration and call handling, including redirect, failure and successful call setup.

Availability is defined as the probability that the service is available to use. It depends on the failure probability distribution and the recovery time distribution. We use the term *reliability* to mean availability.

SIP [3] defines three roles for a SIP server: registration, redirect and proxy. In practice, these roles are combined into a single entity, called the SIP server. The capacity requirement for the SIP server depends on its role in the network (Fig. 1.2). In real implementations one would expect to find small to medium scale corporate or enterprise proxies that provide rich integration with voice mail, calendar or event notification, proxies co-located with NAT (Network Address Translator) or firewall, stateless load balancing proxies managing some resources (e.g., a set of media gateways to PSTN) and high capacity proxies at various points in a carrier network for call routing. For example, 3GPP's IP Multimedia Subsystem (IMS) uses SIP for Call Session Control Function (CSCF) to support millions of users and defines different server roles such as outbound proxy (P-CSCF) in visited network, interrogating proxy (I-CSCF) as the first point of contact for incoming calls in the home network, and serving proxy (S-CSCF) providing services based on subscriber's profile [10, 11].

Reliability: The mean time between failures (MTBF) of the system can be obtained based on historical data, but is very difficult to predict in a complex systems like ours with a number of distributed interacting components. We focus on reducing the mean time to recover (MTTR) for the CINEMA components such as the SIP server. A few seconds of failover latency may not be noticeable for enterprise systems, but is undesirable for carrier proxies. A number of components such as DNS time-to-live, ARP (Address Resolution Protocol) cache, DHCP (Dynamic

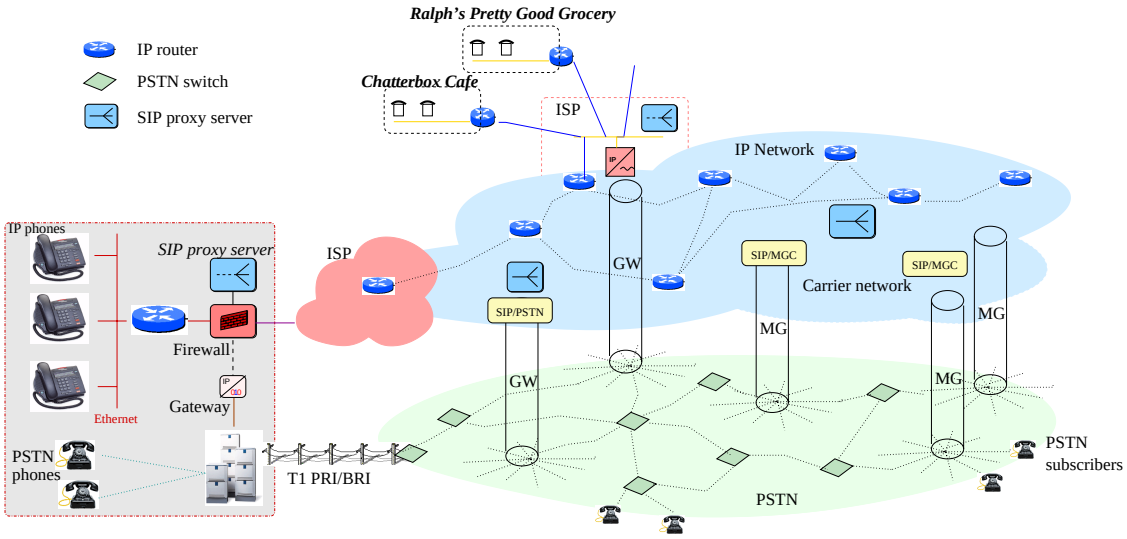


Figure 1.2: SIP network architecture

Host Configuration Protocol) timers, and SIP registration and call setup latency affect MTTR depending on the failover architecture. We explore this further in Section 3.3.

Scalability: SIP registration provides soft state which is periodically refreshed by registration refreshes. With the default one hour registration refresh timeout and the server capacity to handle one hundred registrations per second, i.e., 0.3 million registrations per hour, the server can serve 0.3 million users. A hundred registrations per second capacity roughly translates to 10 ms system time because the server can spend 10 ms per registration on an average. The capacity is further limited due to two registration requests using digest authentication or high registration rate for mobile users.

A call setup and termination may involve up to six signaling messages through the SIP proxy server. This can further increase with retransmissions or forking. 3GPP's IMS call flow has additional messages for early media and reliability of provisional responses resulting in about 14 messages per call. Moreover, advanced services such as programmable call routing further imposes additional processing demands on the server. Thus, the SIP server performance depends on CPU, memory, I/O and network bandwidth resources with one or the other dominating depending on the role. We tackle the reliability and scalability problems of SIP servers in Chapter 3.

1.2 Peer-to-Peer IP Telephony

The majority of the system cost of this server-based architecture is in maintenance and configuration, typically by a dedicated system administrator in the domain. It also means that quickly setting up the system in a small environment (e.g., for emergency communications or at a conference) is not easy. On the other hand, peer-to-peer (P2P) systems [12] are self-organizing. Moreover, they are inherently scalable to large user populations, and reliable because of the lack of a single point of failure. P2P systems, in the purest form, have no concept of servers. All participants are peers and communicate in a distributed, potentially untrusted environment, to achieve a certain objective such as locating music files or users.

Peer-to-peer Internet telephony using the Session Initiation Protocol (P2P-SIP) [13, 14, 15, 16, 17] has been proposed to avoid the maintenance and configuration cost of the server-based SIP architecture, and to prevent catastrophic failures of server-based systems. There are two approaches for combining SIP and P2P: replace the SIP location service by a P2P protocol (*SIP-using-P2P*) [16], and additionally, implement the P2P protocol itself using SIP messaging (*P2P-over-SIP*). In the first case, P2P is used only for lookups and updates of SIP user's IP addresses, similar to LDAP (Lightweight Directory Access Protocol) or SQL (Structured Query Language) databases used in existing SIP proxies. A scalable and global P2P location service automatically makes the SIP lookups scalable. In the second case, the P2P maintenance protocol can further exhibit two modes: (1) tunnel the P2P protocol messages in SIP, e.g., as a message body or headers, or (2) reuse the semantics of some of the SIP messages and headers to convey proximity and location information [14]. We describe our P2P-SIP architecture in Chapter 4.

The P2P deployment architecture can be another dimension to classify P2P-SIP systems. Consider a simple server-based SIP call as shown in Fig. 1.3 (a). This is similar to the earlier example in Fig. 1.1 except that the caller's user agent is configured to use the outbound proxy, which locates the callee's proxy via DNS. Either the user agent or the proxy server can use the P2P network for lookup as shown in Fig. 1.3 (b) and (c). **P2P clients** are SIP user agents that do not require any server and directly perform P2P lookups and updates. **P2P proxies** are SIP proxy servers that perform P2P lookups and updates, transparent to the user agent, e.g., in a zero-configuration server farm of a VoIP provider. The tradeoffs are ease of deployment and integration

with existing SIP clients or proxies, and reusability of other protocols and applications. These architectures and components should interoperate with each other.

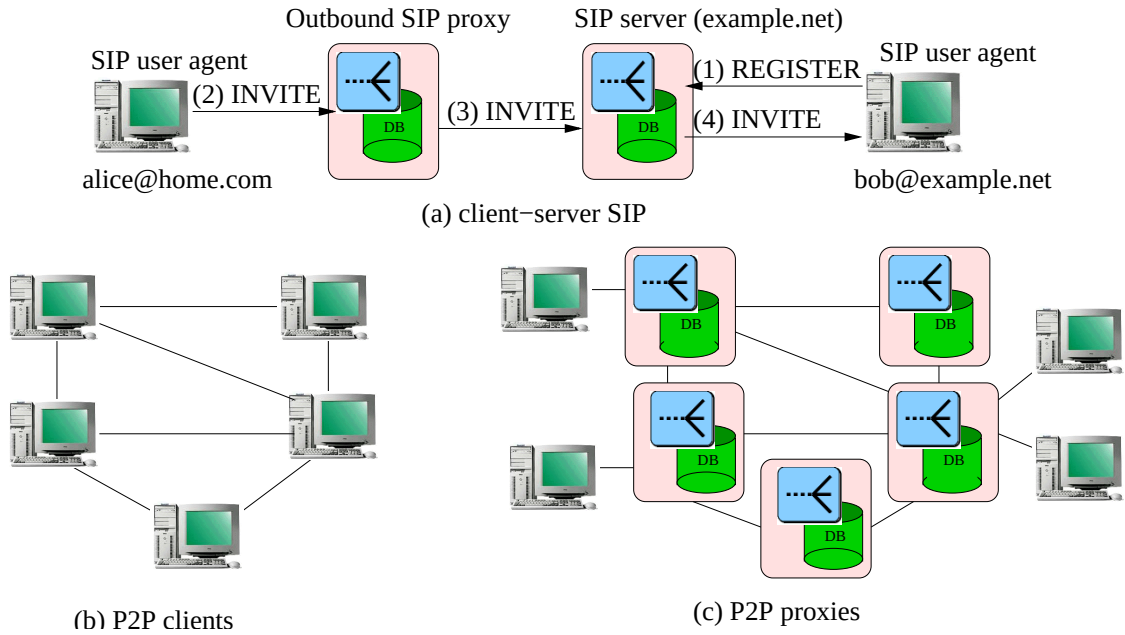


Figure 1.3: P2P-SIP deployment architectures

The architecture benefits from P2P scalability and robustness against catastrophic failures. We believe that P2P file sharing systems such as Kazaa [18] and Gnutella [19] are widely popular because they provide free music and video content without requiring maintenance of a content server, and they automatically detect NAT and firewall settings without any user intervention. Similarly, P2P-SIP has additional advantages over existing Internet telephony architectures as follows:

No maintenance or configuration: The system works out-of-the-box without requiring any tedious server installation, including NAT and firewall configuration. Our work extends the goals of the IETF Zeroconf [20] Working Group to multimedia communication and collaboration systems.

Interoperability: Unlike other P2P systems such as Skype [21], our architecture uses SIP messages for communicating with other peers. This readily interworks with any existing IP telephony infrastructure such as SIP-PSTN gateways.

These advantages come at the cost of increased *resource lookup delay*, security threats and reliability issues. Unlike $O(1)$ lookup cost in a classical client-server based systems, the P2P lookup cost can be much higher (e.g., Chord [22, 23], which is a P2P algorithm, has lookup latency of $O(\log N)$ where N is the number of peer nodes in the system). A distributed P2P architecture makes the system more prone to *security* issues such as trust (privacy: how much information does the untrusted peer need to know about me? and confidentiality: what if the peer who knows my information misuses it?) and denial of service (DoS) attacks (were those thousands of call routing requests that I received, legitimate?). A reliable framework for authentication without centralized elements is a challenge. In addition, we lose some of the traditional IP telephony services. For example, some of the programmable call routing techniques such as SIP-CGI [24] available for SIP telephony do not work in the P2P-SIP system as we do not want to run a potentially malicious script uploaded by some peer on our machines. Finally, the *reliability* of the IP telephony system is very important. People are unlikely to use it if the probability of successful call setup is not at par with that in the regular telephone network. Further details are in Chapter 4.

1.3 Internet Telephony Interoperability

The architecture for Internet telephony can be extended to an interoperable multimedia collaboration system using existing protocols such as SIP and RTP. Besides basic user registration and call setup, SIP supports a number of advanced services such as multi-party conferencing and offline messaging that are required in a collaboration system. Thus, SIP-based enterprise Internet telephony infrastructure can be used to provide a comprehensive multi-platform collaboration. There are two modes of collaboration. A *synchronous* or tightly coupled collaboration is highly interactive and requires the active presence of the other members of the group. On the other hand, an *asynchronous* or loosely-coupled collaboration is part of some collective activity directed towards some shared goal or common purpose, but does not require the active presence of the other members of the group. A *comprehensive* collaboration environment provides both synchronous and asynchronous collaboration tools and integrates the two so that users can easily alternate between the two.

One reason many earlier collaboration systems have not succeeded is that they were hard to use for people when the teams and groups span organizational boundaries. Also, they often require installing a lot of software, usually only available for limited set of platforms such as Windows, or work for only one vendor's tools, or one application such as collaborative software development [25]. On the other hand, a comprehensive multi-platform collaboration architecture such as ours, provides building block tools for any type of multimedia collaboration based on existing protocols, instead of focusing on specific types. It supports platform heterogeneity and device-transparency. For example, consider an IP telephony conference with some participants on phone, and some others using desktop multimedia clients. Late-arriving participants can browse through the past meeting proceedings, and non-participating group members can be automatically notified of meeting minutes and other important document locations via email. Users can access and interact even if they temporarily have only a phone or email. Two important requirements relevant to this thesis are summarized below:

Multi-party conferencing: The system should allow multi-party audio, video and text conferencing. Additionally, it should allow shared white-board, shared applications and screen sharing. It should be possible to record, and later playback, the proceedings of a conference.

Unified messaging: Traditional answering machines and voice mail services of PSTN can be enhanced in the Internet telephony by providing multimedia mail, and integration with web and email. This gives an opportunity to reuse the existing protocols for an interoperable unified messaging architecture. Programmable interactive voice response dialogs can be used to allow access and control from telephones.

1.4 Original Contributions

This chapter has described the high-level requirements for scalable, reliable and interoperable Internet telephony. In this section, I describe my explicit contributions to this topic.

1.4.1 Failover and Load Sharing in SIP Telephony

Consider the example of Fig. 1.1 (p. 3). If the server fails for some reason, the call initiation or termination messages cannot be proxied correctly. We can improve the service availability by adding a second server that automatically takes over in case of the failure of the first server. Secondly, if there are thousands of registered users and a single server cannot handle the load, then a second server can work along with the first server such that the load is divided between the two. We apply some of the failover and load sharing techniques to SIP servers. In particular, we evaluate DNS-based redundancy for user registration and call setup messages among multiple SIP servers that use SQL databases to maintain user records. We present a two-stage SIP server farm architecture where the first stage statelessly proxies the request to one of the several second stage servers. These techniques apply beyond telephony, for example, for SIP-based instant messaging and presence that use the same SIP servers for registration and message routing.

Earlier work showed that our SIP server, sipd, supported about 300 registrations and 90 call requests per second [26]. In this thesis, I further improved the system performance by distributing load among multiple redundant servers and using an event-based software architecture. My main contribution is to show that the two-stage architecture scales linearly with the number of servers.

1.4.2 Peer-to-peer Internet Telephony using SIP (P2P-SIP)

To avoid the maintenance and configuration costs of a server-based architecture, we developed a peer-to-peer (P2P) architecture for Internet telephony using SIP. We identify differences between rendezvous systems such as Internet telephony and traditional file sharing systems in P2P context. We analyzed various design alternatives and present a detailed design for both P2P-over-SIP and SIP-using-P2P, based on our implementation. The P2P-over-SIP implementation has a built-in Chord [22] as the underlying distributed hash table (DHT), whereas the SIP-using-P2P uses an external DHT, in particular OpenDHT [27]. Our work is the first published attempt to apply P2P concepts to SIP-based systems in the P2P-over-SIP architecture. Our novel hybrid architecture allows both traditional SIP telephony as well as user lookup on P2P network if the local domain does not have a SIP server. In P2P-over-SIP, we show that SIP can be used to implement various

DHT functions such as peer discovery, user registration, node failure detection, user location and call setup by replacing DNS [28] and the SIP server database with P2P for the next hop lookup in SIP. For SIP-using-P2P, we provide an XML-based data format for storing SIP information such as user contact location and security keys, so that different implementations can interoperate in a global P2P-SIP network. We also provide guidelines for advanced services such as offline message notification and multi-party conferencing in P2P-SIP. We identify the tradeoff in using P2P-SIP in terms of increased call setup delay and security threats.

1.4.3 Enterprise Internet Telephony and Multi-platform Collaboration

We identify the requirements for an enterprise Internet telephony infrastructure and describe our server-based Columbia InterNet Extensible Multimedia Architecture (CINEMA). It also serves as a corporate or campus infrastructure for existing and future services like instant messaging, presence, video mail and streaming media.

CINEMA consists of a set of SIP-based servers that provide a pathway to a post-PBX era of communications. It provides a comprehensive environment for creating and deploying rich Internet multimedia services including programmable Internet telephony services, audio/video conferencing, IP-based voice mail, and unified messaging. Fig. 1.4 shows the architecture, the interaction among the components and my contribution. (Appendix A further describes some of the software tools I implemented in CINEMA.)

SIP server: Jonathan Lennox is the primary architect of our SIP proxy, redirect and registration server, sipd. It receives user registration messages from user agents and proxies or redirects the incoming calls for registered users thus acting as a call router. I have further improved on the reliability and scalability aspects of sipd, as mentioned earlier.

SQL database: sipd uses the MySQL [29] database for storing the current network addresses and phone numbers where a user can be reached. Other per-user information and server configuration related to voice mail and conferences are also stored in the database.

PSTN gateway: A Cisco 2600 router with SIP/PSTN capability is connected to the telephone

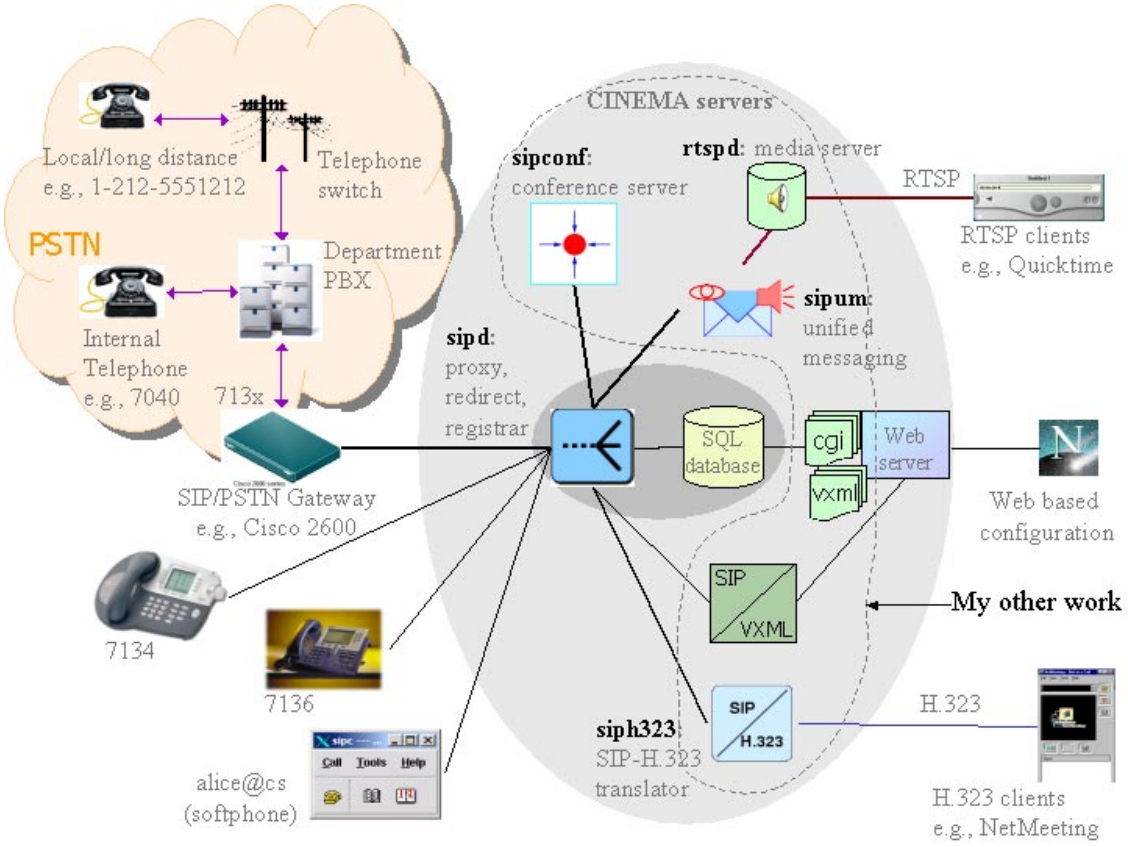


Figure 1.4: CINEMA architecture

switch (PBX) with a T1 trunk and to the departmental LAN. This could be any SIP-speaking gateway.

User agents: We use Columbia software SIP user agent (sipc [30, 31]) and Ethernet phones from Cisco, Pingtel and 3Com in our test bed.

Media server: We developed a general-purpose RTSP [32] streaming media server, rtspd, which we use for the storage and delivery of announcements and voice mail messages [33].

I am the primary author of the unified messaging server, conference server, SIP-H.323 translator and interactive voice response (IVR) module, which are described next.

Multi-Party Conferencing

Our collaboration architecture supports both synchronous collaboration such as multimedia conferencing, instant messaging (IM), shared web-browsing, and asynchronous collaboration such as discussion forums, shared files, voice and video mails, and allows seamless integration of the two. For example, the same group of people can be addressed by video conference, IM and email, with appropriate archival of the transactions.

I developed a multi-party multimedia conference server, sipconf [34], that forms the core of synchronous collaboration in CINEMA. I also evaluated the performance of our conference server and showed that it can support large scale conferences with thousands of simultaneous participants, using a two-layer cascaded conferencing architecture.

Unified Messaging

Traditional answering machines and voice mail services are closed systems, tightly coupled to a single end system, the local PBX or local exchange carrier. It is hard to perform simple operations, like forwarding voice mail outside the local PBX, filtering or sorting messages. Configuration is tedious, e.g., one can not readily switch between a set of outgoing messages. Moreover, voice mail and call answering services are implemented as stand-alone proprietary systems. The service must be provided by the PBX, local phone company or the local handset or one must obtain a separate voice mail number.

On the other hand, Internet protocols, such as electronic mail for Internet messaging and SIP (Session Initiation Protocol [3, 4]) for Internet telephony, have an open architecture. Configuration is simpler compared to the PSTN and the protocols are extensible. There can be a separation between the internet service provider and the messaging or telephony service provider. Internet telephony is replacing the old telephone systems (PSTN), particularly in corporate and institutional environments. So, it is important to design a voice mail system for Internet telephony, addressing some of the shortcomings.

Unified messaging extends the classical voice mail system to more Internet-based messaging service, integrating voice, video, web and electronic mail. We developed a novel and modular unified messaging architecture for multimedia mail using existing Internet protocols, in

particular, SIP and RTSP (Real-Time Streaming Protocol [32]), that allows users message access from any Internet connected device using standard media players or user agents. SIP is used for setting up multimedia calls over the Internet. RTSP controls the delivery of streaming media and provides facilities to play back, record, or perform other operations on multimedia content. I developed a centralized answering machine and voice mail system [33], sipum, based on this unified messaging architecture that uses rtspd media server for recording and playback.

Our approach differs from other traditional answering machines. We use the standard SIP forking proxy behavior that does not require modifying the proxy server or the user's phone. Use of an external media server helps in keeping the voice mail system out of the media path. Use of RTSP enables the recording of the message once and the use of the pointer or URL (Universal Resource Locator) when forwarding the message without actually forwarding the multimedia file. This is desirable for low bandwidth situations where downloading a whole video mail is very expensive, particularly if the recipient decides that she doesn't want to listen to the message after hearing the first few seconds. Moreover, the multimedia mail can be accessed with any RTSP based media player, e.g., Apple's QuickTime.

Integrating VoiceXML with SIP Services

People are familiar with traditional interactive voice response (IVR) systems found in voice mail access, dial-in conferences, phone-based customer support and tele-banking. VoiceXML is an XML-based language developed by the W3C [35] to create voice dialogs that feature synthesized speech, digitized audio, recognition of spoken and touch tone key input and recording of audio for telephony applications. It enhances the traditional proprietary and closed IVR systems to an open programmable architecture. It brings the advantage of web technologies to a telephony user by providing programmable dialogs, similar to HTML forms or CGI scripts.

A SIP-based VoiceXML (or SIP-VoiceXML) browser allows a SIP user to take part in application-specific IVR systems, e.g., voice mail or tele-banking. It also brings the advantage of VoiceXML technology to a regular telephone user via a SIP-PSTN gateway. I implemented the first SIP-VoiceXML browser, sipvxml [36], to enhance the services of our CINEMA test-bed. In particular, we have extended our multimedia conferencing server, sipconf [34], and unified

messaging (voicemail) server, sipum [33] to provide enhanced services and convenience to a telephone user.

Interworking between SIP/SDP and H.323

The International Telecommunication Union (ITU-T) Recommendation H.323 [37] defines packet-based multimedia communication systems and is based heavily on previous ITU-T multimedia protocols. In particular, H.323 call signaling is inspired by H.320 [38] for ISDN (Integrated Services Digital Network), and call control by H.324 [39] for GSTN (General Switched Telephone Network) terminals. SIP [3], developed in the IETF, builds on a simple text-based request-response architecture similar to other Internet protocols such as HTTP [7] and RTSP [32]. SIP provides a similar set of basic services as H.323 [40, 41].

We developed a translation mechanism for interoperability between SIP and H.323. I developed sip323, the first implementation of a signaling gateway [42] between SIP and H.323. This allows once popular H.323 clients such as Microsoft NetMeeting to interwork with our SIP-based CINEMA infrastructure. However, with the gradual disappearance of H.323 systems, the research interest in SIP-H.323 translator has faded. H.323 systems are still used particularly by carriers who have already made huge investments in H.323-based infrastructures, and by vendors such as Polycom developing room-based conferencing systems. Thus, there is still a need for interworking between SIP and H.323.

1.5 Overview of the Thesis

This thesis is organized as follows. After this introduction, I give background information on SIP in Chapter 2. Then, the thesis is divided into three parts: server redundancy, peer-to-peer and enterprise IP telephony.

1. In the server redundancy part, I describe our failover and load sharing architecture for SIP telephony and evaluate its performance in Chapter 3.

2. In the peer-to-peer part, I give an overview of related work and design choices for P2P

Internet telephony using SIP in Chapter 4. Then, I describe our SIP-using-P2P and P2P-over-SIP architectures in Chapters 5 and 6, respectively.

3. In the enterprise IP telephony part, I provide background on different conferencing models, VoiceXML and media streaming in Chapter 7. Chapter 8 presents the related work on Internet telephony and multimedia collaboration. Chapter 9 describes our multi-platform collaboration architecture in CINEMA. Subsequent chapters 10 and 11 give details on conference server scalability and SIP-H.323 translation, respectively.

I present some general conclusions and observations in Chapter 12. Implementation aspects of various components such as Columbia SIP library and related tools, MySQL replication, P2P-SIP data format and SIP-H.323 translation are presented in appendices A, B, C and D, respectively. A glossary of terms and bibliography of references can be found in appendices E and F, respectively.

Chapter 2

Background: Session Initiation Protocol (SIP)

For an Internet voice call, it is sufficient for a participant to know the audio codecs supported by the other participant and the IP address and port number to which audio packets should be sent. The problem with this is that IP addresses are hard to remember and may change if the user is mobile or uses more than one device. SIP allows use of a higher level address of the form *user@domain* for user mobility. For instance, a user can call *bob@office.com* no matter what communication device, IP address or phone number Bob is currently using. The current locations of the users are maintained by the SIP registration servers, also known as registrars. The user's communication devices register with registrar servers periodically by providing the address at which he/she can be reached. A more detailed description of SIP can be found in [3, 43, 4]. In this chapter, we give an overview of the features relevant for this thesis.

SIP message: request and response

SIP is a client-server and request-response protocol, similar to the Hyper Text Transfer Protocol (HTTP) [7]. A user agent client (UAC) generates a request, and sends it to the user agent server (UAS). The server processes the request and sends the response back to the client. There can be zero or more intermediate provisional responses, followed by a final response to a request. A

request and its responses constitute a *transaction*.

SIP defines methods for session establishment, control and termination. The methods are invoked by the UAC for a resource identified by an uniform resource identifier (URI) [44] on the UAS. As shown in Fig. 1.1 (p. 3), when Bob's user agent is powered up, it (UAC) sends the REGISTER method to the server (UAS) at example.net to update the contact location of Bob, identified with URI sip:bob@example.net. The server updates the contact location in its database. Alice's user agent (UAC) sends the INVITE method for sip:bob@example.net to Bob's server to initiate a call to Bob. The server locates the current contact of Bob and proxies the message to Bob's user agent (UAS). Since a SIP user agent (or end system) sends as well as receives the SIP request, it contains both UAC and UAS. Similarly, the server contains both UAC and UAS. The BYE request terminates the call, and can be invoked by the user agent of either Alice or Bob. Alice may cancel a pending call using CANCEL before Bob accepts the INVITE. ACK requests are used for reliability of INVITE responses [3] because, unlike HTTP, which runs on TCP, the SIP message can go over UDP also. There is an OPTIONS method to get the capabilities of the remote party without actually initiating a call. Additional methods have been defined to extend SIP for instant messaging (MESSAGE), presence (SUBSCRIBE, NOTIFY, PUBLISH), call transfer (REFER), etc.

An example INVITE message is shown in Fig. 2.1. Similar to HTTP, it is text-based with a request line containing the resource identifier as request-URI, sip:bob@example.net, followed by a list of headers and finally a body. The To header contains the callee. The From header contains the caller. The Subject identifies the subject of the call similar to an email subject. The Call-ID contains a unique call identifier to identify this association between the caller and callee. A SIP *dialog* is uniquely identified using the two end points (To and From) and Call-ID. INVITE and SUBSCRIBE are two methods that can create a new SIP dialog. Within a dialog, subsequent SIP requests have increasing CSeq header values in each direction, i.e., caller-to-callee and callee-to-caller.

The SIP response is similar to the request, except that the first line is a response line containing a response code and a reason phrase. SIP reuses HTTP's response codes and adds some new responses. In particular, a 200 response code indicates a success response.

```
INVITE sip:bob@example.net SIP/2.0
Via: SIP/2.0/UDP pc33.home.com;branch=z9hG4bKnashds8
Max-Forwards: 70
To: Bob <sip:bob@example.net>
From: Alice <sip:alice@home.com>;tag=1928301774
Call-ID: a84b4c728ca8@mypc.home.com
CSeq: 613 INVITE
Contact: <sip:alice@pc33.home.com>
Content-Type: application/sdp
Content-Length: 148

v=0
o=user1 53655765 2353687637 IN IP4 192.1.2.3
s=Weekly conference call
c=IN IP4 192.1.2.3
t=0 0
m=audio 8080 RTP/AVP 0 8
m=video 8082 RTP/AVP 31
```

Figure 2.1: Example SIP Message with SDP

SIP requests and responses may carry message bodies using MIME (Multipurpose Internet Mail Extension [45]) types. The body carries additional information such as a multimedia session description in the INVITE request and its success response, or presence state in the NOTIFY request.

The SIP session negotiates media capabilities of the caller and callee using the Session Description Protocol (SDP) [46]. SDP contains the various media types (e.g., audio, video), supported codecs for these media types, and the transport addresses (i.e., IP address or host name, port number and protocol) for receiving packets for these sessions. The caller offers a media session in INVITE's message body, and the callee answers it in the successful response's message body [47]. For example, the offer in Fig. 2.1 indicates that the caller can support one audio session with G.711 μ-law or A-law codec (payload types 0 and 8, respectively), and one video session with H.261 codec (payload type 31). The callee can select a subset of this media capability and indicate it in the response. Alternatively, if it does not want to support, say, video session, it can reset the transport address for the video session. After the call is set up, either party may change the session description by sending another INVITE method, known as re-INVITE.

Locating SIP servers

A SIP UAC uses DNS [28] to locate the SIP server for a URI. For example, if the request-URI is sip:bob@example.net, the client uses a DNS query for example.net to locate the SIP server for that domain. In particular, the DNS Naming Authority Pointer (NAPTR) record for example.net is queried. If this query fails, the service record (SRV) is queried for _sip._udp.example.net assuming the transport as UDP. The DNS NAPTR and SRV records have priority and weights to allow failover and load sharing. In the absence of these records, the client can fall back to querying the A and AAAA records for the IPv4 and IPv6 addresses, respectively.

SIP functions and states

A SIP server contains different functions: registrar, proxy and redirect. The *registrar* function deals with incoming REGISTER messages. The *proxy* function proxies the incoming non-REGISTER requests to the current contact location of the destination. The current location can be learned in various ways, including explicit update by the user agent via REGISTER. A *redirect* server responds with the list of current locations to the caller, so that the caller can directly re-initiate the request to one or more of those locations. Typically, a single application implements all these functions.

A SIP server typically destroys all transaction state after the transaction is over. Soft state is used to maintain transaction state. Typically, all transaction state is maintained for up to approximately 32 seconds after the final response is sent, during which the server can respond to any retransmitted request if the earlier response got lost. SIP also defines a *stateless* proxy function which does not even maintain any transaction state. SIP transactions can complete even if a server crashes and reboots in the middle, losing all transaction state. SIP messages have sufficient information to allow a rebooted server to treat the message correctly. This also means a server can safely clean up old state which has collected due to unusual failures or cases where the caller lets the phone ring for a long time.

Multiple locations can be registered for a single user, for instance, if the user has many SIP phones. The SIP server, in proxy mode, forwards the call request to all the registered locations. If the user picks up one of the phones, the server cancels branches to all the other phones. This

behavior is known as *forking* proxy mode. On the other hand, a sequential proxy mode tries the registered locations sequentially.

The media path for audio and video is different from the SIP signaling path because media is exchanged directly between the user agents typically using the Real-time Transport Protocol (RTP [1, 2]) using the session parameters derived from SDP. A SIP server does not maintain any dialog or call state. Thus, it treats individual requests such as INVITE and BYE as independent transactions. Subsequent request in a dialog (or call) can directly be sent between the two end-points instead of going through the server. However, a server can use the Record-Route header to remain in the signaling path for subsequent transactions of the call, e.g., for call accounting.

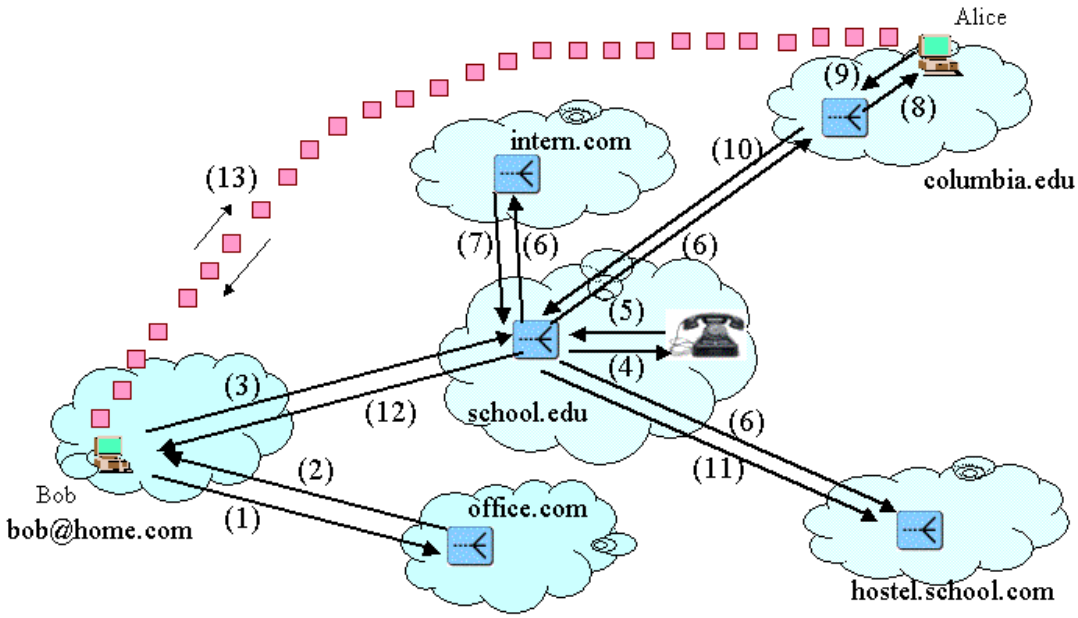


Figure 2.2: Example SIP call routing

An example call routing in SIP

It is possible to encounter multiple SIP servers (either in redirect or proxy mode) in a given call attempt. Fig. 2.2 shows a more complex call routing scenario in SIP.

1. Bob (bob@home.com) tries to reach Alice (alice@office.com).
2. The server at office.com redirects Bob, indicating that Alice can be reached at alice@school.edu.
3. Bob's user agent tries the new location.
4. Alice has registered four contacts, with one of them (her desk phone) as her preferred location. Thus, the server at school.edu tries the more preferred location for Alice at her desk phone.
5. The phone is idle, and sends a "ringing" response. However, since it is not picked up, the server times out.
6. The server then forks the call request to all the remaining three locations simultaneously. The locations are Alice.Cueba@intern.com, alice@columbia.edu and ac114@hostel.school.com.
7. The phone at intern.com responds back saying that the user is not available.
8. The server at columbia.edu forwards the call to Alice's desktop computer.
9. A pop-up window appears on Alice's machine indicating an incoming call from Bob. She accepts the call by clicking on the "Accept" button of the user interface.
10. The server at columbia.edu forwards the response to the upstream server at school.edu.
11. The server at school.edu on receiving the successful response, cancels out all the other pending call requests. In this example, it cancels the call request branch sent to hostel.school.com. The phone at hostel.school.com stops ringing at this time.

12. The server of school.edu then forwards the successful response to the upstream host (Bob's user agent).

13. At this point, the call is successfully established. Now media (audio and/or video) can be exchanged between the two endpoints. The call termination message is not shown.

In the above example we assume a wide-area network composed of a variety of environments such as campus, corporate and enterprise running SIP servers.

Programmable call handling

When receiving an incoming call request, the SIP server finds the current user location and either proxies (forks if multiple contacts), redirects or rejects the call initiation message. Although this simple model satisfies most user's needs, some advanced users may want a more complex scenario. For example, "reach me at my office phone during office hours and call me at my home after office hours, or don't disturb me when a tele-marketer calls." This can be implemented by uploading a piece of software on the server, which governs its behavior based on the time-of-day or caller identification. SIP allows many different ways to achieve this, for example, via the XML-based Call Processing Language (CPL [48, 49]) and SIP-CGI [24].

The Call Processing Language (CPL) is a language that can describe and control Internet telephony services. It is implementable on either network servers or user agent servers. It is simple, extensible, easily editable by graphical clients, and independent of operating system or signaling protocol. It is suitable for running on a server where users may not be allowed to execute arbitrary programs, as it has no variables, loops, or ability to run external programs. Fig. 9.10 shows an example CPL script for time-of-day based call routing. The idea is to proxy the call to the registered location during office hours and forward the call to voicemail otherwise.

SIP-CGI is similar to HTTP-CGI and can be written in any language. It has the same potential security problem as HTTP-CGI, and it should be allowed only in a trusted environment since users are allowed to execute arbitrary code. The SIP server can run the script as an external process passing all the parameters needed by the script (e.g., caller URI, subject headers, etc.) and reading back the response from the standard output of the process. The response indicates how to handle the call, for example, to proxy, redirect or reject it.

```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

<cpl>
 <incoming>
  <time-switch tzid="America/New_York"
       tzurl="http://zones.example.com/tz/America/New_York">
   <time dtstart="20000703T090000" duration="PT8H" freq="weekly"
       byday="MO,TU,WE,TH,FR">
    <lookup source="registration"><success><proxy /></success></lookup>
   </time>
   <otherwise>
    <location url="sip:jones@voicemail.net"><proxy /></location>
   </otherwise>
  </time-switch>
 </incoming>
</cpl>
```

Figure 2.3: Example CPL script: call routing based on time-of-day

More than just a voice call

One advantage of SIP over PSTN (Public Switched Telephone Network) based tele-conferencing is that it allows creating new flexible services including traditional PSTN services such as interactive voice response (IVR), call transfer, music-on-hold as well as more Internet-specific services like presence-enabled calls and integration of voice-mails and emails. Multimedia conferences on the Internet [50] can be created using SIP. Using SIP for instant messaging allows easy integration with other SIP-based telephony components. Various components such as media streaming server, call-routing engines and user location server can be combined in various ways to create new service architectures.

Sipc is the Columbia SIP user agent that can be used for Internet telephony calls, multimedia conferences, presence, instant messaging, and shared web browsing. It supports a range of media types, such as audio, video, text and white board, and can be easily extended to handle additional media types. It uses external media tools such as vic [51] for video, Robust Audio Tool (RAT [52]) for audio and wb [53] as a shared white board. Beyond multimedia communication, it can also perform network appliance control, or act as Session Announcement Protocol (SAP)-

based Internet radio or television [54]. It also supports emergency services such as E911 [55].

After this brief introduction to SIP, I present the first part of the thesis on SIP server redundancy.

Part I

Server Redundancy

This part describes our failover and load sharing architecture to achieve carrier-grade reliability and scalability of SIP servers using commodity hardware.

Chapter 3

Failover and Load Sharing in SIP-based IP Telephony

3.1 Introduction

In this chapter, we apply some of the existing web server redundancy techniques for high service availability and scalability to the relatively new IP telephony context. In particular, we consider SIP [3] server failover techniques based on the clients, DNS (Domain Name Service), database replication and IP address takeover, and load sharing techniques using DNS, SIP identifiers, network address translators and servers using the same IP address, in Sections 3.3 and 3.4, respectively. These techniques also apply beyond telephony, for example, for SIP-based instant messaging and presence that use the same SIP servers for registration and message routing.

We describe our two-stage reliable and scalable SIP server architecture in which the first stage proxies the request to one of the second stage server group based on the destination user identifier. We quantitatively evaluate the performance improvement of the load sharing architecture using our SIP server in Section 3.5. We also quantitatively compare the effect of SIP server architecture such as event-based and thread pool on server performance in Section 3.6. This further improves the performance using the event-based architecture for a single server. Additionally, we present an overview of the failover mechanism we implemented in our test-bed using the open source MySQL database. Next, we give an overview of related work in reliability and

scalability of SIP-based IP telephony.

3.2 Related Work

SIP servers are similar to web servers. Failover and load sharing for web servers is a well-studied problem [56, 57, 58, 59]. TCP connection migration [60], process migration [61], IP address takeover [62] and MAC address takeover [63] have been proposed for high availability. Load sharing via connection dispatchers [64] and HTTP content or session-based request redirection [65, 66, 63] are available for web servers. Some of these techniques such as DNS-based load sharing [67, 68] also apply to other Internet services like email and SIP. Some of the load sharing techniques such as dynamic load balancing [69] can also be applied to SIP, but we have not investigated this. Although SIP is an HTTP-like request-response protocol, there are certain fundamental differences that make the problem slightly different. For example, SIP servers can use both TCP and UDP transport, the call requests and responses are usually not bandwidth intensive, caching of responses is not useful, and the volume of data update (REGISTER message) and lookup (INVITE message) is often similar, unlike common read-dominated database and web applications.

Unlike call stateful PSTN switches, the SIP proxy servers are only transaction stateful [3]. Thus, if the server fails during a transaction, only that transaction needs to be restarted without affecting the existing call to which the transaction belongs. Some work has already been done in the context of SIP server availability [70, 71]. In particular, Ohlmeier [70] presents the requirements for high availability SIP servers and proposes failover using anycast and data replication. For SIP server failover, IP anycast does not work well with TCP and the backend database requires synchronization between the primary and backup servers for authentication, thus making the design complicated. To reduce the session setup time in case of server failure, IETF's Reliable Server Pooling (Rserpool) has been proposed for call stateful SIP servers [71]. Although this is useful for sharing call state between the primary and secondary server, a failed transaction still needs to be restarted. In Section 3.3.5, we describe how to apply the Rserpool [72, 73]) architecture for SIP telephony. The primary disadvantage of Rserpool is that it requires new protocol support in the clients. Mid-call failover [74] and server selection policy [75] for call stateful SIP

servers to improve the call success rate are proposed, but these proposals do not apply to the call stateless SIP proxy servers which are more scalable. We have implemented SIP server failover using database replication.

SIP-based telephony services exhibit three bottlenecks to scalability: signaling, real-time media data and gateway services. The signaling part requires high request processing capacity in the SIP servers. The data part requires enough network bandwidth and capacity (CPU and memory) in the end systems. The gateway part requires optimal placement of media gateways and switching components [76]. This thesis focuses only on the signaling part. SIP allows redirecting a request to a less loaded server using the 302 response, or transferring an existing call dialog to a less loaded endpoint or gateway [3, 77]. An overloaded SIP server can respond with the 503 response to signal the upstream sender to reduce the request rate when this downstream server is overloaded. However, this mechanism suffers from load amplification and oscillation problems in a cluster [78]. The SIP server can utilize the failover and load sharing research done in databases. For example, the MySQL SQL database allows replication and clustering [29, 79].

Identifier-based load balancing has been used for emails. We combine this with DNS-based server redundancy for a two-stage reliable and scalable architecture. Novelty of our work lies in the application of existing techniques to relatively new Internet telephony, and quantitative performance evaluation of the architecture for SIP-based telephony. We also present an overview of our implementation of failover and describe some practical issues.

SIP server performance can be quantified using metrics such as calls per second and registrations per second. SIPstone [9] defines such metrics and provides a benchmarking technique for SIP proxy and registration servers, which we use in our measurements. Optimizations such as memory pool, counted strings and lazy parsing have been proposed for SIP servers [80, 26]. These optimizations can further improve our load sharing performance. Event and thread-based architectures, particularly for web servers, are well known in systems research [81, 82, 83, 84, 85]. We study the effect of server architecture on SIP server performance using commodity hardware and standard POSIX (Portable Operating System Interface) threads in Section 3.6.

3.3 Availability: Failover

High availability is achieved by adding a backup component such as the SIP server or user record database. Depending on where the failure is detected and who does the failover, there are various design choices: client-based, DNS-based, database failover and IP address takeover.

3.3.1 Client-based Failover

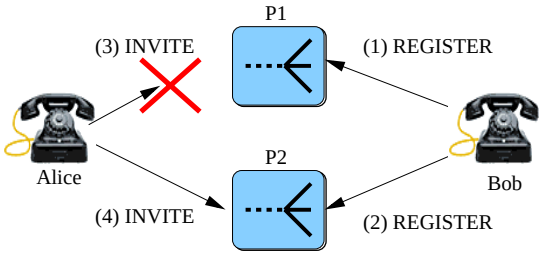


Figure 3.1: Client-based failover

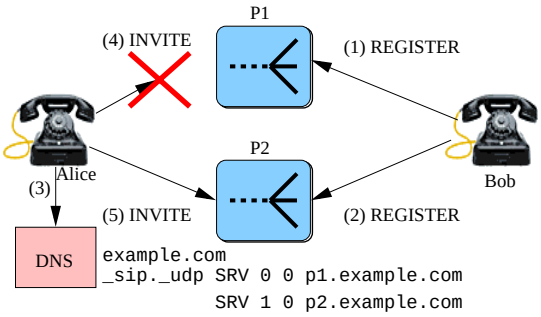


Figure 3.2: DNS-based failover

In the client-based failover (Fig. 3.1), Bob's phone knows the IP addresses of the primary and the backup servers, P_1 and P_2. It registers with both, so that either server can be used to reach Bob. Similarly, Alice's phone is also preconfigured with the addresses of the two servers. It first tries P_1, and if that fails it switches to P_2.

All failover logic is built into the client. The servers operate independent of each other. This method is used by the Cisco IP phones [86]. Configuring phones with the two server addresses works well within a domain. However, DNS is used to allow adding or replacing backup servers without changing the phone configurations as described next.

3.3.2 DNS-based Failover

DNS provides two record types, naming authority pointer (NAPTR) and service (SRV), relevant to SIP requests. A SIP client can use the priority parameters of these DNS records to determine the primary and secondary servers. DNS-based failover using NAPTR and SRV records is the most clean and hence, preferred way, to failover [28]. For instance, Alice's phone can retrieve the

DNS SRV [67] record for *_sip._udp.example.com* to get the two server addresses (Fig. 3.2). In the example, P_1 will be preferred over P_2 by assigning a lower numeric priority value to P_1.

Alternatively, dynamic DNS can be used to update the A-record for *home.com* from the IP address of P_1 to P_2, when P_1 fails. P_2 can periodically monitor P_1 and update the record when P_1 is dead. Setting a low time-to-live (TTL) for the A-record bindings can reduce the failover latency due to DNS caching [87].

3.3.3 Failover based on Database Replication

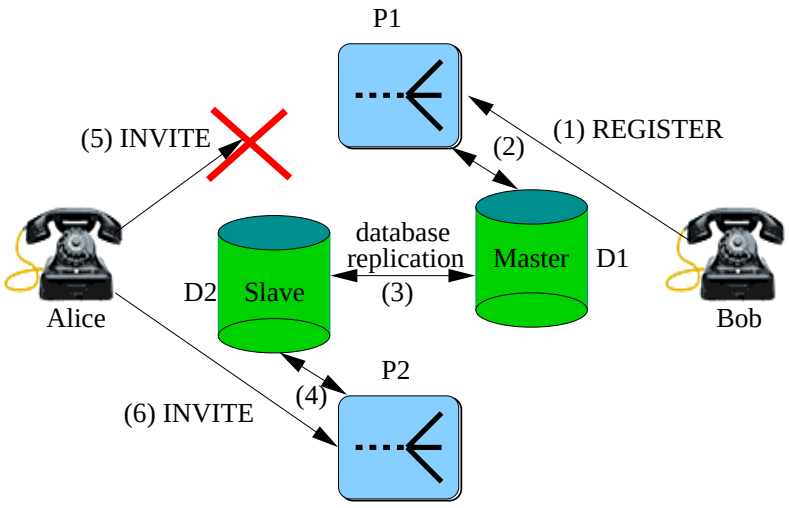


Figure 3.3: Failover based on database replication

Not all SIP phones are capable of registering with multiple servers. Moreover, to keep the server failover architecture independent of the client configuration, the client can register with only P_1, which can then propagate the registration to P_2. If a database is used to store the user records, then replication can be used as shown in Fig. 3.3. Bob's phone registers with the primary server, P_1, which stores the mapping in the database D_1. The secondary server, P_2, uses the database D_2. Any change in D_1 is propagated to D_2. When P_1 fails, P_2 can take over and use D_2 to proxy the call to Bob. There could be small delay before D_2 gets the updated record from D_1.

3.3.4 Failover using IP Address Takeover

If DNS-based failover cannot be used due to some reason (e.g., not implemented in the client), then IP address takeover [62] can also be used (Fig. 3.4). This only works if both the primary and secondary servers are in the same subnet. Both P_1 and P_2 have identical configuration but run on different hosts on the same Ethernet. Both servers are configured to use the external master database, D_1. The slave D_2 is replicated from D_1. The clients know the server IP address as P_1's 10.1.1.1 in this example.

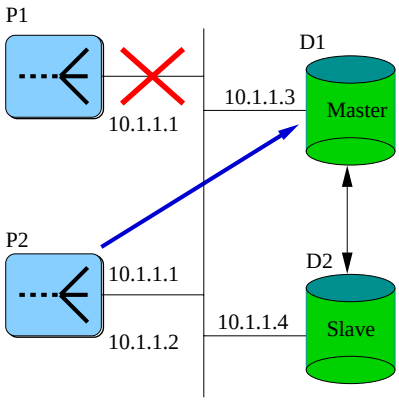


Figure 3.4: When the primary server fails

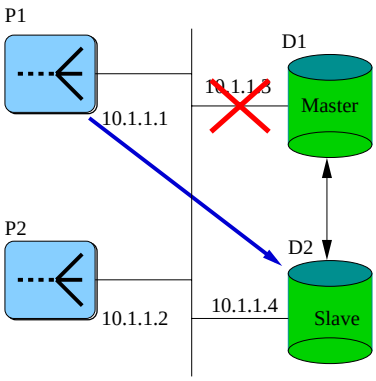


Figure 3.5: When the master database fails

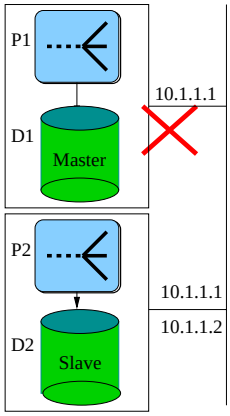


Figure 3.6: co-located database and proxy

P_2 periodically monitors the activity of P_1. When P_1 fails, P_2 takes over the IP address 10.1.1.1. Now, all requests sent to the server address will be received and processed by P_2. When D_1 fails, P_1 detects and switches to D_2 (Fig. 3.5). IP takeover is not used by D_2 since the SIP servers can be modified to switch over when D_1 fails. The ARP cache may introduce additional latency in failover.

The architecture is transparent to the rest of the network including the clients and DNS, and can be implemented without external assumptions. However, if the replication is only from the master to the slave, it requires modification in the SIP server software to first try D_1, and if that fails use D_2 so that all the updates are done to the master server. To avoid replicating the database, P_1 can propagate the REGISTER message also to P_2.

Alternatively, to avoid the server modification, the server and the associated database can

be co-located on the same host as shown in Fig. 3.6. If the primary host fails, both P_2 and D_2 take over. P_1 always uses D_1, whereas P_2 always uses D_2.

3.3.5 Reliable Server Pooling

The IETF's Reliable Server Pooling (Rserpool) working group is developing architecture and protocols for the management and operation of server pools to support highly reliable applications, and for client access mechanisms to a server pool. In the context of Rserpool architecture [73, 71], Fig. 3.7 shows the client phone as the pool user (PU), P_1 and P_2 as the pool elements (PEs) in the *SIP server pool*, and D_1 and D_2 as PEs in the *database pool*. P_1 and P_2 register with their home name server, NS_2, which supervises them, and informs the other name servers about these PEs. Similarly, D_1 and D_2 also register with the name server (NS). The SIP servers are the pool users of the database pool. A pool element is removed from the pool if it is out of service.

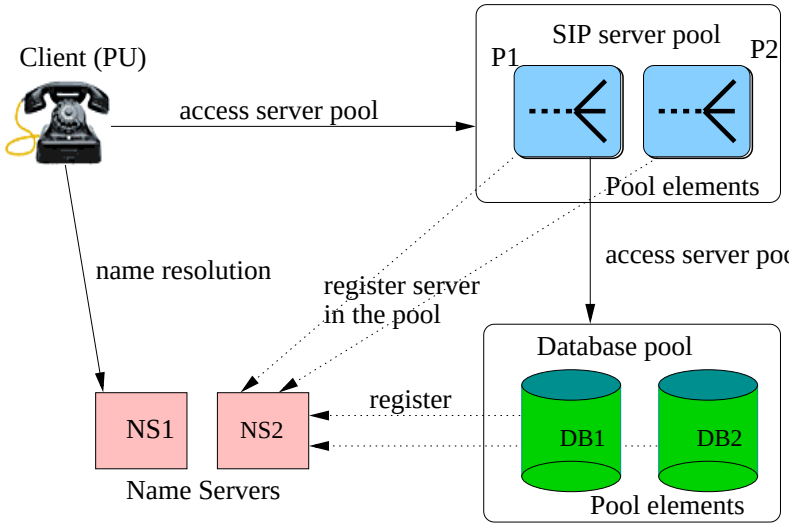


Figure 3.7: Reliable server pooling for SIP

When the client wants to contact the SIP server pool, it queries one of the name servers, NS_1, to get the list of P_1 and P_2 with relative priority for failover and load sharing. The client chooses to connect to P_1 and sends the call invitation. If P_1 fails, the client detects this and sends the message to P_2. For stateful services, P_1 can exchange state information with another server, P_2, and return the backup server, P_2, to the client in the initial message exchange. This way

the client knows which backup server to use in case of failure. P_1 can also give a signed cookie similar to HTTP cookie to the client, which sends it to the new failover server, P_2, in the initial message exchange. This is needed for call stateful services such as conferencing, but not for SIP proxy servers.

The SIP server, P_1, queries the NS to get the list, D_1 and D_2, for the "database pool". D_1 and D_2 are backed up and replicated by each other, so they can return this backup server information in the initial message exchange.

The primary limitation of this architecture is that this requires new protocol support for name resolution and server access in the clients. A translator can be used to interoperate with the clients that do not support reliable server pooling. However, this makes the translator a single point of failure between the client and the server, hence limiting the reliability. Secondly, the name space is flat unlike DNS hierarchy, and is designed for a limited scale (e.g., within an enterprise), but may be combined with wide area DNS based name resolution, for example.

3.3.6 Implementation

I have used some of the above techniques in our Columbia InterNet Extensible Multimedia Architecture (CINEMA). The architecture [88, 89] consists of our SIP server, sipd and a MySQL database for user profile and system configuration. The configuration and management are done via a web interface that accesses various CGI (Common Gateway Interface) scripts written in Tcl (Tool Command Language) [90] on the web server. All the servers may run on a single machine for an enterprise setup.

For failover, I use two sets of identical servers on two different machines as shown in Fig. 3.8. The database and SIP server share the same host. The databases are replicated using MySQL 4.0 replication [29] such that both D_1 and D_2 are master and slave of each other. MySQL propagates the binary log of the SQL commands of the master to the slave, and the slave runs these commands again to do the replication. The details of two-way replication is in Appendix B.

MySQL 4.0 does not support any locking protocol between the master and the slave to guarantee the atomicity of the distributed updates. However, the updates from the SIP server are additive, i.e., each registration from each device is one database record, so having two devices for

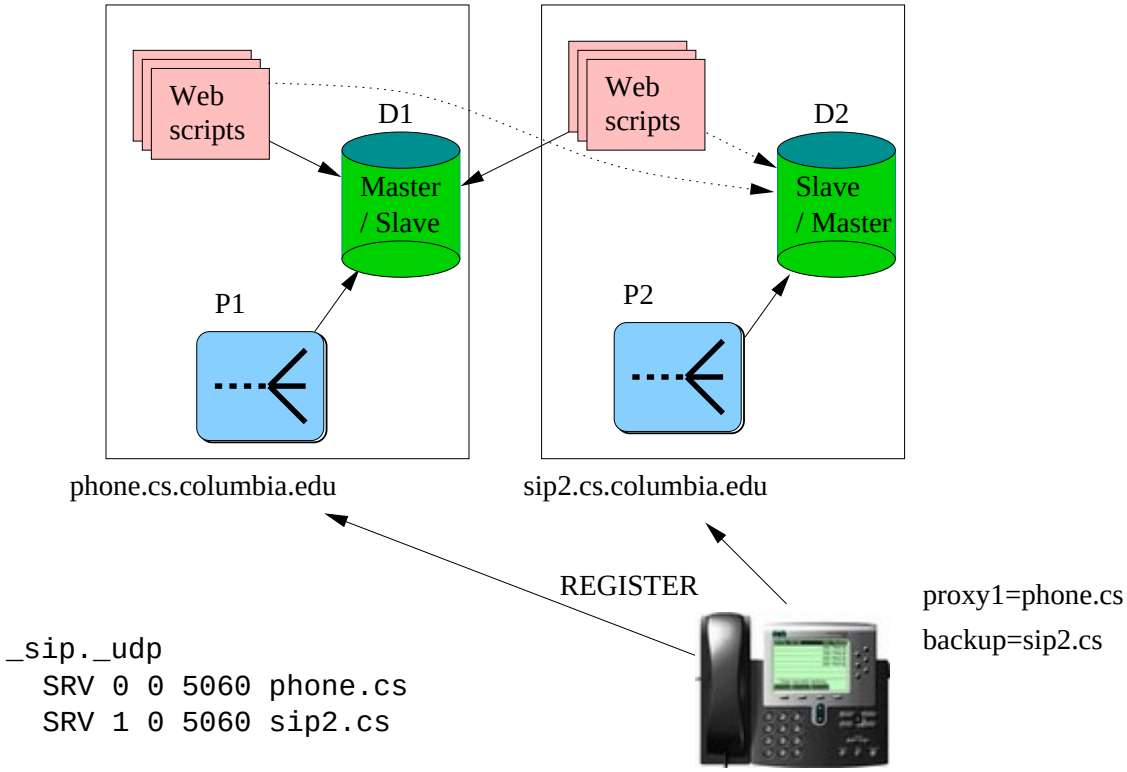


Figure 3.8: Failover in CINEMA

the same user register with two database replicas does not interfere with the other registration. For example, if *bob@home.com* registers *bob@location1.com* with D_1 and *bob@location2.com* with D_2, both D_1 and D_2 will propagate the updates to each other such that both D_1 and D_2 will have both of Bob's locations. There is a slight window of vulnerability when one contact is added from D_1 and the same contact is removed in D_2. Then, after the propagation of updates the two databases will be inconsistent with different contacts for the user. It turns out that this does not occur for the simple failover as I describe next. We can safely use the two-way replication as long as updates are done by only the SIP server.

For a simple failover case, the primary server P_1 is preferred over the secondary server P_2. So all the REGISTER requests go to P_1 and are updated in D_1. The replication happens from D_1 to D_2, not the other way. Only in the case of failure of P_1, will the update happen to D_2 through P_2. But D_1 will not be updated by the server in this case. By making sure that database becomes consistent before the failed server is brought up, we can avoid the database

inconsistency problem mentioned above.

Web scripts are used to manage user profiles and system configuration. To maintain database consistency, the web scripts should not be allowed to modify D_2 if D_1 is up. To facilitate this I modified the MySQL-Tcl client interface to accept a list of connection attributes. For example, if D_1 and D_2 are listed, then the script tries to connect to D_1 first, and if that fails then tries D_2 as shown in Fig. 3.8. For our web scripts, the short-lived TCP connection to MySQL is active as long as the CGI script is running. So the failover at the connection setup is sufficient. For long-lived connection, the implementation should be modified to provide failover even when the TCP connection breaks.

3.3.7 Analysis

The architecture provides high reliability due to redundancy. Assuming the reliability of primary and backup sets of servers as R, i.e., the probability that the server is running is R, $0 \leq R \leq 1$, the overall reliability is $(1 - (1 - R)^2)$.

Server failure affects the *call setup latency* (since the client retries the call request to the secondary server after a timeout) and the *user availability* (the probability that the user is reachable via the server given that her SIP phone is up). If the primary server is down for a longer duration, the DNS records can be updated to promote the secondary server to primary. Fig. 3.9 shows that the client retries the call after a timeout, T_R, to the secondary server if the primary server does not respond. If the individual server reliability is R, client retry timeout is T_R, and DNS time-to-live (TTL) is T_D, then the average call setup latency increases by $T_R(1-R)\mathrm{P}[t_M < T_D]$ (assuming no network delay and $R \approx 1$), where $\mathrm{P}[t_M < T_D]$ is the probability that the time, t_M (random variable), to repair the server is less than the DNS TTL. For example, if the repair time is exponentially distributed with mean T_M, then $\mathrm{P}[t_M < T_D] = 1 - e^{-\frac{T_D}{T_M}}$ assuming that the mean time to failure is much larger than the mean time to repair (i.e., $(1 - R)T_M \approx 0$). If an explicit failure feedback such as ICMP "host unreachable" is received by the client, the client tries the secondary server immediately instead of waiting for the timeout, T_R.

User availability is mostly unaffected by the primary server failure, because most registrations are REGISTER refreshes. Fig. 3.10 shows that if the primary server fails after refreshing

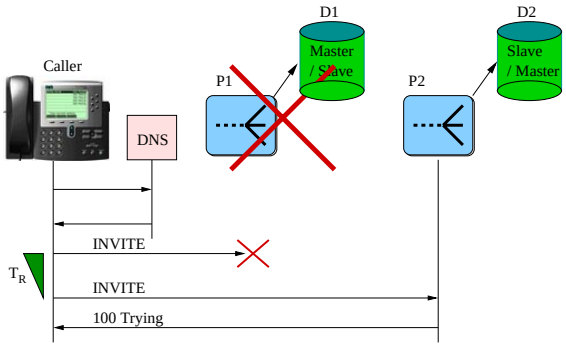


Figure 3.9: Call setup latency on failover

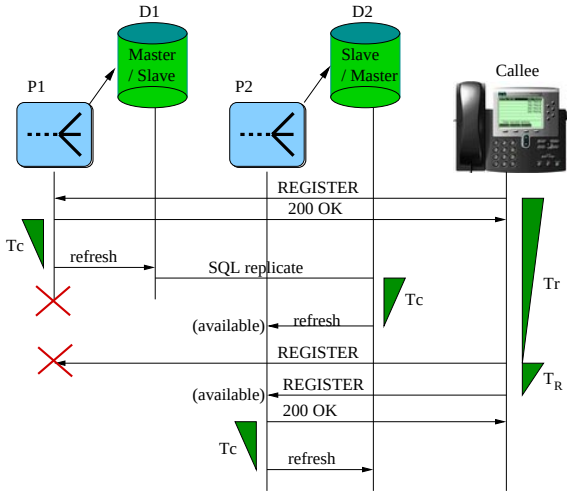


Figure 3.10: User unavailability on failure

the database with the user record, then the user record is still available on the secondary server. However, if the primary server fails after the phone registers a new contact for the first time, but before the registration is propagated to the secondary server, then the phone contact location is unreachable until the next registration refresh. In this case, assuming that the server uptime is exponentially distributed, and given the memoryless property, the time-to-failure has the same distribution. Suppose the mean-time-to-failure is T_F and the database replication latency is T_d, then the probability that the server goes down before the replication is completed (given that it is up at $t = 0$) is $\mathrm{P}[\text{lifetime} < T_d] = 1 - e^{-\frac{T_d}{T_F}}$. For example, if T_F is one week, and T_d is ten seconds, then this probablity is $0.0000165 \approx 0$. If this happens, the user record is unavailable for at most $T_r + T_R$, where T_r is the registration refresh interval (typically one hour), and T_R is client retry timeout, which is about 10 s for Cisco phones. After this time, the client refreshes the registration and updates the secondary server making the user record available.

We use an in-memory cache of user records inside the SIP server to improve its performance [88, 26]. This causes more latency in updating the user registration from P_1 to P_2. If the failure happens before the update is propagated to P_2, then it may have the old and expired record. However, in practice the phones refresh registrations much before the expiry and the problem is not visible. For example, suppose the record expires every two hours and the refresh happens every 50 minutes. Suppose P_1 receives the registration update from a phone and fails before

propagating the update to D_1. At this point, the record in D_2 has 70 minutes to expire so P_2 can still handle the calls to this phone. The next refresh happens in 50 minutes, before expiration of the record in D_2. If a new phone is setup (first time registration) just before failure of P_1, it will be unavailable until the next refresh. Suppose T_d and T_F are defined as before, and T_c is the database refresh interval, then the probability that the server goes down before the replication is completed is $1 - e^{-\frac{T_d+T_c}{T_F}}$.

Since, most of the time, the same contact information is conveyed in registration refreshes, we can reduce the number of database transactions. For example, the expiration time can be kept in memory instead of propagating to the database. When the registration is deleted, expired or changed in the memory, the information is propagated to the database. Thus, the database traffic is reduced considerably. This mechanism can be used for $N + 1$ servers where one server can act as a backup for N primary servers. Since the load on the backup server is considerably lower, this works well. One disadvantage is that if the server fails, the database will still have the expired user registrations, because the expiration is not stored in the database. We have not implemented this mechanism.

With the Cisco phone [86] that has the primary and backup proxy address options (Section 3.3.1), the phone registers with both P_1 and P_2. Both D_1 and D_2 propagate the same contact location change to each other. However, since the contact record is keyed on the user identifier and contact location, the second write just overrides the first write without any other side effect. Alternatively, the server can be modified to perform an immediate synchronization between the in-memory cache and external database if the server is not heavily loaded.

The two-way replication can be extended to more servers by using circular replication such as D_1-D_2-D_3-D_1 using the MySQL master/slave configuration [29]. Thus, if each server reliability is only 98%, a three-way replication gives the total reliability of $1-0.02^3 = 0.999992$, i.e., "5 nines". To provide failover of individual servers (e.g., D_1 fails but not P_1), the SIP server P_1 should switch to D_2 if D_1 is not available.

3.4 Scalability: Load Sharing

In failover, the backup server takes over in the case of failure whereas in load sharing all the redundant servers are active and distribute the load among them. Some of the failover techniques can also be extended to load sharing.

3.4.1 Network Address Translation

A network address translator (NAT) device can expose a unique public address as the server address and distribute the incoming traffic to one of the several internal private hosts running the SIP servers [91]. Eventually, the NAT itself becomes the bottleneck making the architecture inefficient. Moreover, the transaction-stateful nature of SIP servers require that subsequent retransmissions should be handled by the same internal server. So the NAT needs to maintain the transaction state for the duration of the transaction, further limiting scalability.

3.4.2 Multiple Servers with the Same IP Address

In this approach, all the redundant servers in the same broadcast network (e.g., Ethernet) use the same IP address. The router on the subnet is configured to forward the incoming packets to one of these servers' MAC address. The router can use various algorithms such as "round robin" or "response time from server" to choose the least loaded server.

To avoid storing SIP transaction states in the subnet router, this method is only recommended for stateless SIP proxies that use only UDP transport and treat each request as independent without maintaining any transaction state.

In Section , we describe our two-stage architecture. In the absence of DNS SRV and NAPTR, we can use the same IP address method for the first stage (Fig. 3.13) in the two stage architecture. The same IP address method is less efficient since the network bandwidth of this subnet may limit the number of servers in the cluster. Moreover, this method does not work if the network itself is unreachable. Hence, DNS-based load sharing is recommended.

3.4.3 DNS-based Load Sharing

The DNS SRV [67] and NAPTR [68] mechanisms can be used for load sharing using the priority and weight fields in these resource records [28], as shown below:

```
example.com
_sip._udp 0 40 a.example.com
          0 40 b.example.com
          0 20 c.example.com
          1  0 backup.somewhere.com
```

The above DNS SRV entry indicates that the servers `a`, `b`, `c` should be used if possible (priority 0), with `backup.somewhere.com` as the backup server (priority 1) for failover. Within the three primary servers, `a` and `b` are to receive a combined total of 80% of the requests, while `c`, presumably a slower server, should get the remaining 20%. Clients can use weighted randomization to achieve this distribution.

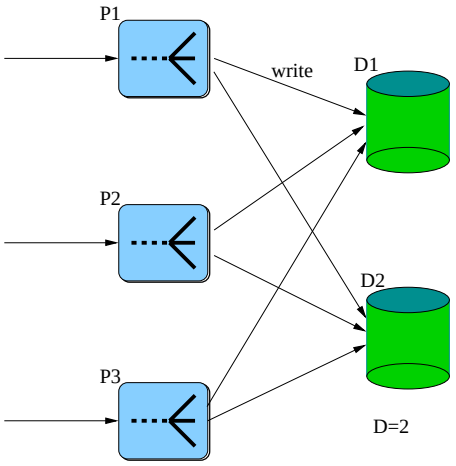


Figure 3.11: DNS-based

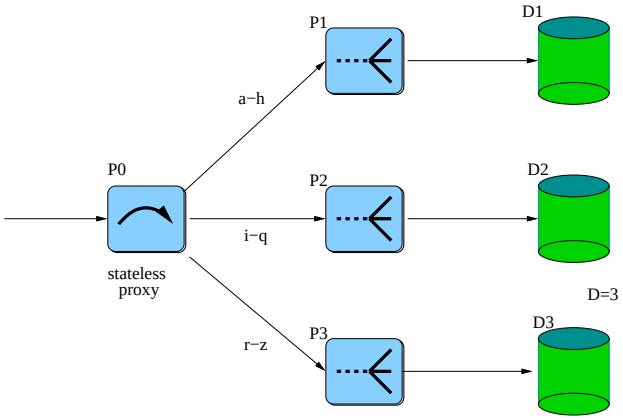


Figure 3.12: Identifier-based load sharing

However, simple random distribution of requests is not sufficient since the servers need to access the same registration information. Thus, in the example above, each server would have to replicate incoming REGISTER requests to all other servers or update the common shared and replicated database(s). In either case, the updates triggered by REGISTER quickly become the bottleneck. The SIP phones typically do REGISTER refresh once an hour. Thus, for a wireless operator with one million subscribers, it has to process about $\frac{10^6}{3600} = 280$ updates per second.

Fig. 3.11 shows an example with three redundant servers and two redundant databases. For every REGISTER, it performs one read and one write in the database. For every INVITE-based call request, it performs one read from the database. Every write should be propagated to all the D databases, whereas a read can be done from any available database. Suppose there are N writes and $r \cdot N$ reads, and if the same number of INVITE and REGISTER are processed then $r = 2$. Suppose, the database write takes T units of time, and database read takes $t \cdot T$ units. Total time per database will be $(\frac{tr}{D} + 1)TN$. This shows that no matter how many servers are used, the performance is limited by the write capacity of one database. For example, even with very large D, the total time is at least TN for N writes, limiting the cluster performance to $\frac{1}{T}$ registrations per second.

The architecture in Fig. 3.11 also provides high reliability due to redundancy. Assuming that the mean-time-to-repair is much less than mean-time-to-failure, and the reliability of individual proxy server is R_p and database server is R_d, and suppose there are P proxy servers and D database servers, the reliability of the system becomes $(1 - (1 - R_p)^P)(1 - (1 - R_d)^D)$. The reliability increases with increasing D and P.

3.4.4 Identifier-based Load Sharing

For identifier-based load sharing (Fig. 3.12), the user identifier space is divided into multiple non-overlapping groups. A hash function maps the destination user identifier to the particular group that handles the user record. The example in Fig. 3.12 uses the hash function based on the first letter of the user identifier. For example, P_1 handles `a-h`, P_2 handles `i-q` and P_3 handles `r-z`. A high speed first stage server (P_0), proxies the call request to P_1, P_2 and P_3 based on the destination user identifier without contacting any database. If a call is received for destination *bob@home.com* it goes to P_1, whereas *sam@home.com* goes to P_3. Each server in the second stage has its own database and does not need to interact with the others. To guarantee almost uniform distribution of call requests to different servers, a better hashing algorithm such as SHA1 can be used or the groups can be re-assigned dynamically based on the load.

Suppose N, D, T, t and r are as defined in the previous section. Since each read and write operation is limited to one database and assuming uniform distribution of requests to the

different servers, total time per database will be $(\frac{tr+1}{D})TN$. With increasing D, this scales better than the previous method. Since the writes do not have to be propagated to all the databases and the database can be co-located on the same host with the proxy, it reduces the internal network traffic.

However, because of lack of redundancy this architecture does not improve system reliability. Assuming that the mean-time-to-repair is much less than mean-time-to-failure, and the reliability of the first stage proxy, second stage proxy and database server as R_0, R_p and R_d, and suppose there are D groups, then the system reliability becomes $R_0 \cdot (R_p)^D \cdot (R_d)^D$. The least reliable component affects the system reliability the most and the reliability decreases as D increases.

The only bottleneck may be the first stage proxy. We observed that the stateful performance is roughly similar to stateless performance (Section 3.5), hence a single stateless load balancing proxy may not work well in practice. We use a cluster of proxies in the first stage as described next.

3.4.5 Two-stage Reliable and Scalable Architecture

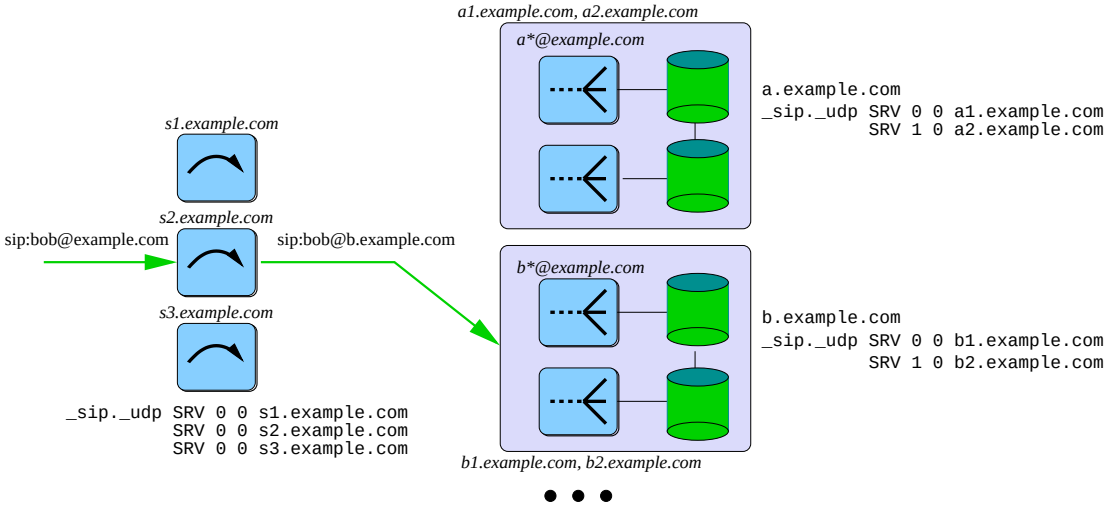


Figure 3.13: Two-stage reliable and scalable architecture

Since none of the mechanisms above are sufficiently general or infinitely scalable, we propose to combine the two methods (Fig. 3.11 and 3.12) in a two-stage scaling architecture

(Fig. 3.13) to improve both reliability and scalability. The first set of proxy servers selected via DNS NAPTR and SRV performs request routing to the particular second-stage cluster based on the hash of the destination user identifier. The cluster member is again determined via DNS. The second-stage server performs the actual request processing. Adding an additional stage does not affect the audio delay, since the media path (usually directly between the SIP phones) is independent of the signaling path. Use of DNS does not require the servers to be co-located, thus allowing geographically distributed cluster. This provides availability even if the networks of some of the servers in the cluster is unreachable.

Note that the first stage server uses the destination user identifier to select the second stage server only for an inbound request to an user in this domain. In a SIP call setup, an optional outbound proxy may be used by the provider to apply policy decisions such as billing to all the outbound calls by the users in the domain. If the two-stage cluster is acting as an outbound proxy of the domain, then the first stage server selects the second stage server based on the hash of the source user identifier instead of the destination user identifier.

Suppose there are S first stage proxy servers, P clusters in the second stage, and B proxy and database servers in each cluster. The second stage cluster has one primary server and $B-1$ backups. All the databases in a cluster are replicated using circular replication. Suppose the REGISTER message arrivals are uniformly distributed (because of the uniform registration refresh rate by most user agents) with mean λ_R and INVITE (or other requests that need to be proxied such as MESSAGE) arrivals are Poisson distributed with mean λ_P, such that the total request rate is λ=λ_R+λ_P. Suppose the constant service rates of first stage server be μ_s, and the second stage server be μ_r and μ_p for registration and proxying, respectively. We assume a hash function so that each cluster's arrival rate is approximately $\frac{\lambda}{B}$. Note that Fig. 3.8 is a special case where S=0, P=1 and B=2. Similarly, Fig. 3.12 is a special case where S=B=1.

The goal is to quantitatively derive the relationship between different service parameters (μ), system load (λ), and redundancy parameters (S, B, P). We want to answer the questions such as (1) when is first stage proxy needed, and (2) what are the optimal values for redundancy parameters to achieve a given scalability. Our goal is to achieve carrier grade scalability (10 million BHCA) using commodity hardware. I provide our performance measurement results for

scalability parameters (S and P) and system load (λ) in the next section.

Suppose each server is R =99% reliable, and $S = P = B = 3$, then overall system reliability is $(1-(1-R)^S) \cdot (1-(1-R)^B)^P$ = 99.9996%, i.e., "five nines".

We do not consider the case of load sharing by different proxies in the same cluster, because load sharing is better achieved by creating more clusters. For handling sudden load spikes within one cluster, the DotSlash on-demand rescue system [92] is more appropriate where a backup server in the same or another cluster temporarily shares the load with the primary server of the overloaded cluster.

3.5 Performance Evaluation

In this section, I quantitatively evaluate the performance of our two-stage architecture for scalability using our SIP registration and proxy server, sipd, and SIPstone test suite [9].

3.5.1 Test Setup

I performed the SIPstone Proxy 200 tests, over UDP. The SIPstone test suite has *loaders* and *call handlers*, to generate SIP requests and to respond to incoming requests, respectively. The server under test (SUT) is a two-stage cluster of our SIP servers, sipd, implementing the reactive system model [26]. An example test setup is shown in Fig. 3.14. Each instance of sipd was run on a dedicated host with Pentium 4, 3 GHz CPU, on a 800 MHz motherboard, with 1 GB of memory, running Redhat Linux 2.4.20. The hosts communicated over a lightly loaded 100base-T Ethernet connection. A single external MySQL database, running version 3.23.52 of the MySQL server was shared by all the sipd instances. But this is not an issue because the Proxy 200 test does not modify the database, but uses the in-memory cache of sipd [88].

To focus on only the scalability aspects I used one server in each group of the second stage (Fig. 3.13, B=1). I use the convention S_nP_m to represent n first stage servers, and m second stage groups with one server per group. S_0P_1 is same as a single SIP proxy server without any first stage load balancer.

On startup, a number of call handlers (in our tests, four) register a number of destination

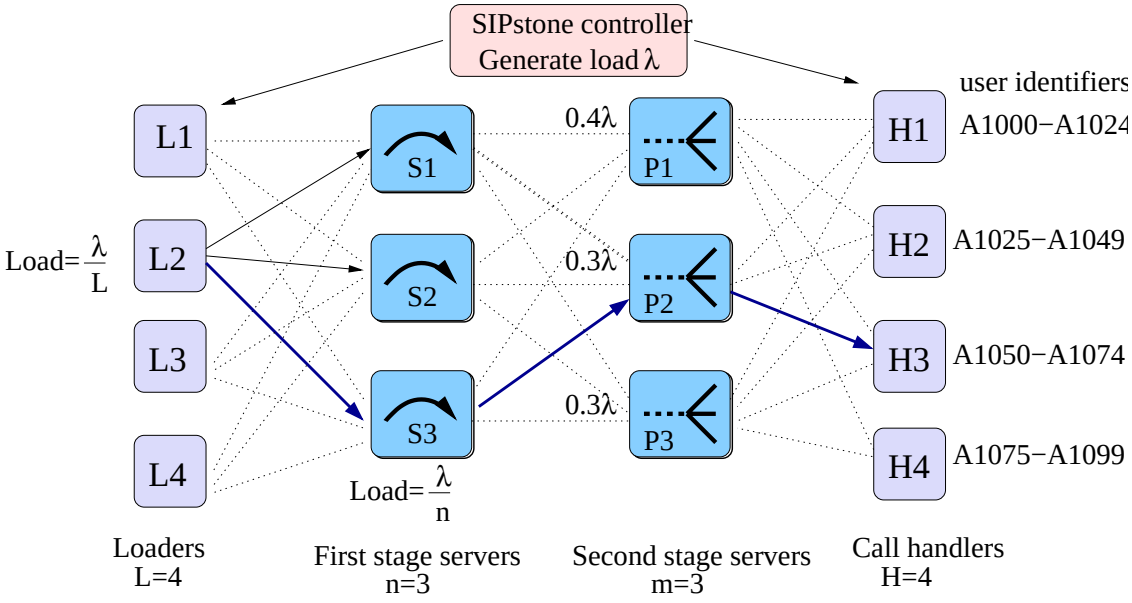


Figure 3.14: Example test setup for S3P3

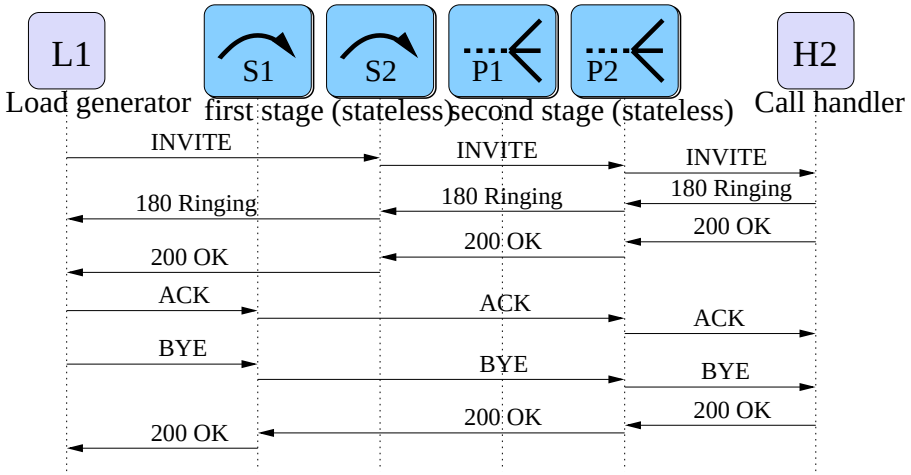


Figure 3.15: Example message flow for S2P2: in the first stage INVITE goes via S2, whereas ACK and BYE via S1, but in the second stage all the requests go via P2 based on the consistent hash of the destination user identifier.

locations (from non-overlapping user identifier sets as shown in Fig. 3.14) with the proxy server. Then for the Proxy 200 test, a number of loaders (in our tests, four) send SIP INVITE requests using Poisson distribution for call generation to the SUT, randomly selecting from among the registered addresses as shown in Fig. 3.15. If there is more than one first stage server ($n > 1$), then

the loader randomly selects one of the first stage servers. The first stage server proxies the request to one of the second stage servers based on the destination user identifier. The second stage server forwards each request to the appropriate call handler responsible for this user identifier. The call handler immediately responds with 180 Ringing and 200 OK messages. These are forwarded back to the load generators in the reverse path. Upon receiving the 200 OK response, the load generator sends an ACK message for the initial transaction and a BYE request for a new transaction. The BYE is similarly forwarded to the call handler via the two-stage servers to reflect the record-route behavior in real operational conditions [9]. The call handler again responds with 200 OK. If the 200 OK response is not received by the loader within two seconds, or if any other behavior occurs, then the test is considered a failure. The loader generates the request for one minute for a given request rate. The server is then restarted, and the test is repeated for a higher request rate. I used an increment of 100 calls per second (CPS).

This process is repeated until 50% or more of the tests fail. Although [9] requires 95% success, I measured until 50% to show that the throughput is stable at higher loads. There is no retransmission on failure [9]. The complete process is repeated for different values of n and m in the cluster configuration, S_nP_m.

3.5.2 Analysis

Fig. 3.16 compares the performance of the different S_nP_m configurations. It shows the average of three experiments for each configuration at various call rates. A single sipd server handles about 900 calls/second (CPS) (see S_0P_1 in Fig. 3.16), which corresponds to about three million BHCA. When the load is more than the server capacity, the throughput remains almost constant at about 900 CPS. When the throughput is same as load, i.e., 100% success rate, the graph is a straight line. Once the throughput reaches the capacity (900 CPS), the graph for S_0P_1 flattens indicating lower success rate for higher load. At a load of 1800 CPS, the system gives only 50% success rate (i.e., throughput is half of load), and the experiment stops. Note that for all practical purposes, success rate of close to 100% is desired.

When the server is overloaded, the CPU utilization is close to 100%. Introducing an extra server in the second stage and having a first stage load balancing proxy puts the bottleneck on

the first stage server which has a capacity of about 1050 CPS (S_1P_2 in Fig. 3.16). An additional server in the first stage (S_2P_2) gives the throughput of approximately double the single second stage server capacity. Similarly, S_3P_3 has capacity of approximately 2800 CPS which is about three times the capacity of the single second stage server, and S_2P_3 has capacity of 2100 CPS which is double the capacity of the single first-stage server.

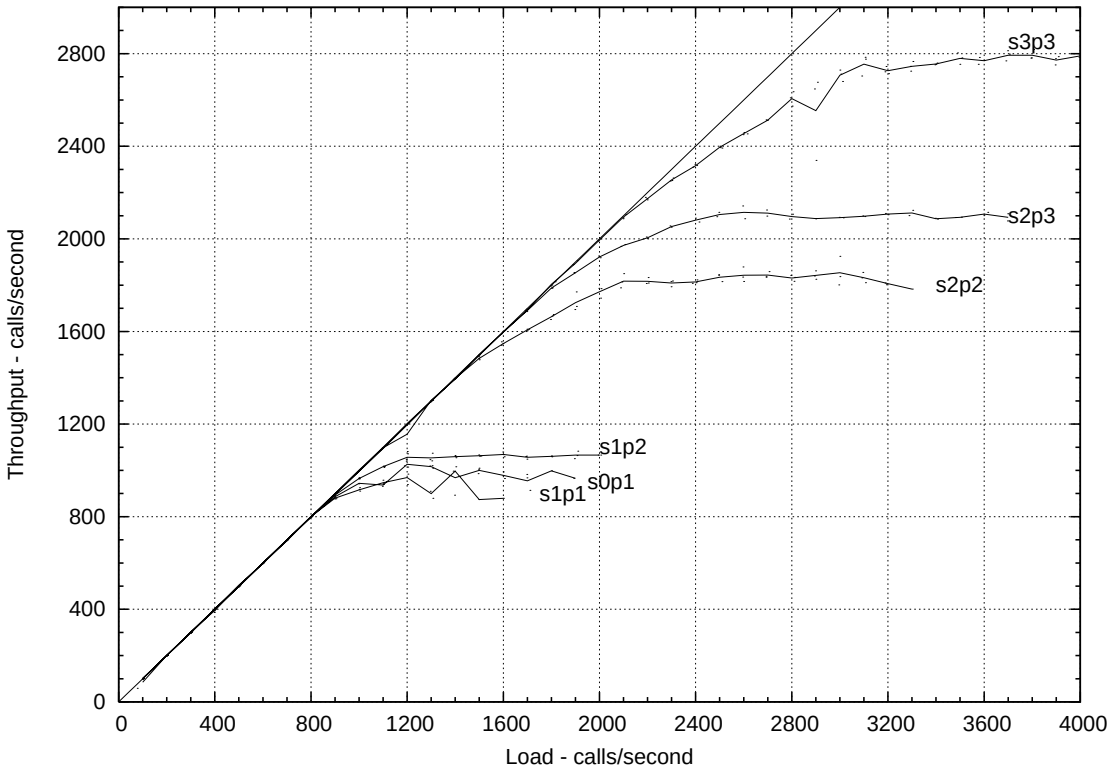


Figure 3.16: Server throughput in S_nP_m configuration (n first stage and m second stage servers. The results show that the performance increases linearly with the number of servers, i.e., s2p2 is twice and s3p3 is thrice that of s1p1 and s0p1 performance.)

The results show that we can achieve linear scaling by putting more servers in the first and second stages in our architecture. Below, I present the theoretical analysis for the two-stage architecture.

Suppose the first and second stage servers in S_nP_m have capacity of C_s and C_p, respectively (usually, $C_s \geq C_p$). The servers are denoted as S_i and P_j, $1 \leq i \leq n$, $1 \leq j \leq m$, for the first and second stage, respectively. Suppose the incoming calls arrive at an average rate λ,

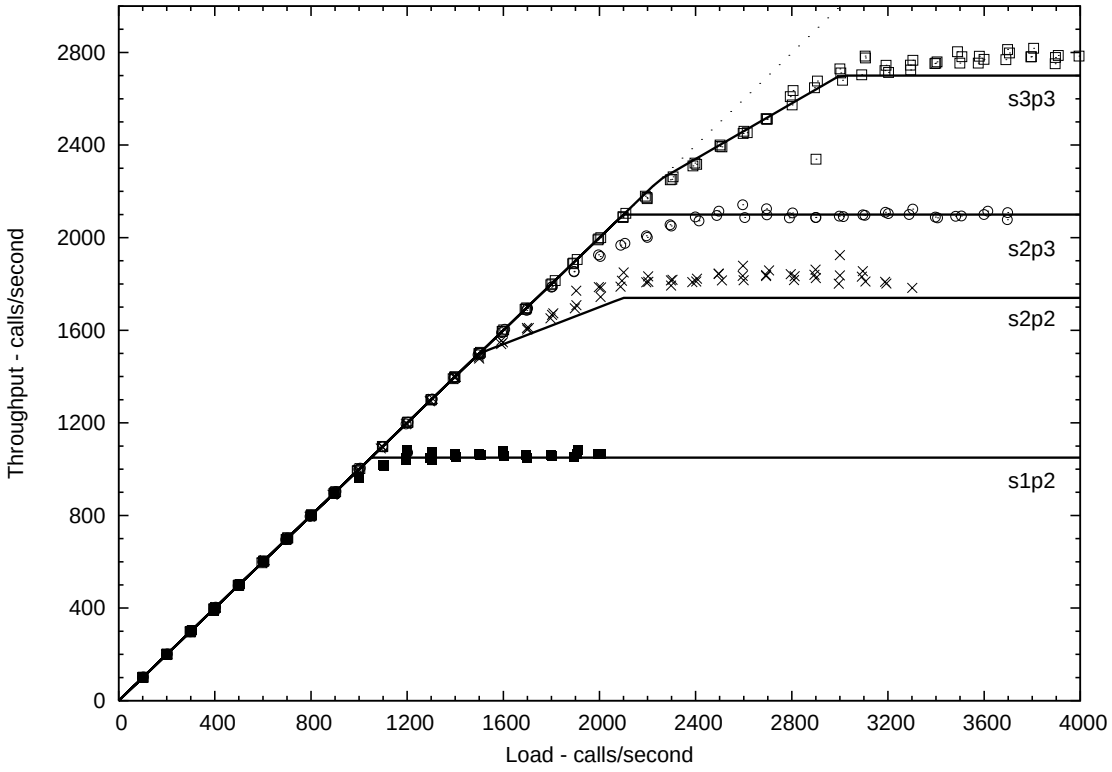


Figure 3.17: Theoretical and experimental capacity for configuration $S_n P_m$

with exponential inter-arrival time. Suppose the load is uniformly distributed among all the n first stage servers, so the each first stage server gets a request rate of $\frac{\lambda}{n}$. Suppose the hash function distributes the requests to the second stage server such that the i^{th} server, P_i, gets a fraction, f_i, of the calls (Note that $\sum f_i = 1$). Assuming that all the users are equally likely to get called, and the hash function uniformly distributes the user identifiers among the second stage servers, then all f_i will be same (i.e., $f_i = \frac{1}{n}$). However, differences in the number of incoming calls for different users will cause non-uniform distribution.

The throughput, τ, at a given load, λ, is the combined throughput of the two stages. The throughput of the first stage is $\lambda' = \min(\lambda, nC_s)$, which is load (input) to the second stage. The server, P_j, in the second stage has throughput of $\min(\lambda' f_j, C_p)$. Thus,

$$\tau(\lambda) = \sum_{j=1}^{m} \min(f_j \min(\lambda, nC_s), C_p)$$

Without loss of generality, we assume that $f_i \geq f_j$ for $i > j$. The resulting throughput

vs load graph is given by $m+1$ line segments, L_i: $(\lambda_i, \tau_i) \rightarrow (\lambda_{i+1}, \tau_{i+1})$, for i=0 to m, where (λ_k, τ_k) is given as follows:

$$\begin{aligned}
(0,\ 0) \quad & for\ k = 0 \\
(\tfrac{C_p}{f_k},\ \tau_{k-1} + (\lambda_k - \lambda_{k-1})F_k) \quad & for\ 1 \le k \le m; f_k \ge \tfrac{C_p}{nC_s} \\
(nC_s,\ \tau_{k-1} + (\lambda_k - \lambda_{k-1})F_k) \quad & for\ 1 \le k \le m; f_k < \tfrac{C_p}{nC_s} \\
(\infty,\ \tau_m) \quad & for\ k = m+1 \\
where\ F_k = (1 - \textstyle\sum_{i=k}^{m} f_i) &
\end{aligned}$$

The initial line segment represents 100% success rate with slope 1. Note that the region beyond 100% success rate is not of practical interest. At the request load of $\frac{C_p}{f_1}$, server P_1 reaches its capacity and drops any additional request load. So the capacity increases at rate equal to the remaining fraction of requests that go to the other non-overloaded servers, P_k, $k = 2, 3, ..., m$. This gives the slope $F_1 = (1 - (f_2 + f_3 + ... + f_m))$ for the second line segment. Similarly, P_2 reaches its capacity at load $\frac{C_p}{f_2}$, and so on. When all the second stage servers are overloaded the throughput remains constant, giving the last line segment. At the request load of nC_s, all the first stage servers, S_i, reach their capacity limit. If the second stage server P_j's capacity, C_p is more than the load it receives at that time, $f_j(nC_s)$, then the system throughput is not limited by P_j.

I used a set of hundred user identifiers for test. The hash function I used distributed these identifiers as follows: for $m = 2$, f is roughly {0.6, 0.4}, and for $m = 3$, f is roughly {0.4, 0.3, 0.3}. Note that with 1000 or 10,000 user identifiers, the same hash function distributed the set more uniformly as expected, but our skewed distribution of hundred identifiers helps us verify the results assuming a non-uniform call distribution for different users. The capacity of C_s and C_p are 900 CPS and 1050 CPS, respectively. The resulting theoretical performance is shown in Fig. 3.17 for S_1P_2, S_2P_2, S_2P_3 and S_3P_3 with a system capacity of 1050, 1740, 2100 and 2700 CPS, respectively. Although S_2P_2's second stage can handle $900 \times 2 = 1800$ CPS, the throughput of the first stage is only $1050 \times 2 = 2100$, out of which 60% (i.e., 1260 CPS) goes to $P1$ which drops $1260 - 900 = 360$ CPS. So the system throughput is $2100 - 360 = 1740$ CPS. Our experimental results are plotted as data points (not average, but individual throughput values) in the same graph for comparison.

3.5.3 Non-uniform Call Distribution

If the call requests to the user population among the different second stage servers is non-uniformly distributed, then the system starts dropping the call requests at a load lower than the combined capacity of the second stage servers. To prevent this, the user data should be redistributed among the second stage servers to provide an uniform distribution on an average, e.g., by changing the hash function. Fig. 3.18 compares the two experiments for the S_2P_2 configuration: one with the earlier skewed hash function that distributed the user identifiers in ratio 60:40 and another hash function (Bernstein's hash [93]), that distributed the user identifiers in ratio 50:50. For uniform call distribution, the graph shows 100% success rate until about the peak capacity of 1800 CPS, followed by a flat throughput. This is the ideal behavior for uniform call distribution.

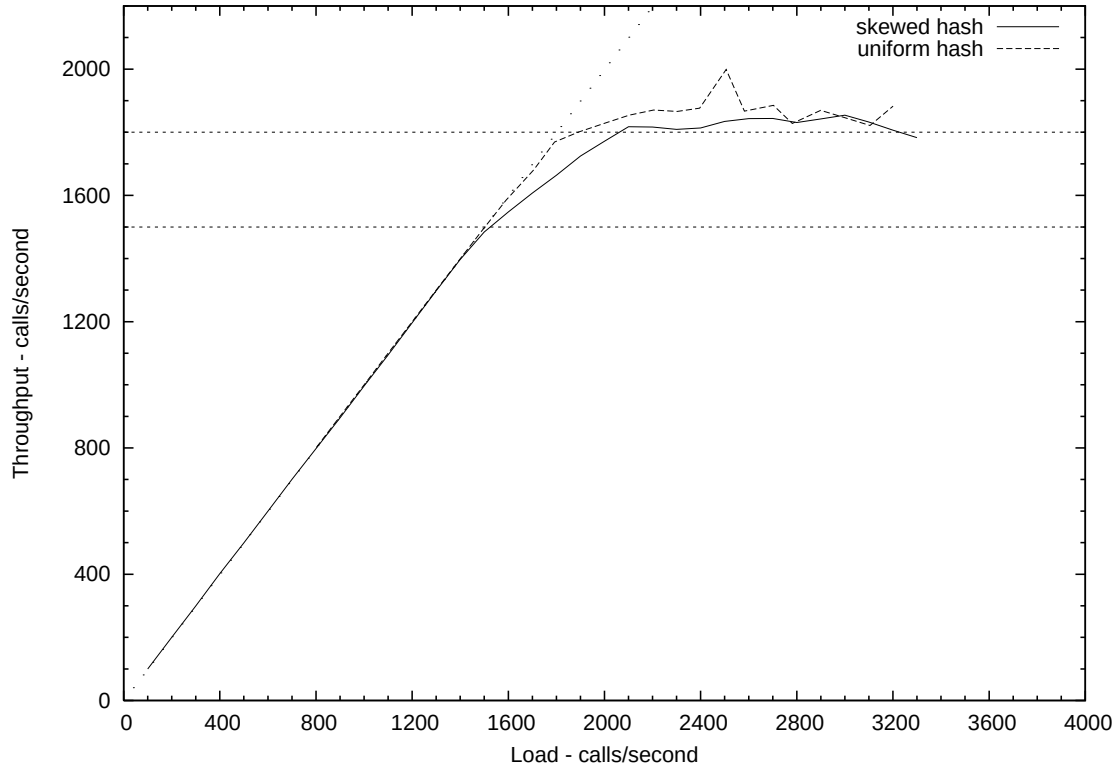


Figure 3.18: Effect of user identifier distribution among second stage servers for S2P22. Uniform distribution gives the best performance, i.e., success rate is close to 100% until the peak performance (1800 CPS), whereas for non-uniform distribution the success rate reduces as soon as one of the server is overloaded (at 1500 CPS).

If the number of second-stage groups changes frequently, e.g., due to failure or maintenance, then a consistent hashing function [94] is desirable as it avoids large redistributions of the user identifiers among the servers.

3.5.4 Performance of Stateful Proxy

So far I have shown the test results using the stateless proxy mode. A SIP request over UDP that needs to be proxied to only one destination (i.e., no request forking), can be proxied statelessly. Our SIP server, sipd, can be configured to try the stateless mode, if possible, for every request that needs to be proxied. If a request could *not* be proxied statelessly, sipd falls back to the transaction stateful mode for that request. Stateful mode requires more processing and state in the server, e.g., for matching the responses against the request.

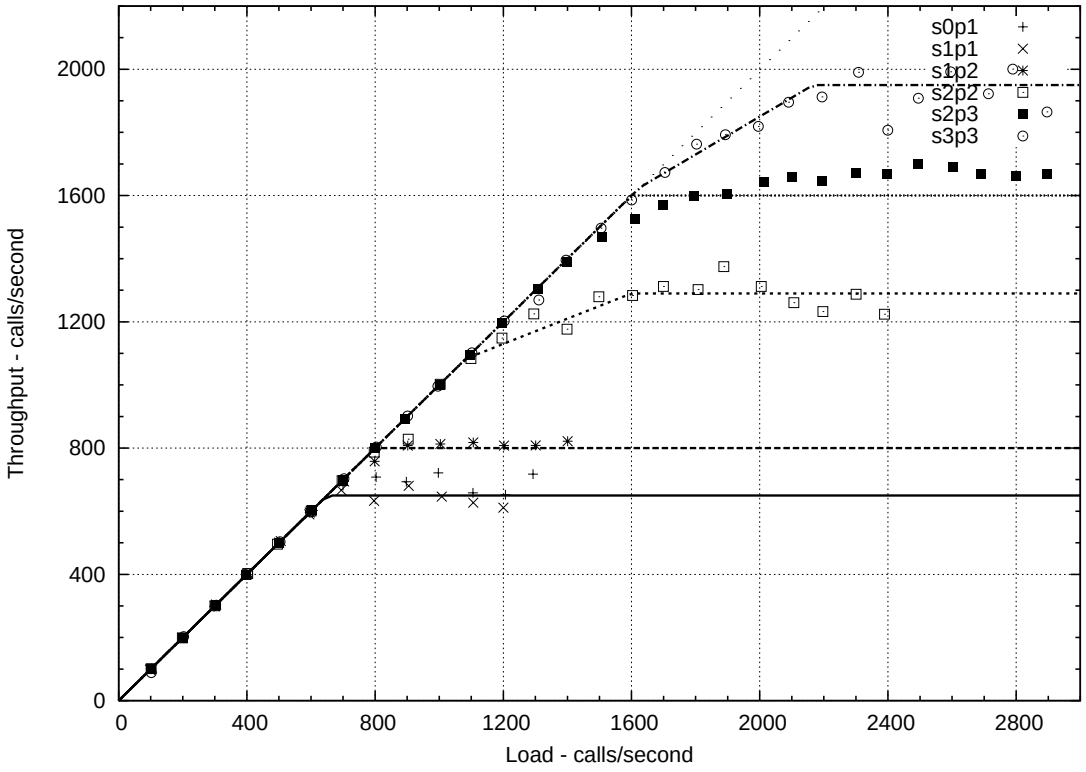


Figure 3.19: Performance of S_nP_m with stateful proxy in second stage. The results show that the performance increases linearly with the number of servers, i.e., s2p2 is twice and s3p3 is thrice that of s1p1 and s0p1 performance.

I ran one experiment by disabling the stateless proxy mode in the second stage. Fig. 3.19 shows the experimental results along with the theoretical throughput using the earlier hash function. The first and second stage server capacities are C=800 and C'=650 CPS, respectively. The first stage server capacity is less if the second stage is stateful (800 CPS) compared to the case when the second stage is stateless (1050 CPS), because the stateful second stage server generates two additional 100 Trying SIP responses for INVITE and BYE in a call that increases the number of messages handled by the first stage server (See Fig. 3.20 and 3.15). If a fraction, f_s, of the input load needs to be handled using stateful mode (e.g., due to request forking to multiple callee devices), then the effective server capacity becomes $(1 - f_s)C + f_sC'$.

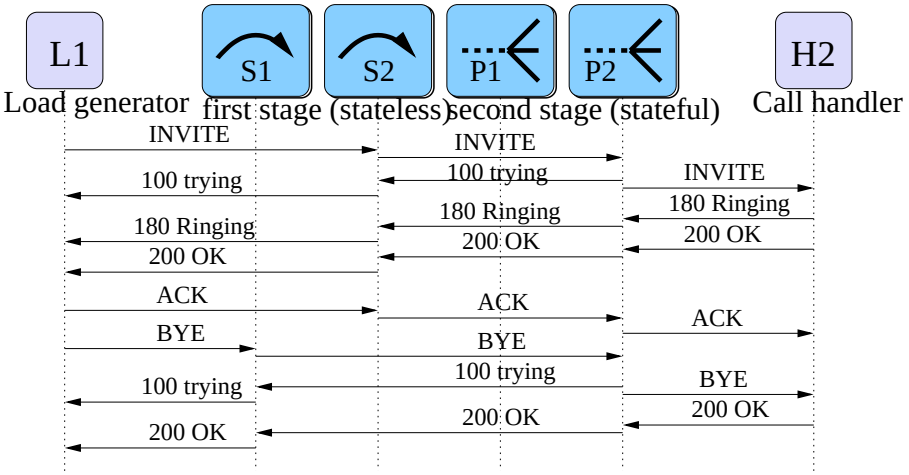


Figure 3.20: Stateful proxy message flow

Our more recent optimizations enhance the single second stage server throughput to 1200 CPS and 1600 CPS for stateful and stateless proxy, respectively, as shown in Section 3.6.

3.5.5 Effect of DNS Lookups

In some of our initial experiments not reported in this thesis, the call handler registered the DNS host name with the proxy server so that the server performed a DNS lookup for locating the call handler host. We observed comparatively poor performance, e.g., a single proxy server capacity with DNS was 110 CPS on the same hardware, compared to 900 CPS without DNS. There were two problems in our implementation: (1) it used a blocking DNS resolver that waits for the query

to complete so the internal request queue builds up if the DNS latency is more than the average interarrival duration; and (2) it did not implement any host-cache for DNS, so the second stage server did DNS lookup for every call request. We also observed some fluctuations in throughput even before the server reached its capacity. This was due to the fact that the DNS server was not in the same network, and the DNS lookup procedure took between 10 to 25 ms for each call. In our tests, sipd sent about 28 DNS queries for each call due to multiple resolver search domains (six in our tests) and DNS records (e.g., sipd tries NAPTR, SRV and A records, falling back in that order) used in the implementation.

> For example, if the destination domain is example.com, DNS query is sent for (1) NAPTR record for example.com, (2) SRV record for _sip._udp.example.com, (3) SRV record for sip.udp.example.com, and finally (4) A record for example.com, in that order in our implementation. Since the Linux machine used for testing had six different search domains, seven queries are sent sequentially for each type of record, resulting in 28 queries. For example, if search domains are columbia.edu, and cs.columbia.edu, then a query for example.com generates three DNS queries for example.com, example.com.columbia.edu, and example.com.cs.columbia.edu, sequentially if the previous query fails. Thus, if NAPTR and SRV records do not exist for example.com, this results in six wasted queries, before a successful query for DNS A record of example.com.

I implemented a simple DNS host-cache in sipd and observed same performance as that without DNS (i.e., 900 CPS for single second stage server). In practice, the first-stage servers access records for the second-stage servers within the same domain, thus, doing localized DNS queries in the domain. It will be interesting to measure the host-cache performance for the real callee host names by the second-stage servers, instead of a few call handler host names that were cached after the first lookups until the end of the test run in our tests. One can use an event-based DNS resolver such as `adns` [95] to improve the performance and eliminate the potential bottleneck due to DNS access. Another common technique is to use a local resolver that implements a DNS cache.

3.5.6 Other SIPstone Tests

We also performed one experiment with Registration test without authentication. The performance is shown in Fig. 3.21 along with the expected throughput values. We used capacity values as C_s=2500 registrations/second (RPS) and C_p=2400 RPS for first and second stage servers, respectively. Authentication requires two transactions, thus reducing the capacity to half. Thus, the S_3P_3 configuration will be able to support more than 10 million subscribers assuming one hour registration refresh interval.

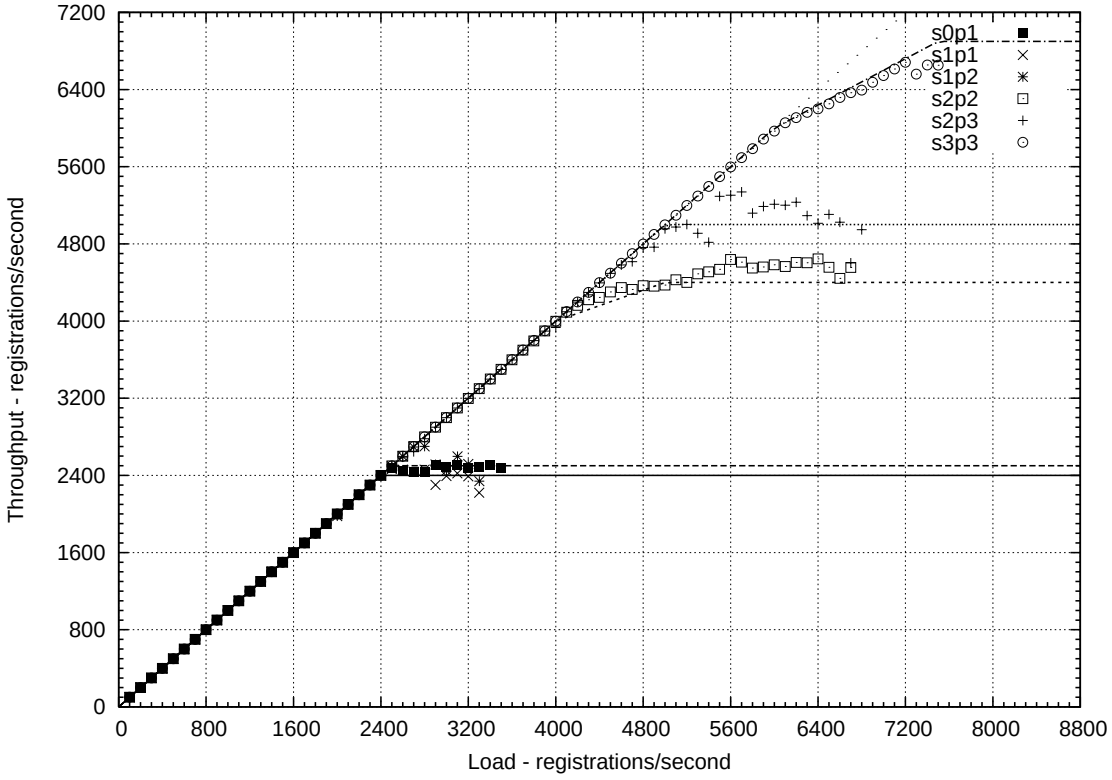


Figure 3.21: Performance for S_nP_m with registration server in second stage. The results show that the performance increases linearly with the number of servers, i.e., s2p2 is twice and s3p3 is thrice that of s1p1 and s0p1 performance.

Note that the second stage registrar is always stateful. Moreover, we used the database refresh rate to be more than the test duration, thus, removing the database synchronization variable from the results. The first stage proxy server capacity for the registration test is more because the number of messages per transaction that it handles is two in the registration test compared to six

in the Proxy 200 test (see Fig. 3.22 and 3.15).

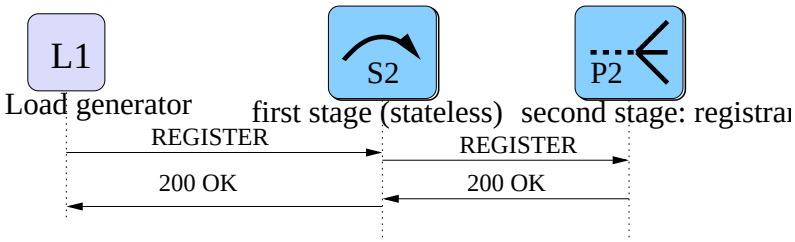


Figure 3.22: REGISTER message flow

The Proxy 200 test determines the BHCA (busy hour call attempts) metric, whereas the registration test determines the number of registered subscribers for the system.

3.6 Server Architecture

There are two components in providing high capacity IP telephony services: network-related components such as bandwidth, server location and load sharing, and server-related components such as server hardware (CPU, memory), features vs. performance tradeoff, non-blocking I/O and software architecture. In general, scaling any Internet service involves individual server performance enhancements and load distribution among multiple servers. We described and evaluated SIP load sharing in Sections 3.4 and 3.5. This section deals with performance enhancements on an individual SIP server on commodity server hardware. In particular, we evaluate the effect of software architecture - events, threads or processes - for the SIP proxy server. We try to answer the following questions: (1) For a SIP-style server, which of the basic architectures is likely to perform better in a given situation? (2) Does performance scale with CPU speed or is it memory dominated? (3) What can be done to improve the performance on a multiprocessor machine?

We built a very basic SIP server in different software architectures using the same set of libraries for SIP processing. This helps us in understanding the effect of the server architecture on performance. The server includes a parser module and has many simplifications such as memory-only lookups without any database write-through, no SIP Route header handling, minimal configuration, only UDP transport (i.e., no TCP or TLS), no programmable scripts, and no user authentication. We used standard POSIX threads, which map to kernel-level threads on

Solaris and Linux. On a multi-processor hardware, concurrency is utilized via multi-threading and multi-processing software architecture. Our goal is to use commodity hardware without any custom tweaks, hence optimized user-level threads and CPU scheduler reconfiguration are not investigated.

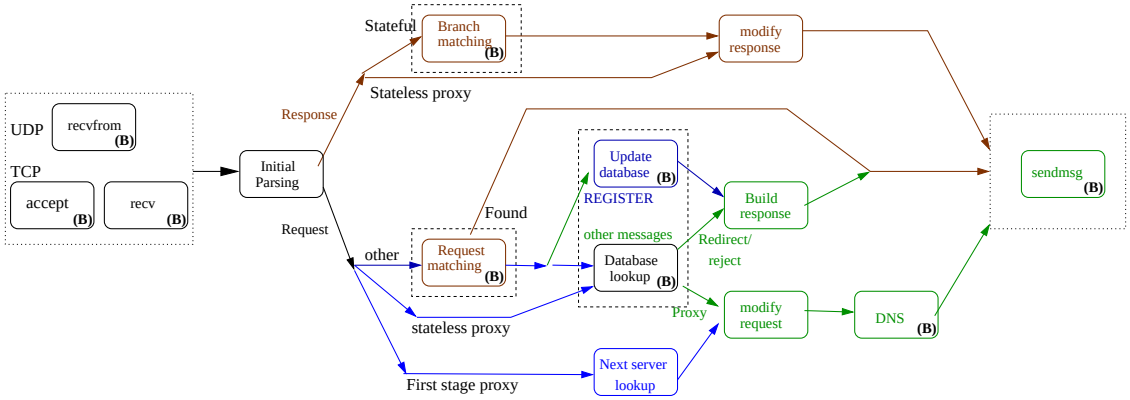


Figure 3.23: Processing steps in a SIP server. The potentially blocking operations either due to I/O, events or locks are marked with B

3.6.1 Processing Steps

Figure 3.23 describes the steps involved in processing a SIP request in any SIP server. It includes both transaction stateless and stateful processing. The server receives a message on a UDP or TCP socket. We use only UDP in our tests. The message is parsed using our unoptimized SIP parser. If the message is a SIP request, it is matched against the existing transactions. If a matching transaction is found, the request is a retransmission and the last response, if any, in the transaction is returned for the request. If a match is not found, and the request is a SIP REGISTER request, then the user contact records are updated for this registration and a response is sent. For any other request, the user record is looked up. Depending on the policy chosen, the call is then proxied, redirected or rejected. In the proxy mode, the server looks up the callee's current contact locations, forwards the request and waits for the response. During this process, the server may need to perform additional retransmissions for reliability. When receiving a response, the server looks up the appropriate matching transaction for this response and forwards the response. If the policy decides to redirect the request instead of proxying it, the server sends the response to the

caller listing the new contact location(s). The first stage load balancing server selects the next stage server based on the destination user identifier, without doing any database query. The steps are based on our SIP server implementation, but are likely to be similar for other implementations.

These processing steps can be implemented in various software architectures for both stateless and stateful proxy modes.

3.6.2 Stateless Proxy

A stateless proxy does not maintain any transaction state, and has a single control flow per message. That means that once a message has been received, it can be processed to the end without interfering with other messages. We used only UDP transport for our tests and did not perform any DNS lookups. As shown in Figure 3.15, a single Proxy 200 test involves six messages. In Figure 3.23, for an incoming call request, the steps performed are recvfrom, initial parsing, database lookup, modify request and sendmsg. Similarly, for an incoming call response the steps performed are recvfrom, initial parsing, modify response and sendmsg. A first stage load balancer proxy is also a stateless proxy, but it does not include the database lookup stage. The processing steps can be implemented in different software architectures as follows:

Event-based: A single thread for the whole system listens for incoming messages and processes it to the end. There is no locking or mutual exclusion (mutex). This does not take advantage of the underlying multiprocessor architecture. If DNS is used, then the same thread also listens for events such as the DNS response and timeout.

Thread per message: A main thread listens for incoming messages. A new parsing thread is created to do the initial parsing. Then another processing thread is created to perform the remaining steps depending on whether the message is a request or a response. This architecture performs independent logical operations in separate threads, making the program easy to understand. The thread terminates after the steps are completed. DNS lookups, if any, are performed synchronously in the processing thread. Locks (i.e., mutexes) are used for accessing shared data such as the database. Potentially blocking operations include DNS, sendmsg, and database lookup.

Pool-thread per message: This is similar to the previous method, except that instead of creating a new thread, it reuses a thread from a thread pool. A set of threads are created in the thread pool on server initialization and persist throughout the server lifetime. This reduces the thread creation overhead and is the original architecture of our SIP server, sipd [26]. To further reduce lock contention, the user data can be divided into multiple sets (say, 100), each with its own transaction tables or user records. Thus, access to user records in different sets do not contend for the same lock.

Process pool: On server initialization, a pool of identical processes is created, all listening on the same socket. When a message is received, the OS gives the socket message to one of the listening processes and that process performs all the processing steps for that message. Shared memory is used for sharing the database among multiple processes. This is the architecture of the SIP express router [96].

Thread pool: This is similar to the previous method, but it uses threads instead of processes. Only one thread can call recvfrom on the listening socket. If a thread has called recvfrom, then another thread is blocked from calling this function until the first thread finishes receiving the next socket message.

Software architecture /Hardware	1xP	4xP	1xS	2xS
Event-based	1550	400	150	600
Thread per message	1300	500	100	500
Pool-thread per message (sipd)	1400	850	110	600
Thread pool	1500	1300	152	750
Process pool	1600	1350	160	1000

Table 3.1: Performance (CPS) of stateless proxy for Proxy 200 test

We ran our tests on four different platforms as follows: (1xP) Pentium 4, 3 GHz, 1 GB running Linux 2.4.20, (4xP) four-processor Pentium 450 MHz, 512 MB running Linux 2.4.20, (1xS) ultraSparc-IIi, 300 MHz, 64 MB running Solaris 5.8, and (2xS) two-processor ultraSparc-III+, 900 MHz, 2 GB running Solaris 5.8. The results of our tests are shown in Table 3.1. The numbers presented in this section are different from earlier load sharing experiments of sipd

in Section 3.5, because these tests were done after some optimizations such as per-transaction memory pool to reduce memory deallocation and copy [26]. We used a small pool size for both process pool and thread pool, because the performance degraded if the pool size was more than two times the number of processors. The performance of different architectures relative to the event-based model on different platforms is shown in Figure 3.24 (a).

For a single processor system (1xP and 1xS), the performances of event-based, thread pool and process pool are roughly similar. We found that the thread pool model had a higher number of context switches compared to process pool. In the process pool model the same process keeps getting scheduled for handling subsequent requests. This resulted in the slight difference in the performance. The process pool model performs the best. The thread-per-message and pool-thread-per-message models have many fold higher context switches resulting in much poorer performance. This is because every message processing must involve at least two context switches. One interesting observation is that both the single processor systems (1xP and 1xS) took approximately 2 MHz CPU cycle per CPS (call per second) load.

For a multiprocessor system, the performance of the process pool implementation scales linearly with the number of processors. The performance of the pool-thread-per-message model is much worse than process pool because the former does not fully utilize the available concurrency of multiprocessor hardware. The processor running the main listening thread becomes the bottleneck.

3.6.3 Stateful Proxy

Unlike the stateless proxy, a transaction stateful proxy needs to maintain the SIP transaction state for the duration of the transaction. We used only UDP transport for our tests and did not perform any DNS lookups. As shown in Figure 3.20, a single Proxy 200 test involves six incoming and eight outgoing messages. In Figure 3.23, compared to the stateless proxy, the stateful proxy performs additional steps such as transaction (or client branch) matching. The transactions data structures are locked for exclusive access in a multi-threaded system. The processing steps can be implemented in different software architectures as follows:

Event-based: Most of the blocking operations are made non-blocking using events. A single

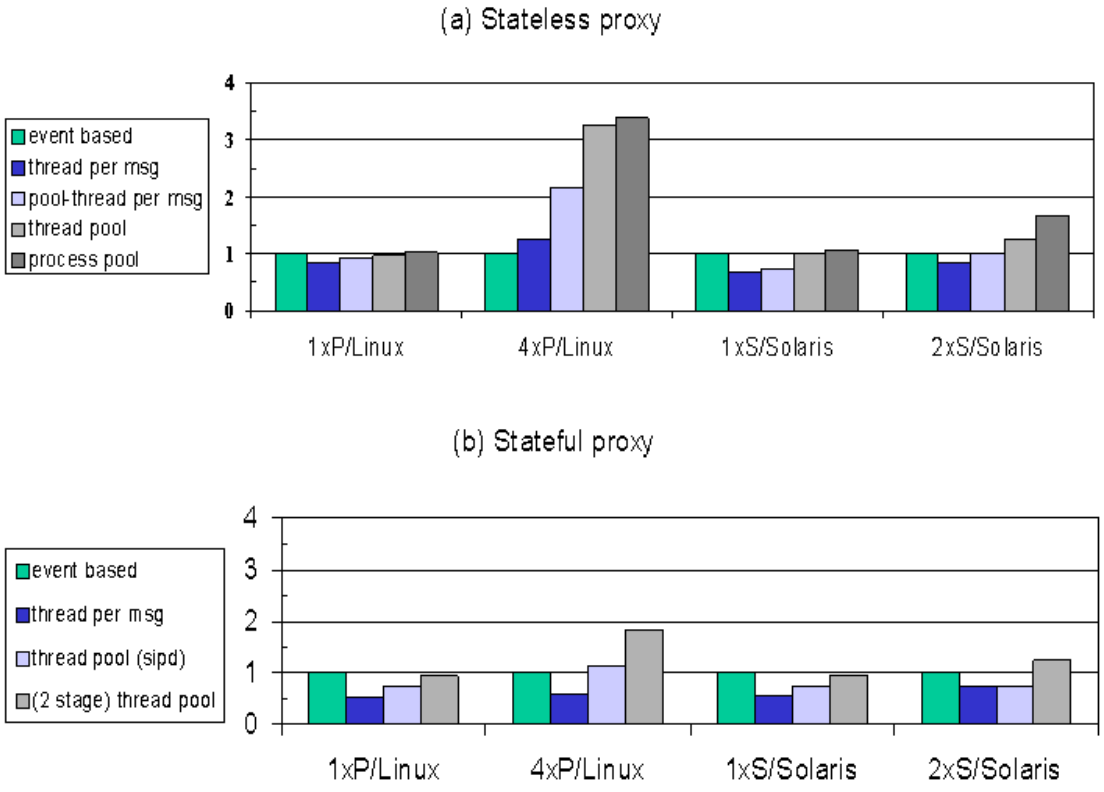


Figure 3.24: Performance of software architectures relative to event-based on different hardware. For example, the performance of stateless proxy on 4xP hardware in the thread pool architecture is approximately three times that in the event-based architecture on the same hardware.

thread for the whole server handles events from a queue (e.g., timer events) as well as messages from the listening socket. There is no locking or mutexes. There are only two operations that remain blocking: listening for incoming message on the socket, and listening for events on the event queue. A single threaded event-based system does not take advantage of the underlying multiprocessor architecture. Having multiple threads serving events results in lock contention while accessing the same transaction structures.

Thread per message (or transaction): A main thread listens for incoming messages. If the message is a request not matching any previous transaction, then a new thread is created to handle the new transaction associated with this message. The thread persists as long as the transaction exists. Similarly, a process-per-message model can be defined that creates

a new process for each incoming connection and message.

Thread pool: This is similar to the previous method, except that instead of creating a new thread, it reuses a thread from the thread pool. This reduces the thread creation overhead. Locks are used for accessing shared data. Potentially blocking operations include DNS lookup, sendmsg, request matching and database access. This is the original architecture of our SIP server, sipd [26].

(Two-stage) thread pool: A pool of identical threads is created. Each thread handles a specific subset of the user population based on the hash value of the user identifier as shown in Fig. 3.25, similar to the second stage of our load sharing architecture. A request is processed in two stages. The first stage thread listens for incoming messages, does minimum parsing, and chooses the second stage thread based on the destination user identifier. The message is then handed over to the particular second stage thread. The second stage is purely event-based with no other locking. Since a single thread handles the requests for the same set of users, we do not need to lock the database or transaction data structures. The number of threads in the thread pool is determined by the number of processors.

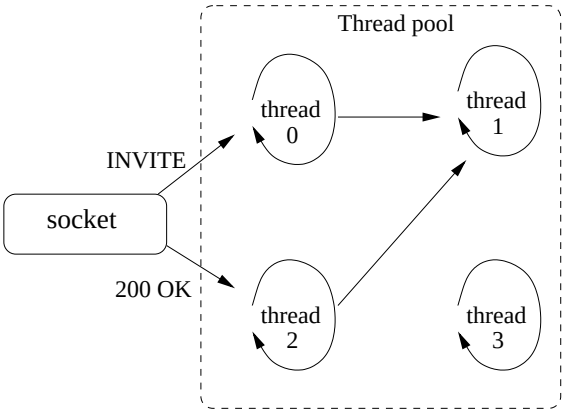


Figure 3.25: Two-stage thread pool software architecture: the example consists of four threads, numbered 0 to 3, in the thread pool. Any available thread receives the message, parses it and based on the hash of the SIP Call-ID value in the message, forwards the message to the appropriate thread. In the example, the hash is 1, thus both SIP INVITE request and 200 OK response go to the thread number 1.

The models can be further extended to processes as follows. We have not evaluated these extensions yet:

Process pool: A pool of identical processes is created, each listening on the same socket. When a message is received, the server performs all the processing steps for that message. Shared memory is used for sharing the transaction and user contacts among multiple processes. This is the architecture of the SIP express router [96].

Two-stage event and process-based: This is similar to the two-stage thread pool model, but using processes instead of threads. The operating system delivers an incoming message to any of the first stage processes listening on the UDP port. That process forwards the message to one of the second stage processes using pipes or Unix sockets, based on the hash of the SIP Call-ID in the message. Thus, all the messages in a transaction always go to the same second stage process, and that process does not need to share state with any other process. Multiple first stage processes can be used to allow more concurrency.

A generic design of thread-per-message is easy to understand and implement. However, this model suffers from poor performance at higher load [82]. As the load increases the number of threads in the system also increases. If the thread blocks waiting for a network response, the maximum number of simultaneous requests active in the system is small. Transaction lifetime further reduces the system capacity. For example, if the operating system supports 10,000 threads, and the SIP transaction lifetime is about 30 seconds, then there can be at most 10000/30 = 333 transactions/second processed in the system. Unlike a web server, this is further exacerbated in a SIP server by the fact that about 70% of calls are answered within roughly 8.5 seconds [97] while unanswered calls ring for 38 seconds. Thus, a bad design results in insufficient number of threads. This leads to higher call blocking or call setup delays at high call volume. Thus, we need to use a true event-driven architecture which requires the threads to be returned to the free-threads pool whenever they make a blocking call.

Table 3.2 and Figure 3.24 (b) compare the performance of stateful proxy in different architectures on the same set of hardware, except that 1xS is replaced by a single-processor ultraSparc-IIi, 360 MHz, 256 MB, running Solaris5.9. Event-based system performs best for sin-

gle processor machine. For an N-processor machine, the thread pool performance is much worse than N times the single-processor performance due to memory access contentions.

Software architecture /Hardware	1xP	4xP	1xS	2xS
Event-based	1150	300	160	400
Thread per message	600	175	90	300
Thread pool (sipd)	850	340	120	300
2-stage thread pool	1100	550	155	500

Table 3.2: Performance (CPS) for stateful proxy for Proxy 200 test

3.6.4 The Best Architecture

The two-stage thread pool model for the stateful proxy and the thread pool model for the stateless proxy combine the event and thread pool architectures. They provide an event-loop in each thread, and has a pool of threads for concurrency on multiprocessor machines. The lock contention is reduced by allowing the same thread to process all the steps of a message or transaction after initial parsing. For a multi-threaded software architecture this seems to give the best performance as per our tests. We have not yet evaluated the stateful proxy in process pool model.

The stateless proxy performance is usually limited by the CPU speed, whereas the memory utilization remains constant. On the other hand, the stateful proxy may be limited by either CPU or memory depending of various transaction timers. By default a SIP transaction state is maintained for about 30 seconds. Thus, a load of 1000 CPS creating 2000 transactions per second will require memory for about 60 thousand transactions. Assuming 10 kB for storing each transaction state, this requires 600 MB. In our tests, we have reduced the timer values significantly so that memory is not the bottleneck.

3.6.5 Effect on Load Sharing Performance

The software architecture choice of the SIP server further enhances the load sharing results since the best single stateless proxy capacity is about 1600 CPS on a 3 GHz Pentium 4 with 1 GB memory running Linux 2.4.20. In addition, we have achieved about 4000 CPS throughput for the

first stage proxy in a simplified implementation. This means even S1P2 in stateless proxy mode can achieve close to 3200 CPS, i.e., 11 million BHCA on this hardware configuration. Similarly, S3P3 in stateful proxy mode can achieve close to 13 million BHCA.

3.7 Conclusions

We have shown how to apply some of the existing failover and load sharing techniques to SIP servers, and propose an identifier-based two-stage load sharing method. Using DNS is the preferred way to offer redundancy since it does not require network co-location of the servers. For example, one can place SIP servers on different networks. With IP address takeover and NATs, that is rather difficult. This is less important for enterprise environments, but interesting for voice service providers such as Vonage. DNS itself is replicated, so a single name server outage does not affect operation. We combine DNS, server redundancy and the identifier-based load sharing in our two-stage reliable and scalable server architecture that can theoretically scale to any capacity. A large user population is divided among independent second stage servers such that each server load remains below its capacity.

We have also described the failover implementation and performance evaluation of our two-stage architecture for scalability using the SIPstone test suite in our test bed. Our results verify the theoretical improvement of load sharing for call handling and registration capacity. We achieve carrier grade scalability using commodity hardware, e.g., 2800 calls/second supported by our S_3P_3 load sharing configuration roughly translates to 10 million call arrivals per hour, using six servers. Lucent's 5E-XC$^{\mathrm{TM}}$ switch, a high-end 5ESS, can support four million BHCA for PSTN. This is further increased to 16 million BHCA in our memory pool and event-based architecture. We also achieved the 5-nines reliability goal even if each server has only uptime of 99% (3 days/year downtime) using the two-stage architecture. Other call stateful services such as voicemail, conferencing and PSTN interworking need more work to do failover and load sharing in the middle of the call without breaking the session.

Detection and recovery of wide area path outages [98] is complementary to the individual server failover. Adaptive load sharing based on the workload of each server is not investigated in this thesis. It is not clear how useful this will be for Internet telephony because the call dis-

tribution is more uniform unlike Zipf distribution of web page popularity. Therefore, a good static hash function can uniformly distribute the call requests among the servers. Instead of statically configuring the redundant servers, it will be useful if the servers can automatically discover and configure other available servers on the Internet, e.g., to handle temporary overload [92]. This gives rise to the service model where the provider sells its SIP services dynamically by becoming part of another customer SIP network. The SIP servers in a VoIP provider network can automatically discover, self-organize and configure themselves as first and second stage servers. A peer-to-peer approach for SIP service extends this idea to serverless VoIP infrastructure and proves to be promising for scalability and robustness as we describe in the next part of this thesis.

Part II

Peer-to-peer IP Telephony

This part describes our peer-to-peer Internet telephony architecture using SIP. The goal is to build a self-organizing, robust and scalable peer-to-peer network for Internet telephony using open interoperable protocols.

Chapter 4

Overview of Peer-to-Peer Internet Telephony using SIP

P2P systems inherently have high scalability because the capacity scales with user population, and robustness and fault tolerance because there is no centralized server and the network self-organizes itself. This is achieved at the cost of higher signaling latency for locating the resources of interest in the P2P overlay network. Internet telephony can be made as an application of the P2P architecture where the participants form a self-organizing P2P overlay network to locate and communicate with other participants. We propose a P2P architecture for the Session Initiation Protocol (SIP)-based IP telephony systems. Our P2P-SIP architecture supports basic user registration and call setup as well as advanced services such as offline message delivery, presence, voice and video mails, and multi-party conferencing. We also provide an overview of practical challenges for P2P-SIP such as firewall and NAT traversal, and discuss security.

4.1 Introduction

Existing Internet telephony client-server architecture based on IETF's Session Initiation Protocol (SIP [4, 3]) or ITU-T recommendation H.323 [99] typically employ a registration server for every domain. The user agents (or IP phones) of the users in the domain register their IP addresses with the server so that the other users can reach them. Scalability and reliability of such server-

based systems are achieved using traditional redundancy and failover methods as described in Chapter 3. The majority of the system cost is in maintenance and configuration, typically by a dedicated system administrator in the domain. It also means that quickly setting up the system in a small environment (e.g., for emergency communications or at a conference) is not easy.

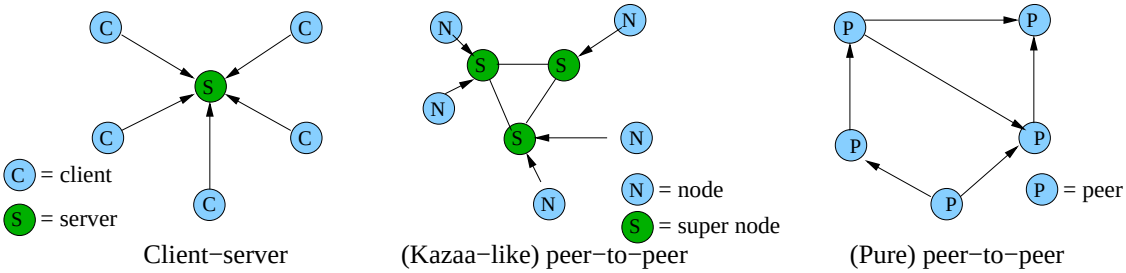


Figure 4.1: Client-server vs peer-to-peer distributed systems

On the other hand, peer-to-peer (P2P) systems [12] are inherently scalable and reliable because of the lack of a single point of failure. P2P systems, in the purest form, have no concept of servers as shown in Fig. 4.1. All participants are peers and communicate in distributed, potentially untrusted environment, to achieve a certain objective such as locating music files or users. Some file transfer systems with central index server such as the old Napster are hybrid P2P systems. However, for the purpose of this thesis we use the definition that *pure* P2P systems do not have any centralized control. Accordingly, existing SIP and H.323-based systems that have centralized user location lookup but end-to-end media transport are *not* P2P.

Peer-to-peer Internet telephony avoids the maintenance and configuration cost of the server-based SIP architecture, and prevents catastrophic failures of server-based systems. There are five major components that can be made peer-to-peer:

User location storage: User location information contains a list of current contact host names or IP addresses of the user. In client-server SIP, the REGISTER message conveys the contact location to the registrar. The user location binding information is updated by the user and read by other prospective callers.

Configuration storage: A user may need to store some configuration information such as his friends list.

Multimedia mail and offline storage: If the user is not available to pick up his phone call, the call may get forwarded to voice mail. Such offline messages are written by the caller, and read and deleted by the recipient.

Media relay discovery: Traversing NATs requires use of external media relays in the public Internet. Such components may be distributed in the P2P network, and discovered as needed by the clients that are in a network with a private address space.

PSTN gateway discovery: A number of VoIP gateways may be available in the Internet to reach the same telephone subscriber. We need to select a gateway for making a call from IP to PSTN using a selection criterion such as lower call cost, lower network latency on IP, less number of hops, same PSTN or IP provider network, or same PSTN area code.

The first three components nicely fit in the hash table data structure, whereas the last two require some notion of proximity. We use distributed hash table (DHT) as the P2P network in our Internet telephony architecture.

There are two approaches to combine SIP and P2P: replace the storage of SIP data by a P2P protocol (*SIP-using-P2P*), and additionally, implement the P2P protocol itself using SIP messaging (*P2P-over-SIP*). In this part, we describe our P2P-SIP architecture using both approaches. We analyze various design alternatives and present the detailed design of our P2P-over-SIP endpoint that uses Chord [22] as the underlying DHT and our SIP-using-P2P endpoint that uses OpenDHT [27] as the external DHT. Chord or OpenDHT can be replaced by any other DHT in our implementation without affecting the architecture as long as the DHT APIs are similar. In addition to the basic call setup and registration, we also outline advanced services such as “missed call” notifications, presence and multi-party conferencing in P2P-SIP.

Our novel hybrid architecture allows both traditional SIP telephony as well as user lookup on P2P network if the local domain does not have a SIP server. For P2P-over-SIP, we show that SIP can be used to implement various DHT functions in P2P-SIP such as peer discovery, user registration, node failure detection, user location and call setup by replacing DNS [28] with P2P for the next hop lookup in SIP without changing the semantics of SIP messages.

We summarize the related work in Section 4.2. Section 4.3 lists the goals for a P2P

architecture for IP telephony. Section 4.4 compares the SIP-using-P2P and P2P-over-SIP architectures. The detailed design and evaluation of SIP-using-P2P and P2P-over-SIP architectures are presented in Chapters 5 and 6, respectively.

4.2 Related Work

A number of studies have been done to analyze and understand different peer-to-peer (P2P) systems [12, 100]. P2P systems can be broadly classified into unstructured networks such as Kazaa and Gnutella with no structure of how the nodes store files, and structured networks such as those using a distributed hash table (DHT). The unstructured systems have concentrated on practical problems such as NAT and firewall traversal but search is typically performed by flooding the request to all the neighboring peers. On the other hand, structured systems such as Chord [22, 23], Content Addressable Network (CAN) [101] and Pastry [102] focus on optimizing the P2P overlay for lookup latency and join or leave maintenance cost [103] instead of using inefficient blind search by flooding. DHTs are well suited for Internet telephony application because the user contacts can be stored and looked up based on the user identifier as the hash key. NAT traversal has not been explored in detail for structured P2P networks.

DHTs provide distributed implementation of hash tables with two sets of high level API: data access (get, put and remove) and service (join, leave and find). Our peer-to-peer Internet telephony architecture uses this API of the underlying DHT. Chord is a DHT that has a ring-based topology where each node stores at most $\log(N)$ entries (or state) in its *finger table* to point to other peers. Lookup is done in $O(\log(N))$ time. The *iterative* and *recursive* lookup styles in Chord [22] directly map to the *redirect* and *proxy* behavior, respectively, in SIP. Research in DHT is complementary to our work, since our architecture can use innovations and optimizations in the underlying DHT.

4.2.1 Skype and Related Systems

Skype [21, 104] is a free P2P application based on Kazaa [18] architecture that allows making calls over the Internet to any other Skype user. Skype has the following problems:

1. The protocol is proprietary unlike open standards such as SIP.

2. It provides a single service, making calls or sending instant messages, and not an architecture for new services.

3. Most importantly, it has centralized elements for login authentication [104] which means that if this element fails, the system may not work.

In a way, the Skype's architecture is no different from the classical SIP telephony architecture, except that Skype's Global Index Server assigns a *super-node* for a new joining node. The super-node, similar to the SIP registrar, proxy and presence server, maintains the presence information for this node, and locates other users by communicating with other super-nodes. A node that has enough capacity and availability can become a super-node. We believe that the lookup is based on some variation of flooding, similar to Kazaa, instead of using the more efficient DHT-based lookup.

The main advantage of Skype is that it implements the equivalent of STUN [105] and TURN [106] servers in the node itself to handle NAT [107], unlike explicit server configuration in existing SIP applications. We use the super-node and ordinary node distinction in our architecture, too.

Others have developed various P2P multimedia communication applications such as flooding-based text chat [108] and peer-to-peer collaboration systems [109, 110, 111] for small groups with centralized components and limited scalability.

4.2.2 P2P-SIP Telephony

SIP-based IP telephony can be treated as a P2P system with static set of super-nodes (SIP servers) where the lookup is based on DNS instead of a hash key. However, using a pure P2P architecture instead of static set of SIP servers improves the reliability and allows the system to dynamically adapt to node failures.

There are some recent P2P Internet telephony applications such as NimX [112] and Peerio [113], but the architectures are not open. Earthlink's experimental SIPshare [114] provides

SIP-based P2P file sharing. It uses SIP messages, SUBSCRIBE and NOTIFY, to build and maintain P2P overlay, file search and content transfer.

Our work is not related to the peer-to-peer third-party call control (3PCC [115]) work in the SIP community, as the latter focuses on using the SIP REFER message to do call control directly between the participating user agents in the client-server SIP architecture, whereas our work focuses on defining a P2P architecture for user location in SIP.

We published our initial architecture of P2P-SIP in 2004 [14]. Since then, P2P-SIP has been discussed extensively in the IETF with a number of internet-drafts submitted on various aspects of P2P-SIP [15, 16, 17]. In particular, [17] is similar to our P2P-over-SIP architecture [14, 116]. Our work on using an external DHT (SIP-using-P2P) is inspired by [16], but fills in details to design and implement such system securely.

4.2.3 IP Telephony vs. File Sharing

There are three broad categories of P2P applications: file sharing, directory service and rendezvous systems. A rendezvous or meeting system initiates communication with users or groups of users and actively synchronizes different activities such as audio and video communications and floor control. For example, a user can send a SIP INVITE message to many potentially nomadic users to invite them to a conference by creating one-to-many bindings. On the other hand, a directory service provides a structured (e.g., hierarchical) repository of information on people or resources [117]. Usually the directory information does not change frequently and slightly stale information is also useful. However, user contact location (i.e., the IP address of her multimedia rendezvous client) may change frequently. SIP is often labeled as a rendezvous system, but uses server-based user lookup. Table 4.1 summarizes the similarity and differences among these types. In particular, for rendezvous systems such as Internet conferencing, data storage is not an issue. A single P2P-SIP node can handle many more requests than a file sharing node due to the low data volume. Caching of location information is not useful because compared to the file access pattern, which often follows the Zipf distribution [118], call access patterns are more uniformly distributed. Moreover, most residential users are likely to get a new DHCP IP address every time they connect to the Internet making the cache entry for this user location stale. The file sharing

Table 4.1: Different applications of P2P

Properties/Types	File sharing	directory	rendezvous systems (for user lookup)
Data storage	Yes	No	No
Caching	Yes	Yes	No
Delay sensitive	No	No	Yes
Reliability	Having multiple independent copies of data helps		Only the intended user must be found

and directory lookup-based systems can tolerate high lookup latency due to the fact that the user does not need to wait for the file to download, and the actual file download time tends to be larger than the lookup latency. On the other hand, an IP telephony caller actively waits for the phone on the other side to ring. For file sharing applications, multiple almost-exact copies of a popular file may be available (e.g., independently ripped by different peers). So node reliability does not matter. On the other hand, in the case of IP telephony, we want to talk to the right person, and not some similarly named person!

4.2.4 Robustness and Scalability

The primary advantage of P2P is robustness and scalability. Load sharing techniques can be applied to DHT to provide better performance [119]. DNS-based [67, 68] or same IP address-based [58] redundancy techniques are not good for P2P because they require significant maintenance on join and leave, are server-based or do not work when the nodes are distributed over the Internet. Load sharing techniques such as those based on load or available capacity with a central dispatcher do not work well for P2P systems due to heterogeneity of the peer nodes and absence of central dispatcher [120]. Our work on integration of SIP and P2P also benefits from the robustness and scalability research in P2P overlays.

4.3 Design Requirements

Based on the review of existing P2P systems such as Skype [21] and Chord [22], we propose the following goals for our P2P-SIP telephony architecture.

Zero configuration: The system should be able to automatically configure itself [20], e.g., by detecting NAT and firewall settings, discovering neighboring peers and performing initial registration.

Heterogeneous nodes: It should be able to adapt to available resources and distinguish between peers with different capacity and availability constraints. This favors the distinction between nodes and super-nodes as in Kazaa.

Efficient lookup: Blind search based on flooding is inefficient [100]. The system should use an underlying DHT to optimize lookup. We choose Chord as the underlying DHT for our P2P-over-SIP system because of its robustness and efficiency in the case of concurrent node joins and leaves [103].

Multiple systems: Unlike a single global system such as Skype, it should support multiple systems, e.g., with multiple user identity providers, and interoperate among them.

Advanced services: It should support advanced telephony services such as offline voice messaging, multi-party conferencing, call transfer and call forwarding as well as advanced Internet services such as presence and instant messaging.

Interoperability: It should easily integrate with existing protocols and IP telephony infrastructure. We choose SIP [3] as the signaling protocol for interoperability.

Besides these explicit goals, there are some implicit scalability and robustness benefits in the P2P-SIP architecture compared to the client-server SIP architecture. To incrementally build the P2P-SIP architecture and to illustrate some design choices, we start from the server-based architecture.

Replicate Registrations vs Search on Call Setup

Going back to the simple call setup example of Fig. 1.1 (p. 3), the single server can become the bottleneck for reliability. It can be improved by having multiple redundant servers. There are two alternatives:

1. replicate all user location information to all the servers, as shown in Fig 4.2, or
2. search for the correct server holding the destination user location when a new incoming call is received, as shown in Fig 4.3.

In the first case, although Fig. 4.2 shows multiple registrations, one can alternatively do database replication to ensure consistent user records among multiple server databases in the cluster. In the second case, either the caller retries all the servers in some order or the first contacted server can do the search.

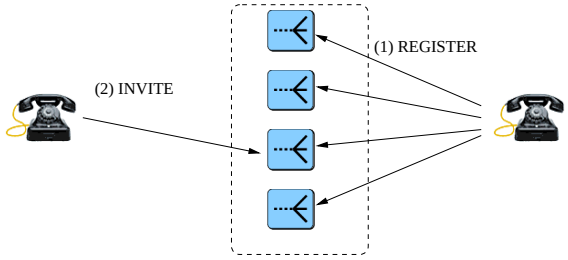


Figure 4.2: Design A: all servers store all user records on registration

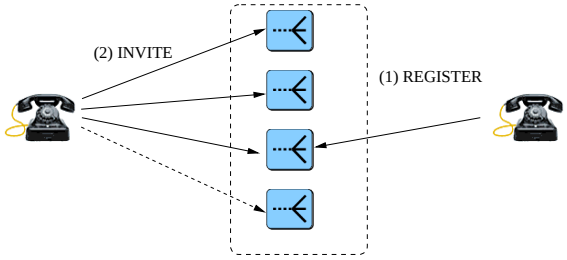


Figure 4.3: Design B: search for the server on call setup

The disadvantage of the first approach is that it involves synchronization overhead for each registration. There is a danger of stale user location record on some servers for a brief interval after the update is done but before all the servers get the updated registration. With registration refreshes every hour per user, this architecture may limit the total number of users supported by the system as the synchronization traffic will soon become a bottleneck. In the second case, the call setup latency is higher due to the sequential search steps. A parallel search will increase the bandwidth requirement. Both the approaches of Fig. 4.2 and 4.3 tend to fail when the number of servers is very large. The first approach and its variations are described in Chapter 3.

What Nodes form the DHT?

We can achieve some combination of the two designs using a DHT such as Chord [22] so that the registration is done on only $O(\log N)$ servers instead of all the N servers, and the search is done for only $O(\log N)$ servers instead of all the N servers. There can be three alternative designs for using a DHT. On one extreme, we can limit the DHT to the server farm as shown in Fig. 4.4. In this case, each client or phone connects to one of the servers. The servers implement a DHT or a scalable distributed data structure [121] to locate the correct user record. The architecture is still client-server. The client needs to discover at least one server, preferably lightly loaded, and connect to it. On the other extreme (Fig. 4.5), a client also acts as a server and implement a "pure" P2P overlay with all the other clients. The first option does not require modifying the clients, but provides a scalable and reliable server farm architecture. But it still has some of the server maintenance and configuration problems, unlike the second option.

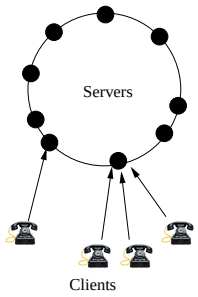


Figure 4.4: Option 1: Only servers in DHT

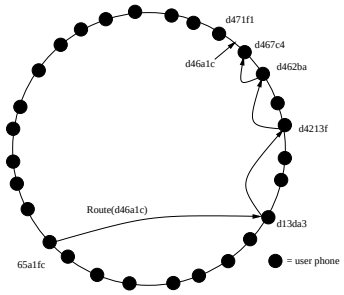


Figure 4.5: Option 2: Complete P2P overlay

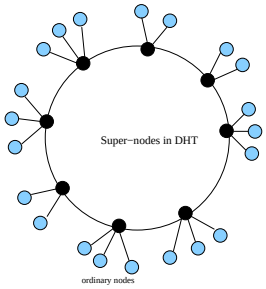


Figure 4.6: Option 3: Intermediate model

One problem with the pure P2P overlay of all nodes is that not all nodes have equal capacity and availability. For example, a node with low bandwidth connection to the Internet or those behind a firewall or NAT may not be able to fully function in a DHT because it may need in-bound connections, significant bandwidth for forwarding P2P messages or significant memory or CPU for maintaining DHT state. This problem can be solved by adopting an intermediate design as shown in Fig. 4.6. Some of the nodes with high capacity (bandwidth, CPU, memory) and availability (uptime, public IP address) are made super-nodes. Only the super-nodes form a DHT. An ordinary node just connects to one of the available super-nodes, similar to Kazaa. This

is similar to the first option except that there is no distinction between clients and servers, and any node can be a super-node or ordinary node, depending on the capacity and availability. Our goal is to allow a P2P-SIP node to work in any of the above configurations.

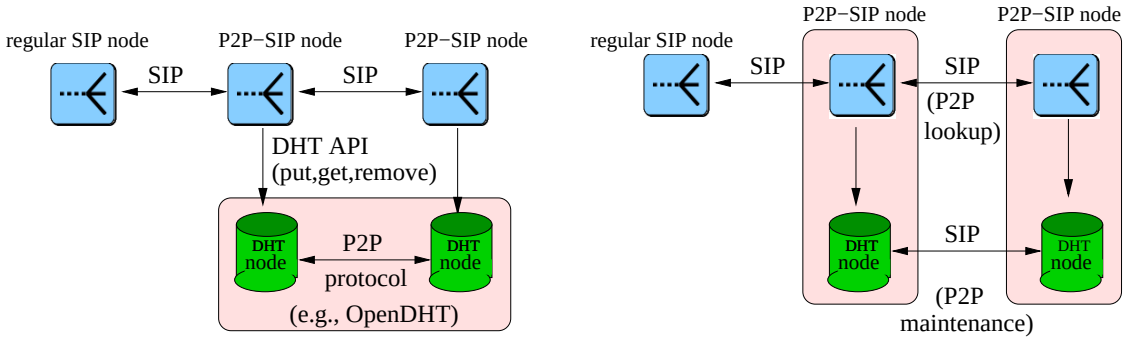
The decision to become an ordinary node or a super node is usually local. When a node starts up it will become an ordinary node. When the ordinary node detects enough capacity and availability (public IP address and uptime), then it can become a super-node. A node with enough capacity and availability may be forced to become a super-node when an existing super-node is leaving or has reached the capacity limit. However, some nodes that are known to have enough capacity and availability can immediately transition to super-node upon startup.

Having two levels, super-nodes and ordinary nodes, does not affect the search latency bounds. The search latency is still $O(\log N)$. However, it improves the performance in practice because the DHT maintenance traffic is reduced if the nodes in the DHT are more stable.

The DHT is logically separate from the SIP operations as described next.

4.4 SIP-using-P2P and P2P-over-SIP

There are two architectures for P2P-SIP: SIP-using-P2P and P2P-over-SIP. These are fundamentally similar because there is a clear separation between the DHT layer and the SIP layer as shown in Fig. 4.7. The difference is that in P2P-over-SIP the P2P maintenance protocol is also implemented using SIP. In this section, we compare the two architectures.



(a) SIP−using−P2P

(b) P2P−over−SIP

Figure 4.7: Difference between SIP-using-P2P and P2P-over-SIP architectures

Transport and transaction overhead

In the SIP-using-P2P architecture, the system can use the optimizations and enhancements done in the external DHT. For example, the message overhead can be reduced for the DHT maintenance. However, the algorithmic overhead of number of messages remains the same and depends on the particular DHT (e.g., Chord) in use.

Some SIP specific timers (e.g., retransmission timeout) may not be acceptable for some DHT-based applications, especially if the timers translates to long DHT lookup and update latency.

Choice of DHT

In the P2P-using-SIP architecture, the node needs to implement the particular DHT connector. If multiple DHTs can be used then such implementations need to potentially implement all such DHT connectors.

Today, there are multiple P2P protocols that do not interoprate and are not meant to interoperate (e.g., Kademlia, Chord, OpenDHT). Moreover, there is no single protocol or mechanism to talk to any DHT. Thus, the SIP-over-P2P architecture gives us an opportunity to build such an interface using SIP.

Feature reuse from SIP

Using SIP to build the DHT allows us to reuse the existing naming, routing, and security issues from SIP. Moreover, the NAT and firewall traversal mechanisms in SIP can also be used to allow a node behind a NAT to become a super-node. More work is needed for this.

Secondly, SIP features such as redirect and proxy modes are readily reusable in a DHT's iterative and recursive modes. Moreover, we can transparently reuse the existing SIP-based components such as voicemail and conferencing servers without having them to understand the DHT protocol to update the DHT indicating that they provide the service.

SIP specific protocol

If the DHT interface (lookup and update) is implemented using SIP, (e.g., REGISTER and INVITE methods), then any other application that wants to use the DHT needs to implement the SIP protocol stack. This is an undesirable implementation complexity. However, for the use case of Internet telephony, SIP-based DHT protocol is acceptable since the implementation will already have a SIP stack.

Security

Using SIP for P2P maintenance burdens the SIP protocol with additional security issues of handling malicious nodes. On the other hand, having a separate DHT simplifies the problem and in some instances (such as managed OpenDHT) solves the problem.

Service model

The SIP-using-P2P architecture promotes free-riding of SIP endpoint on the external DHT. To prevent this, the P2P-SIP nodes themselves should form the DHT and use a well-defined DHT protocol to perform P2P-SIP operations. If the same node implements both SIP and DHT, it is better to use a single protocol to simplify the implementation.

In summary, we need a clear separation between the SIP and DHT layers, but whether to use SIP for the DHT maintenance is not yet clear. Either way the gain or loss is not much compared to the advantages of using P2P-SIP versus centralized SIP. Once we have a clear interface between the SIP and DHT layers, the exact protocol for the DHT maintenance can depend on the deployment scenario, e.g., use SIP if all the nodes in the DHT are only P2P-SIP nodes, but use something else if the DHT is an externally managed P2P network.

We describe details of the SIP-using-P2P architecture in the next chapter.

Chapter 5

SIP-using-P2P: Using an External DHT as a SIP Location Service

5.1 Introduction

In this chapter, we describe the SIP-using-P2P architecture that uses an external P2P network for storing SIP location data. Since the user agents and proxies use a shared P2P network, we need to define the precise data format for such operations for interoperability, i.e., contacts updated by one user agent are readable by another. For storing user contact locations, a distributed hash table (DHT) is enough instead of a full P2P database with various SQL-style search commands. We provide an example data format for such a DHT-based SIP location service, and guidelines for implementing a SIP-using-P2P architecture with a managed external DHT based on our implementation experience. We describe what DHT keys and values should be used and how to sign and encrypt data for P2P-SIP using pseudo-code and examples. We also describe the P2P presence and offline messaging. We do not propose any new algorithms but just apply existing algorithms to P2P-SIP clients and proxies. The assumption is that the DHT nodes are not malicious and correctly perform DHT operations. One example of an external DHT is OpenDHT [27, 122] run on PlanetLab.

We provide background on the DHT API in Section 5.2. Then, we describe the logical operations such as contact management and key storage in Section 5.4. Section 5.3 gives the

motivation for the service model. We explain the P2P-SIP deployment scenarios such as client and proxy with pseudocode in Section 5.5. Section 5.7 presents some implementation issues. Security consideration, advanced services and evaluation are presented in Sections 5.6, 5.8 and 5.9 respectively. We present our proposed XML-based data format in Appendix C.

$H(v)$	SHA-1 of v.
MD5(v)	MD5 hash of v.
$\{v\}_K$	v is encrypted using RSA private key K_S or public key K_P.
$[v]_s, [v]^s$	The subscript encrypts v using shared secret s and the superscript decrypts it.
now	the current timestamp.
δ	a small value for time, e.g., few seconds.
$v\|u$	concatenation of two parameters, v and u, possibly using a delimiter
(v, u)	a tuple containing v and u in that order, possibly stored in XML
$a[..]$	a list or vector variable, a
$v \leftarrow u$	assignment from u to v
$/* ... */$	is used as a comment or remark similar to C

Table 5.1: Notations used in this chapter

5.2 Background: DHT API

The current interface of OpenDHT is described in [27], and summarized here. The put$(k, v, H(s), t)$ method is used to store a value v associated with a key k. The value expires after time-to-live (ttl), t, and can be removed before that time using the secret s. The value for the key k can be retrieved using get(k). It returns a list of tuples, $(v, H(s), t)$, where t is the remaining ttl. The value for the key, k, can be removed using remove$(k, H(v), s, t)$, where t is more than the remaining ttl.

We use the existing interface as the basis to build P2P-SIP services. The interface allows putting multiple values under the same key, i.e., both (k_1, v_1) and (k_1, v_2) can be stored. For example, if Bob has many SIP phones, each phone can store its own contact IP address under Bob's key, and Alice's phone can retrieve all these contacts when making a call. The interface also allows putting the same value under the same key using different secrets. For example, both $(k_1, v_1, H(s_1))$ and $(k_1, v_1, H(s_2))$ can be stored. The secret controls who can remove the value associated with that key. Finally, a put with same key, value and secret, just updates the time-to-live (ttl). The ttl can be mapped to the Expires header in SIP REGISTER request for expiry of

contact bindings.

An authenticated DHT interface [27] is required for protection against malicious users of the DHT and to filter get results at the DHT node. This is a planned future work in OpenDHT.

5.3 Data and Service Models

In a server-based SIP architecture, the SIP server performs three logical operations: registration, proxy (or redirect) and location service, as shown in Fig. 5.1. In addition to storing the contact bindings, the location service includes service logic such as programmable service scripts. Real implementations usually combine all these logical operations into a single server such as our sipd. The protocol for accessing the location service is currently not standardized. A interoperable location service access protocol allows decomposing the server implementation, and helps in implementing P2P communication between two users without going through the SIP server.

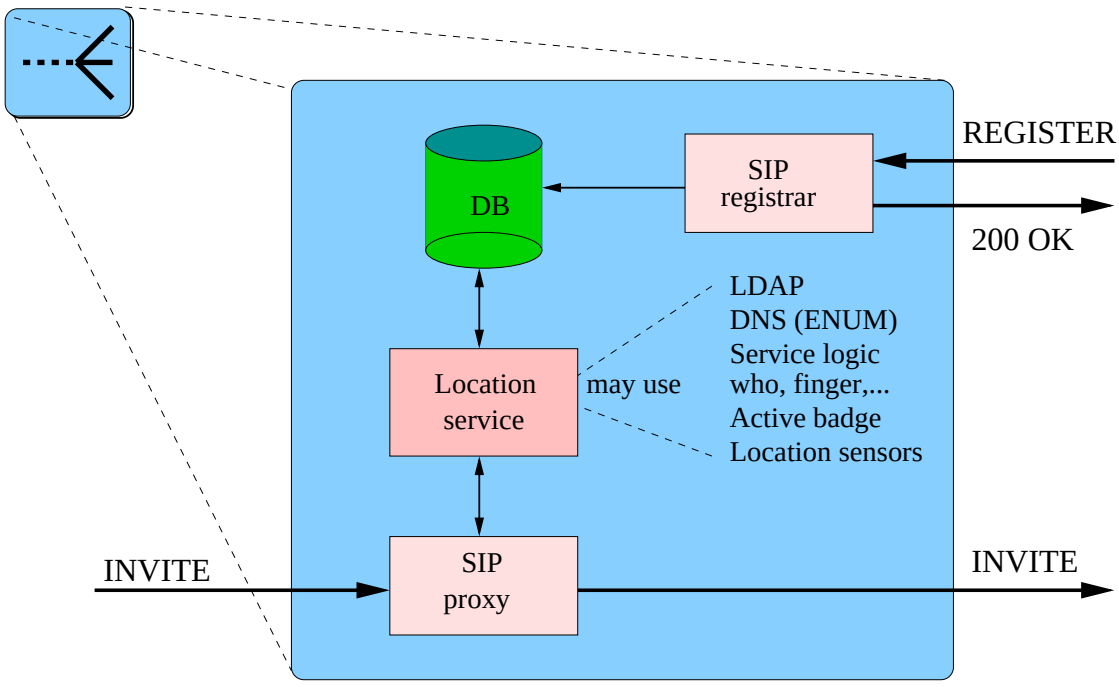


Figure 5.1: Logical operations in a SIP server

There are two approaches to perform location service in P2P-SIP: any user directly updates the DHT (called as *data model*) or forwards the request to the service node responsible for

that user key (*service model*).

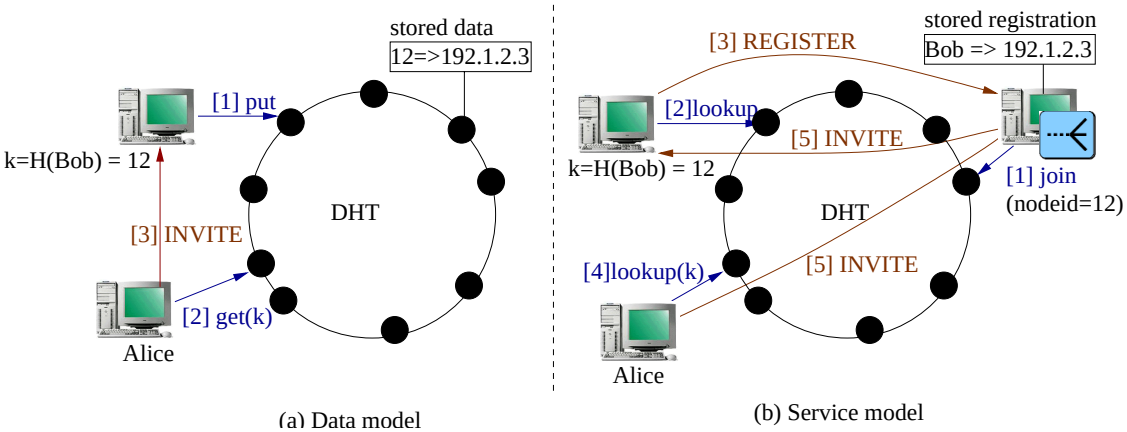


Figure 5.2: Data model vs service model

Data Model

In this model the DHT is used as a shared data storage and the P2P-SIP operations are performed by the user by directly updating the corresponding DHT data. For example, a user stores his contact information and a caller stores the offline messages in the DHT. Similarly a P2P proxy updates the data in the DHT on behalf of the user to provide transparent SIP service to non-P2P users.

There are several limitations to this approach. For example, presence composition [123] or programmable call routing [49] are *not* easy to implement. Moreover, the node needs to discover STUN and TURN servers anyway, but the service discovery does not work well with the data model as we describe in Section 5.4. An alternative service model solves this problem as described below.

Service Model

In this model, every P2P-SIP client or proxy joins the DHT for the p2p-sip service. The p2p-sip service includes SIP registrar, presence agent, offline message storage, and STUN and TURN servers at the minimum.

When a user, Alice, wants to send a SIP message to, say sip:bob@example.net, she looks up the DHT to find the service node responsible for this user identifier, and sends SIP request to that node. The service node acts as the proxy, registrar and presence server for all the users for which it is responsible. The service node also does any safe programmable call handling scripts [49] and presence composition [123].

For signed or encrypted data such as contact information, there are two approaches: either the user sends the signed contacts in the SIP message or the user authorizes the service node to sign the contacts on his behalf. The first approach requires changes in existing SIP clients, whereas the second approach just uses a chain of certificates for verification of signed contacts.

The service model is more extensible than the data model. A P2P-SIP service node readily interworks with any non-P2P clients who just happen to know one or more service node addresses. The service mode readily extends to P2P-over-SIP architecture since only the service interface (join and lookup) is used in the DHT, instead of the data interface (get and put). Note however that the ReDiR interface of OpenDHT is in fact built on top of the data interface and resides purely on the client side without any change in the DHT node implementation. In particular, a balanced tree of service node identifiers is built and embedded on to the DHT. This prevents overloading a single DHT node with all the service node identifiers, and optimizes the lookup cost to O(1) on average. Thus, the service model is suitable for both P2P-over-SIP and SIP-using-P2P, though we describe only SIP-using-P2P in this chapter.

The rest of the chapter describes only the data model. The service model can be built using the underlying data model, because the service nodes also use the specified data format for storage in the DHT.

5.4 Logical Operations

In this section, we identify the logical operations that can be made peer-to-peer for SIP-based Internet telephony. The P2P-SIP design consists of logical operations such as key storage, location service, NAT and firewall traversal, presence and offline message storage.

Location Service (Contact Management)

The DHT interface is used to store the user contact information. For example, Bob stores his contacts under the DHT key, k=H(sip:bob@example.net). This simple scheme allows multiple users to register under the same SIP identifier, say bob@example.net. So it is the responsibility of P2P-SIP to verify the correct identity of the callee. Any public data such as user contacts on the DHT should be signed by the owner so that others can verify its validity.

A P2P client signs the data on behalf of the user. The user should be able to use another client and update his contact information. This mode allows the user to pick his own SIP identifier, as long as he can prove that the identifier belongs to him via certificate(s). There is no dependency on a SIP server. For example, if the user's identifier is bob@example.net, then the domain example.net need not be a valid DNS name or need not have any associated SIP server.

A proxy in a P2P server farm (Fig. 4.4) authenticates the user, and then signs the data put on the DHT. For example, when user alice@home.com registers with the P2P proxy of domain home.com, the proxy signs her contacts using the signer identity as home.com. To allow other proxies in the farm to change or remove the contacts, all proxies of home.com should use the same key for signing. This allows the user to transparently use any of the proxy in the farm.

The caller verifies that the contacts retrieved from the DHT for bob@example.net are signed either by the user identity, bob@example.net, his domain example.net, or a mutually trusted certificate authority (CA) such as VeriSign.

Cryptographic Key Storage

To avoid any central server, the certificates, cryptographic keys, and any user configuration such as "friends list" are also stored on the DHT. For example, Bob can store his certificate on the DHT with k=H(certificate:bob@example.net). Multiple certificates of Bob from different CAs can be put under the same DHT key. Since the information needs to be available to any potential caller, the value is unencrypted. There is a danger of other malicious users polluting the DHT values for this key. However, chained verification of the certificates can be used to retrieve the correct certificate.

The user can also store his private configuration information such as his private key on the

DHT. Thus, he can share the same configuration among multiple clients. However, this sensitive information must be stored encrypted on the DHT. For example, Bob can store his encrypted private key with k=H(private:bob@example.net:secret). In addition to encrypting the private key with a secret, the secret is also used by Bob to generate the DHT key, so that other malicious users can not pollute the values for k. Since the user chosen secret password is much easier to remember for the user than his private key, storing the encrypted private key on the DHT is helpful.

Presence

Presence data of a user contains three pieces of information: (1) watcher list: the list of users interested in knowing the presence status of this user, (2) friends list: the list of users whose presence status is being watched by this user, and (3) watcher authorization list: the authorization information about the users in the watcher list. The separation allows any one to update this user's watcher list, but only this user can update his friends and authorization lists.

Presence data is handled differently because, unlike the contact information, which needs to be available to all the potential callers, the watcher list should be visible only to the presentity (the entity being watched). We use a generic DHT key format to store the subscription request, i.e., watcher list for any event including presence. The DHT key is formatted as "subscribe:*event*:*user*". For example, if Alice wants to subscribe to the presence status of Bob, she puts her signed identity in Bob's watcher list with k=H(subscribe:presence:bob@example.net). The value is encrypted using Bob's public key so that only Bob can decrypt the watcher identity. This mechanism also works for events other than presence.

Additionally, Alice can store her encrypted friends and authorization lists on the DHT similar to the private key storage described earlier. If a new user Sam appears in the watcher list, but is not present in the authorization list, then Alice is prompted to authorize or deny the subscription by this new user, Sam. The result is stored in Alice's watcher authorization list.

Offline Messages

When Alice calls Bob, and Bob is not registered or does not pick up the phone, Alice can store an offline message (text or multimedia) under key k=H(offline:bob@example.net). When Bob comes back, he can retrieve his offline messages. The signing and encryption is similar to the watcher list.

The difference between storage of watcher list (presence data) and offline message is that the watcher list is periodically refreshed by the individual watchers, whereas the offline message is usually removed by the recipient after retrieval.

NAT and Firewall Traversal

Although NAT and firewall traversal is not a generic P2P-SIP logical operations, we believe that NAT and firewall traversal is required for successful deployment of P2P-SIP. Hence, we include this as a basic P2P-SIP operation.

Inbound SIP messages to a client behind a NAT (Network Address Translator) require connection reuse [124] and symmetric response routing [125]. Additionally, SIP phones use mechanisms such as STUN (Simple Traversal of UDP through NAT [105]), TURN (Traversal Using Relay NAT [106]) and ICE (Interactive Connectivity Establishment [107]), to allow media traversal through NATs and firewalls. This requires publically available STUN and TURN servers. Our P2P-SIP node implements both STUN and TURN, and provides these services to other users.

The existing DHT interface of OpenDHT [27] is not sufficient for such service discovery. Consider the trivial approach where every STUN server stores its IP address under k=H(stun). This requires modifying existing STUN servers, or some other centralized entity to register existing STUN servers' IP addresses in the DHT. Secondly, this is not scalable because the DHT node storing this key, k, will soon become overloaded with potentially millions of clients advertising as STUN servers. There are two alternatives: DHT's service interface and hierarchical location-based key. OpenDHT provides additional API (ReDiR [27]) that addresses this scalability problem to join and lookup for a service. Thus, a P2P-SIP node joins OpenDHT for "stun" and "turn" services. Alternatively, if a node detects its location as "New York" and au-

tonomous system (AS) number of his service provider as 1234, it can store its IP address with k_1=H(stun:geo:us.ny.newyork) and k_2=H(stun:as:1234). The use of AS number is useful because users in the same AS are likely to have good connectivity.

Next, we describe the details of P2P-SIP implementation to perform these logical operations such as contact management and key storage in different deployment scenarios.

5.5 Deployment Scenarios

As mentioned earlier, a P2P-SIP node can run in different scenarios such as the P2P client, proxy or an adaptor for the existing SIP phones as shown in Fig. 5.3. In this section, we illustrate these scenarios using pseudo-code and examples.

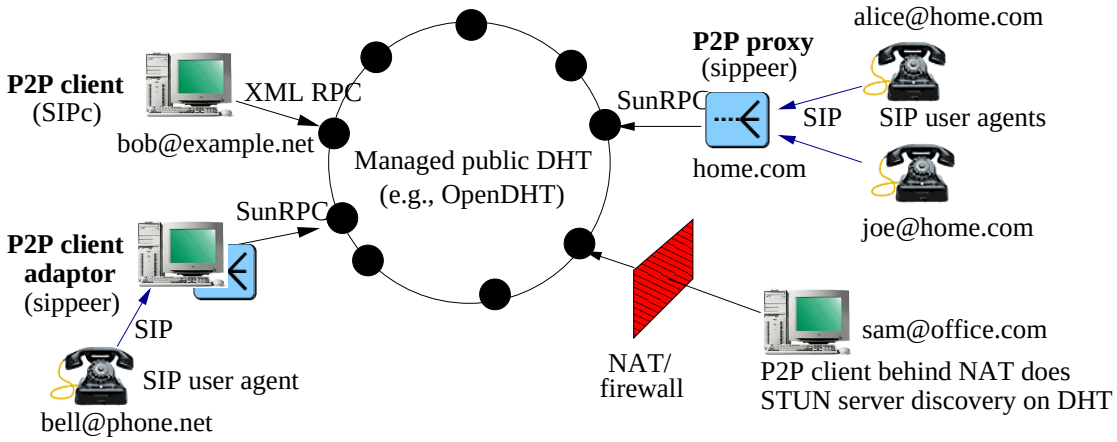


Figure 5.3: P2P-SIP: SIP-using-P2P architecture

5.5.1 P2P Client

Consider a user Bob who picks his identifier, i=bob@example.net. For the first time use, he also picks a secret, s="mypass", and generates his RSA public and private keys, (K_P, K_S). The user's X.509 certificate [126], either self-signed or signed by some trusted authority, is put on the DHT under the key H(certificate:bob@example.net) (see procedure 5.5.1). Other users can get Bob's certificate and hence the public key using his identifier. Bob then encrypts his private key using mypass and puts it in the DHT key, H(private:bob@example.net:mypass). Using the

password in the DHT key prevents someone else from polluting the value under this DHT key.

Procedure 5.5.1: on-startup(identifier: i, password: s)
global: id=n, keys=(K_P, K_S)
$n \leftarrow i$
if $k \leftarrow get(H(\text{private:}i\text{:}s)$ failed **then**
  $(K_P, K_S) \leftarrow generate\ RSA\ keys$
  $put(H(\text{certificate:}i), cert(K_P))$ /* no $H(s) \Rightarrow$ *never remove* */
  $put(H(\text{private:}i\text{:}s), [K_S]_s)$
  /* $[a]_b$ *means encrypt* a *using secret* b */
else
  $K_S \leftarrow [k]^s$ /* $[a]^b \Rightarrow$ *decrypt* a *using* b */
  $K_P \leftarrow extractKey(get(H(\text{certificate:}i))$ verified with K_S

If Bob knows that his certificate issuer's identity may not be known to the prospective callers, he can also put his issuer's certificate on the DHT, say under the key H(certificate:example.net), if the issuer is example.net. Any caller should acquire the chain of certificates until she can trust the issuer.

Procedure 5.5.2: put-contact(id: i, contact: c, ttl: t, password: s)
global: private-key:K_S of signer:n
$e \leftarrow now + t$ **and** $\sigma \leftarrow \{H(i|c|e)\}_{K_S}$
$v \leftarrow (c, e, n, \sigma)$ /* $n = i$ *for P2P client* */
$r \leftarrow H(i|c|e|s)$ /* *password in put* */
$put(H(\text{sip:}i), v, H(r), t)$

Procedure 5.5.3: remove-contact(id: i, contact: c, password: s)
global: keys:(K_P, K_S) of signer:n
$(v, H(r), t) \leftarrow get(\text{sip:}i)$ **and** $(c, e, S, \sigma) \leftarrow v$
if $S = n$ **and** $\{\sigma\}_{K_P} = H(i|c|e)$ **then**
  $v \leftarrow (c, e, n, \sigma)$ **and** $r \leftarrow H(i|c|e|s)$
  $remove(H(\text{sip:}i), H(v), r, t + \delta)$

Now, when Bob wants to register his contact location, say sip:bob@192.1.2.3:5060, he creates an RSA digital signature of this contact. He then creates a value containing his contact, signer's name (which is his own identifier in this case), and the signature. This value is put on

the DHT under the key H(sip:bob@example.net). One problem is that a malicious user can fetch the contacts and signature of Bob, and when Bob's registration expires, registers him again with the old signed contact. Alternatively, the malicious user can use this signed contact to register for some other user, thus messing up with other user's call routing.

To prevent this problem, one can use the authenticated interface of OpenDHT [27]. We use the similar signing procedure on top of the existing interface, until OpenDHT implements the authenticated interface. The signed data includes an absolute expiry time of the registration, the user's identifier and the signer's identifier in addition to the contacts. This will guarantee that the signature can not be used for another user or after it expires. The pseudo-code to add and remove a SIP contact is shown as procedures 5.5.2 and 5.5.3, respectively.

When the registration is refreshed, the planned authenticated interface [27] just updates the TTL of the existing contact record. However, with the existing DHT interface, a registration refresh creates a new record under the key instead of replacing, since the expiration and hence the value is changed. Unless the old record is expiring soon, it is recommended that the old record be explicitly removed to prevent storing dangling contact information in the DHT.

When Alice wants to call Bob, she looks up sip:bob@example.net in the DHT. If Alice knows Bob's public key, from earlier communication, she can use that to verify Bob's signature. Otherwise, she does another DHT lookup for the signer's certificate with DHT key, H(certificate:bob@example.net). If the certificate is found and issuer is trusted, the signature is verified. Otherwise, the issuer's certificate is looked up and the process repeats. Any unverified contact is discarded (procedure 5.5.4). The existing DHT interface may return the same contact multiple times with different expiration, if the old contacts were not removed by the user on registration refresh. After removing such duplicate entries, Alice can call one or more contact location in sequence or parallel. After successfully talking to the right person, Alice remembers his public key, or at least H(public-key), for future communication. This is like the known_hosts file in OpenSSL [126].

Bob may store certificates from multiple issuers in the DHT, in the hope that the caller will recognize at least one of the issuers, and minimize the number of get operations on the DHT. This leads to a friend-to-friend trust model, where after successfully communicating with Bob,

Procedure 5.5.4: get-contacts(id: i)

$V[..] \leftarrow get(H(\text{sip:}i))$ /* *get all contacts* */
$ret \leftarrow ()$
for all u in V **do**
  $(v, H(s), t) \leftarrow u$ **and** $(c, e, S, \sigma) \leftarrow v$
  if $e > now$ **and** S is i or domain of i **then**
    $K_P \leftarrow$ get-public-key(S) /* *procedure 5.5.5* */
    if $\sigma = \{H(i|c|e)\}_{K_P}$ **then**
      append v to ret
return ret

Procedure 5.5.5: get-public-key(id: i)

if $C \leftarrow$ get-certificate(i) **then**
  return public key from C

Alice may herself issue a certificate to Bob. The certificate indicates that Bob is the owner of the private key corresponding to the signed public key of the certificate. Thus, other users who know Alice can verify Bob's certificate.

The keys and certificates are put without any TTL indicating that they should not expire, where as the TTL in the contact data is derived from the desired registration TTL, e.g., usually one hour for SIP REGISTER.

Procedure 5.5.6: get-certificate(X.509 subject's common name: i)

Q.enqueue(i) /* *queue of id's to query* */
$L \leftarrow \{\}$ /* *list of certificates* */
repeat
  $j \leftarrow Q$.dequeue()
  for all c in $get(H(\text{certificate:}j))$ **do**
    L.append(c)
    if c.issuer is **not** known **and** c is **not** self-signed **then**
      Q.enqueue(c.issuer)
until Q is empty **or** chain L is not verified
if L can be verified based on our trusted certificates **then**
  return certificate of i from L

5.5.2 P2P Proxy

When a SIP proxy wants to use the DHT as a location service, it performs similar operations as the client. If there are multiple proxies in the server farm for domain home.com, all of them use the same set of secret (s), public (K_P) and private keys (K_S). The proxy can store the domain's RSA key and X.509 certificate, C, on the DHT (procedure 5.5.1), so that other proxies in the farm can retrieve them. When signing a user contact, the signer's identity is set to home.com.

The proxy also stores the appropriate authentication credentials for the users in the domain for authenticating SIP REGISTER requests. For example, it may store Alice's credentials in DHT key H(digest:alice@home.com:mypass) as shown in procedure 5.5.7. Since most digest authentication [127] implementations use MD5, procedure 5.5.7 stores the MD5 hash of the user credentials, which is sufficient for digest authentication by the proxy.

Procedure 5.5.7: signup-user(identifier: i, password: p)
global: domain=n, secret=s
$h \leftarrow$MD5(i:n:p) **and** put(digest:i:s, $[h]_s$)

When Alice registers with the proxy, the proxy authenticates her using the stored credentials. If the authentication succeeds, it updates the contacts using procedure 5.5.2 like a P2P client (procedure 5.5.8). Similarly, procedure 5.5.3 is used to unregister. Since the contacts are signed over the absolute expiration time, a SIP REGISTER refresh causes one more contact to be added in DHT. The proxy should then remove the old contact using the expiration value of the old registration.

A proxy associated with a domain, home.com, may require that all the incoming registrations belong to its own domain, i.e., user identifier of the form *@home.com. This prevents users having their contacts certified by unrelated third party, e.g., home.com proxy will *not* certify the contacts of bob@example.net.

Alternatively, there can be hosted VoIP services where the proxy may allow any user identifier as long as they signup for the service. In such cases, the service provider should verify that the user is the owner of the identifier, e.g., by sending the signup confirmation on an email to that user@domain identifier.

Procedure 5.5.8: on-register(SIP message: R)

global: domain=n, secret=s, keys=(K_P, K_S)
let i be user id from request-URI of R
$h \leftarrow get$(digest:i:s)
if R **not** authenticated using h **then**
  send response 401 Unauthorized
    realm is n, user is i
else
  $A \leftarrow$ get-contacts(i) /* *existing contacts* */
  let $B \leftarrow$ be R.contacts /* $B := A$ *if "Contact: *"* */
  for all (c,ttl:t) in B **do**
    if $t > 0$ **then**
      put-contact($i, c, now + t, s$) /* *add new* */
      if c exists in A as (c, e, S, σ) **then**
        $v \leftarrow (c, e, S, \sigma)$ **and** $r \leftarrow H(i|c|e|s)$
        remove(H(sip:i), $H(v), r, e - now + \delta$) /* *old* */
    else
      remove-contact(i, c, s) /* *remove expired* */

When the proxy receives a SIP INVITE or other request, it looks up the existing contacts for the destination user and proxies or redirects the call. The lookup is same as that done by P2P clients (procedure 5.5.4).

5.5.3 P2P Client Adaptor

A SIP proxy may also be used as a **P2P client adaptor** for existing SIP phones that do not support P2P-SIP. In that case, the P2P proxy (adaptor) runs along with the SIP phone in P2P clients scenario, e.g., on the same host or within the same trusted network. The adaptor is logically part of the user's phone except that the functionality is split between the phone and adaptor.

It should not be necessary to keep two different passwords, one for digest authentication by the phone to the adaptor and the other by the adaptor to sign the contacts in DHT. To solve this, the adaptor uses basic authentication instead of digest and learns the user's password on the fly on REGISTER. The adaptor behaves like a P2P client instead of a P2P proxy, but implements SIP registrar and proxy. There are two common authentication modes: basic and digest. While digest authentication never transfers the password, basic send password as base64 encoded text.

Although, basic authentication is not supported in SIP, basic over TLS is considered safe and in some case better than digest if the server stores the hashed user credential for digest without encryption. In procedure 5.5.8, the secret s is obtained from the SIP REGISTER request's Authorization header, n is obtained from the From header, and keys (K_P, K_S) are obtained using procedure 5.5.1.

5.6 Security and Trust

In general, DHT provides some protection against malicious nodes since they cannot subvert a specific user identifier, but just the (random) user identifiers that happen to land on their node. In our architecture, we assumed that the DHT is managed, nodes are trusted, and the system will eject bad nodes with reasonably high probability.

Since anyone can pick any user identifier and store the contacts and keys for that identifier on the free public DHT such as OpenDHT, there is some risk of talking to the wrong person. On Unix systems, the known_hosts file contains an encoded ssh fingerprint for each host that this machine has contacted through ssh. Similarly, the P2P-SIP node can store the fingerprint of the user after initial communication. The fingerprint contains the user's identity and public key. The encrypted fingerprint can be put on the DHT for future verifications. If storing the public keys of all the contacted users is not space efficient, SHA-1 is used (procedure 5.6.1). When making a call, the user gets the public key from the DHT and verifies it with the hash stored in his mapping (procedure 5.6.2). The fingerprints can be used as a "friends" list similar to those maintained in popular IM clients such as Yahoo and MSN.

Procedure 5.6.1: sign(identifier: i, public-key: P)
global: private key=K_S of signer
$put(H(i), \{H(i|P)\}_{K_S})$

If the callee can certify his identifier, the caller can decide which one to trust based on the certifying authority in the certificate chain stored on the DHT. For example, if two users signed up for the identifier bob@example.net, where the first is certified by example.net and the second by free-service.com, the caller can pick the first one with high probability of being the correct

Procedure 5.6.2: verify(identifier: i, public-key: P)
global: public key=K_P of signer
for all c in $get(H(i))$ **do**
  if $[c]_{K_P} = H(i|P)$ **then**
    return true
return false

one.

Alternatively, the DHT may provide a service model in which every user first signs up with the DHT providing the mapping between the identifier and his public key. The DHT guarantees that there will be only one user with the given identifier at any time, and can verify his public key when requested. This can be implemented using the existing OpenDHT interface as shown in procedure 5.6.1 and 5.6.2, but requires a signer to sign every new user identifier. We assume for scalability that the new identifiers are not created very often. Also, the signer verifies that the user owns the identifier of the form user@domain, e.g., by requesting confirmation from that email address as mentioned earlier.

One important difference between our approach and Skype [21] is in the use of central servers. Skype uses centralized login server(s) to authenticate the user every time the client is started. On the other hand, centralized certifying authority (CA) in our architecture are contacted only for issuing the initial user or domain certificate. Subsequent user logins just use the DHT without contacting the CA. Thus, this is more scalable than the central login server architecture. In particular, the system can operate even if it is separated from the global Internet.

5.7 Implementation Issues

I have implemented the OpenDHT-based SIP contact management and key storage for P2P client and adaptor modes in our P2P-SIP implementation, SIPPEER [116]. Additionally, with the help of Xiaotao Wu, I have implemented the SIP contact management, key storage, service advertisement and discovery of STUN servers for NAT/firewall traversal, presence, and offline instance message (IM) storage for the P2P client mode in Columbia SIP user agent, sipc [30]. The module that connects to OpenDHT, is called sippeer-connector, and can be replaced by other similar

DHT connectors in future.

The connector connects to the DHT nodes and uses the get, put and remove interface to perform P2P-SIP operations described in this chapter. In this section we describe some of the implementation highlights of SIPPEER [116] and SIPc. The SIPPEER implementation is done in C++, using Sun RPC as the OpenDHT interface [122], whereas SIPc is in Tool Command Language (Tcl), using XML-RPC as the OpenDHT interface. Both use OpenSSL [126] for cryptographic routines. SIPPEER runs on Linux, but can be easily ported to other Unix platforms, and also to Windows, using our portability libraries. SIPc runs on both Unix and Windows platforms.

Redundant Connections

Our implementation periodically downloads the list of OpenDHT nodes from http://www.opendht.org/servers.txt and connects to two or more nodes. It selects the closest node, defined as the one to which the connect socket call takes the least time, from a random subset of the nodes list. It periodically does null RPC calls to check liveness. The list of N (≤ 8) closest nodes is maintained and periodically updated in the host cache.

Alternatively, we can use the DNS lookup for opendht.nyuld.net, which fetches the IP addresses of any two OpenDHT nodes close to the client [122].

Data Format

In the current implementation of SIPPEER, RSA keys are generated using 1024-bit modulus and exponent as 65537. All certificates and RSA keys are currently stored in ASCII PEM (privacy enhanced mail) format. Appendix C describes a better format based on the W3C's recommendations.

In SIPPEER, when a collection of data such as in a tuple or a list needs to be evaluated in a scalar context, e.g., in procedure 5.5.2 for the tuples being put or hashed, the elements in the tuple or list are concatenated together and delimited by nul character. To prevent the ambiguity if the actual data has nul character, data may be base64 encoded before concatenating.

We propose an XML-based data format for interoperability among various P2P-SIP implementations. The details are in Appendix C.

Data Size

One of the restrictions of OpenDHT is that the data size for every put is limited to 1024 bytes. The X.509 certificates sometimes exceed the limit. We wrote another interface layer to put larger data, by splitting it into chunks of no more than 1024 bytes. The original DHT key stores the index containing DHT keys to the individual chunks. Assuming, a 20-byte index (SHA1), a one level indirection can store index of other 50 blocks of 1 kB each, thus giving a total of 50 kB of data under a key. This is more that sufficient for storing user keys, contacts or presence data in P2P-SIP.

We use some ugly hack as shown in procedure 5.7.1 and 5.7.2. If the first byte of the data is the nul character, then the data is assumed to be index of other chunks separated by nul characters.

Procedure 5.7.1: put-large(k,v,H(s),t)

$i = ()$
for all u chunk in v of size $<= 1024$ bytes **do**
  $put(H(u), u, H(s), t)$ **and** i.append(nul+$H(u)$)
$put(k, i)$

Procedure 5.7.2: get-large(k)

list $a[..] \leftarrow get(k)$
for all $(v, H(s), t)$ in a **do**
  if v.first is nul **then**
    $w = ()$
    for all i in v tokenized by nul **do**
      if $u \leftarrow get(i)$ **and** $H(u) = i$ **then**
        $w.append(u)$
    replace v by w in a
return a

Data Expiration

OpenDHT has a maximum TTL of one week for any data item. Although most user contacts have much lower TTL, semi-permanent data such as certificates and RSA keys are limited to

a maximum of one week. To continue using the system the P2P clients and proxies should periodically refresh the certificates and keys on the DHT. Alternatively, there can be service nodes that walk the DHT and slowly refresh all the data.

Storing Time

All expirations and absolute times should be stored in GMT (Greenwich Mean Time) format, because the data, such as contact information containing expiration may be read by a user in a different timezone, e.g., contact information.

Fairness

OpenDHT allocates space quota fairly to different clients, identified by IP addresses. This is achieved by defining a maximum time-to-live (one week) and size (1024 bytes) for stored data, and rate limiting the put for data internally. This means a single proxy handling a lot of users and storing a lot of data, may fail if the quota exceeds. Thus, the current OpenDHT fairness policy favors the P2P client and adaptor modes.

Privacy

Another scenario for the centralized SIP proxy is to use the DHT just as a replacement for back-end database. This is not a P2P mode, as lookup in this client-server mode is still done via DNS and SIP [28]. In this case, the proxy encrypts the data stored on the public DHT, so that others cannot use the data directly. Unlike a P2P proxy, in this mode the proxy works in the server-based architecture. Our SIPPEER, in this mode, encrypts all user contacts on the DHT using a password. This mode does not require signing and verification of the user contacts, since the data is encrypted and not visible to others in the DHT.

Authenticated Interface

Once the authenticated interface is implemented in OpenDHT, some of the procedures of P2P-SIP can be simplified. In particular, the two step put-remove process of register refresh (procedure 5.5.8), will be done using a single put. Also, a get request will return only the desired data

if the public key of the creator is specified. Similarly, certificate and key verification can specify the public-key in get to avoid getting unnecessary data and becomes a single step process.

With the authenticated interface, a caller can invoke $get(i, H(K_P))$ if she knows $H(K_P)$ from previous communication. It is desirable that the SIP phone sends $H(K_P)$, if known, of the intended callee in the outgoing SIP INVITE or other requests to the P2P proxy. For example, the SIP request-URI can carry this as an URI parameter, fingerprint.

5.8 Advanced Services

In addition to the user contact locations and keys, configuration such as "friends" list and media such as voicemails may be stored on the DHT. Any configuration needs to be accessed only by the owner, hence can be encrypted. On the other hand, subscription requests and offline messages are stored and retrieved by two different users, but not accessible by any other users. Thus, the P2P client or proxy encrypts the signed subscription request or offline message using the recipient's public key so that only the recipient may read the request or message.

5.8.1 Offline Messages

The details for offline messages are shown in procedure 5.8.1 and 5.8.2. It allows the caller to store a message and the recipient to read and delete the message. The message, M, is in email format and may have voice attachments. One must be careful in storing large values in the DHT, since the data size may exceed 50 kB now. Using a Merkle tree [128] instead of the one-level indirection we described earlier, solves the problem. The idea is to split a large data into a number of smaller pieces, hash those pieces, and then iteratively combine and rehash the resulting hashes in a tree-like fashion until a single root hash is created.

Procedure 5.8.1: put-offline(caller:(a, A_S), callee:(b, B_P), M)
$k \leftarrow H$(offline:b) **and** $t \leftarrow$ 1 week
$u \leftarrow (a, b, now + t, M)$ **and** $\sigma \leftarrow \{H(u)\}_{A_S}$
$r \leftarrow$ random **and** $v \leftarrow ([(u, \sigma)]_r, \{r\}_{B_P})$
$put(H$(offline:b)$, v, H(H(r)), t$ */* secret is H(r) */*

Procedure 5.8.2: get-offline(user:(b, B_S)
for all (v, t) in get(offline:b) **do**
  $(w, p) \leftarrow v$ **and** $r \leftarrow \{p\}_{B_S}$ **and** $((a, b, e, M), \sigma) \leftarrow [w]^r$
  $A_P \leftarrow$get-public-key(b)
  if $H(a, b, e, M) = \{\sigma\}_{A_P}$ **and** $e > now$ **then**
    /* M is valid; read or replay M */
  $remove(H$(offline:b), $H(v), H(r), t + \delta)$

Alternatively, the caller may store the message, v=(a, b, M), signed and encrypted under any DHT key, $H(v)$, and notify the recipient of the key via email, for example. This method is preferred to avoid congesting the same DHT key for a given user. Another alternative is to build a P2P event notification service to notify the recipient of offline messages when he logs in.

5.8.2 Presence

Subscription request for user's presence is signed and encrypted similar to an offline message, but stored in the DHT key, H(subscribe:presence:alice@home.com), and value as the subscriber's identity, e.g., bob@example.net, if Bob wants to watch Alice.

5.9 Evaluation

In this section, we compare the different deployment scenarios (client, proxy and adaptor) in SIP-using-P2P architecture using the data model. We also describe the performance and reliability of using OpenDHT as an external DHT for P2P-SIP.

Comparison of Deployment Architectures

We consider the number and size of lookups and updates in a typical message flow for different deployment architectures. In our implementation, the lookups for certificates and keys are cached, hence reducing the number of actual DHT lookups for registration refreshes, and outgoing calls to the same destination.

A P2P client typically performs one put operation for every registration refresh, whereas a P2P proxy does one get, put and remove on an incoming SIP REGISTER. Additionally, new

registrations for which there is no cache entry, causes one get for getting the user's private key in a P2P client, and for getting the user's digest credentials in a P2P proxy. In OpenSSL, the RSA private key includes the public key, so there is no need to explicitly fetch the public key once the private key is known on startup of a P2P client. For unregistration, the client and server both make one remove call, and the server additionally makes a get call to get the list of contacts. For signup or first time registration of the user identifier, a P2P client invokes two additional put for RSA keys, whereas a P2P proxy invokes one additional put for the user's digest credentials.

An outgoing call typically involves one get for the contacts and one for the signer's public key, assuming that there is no intentional collision of the signer's public key. The signer is either the callee or the domain of the callee's identifier.

If certificates are used and assuming that a user uploads his own certificate as well as that of the domain he belongs to, and a proxy uploads the domain certificate, then the user signup typically takes two puts by the client. The proxy uploads its certificate once for its domain. An outgoing call to a unknown callee but known domain may involve one extra get for the callee certificate, and to an unknown domain may involve two extra get for both user and domain certificates. In OpenDHT, a single get and put for a certificate resolves to three calls because the data size typically exceeds the limit of 1024 bytes.

Suppose the user's login rate is Poisson distributed with mean λ logins per second, and he remains online with duration that is exponentially distributed with mean interval t_{on} seconds. Suppose the registration refreshes are periodically done every t_r seconds, and the maximum TTL allowed in the DHT is t_{max} seconds. Suppose, out of the total call rate of c calls per second by the user agent, a fraction β of the calls is to unknown user and domains with user certificates and α of the calls to unknown users with domain certificates plus unknown users but known domain with user certificates. Suppose, the user has on an average k contacts. The rate of DHT calls by a P2P client and proxy can be given as follows:

	client	proxy
get	$\lambda + kc(1 + 2\alpha + 2\beta)$	$3\lambda + S(t_{on}, t_r) + kc(1 + 2(\alpha + \beta))$
put	$\lambda + S(t_{on}, t_r) + 8/t_{max}$	$\lambda + S(t_{on}, t_r) + 4/t_{max}$
rm	λ	$2\lambda + S(t_{on}, t_r)$

where $S(t_{on}, t_r) = 1 + \sum_{n=1}^{\infty} \mathrm{P}(t_{on} > nt_r) = \sum_{n=0}^{\infty} \{e^{-nt_r/t_{on}}\}$

Typically, t_{max} is very large (one week for OpenDHT) and t_r is one hour in SIP. A mobile user with high λ generates three times more get and two times more remove for registrations and unregistrations when using a P2P proxy instead of a client. This is because a proxy needs to return a list of current contacts in REGISTER response, and remove the old contacts after put, whereas a client does not generate response and puts just before the old contact expires for registration refreshes, hence it does not have to remove the old contact. An office phone which remains always on typically generates an extra get and remove per hour when using a proxy instead of a client since a registration refresh causes an extra get and remove by a proxy. The rate of DHT calls by an adaptor is similar to that by a proxy.

Performance Evaluation

The maximum request rate is determined by both the number of DHT calls and the data size. Most data sizes are small and less than 1 KB in OpenDHT. Moreover, the network bandwidth also depends on the particular DHT algorithm in use.

If authenticated interfaces are implemented in OpenDHT, then no remove needs to be done for SIP registration by a P2P proxy. However, major benefit of authenticated interfaces is in get bandwidth since the DHT will not return unnecessary or polluted data.

The OpenDHT itself gives a low average latency of few hundred milliseconds, and 95th percentile latency of less than 10 seconds [27]. We found similar performance in our quick test of OpenDHT latency. This is reasonable for a SIP call setup. However, doing DHT lookup for every instant message (IM) is not desirable. Instead, only the first IM in the session invokes DHT get for remote contact information, and subsequent IMs reuse the cached value. Similarly, the IM

sent to the user in friends list can reuse the contact address of the user obtained on last presence notification.

When the client starts up, it gets the presence information of all the users in the friends list, because this user is more likely to call or send IM to one of the users in his friends list. The actual performance depends on three important parameters: how often the user changes his contact information, how many friends the user has in his friends list, and how often the user sends a message or call request to another user who is not in his friends list.

Reliability

OpenDHT does data replication for reliability. This means the P2P-SIP node itself does not have to do any replication. The redundant connection (Section 5.7) takes care of fail-over to the next DHT node if the closest DHT node dies. The service discovery module for locating STUN servers also fails over to the next serving node if the first looked up server does not respond.

Thus, the SIP-using-P2P architecture provides secure, scalable and robust P2P-SIP with tolerable call setup latency. We describe the P2P-over-SIP architecture in the next chapter.

Chapter 6

P2P-over-SIP: DHT Maintenance using SIP

6.1 Introduction

Unlike in SIP-using-P2P, the P2P-over-SIP architecture implements the underlying DHT using SIP. Our P2P-over-SIP architecture supports basic user registration, lookup and call setup as well as advanced services such as offline message delivery, voice/video mails and multi-party conferencing. It uses SIP as the underlying protocol so that it interoperates with existing infrastructure such as SIP-PSTN gateways and server-based IP PBX such as Asterix and FWD (Free World Dialup).

A P2P-SIP node can also act as an adaptor that allows existing or new SIP user agents to connect to the P2P-SIP network without modifying the user agent. For example, it can run on the same host as the PC-based SIP user agent and act as the outbound proxy for the SIP user agent. It can also act as a standalone SIP user agent, proxy or registration server. We have implemented a command line user interface based P2P-SIP node, SIPPEER, using the algorithms described here.

We describe the design and implementation of SIPPEER for basic user registration and call setup using pseudo-code and example messages. We also describe how to extend it for advanced services such as presence and event notifications, firewall and NAT traversal and interdomain operations. The modular design allows reusable and replaceable components. For example,

Chord could be replaced by another distributed hash table (DHT) without affecting the rest of the implementation. The open architecture allows installing new services without affecting the existing design. For example, a new voice mail module can be added to the existing node. Finally, we discuss the security aspects and advanced services such as firewall and NAT traversal in the context of P2P-SIP.

We do not propose any change in SIP. It uses existing SIP concepts such as proxy, registrar and user agent, and messages such as REGISTER to create a P2P-SIP network among the participanting nodes. The P2P-SIP node uses existing SIP headers such as To, Contact and Reason to convey various Chord parameters.

Section 6.2 presents various design alternatives. Section 6.3 gives an overview of the P2P-SIP architecture, user registration and call setup. Section 6.4 describes the detailed design of the DHT using pseudo code and example messages. Section 6.5 and 6.6 describe the user registration and call setup in P2P-SIP. Section 6.7 provides guidelines to extend P2P-SIP for advanced services. Section 6.9 analyzes various security threats and their proposed solutions. Section 6.10 predicts performance of the system in terms of scalability, reliability and call setup latency.

6.2 Background and Design Alternatives

Background on Chord: a Structured P2P Algorithm

Structured P2P algorithms such as Chord [22] focus on optimizing the P2P overlay for lookup latency and join or leave maintenance cost [103] instead of using inefficient blind search by flooding. We use Chord as the underlying distributed hash table (DHT) in our implementation for lookup. Chord has a ring-based topology where each node stores at most $\log(N)$ entries (or state) in its *finger table*, which is like an application level routing table, to point to other peers. Lookup is done in $O(\log(N))$ time.

Consider an example Chord network with six bit identifiers as shown in Fig. 6.1. The identifier range is [0-63]. The node identifier is hash of the node's IP address. The data key is also hashed to a key identifier in the identifier range. Chord suggests using SHA1 as the hash

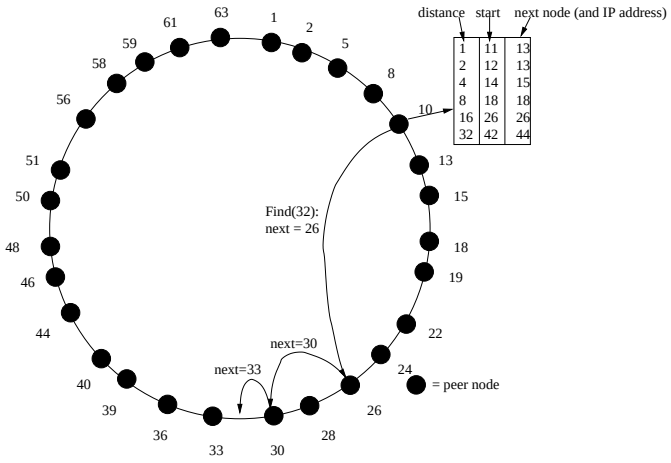


Figure 6.1: Example Chord network

function that generates 160-bit identifiers. The nodes arrange themselves in the identifier circle (or ring) as shown. A node with identifier N_{id} and predecessor N_{pred} is responsible for storing all the keys in the range (N_{pred},N_{id}]. For example, node 22 should store keys 20, 21 and 22 in this example.

Every node maintains a finger table of $\log(N) = 6$ entries pointing to the next-hop node location at distance 2^{i-1} (for i=1,2,..,6) from this node identifier. Node 10's finger table is shown in Fig. 6.1. The finger table contains first nodes with identifiers greater than or equal to 11, 12, 14, 18, 26, 42 for index i=1,2,...,6, respectively. If the node with identifier 11 does not exist then the next available node identifier (13 in this case) is used. The nodes in the finger table are 13, 13, 15, 18, 26, 44, respectively, because nodes with identifiers 11, 12, 14, 42, 43 are not present in the example network. The "next node" column contains both the node identifier and the IP address of the next hop node.

When node 10 wants to find key 32, it looks up the finger table to find the closest match as start value of 26, and sends the query to node 26. Similarly, node 26 in turn sends it to node 30, which finally sends it to node 33. Node 33 is the successor node for the identifier 32 in the network, hence is responsible for storing information about key 32. At each step the distance to the destination is reduced by approximately half, resulting in $O(\log(N))$ lookup latency, if there are N nodes in the ring.

The rest of the architecture describes the mapping between the Chord algorithm and SIP

message processing. We evaluate different design alternatives for user lookup and registration to meet P2P-SIP goals.

Why is node identifier independent of user identifier?

In our first attempt to design P2P-SIP, we derived node identifier using the hash of user identifier. Users registered their identifiers with the system so that other users could locate them. As shown in Fig. 6.2 when the user started her client application and indicated her "screen name" as *alice@office.com*, the node computed the DHT key (e.g., using SHA1 as in Chord) from the name and joined the DHT using this user key as the node key. Alice's key was 42 in the example. When another user, say Bob, wanted to locate Alice, Bob's node used the same hash function to calculate the same key, 42, for Alice, and invoked the `find(42)` method on the DHT. The DHT algorithm located node 42 and then, Bob's application could talk to Alice's application.

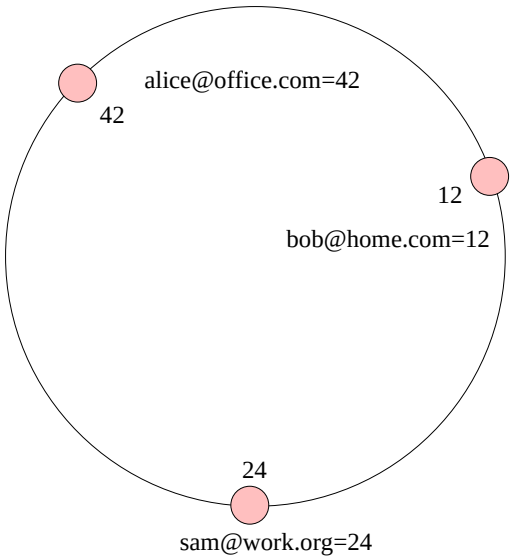


Figure 6.2: No REGISTER

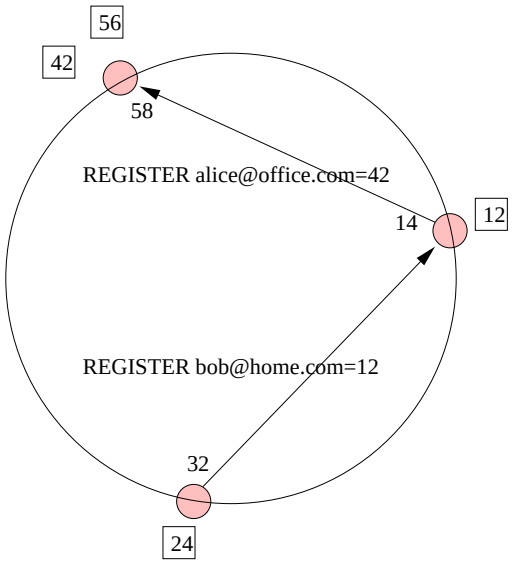


Figure 6.3: With REGISTER

This scheme could not support offline messages or multiple clients registered for the same SIP user identity, *sip:alice@home.com*. For example, if Alice is not present then Bob cannot leave a message for her. On the other hand, in Fig. 6.3, the node key and user key are computed separately. The SIP REGISTER message is used for inserting a node as well as registering a user identity in the DHT. Each node in the DHT acts as a registrar. When Alice starts her application,

the node uses its IP address to compute the node key, 14. In other DHT algorithms such as CAN [101], it may randomly choose a key. It then inserts itself into the DHT based on its node key by sending one or more SIP REGISTER messages to its prospective neighbors in the DHT. The node then computes the key on Alice's name and sends a SIP REGISTER message to the other node, with key 58, that is responsible for the user key 42. For example, Alice's node has a node key of 14 where as Alice's user key is 42, so the node 14 sends a REGISTER message for key 42. The node 58 that is responsible for key 42 accepts the registration and maintains the state that user Alice can be found at node 14's IP address. Even if Alice's application (node 14) is not available, Bob can still leave offline message with node 58 that can later be delivered to Alice when she comes online. Similarly, there can be multiple registrations for the same user key 42, if Alice has multiple active clients.

As an alternative to the SIP REGISTER message, one can use the SIP PUBLISH message to publish the user location and presence status [129]. Both the messages are handled in the similar way for the purpose of this chapter, so the choice does not affect the overall architecture.

6.3 Architecture Overview

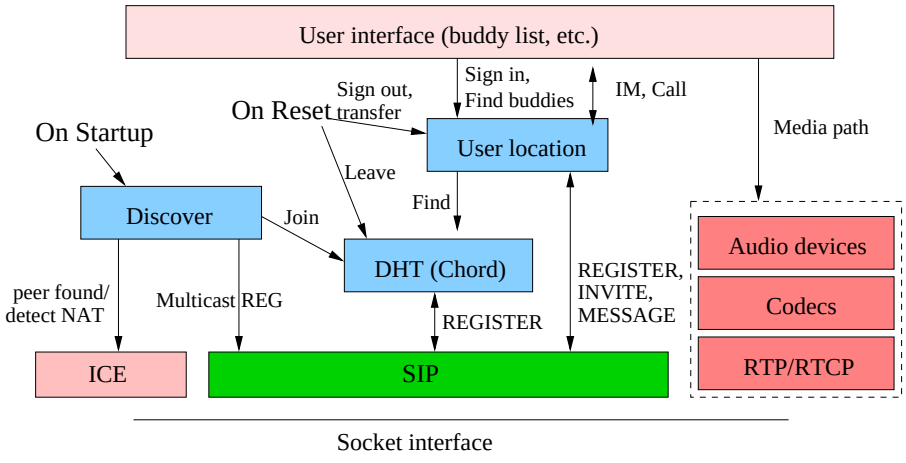


Figure 6.4: Block diagram of a P2P-SIP node

Fig. 6.4 shows the block diagram of the different components in the P2P-SIP node. When the node starts up and the user signs-in with her identifier, the discover module is activated to

initiate NAT and firewall detection, peer discovery and SIP registration. Multicast SIP REGISTER request, cached peer addresses from last boot cycle and pre-configured bootstrap addresses are used to discover an initial set of nodes. The user interface module keeps track of the user's "friends list" and invokes the user location module to locate these friends. User location is obtained using the SIP module or, if this node is a super-node, the DHT module. Typically a node with public IP address, sufficient bandwidth and uptime is made a super-node to form the DHT. In our implementation, we make a node with public IP address a super-node.

The node architecture can be logically divided into two parts: DHT maintenance and user account maintenance. The DHT module maintains the peer information (e.g., Chord *finger table*) and performs DHT operations such as find, join and leave. It provides the underlying topology for communication. The user account maintenance module deals with maintaining local user accounts as well as storing remote user registrations. It acts as a SIP registrar and proxy server.

6.3.1 SIP Layer

SIP is used as the underlying protocol for maintaining the DHT, locating another user, registering the user, call setup and instant messaging. The SIP REGISTER message is used in two contexts by the node: query and update. If a Contact header is present in the message, then it is an update request indicating that the sender wants to update the bindings for the node identifier in the To header. Otherwise, it is a query request, where the sender is requesting to get the Contact information of the node identifier in the To header. Initial discovery uses the REGISTER message for query. This behavior is semantically same as that defined by SIP.

> One can argue that the SIP OPTIONS message should be used in place of the REGISTER message. Since the SIP OPTIONS message semantics is to query the media sessions and supported methods of the recipient end point, but not to retrieve the contact locations of the recipient, we do not use OPTIONS. Moreover, the multicast SIP registration semantics can also be used for discovery of the initial peer node.

Once the user's contact location lookup is done, the call setup or instant messages can be sent directly to the user's phone. SIP REGISTER refresh and OPTIONS messages are used to detect node failure. When a super-node shuts down, the registrations are transferred to other

super-nodes in the DHT as appropriate. Other SIP functions such as third-party-call control and call-transfer can be implemented in the similar way. The media path (audio device, codecs and transport) is independent of the P2P-SIP operation, except that it uses the ICE module in Fig 6.4.

Node and user identifiers are represented using SIP URIs. For example, if a node is listening at transport address 192.1.2.3:8054 for SIP messages and the Chord's hash function computes the key H(192.1.2.3) as 17, then the node's URI becomes sip:17@192.1.2.3:8054. A node identifier or key (e.g., 10) in the domain example.com, whose transport address is not known is represented as sip:10@example.com. This is needed, for example, to lookup node address for node identifier 10 in the DHT, because the node IP address is not already known. Every local P2P-SIP network is represented using a DNS domain name, whereas example.invalid is used for the key that has no domain, e.g., in the global DHT. Such node identifiers are useful for DHT maintenance, e.g., to query another node's transport address to populate this node's finger table entries.

User Alice can register her identifier as sip:alice@example.com. The user's email is used as the identifier so that she can use the authentication mechanism described in Section 6.9.

6.3.2 Node Startup and Peer Discovery

In practice, the client node can try to use both P2P overlay and the SIP-based user lookup. When the node starts up and the user enters her identifier such as *alice@example.com*, the node finds the possible SIP server addresses using DNS [28] and sends a SIP REGISTER message as shown in Fig. 6.5. If the SIP registration succeeds, the node can be reachable using standard SIP mechanism in addition to the P2P mechanism.

The node also tries to discover possible super-nodes so that it can join the P2P overlay. When the node discovers any one other node in Chord, it can join the Chord DHT based on the node key. A number of approaches can be re-used from various existing proposals as follows:

- Multicast with very small time-to-live (TTL) value (e.g., within a LAN) can be used to discover local peers and get more super-node information from these peers. SIP defines multicast registration address for IPv4 as 224.0.1.75. Multicast-based node discovery may result in many disconnected DHT components. To prevent this, only existing DHT nodes

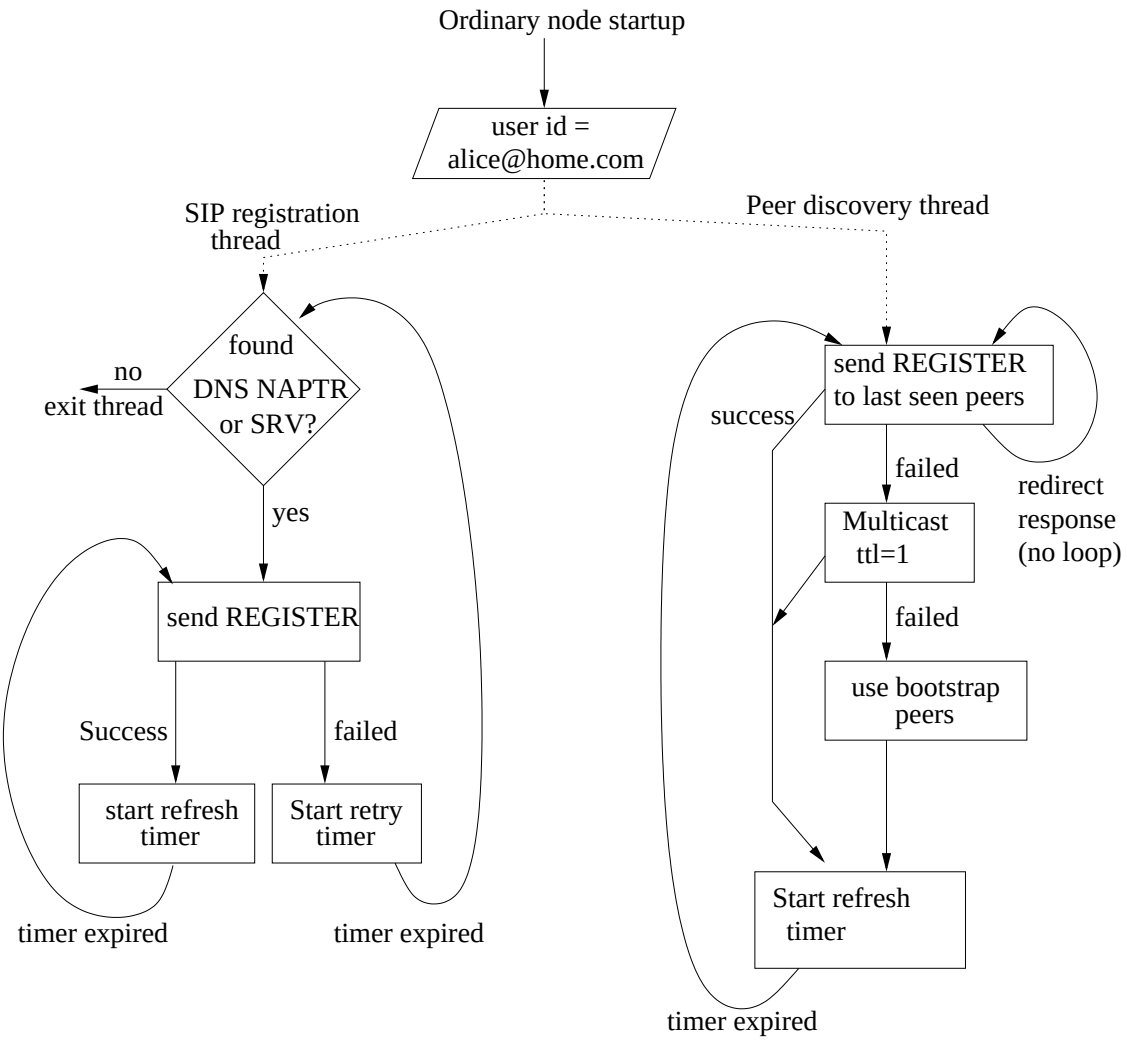


Figure 6.5: Node startup and outgoing registration

(super-nodes) should respond to multicast discovery requests (i.e., ordinary nodes should not get discovered). Limited multicast on wide-area means the system cannot rely on multicast alone.

- Some sort of service discovery can be used, e.g., SLP, to locate super-nodes [130].
- If the peer addresses are cached, then more super-node information can be obtained from those peers assuming the peers are still active and have not changed their locations since last seen.

- As the last resort, some pre-configured bootstrap peers can be obtained from DNS query to a well known domain, e.g., *sippeer.net*, or can be pre-configured in the application software (e.g., as implemented in Skype).

The super-node information is cached for subsequent registrations when the user logs out and logs in again. Hence, the discovery is going to be a one time affair for most installations unless all the cached super-nodes are found to have moved or disappeared.

6.3.3 User Registration

Once a node detects a set of super-nodes, it picks one and sends SIP REGISTER messages to register with it. The To and From headers in the message correspond to the local user identifier, e.g., *sip:alice@home.com*. The Request-URI corresponds to the super-node's address, e.g., *sip:192.2.1.2:5060*.

An ordinary node is just a SIP user agent, whereas a super-node serves as the SIP user agent as well as registrar for other nodes. A super-node sends the REGISTER messages on behalf of the attached nodes to the destination super-nodes in the DHT. It also joins the DHT with other super-nodes and actively takes part in user location lookup.

Ordinary nodes periodically send REGISTER refreshes as keepalive messages to detect any super-node failures. Super-nodes can periodically send the SIP OPTIONS message among themselves or to the attached nodes to monitor liveness. The refresh interval can be adjusted based on the system load. The keepalive OPTIONS message is not sent to a node if some other message was exchanged with that node in the last keepalive interval.

When an ordinary node receives a REGISTER message either for user registration or node registration, it sends the SIP redirect response to redirect the sender to its own super-nodes as shown in Fig. 6.6. When a super-node receives a REGISTER message from an ordinary node and the sender is part of its attached nodes, the super-node proxies the message to the appropriate nodes in the DHT as per user key of the sender. If the sender is not part of this super-node's attached nodes, it can decide either to accept the new node or reject it. If it wants to reject it, it redirects it to some other super nodes which may be less loaded than this super-node. The sender does loop detection to avoid getting into redirection loop.

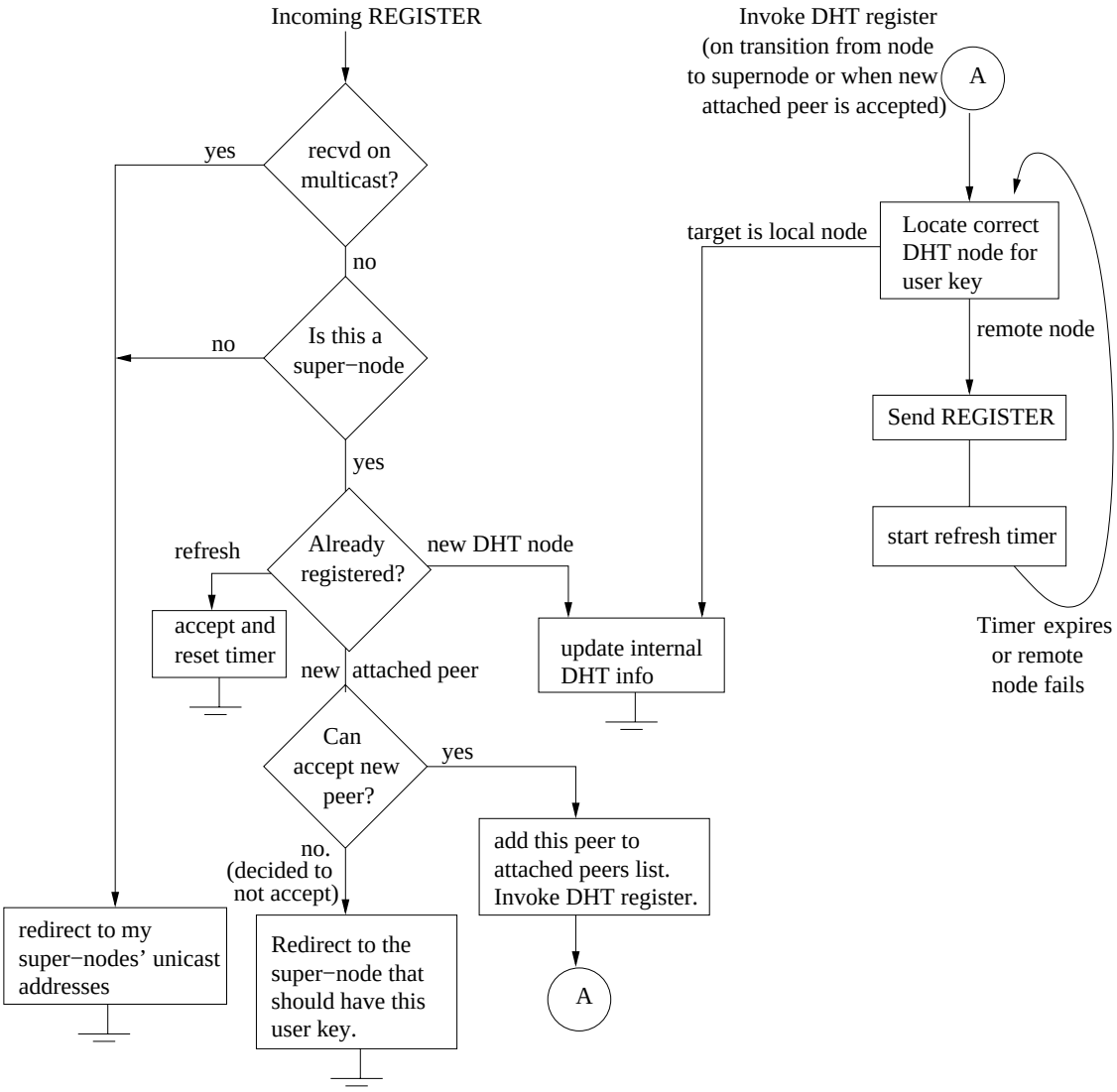


Figure 6.6: Incoming registration

6.3.4 Node Shutdown or Failure

When an ordinary node leaves the system it can just un-REGISTER with the attached super-node which in turn can propagate the un-registration to the corresponding nodes holding this node's key. A failure of an ordinary node does not affect the rest of the system. In any case, the attached super-node can detect the failure by the absence of periodic refresh. It can further confirm the failure by sending an OPTIONS message to the failed node to see if there is any response.

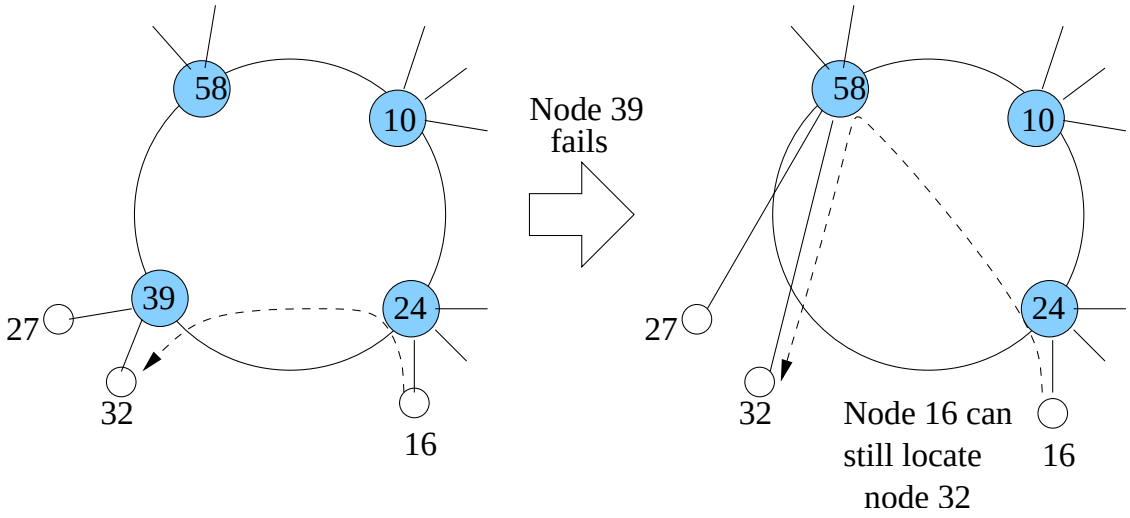


Figure 6.7: Failure of a super-node in the DHT

When a super-node leaves, the state needs to be updated in the attached ordinary nodes as well as the other super-nodes in the DHT that are neighbors of this node. If a super-node is shutting down, it gracefully transfers the user records that it holds to the other nodes in the P2P overlay. This guarantees that others users can locate the record when the DHT node is gracefully shutting down. It sends a SIP REGISTER message to the DHT nodes that will be holding the user records after this node leaves. It does not need to inform the attached ordinary nodes. The attached nodes will detect the failure on the next registration refresh and try to discover and connect to other super-node that holds the record.

When a super-node fails abnormally (Fig. 6.7), the neighboring DHT nodes detect the failure by detecting failed keepalive message and adjust the DHT to accommodate for the keys that were held by the failed nodes. However, the mapping is lost unless the originating node sends REGISTER refresh. The REGISTER refresh goes to the new super-node that handles the corresponding key in the DHT. This make some services such as offline messaging temporarily unavailable (Section 6.7).

To distinguish a SIP-only application with a P2P-SIP application, we can use the Supported header in the OPTIONS or REGISTER message.

6.3.5 User Location and Call Setup

User can watch the presence status of other users by specifying their identities in his “friends” list. If the user already has a friends list, the node tries to locate those friends on startup. Initially we assume that the friends list is stored in the local computer for this user. Later we extend this in Section 6.7 to store any user information (including the friends list) on the P2P network to provide device independence to the user. The IP addresses of all the friends are cached for future use.

The only important step for the purpose of this chapter is locating the node that has the user location record for the destination user. Once the call setup is complete, media packets are sent end-to-end. A node sends the SIP MESSAGE or INVITE message for instant message or multimedia call, respectively. If the destination address is cached because, for example, this node made a recent call or instant message to that destination, then the cached address is used. If the client at the cached IP address does not respond (because there is no client running or the client is not a P2P-SIP node), then the cache entry is removed and discovery is restarted.

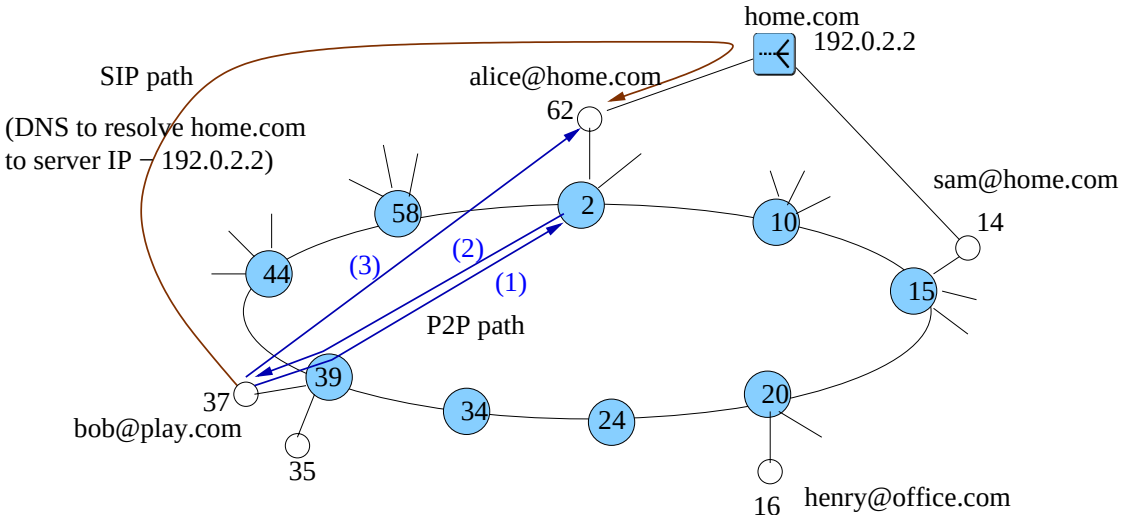


Figure 6.8: User location and call setup

DNS-based lookup [28] and P2P lookup is done simultaneously as shown in Fig. 6.8. For P2P lookup, an ordinary node sends a INVITE or MESSAGE to the attached super node, which acts as a SIP proxy. A super-node locates the destination node holding the key in the underlying

DHT. Once the mapping is obtained, it can either proxy or redirect the message. Redirection is the preferred way as it takes the super-node out of the call loop. However, in some cases such as those involving firewall and NAT, proxy is the only option as we show in Section 6.7.6.

Other SIP functions such as third-party-call control and call-transfer are implemented in the similar way. For example, the SIP REFER message for call transfer is routed similar to INVITE on the P2P overlay. Most of the messages are handled end-to-end directly by the communicating nodes without going over the P2P overlay. Only dialog initiating messages such as INVITE or SUBSCRIBE, or out-of-dialog messages such as first MESSAGE for instant messages need to use the P2P lookup service.

6.4 Details of the DHT Module

The DHT module takes care of implementing three abstract methods: Join, Leave and Find using SIP. When the node starts up it needs to discover at least one other node in the DHT. Then, it joins the DHT through that node. When the node is gracefully shutting down, it leaves the DHT. Higher layer application such as user account maintenance module uses the Find method to locate the next hop node to send user registration or proxy other SIP messages for call setup or instant messaging.

This Section describes the details of the DHT module implementation. In particular, we explain the mapping of the Chord algorithm to SIP messages and processing. We illustrate with simple examples using 5-bit identifiers. We represent the node N's identifier as N_{id}, transport address (IP and port number) as N_{addr}, IP address as N_{ip}, predecessor as N_{pred}, successor as N_{succ}, finger table entry of this node for index i as F^i, and corresponding start, end and next hop node URI as F^i_{start}, F^i_{end} and F^i_{node} respectively. The successor list is represented as N^{list}_{succ}, and i^{th} successor as N^i_{succ}. Note that N^1_{succ} is same as N_{succ}. Finger table entry for another node N is denoted as $N :: F^i$. Note that N_{succ} is same as $N :: F^1_{node}$.

The pseudo code to set the finger table entry and to query the closest preceding finger are described in procedures 6.4.1 and 6.4.2, respectively. If F^i_{node} is set to $node$, then all subsequent F^j_{node} are also set to $node$ as long as F^j_{start} is before $node$ in the Chord ring. To find the closest preceding finger for a given key, the finger table is scanned in the reverse order and the alive node

Procedure 6.4.1: N.set-fingers (i:start index, $node$:node location)

/ Set the finger table entries starting at i */*
/ Returns the index of the last finger entry that gets set. */*
$F^i_{node} \Leftarrow node$
while $i \leq m - 1$ **do**
  if $F^{i+1}_{start} \in [N_{id}, F^i_{node}]$ **then**
    $F^{i+1} \Leftarrow F^i_{node}$
  else
    return i
  $i \Leftarrow i + 1$
return i

with highest identifier preceding the key is returned. Lookup is also done in the successor list, N^{list}_{succ}.

Procedure 6.4.2: N.closest-preceding-finger (key)

/ Find the closest preceding finger for the key */*
$node \Leftarrow N_{id}$
for $i \leftarrow m$ down to 1 **do**
  if F^i_{node} is $alive$ **and** $F^i_{node} \in (N_{id}, key)$ **then**
    $node \Leftarrow F^i_{node}$
    break
for all s in N^{list}_{succ} **do**
  if s is alive **and** $s \in (N_{id}, node)$ **then**
    $node \Leftarrow s$
    break
return $node$

6.4.1 Initialization

When the node starts up, it allocates any available port for receiving SIP messages on TCP and UDP. Our SIPPEER application can accept -p command line option to configure a fixed receiving port number, instead of using any available port. Typically, there are three listening threads, (1) for TCP on INADDR_ANY interface and some port p, (2) for UDP unicast on INADDR_ANY interface and same port p, and (3) for UDP multicast on address 224.0.1.75 and port 5060. To allow receiving both multicast and unicast packets on ports p and 5060, threads (2) and (3) bind

to multicast as well as unicast addresses. If p is 5060, then thread (3) is not created. The UDP sockets are bound non-exclusively on port 5060, so that multiple instances of the node can run on the same host.

The node calculates its node identifier using the IP address of the local interface. For testing purpose we use both IP and port, so that we can start nodes with different node identifiers on the same host. However, in practice only the IP address should be used. This prevents a single IP address from disturbing random parts in the DHT if a malicious node does frequent join and leave to cause high churn.

6.4.2 Peer Discovery

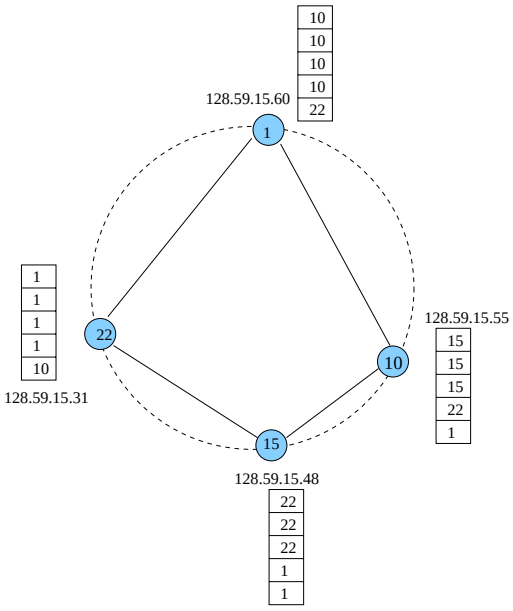


Figure 6.9: Example Chord network with 4 nodes

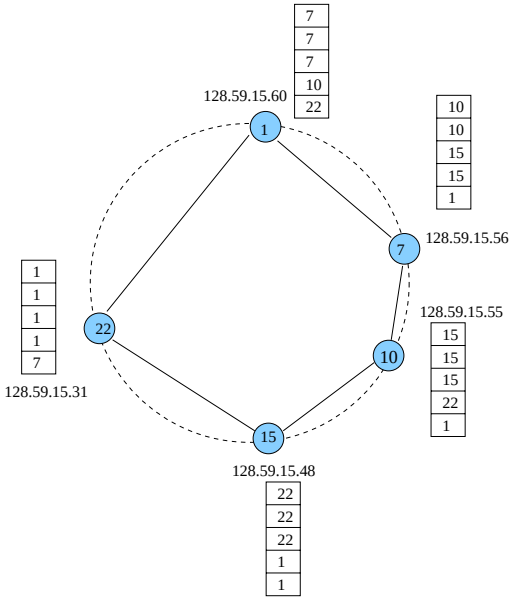


Figure 6.10: After node 7 joins the network

Consider an example Chord network with four nodes as shown in Fig. 6.9. The node identifiers are 10, 22, 1 and 15, and the node IP addresses are 128.59.15.55, 128.59.15.31, 128.59.15.60 and 128.59.15.48, respectively. When a new node, 7, (with transport address 7_{addr}=128.59.15.56:44452) starts up, it invokes its Discover method (procedure 6.4.3) to discover possible peers.

Procedure 6.4.3: N.Discover

if discovery is allowed **then**
  send REGISTER sip:224.0.1.75
    To: N_{id}
else
  for $i := 1$ to m **do**
    $F^i_{node} \Leftarrow N_{id}$
  $N_{pred} \Leftarrow N_{id}$
  trigger join complete event

The Discover method of node 7 sends a SIP REGISTER message with request-URI as sip:224.0.1.75 (SIP REGISTER multicast IPv4 address) and the To header as the local node identifier, 7_{id}. The From header is always the local node identifier, 7_{id}, if the request is generated for the node. (The mandatory SIP headers that are not needed for understanding P2P-SIP are not shown, but must be sent as per SIP specification.)

```
REGISTER sip:224.0.1.75 SIP/2.0
To: <sip:7@128.59.15.56>
From: <sip:7@128.59.15.56>
```

If the application is started with -N option to suppress node discovery, the node state is initialized to reflect a singleton node in the DHT. In that case, all finger table entries and predecessor in the node point to this node's location.

If some other node, say node 22, receives the multicast REGISTER request, and is already part of the DHT, it responds with its own unicast address, 22_{addr}=128.59.15.31, in the SIP 302 redirection response.

```
SIP/2.0 302 Redirect to unicast
Contact: <sip:128.59.15.31>
```

If the node receives multiple final responses, it can choose which one to use. Our implementation uses the first received response. If the node does not get any response within a timeout (we use 30 seconds), it uses other means of discovery. The following possibilities exist but are not yet implemented:

Service discovery: The node can have a service location protocol (SLP [131]) user agent (UA), that discovers other nodes in the domain. Once the node joins the DHT it should register with the SLP directory agent so it other nodes can discover this node. For the Internet, some wide-area service discovery protocol is more suitable [132].

Bootstrap nodes: The node can be pre-configured with a set of IP addresses or domain names to probe for possible peers. For example, the node can query DNS for sippeer.net domain's SIP servers and send the initial REGISTER message to them. At least one of the the initial bootstrap P2P-SIP nodes is assumed to be active for this scheme to work. This may introduce the centralized component, but is limited in scope only to the initial bootstrap process. Once the node starts up it caches other peers addresses for subsequent reboots.

If the node cannot discover any other peer, it assumes that it is the first node in the DHT and initializes its data structures (Chord finger table and predecessor location) accordingly. It also re-schedules the discovery procedure for a later time, say after five minutes.

6.4.3 Joining the DHT

Once other peer(s) are discovered, the node selects one and sends a SIP REGISTER message to its unicast address. For example, node 7 sends the following message to 22_{addr}=128.59.15.31.

```
REGISTER sip:128.59.15.31 SIP/2.0
To: <sip:7@128.59.15.56>
From: <sip:7@128.59.15.56>
```

When node 22 receives the REGISTER on its unicast address, it extracts the *destination key*, 7, from the To header. Depending on the destination key value, k, there are three cases for node N to process the request (procedure 6.4.4): (1) if $k \in (N_{pred}, N]$, then node N is responsible for storing k, (2) if $k \in (N, N_{succ}]$, then node N_{succ} is responsible for storing k, otherwise (3) some other node is responsible for storing k. For case (1), node N responds with a SIP 200 success response containing the Contact header as N_{id} and the predecessor Contact parameter as N_{pred}. Thus, it inserts node 22 immediately before node N in the ring. If the key, k, is same as N (subset of case (1)), but the addresses are different (e.g., two nodes happen to have

Procedure 6.4.4: N.OnRegister (R:registration object, M:request message)
if join is not complete **then**
  ignore M
else if M is a query, i.e., M.Contact is empty **then**
  $to \Leftarrow M$.To.user
  if $to \neq N_{id}$ **and** $to \notin (N_{id}, N_{succ}]$ **then**
    $node \Leftarrow$ closest-preceding-finger(k) */* procedure 6.4.2 */*
  else if to equals N_{succ}'s id, but has different address **then**
    $node \Leftarrow N_{succ}$
  else
    $node \Leftarrow N_{id}$
  if $node$ = N_{id} **then**
    if $to = N$ **then**
      send response 200 OK
        Contact: N_{id}; predecessor=N_{pred}
    else
      send response 200 OK
        Contact: N_{succ}; predecessor=N_{id}
  else
    proxy M **to** $node$

the same hash value for the node identifier), then a global failure (SIP 600 response) is returned, with the Contact header as N_{id}. For case (2), it responds with a SIP 200 success response containing the Contact header as N_{succ} and the predecessor parameter as N_{id}. Thus, it inserts node 22 immediately after node N in the ring. For case (3), it proxies the request to the next hop node based on the finger table $N :: F$. Eventually the request reaches the node responsible for k, which can respond back with the correct Contact header.

In our example, key 7 does not belong to node 22 or successor 1 (this is case (3)), so the finger table is used to find the next hop node. Since the largest i for which $N + 2^{i-1} \leq k$, where N=22 and k=7, is $i = 4$, the next hop is $22 :: F^4_{node}$=1_{addr}, hence the request is proxied to 1_{addr}. Node 1 decides that key k=7 belongs to the successor node 10, (case (2) because $7 \in (1, 10]$) and responds with the success response containing Contact as 10_{addr}. Node 22 forwards this response back to node 7. Note that the predecessor information needed by the Chord algorithm is conveyed in the predecessor parameter of the SIP Contact header.

```
SIP/2.0 200 OK
```

```
To: <sip:7@128.59.15.56>
Contact: <sip:10@128.59.15.55> ;predecessor=sip:1@128.59.15.60
```

Procedure 6.4.5: N.OnRegisterSuccess (R:registration object, M:response message)
if R was a query, i.e., R.Contact is absent **then**
  if M.To = N_{id} **and** N_{succ} is empty **then**
    / set the finger table. */*
    $k \Leftarrow$ set-fingers(1, M.Contact) + 1 */* procedure 6.4.1 */*
    if $k \leq m$ **then**
      / more empty entries in finger table. */*
      / query for the next empty entry. */*
      $id \Leftarrow N + 2^{k-1}$
      send REGISTER M.Contact
          To: sip:id@sippeer.net
      trigger join complete event
    else
      / stabilize here with predecessor. */*
      send REGISTER M.Contact
          To: N_{pred} (or N_{id} if predecessor is empty)
          Contact: N_{id}; predecessor=N_{pred}
  else
    if $\exists i$, such that $F^i_{start} = M$.To **then**
      if $i \leq m$ **then**
        / found a pending query for empty finger table entry. */*
        $i \Leftarrow$ set-fingers(i, M.Contact) */* procedure 6.4.1 */*
        if $i < m$ **then**
          / more empty entries in finger table. */*
          / query for next empty entry. */*
          $id \Leftarrow F^{i+1}_{start}$
          send REGISTER N_{succ}
              To: sip:id@sippeer.net
        else if $i = m$ **then**
          if join is not complete **then**
            / stabilize here. */*
            send REGISTER N_{succ}
                To: N_{pred} (or N_{id} if predecessor isempty)
                Contact: N_{id}; predecessor=N_{pred}
            trigger join complete event

When the discovering node, 7, receives the SIP 200 response with To header as 7_{id}, it updates its finger table with the successor node locations and goes on to find remaining nodes

in the finger table (procedure 6.4.5). For example if the To header in the response is this node identifier, 7_{id}, and the successor for this node 7_{succ} is empty, then the successor is set to be the Contact header in the response, $7_{succ} := 10$. Now, $10 \geq 7 + 2^{i-1}$ for i=1 and 2, so node 7 updates its finger table $F^1_{node} := F^2_{node} := 10_{id}$. The next unassigned finger entry for index i=3 needs to be discovered. Node 7 sends a SIP REGISTER message for F^3_{start}=11. The domain sippeer.net is used as logical domain for node 11_{id} to indicate that the IP address of this key 11 is not known. Alternatively, a domain name such as sippeer.invalid can be used to prevent conflict with a real domain name.

```
REGISTER sip:128.59.15.55 SIP/2.0
To: <sip:11@sippeer.net>
From: <sip:7@128.59.15.56>
```

Eventually node 7 receives the response for this registration, indicating that node 15 is responsible for key 11:

```
SIP/2.0 200 OK
To: <sip:11@sippeer.net>
Contact: <sip:15@128.58.15.48> ;predecessor=sip:10@128.59.15.55
```

When the node gets the SIP 200 response for this REGISTER, it realizes that the To header corresponds to F^3_{start}, and updates the finger table based on the Contact header of the response, $F^3_{node} := 15_{id}$. Since $15 \geq F^4_{start}$, it updates $F^4_{node} := 15_{id}$. Finally, node 7 sends another SIP REGISTER message to discover node $F^5_{start} = 23$, and updates the finger table on response as $F^5_{node} := 1_{id}$.

6.4.4 Stabilization

Chord implements a distributed stabilization algorithm to gradually update the finger tables of various nodes after some node joins or leaves. It also allows a node to add itself to other nodes' finger tables. The stabilization algorithm is periodically started by each node. In our example, after node 7 fills all the finger table entries, it tries to stabilize the Chord DHT if the Join procedure is not yet complete. To initiate the stabilization process, it sends a SIP REGISTER message to

N_{succ}, sets the To header as N_{pred} (or N_{id}, if the predecessor is not known or is empty) and the Contact header pointing to N_{id}. Assuming the predecessor 7_{pred} is known as node 1, then the node 7 sends the following request:

```
REGISTER sip:128.59.15.60 SIP/2.0
To: <sip:1@128.59.15.60>
Contact: <sip:7@128.59.15.56> ;predecessor=sip:1@128.59.15.60
```

Procedure 6.4.6: N.Stabilize

/ This is called periodically by the ChordNode thread. */*
if join is completed **then**
  if $N_{succ} \neq N_{id}$ **then**
    send REGISTER N_{succ}
      To: N_{succ}
      Contact: N_{id}; predecessor=N_{pred}
  else if N_{pred} is not empty **and** $N_{pred} \neq N_{id}$ **then**
    / this is a singleton node in the ring */*
    set-fingers(1, N_{pred}) */* procedure 6.4.1 */*
  if $N_{pred} \neq N_{succ}$ **and** N_{pred} is not empty **then**
    send REGISTER N_{pred}
      To: N_{pred}
      Contact: N_{id}; predecessor=N_{pred}

When the node joins the DHT, it also starts its stabilization algorithm. The stabilization algorithm is periodically invoked by the node to refresh finger table entries, successor and predecessor locations. The stabilization algorithm just initiates the SIP registration for the successor and predecessor nodes with the local contact address in the Contact header as shown below and detailed in procedure 6.4.6). It avoids sending duplicate messages if the successor and predecessor nodes are the same, which happens only when there is only one node in the ring.

```
REGISTER sip:10@128.59.15.55 SIP/2.0
To: <sip:10@128.59.15.55>
Contact: <sip:7@128.59.15.56> ;predecessor=sip:1@128.59.15.60

REGISTER sip:1@128.59.15.60 SIP/2.0
```

```
To: <sip:1@128.59.15.60>
Contact: <sip:7@128.59.15.56> ;predecessor=sip:1@128.59.15.60
```

If the node, N, discovers that the predecessor node is not empty or not same as this node, and the successor is this node (i.e., $(N_{pred} \neq \phi | N_{pred} \neq N_{id}) \& N_{succ} = N_{id}$), then it concludes that there is only one node in the DHT. In that case it sets the successor as the predecessor node and adjusts the finger table accordingly: $N_{pred} := N_{id}$.

When the successor or predecessor of this node receives this SIP REGISTER message with the Contact header, it updates its state (procedure 6.4.7). In particular, if the sending node identifier is closer to the receiving node than the existing predecessor in the Chord ring, then predecessor is set as the sending node identifier. The 200 response contains the successor-list so that the original stabilizing node can update its state with the successor's successor list. The successor-list is sent using the Contact headers with different preference values, $q : 0 \leq q \leq 1$. The preference value indicates how close the successor is to the key, and hence how likely it is to store the data for this key. Higher value indicates higher preference. Suppose there are k successors, then i^{th} successor has $q := 1 - \frac{i}{k}$ for $i = 0, 1, .., k-1$. Chord specifies the successor-list to be of size $O(\log(N))$. Node 10 sends the following response to node 7, indicating node 7's successor list: $\{10, 15, 22, 1\}$.

```
SIP/2.0 200 OK
To: <sip:10@128.59.15.55>
From: <sip:7@128.59.15.56>
Contact: <sip:10@128.59.15.55> ;q=1 ;predecessor=sip:7@128.59.15.56
Contact: <sip:15@128.59.15.48> ;q=.8
Contact: <sip:22@128.59.15.31> ;q=.6
Contact: <sip:1@128.59.15.60>  ;q=.4
```

When the stabilizing node, 7, receives the SIP 200 success response from its successor, 10, it updates its successor list using the Contact headers in the response (procedure 6.4.8). If node 7 discovers that successor node 10's predecessor lies between this node and the successor, $(7, 10)$, then node 7 sets its successor pointer to 10's predecessor.

Procedure 6.4.7: N.OnRegister (R:registration object, M:request message)

/ This is appended to procedure 6.4.4 */*
if M.Contact is present **then**
  if M is not unregister, i.e., expires $\neq 0$ **then**
    if N_{pred} is empty **or** M.From $\in (N_{pred}, N_{id})$ **then**
      $N_{pred} \Leftarrow M$.From
  if M.To = N_{id} **then**
    send response 200 OK
        Contact: N_{id}; predecessor=N_{pred}; q=1.0
        Contact: successor-list[0]; q=.8
        Contact: successor-list[1]; q=.6
        ...
  else
    send response 200 OK
        Contact: N_{succ}; predecessor=N_{id}; q=1.0
        Contact: successor-list[1]; q=.8
        ...

Procedure 6.4.8: N.OnRegisterSuccess (R:registration object, M:response message)

/ This is appended to procedure 6.4.5 */*
if R was not a query **then**
  if N_{succ} = M.To **then**
    $pred \Leftarrow M$.Contact.predecessor
    if $pred \neq N_{id}$ **and** $pred \in (N_{id}, N_{succ})$ **then**
      set-fingers(1, $pred$) */* procedure 6.4.1 */*
    if $pred = N_{id}$ **then**
      $N_{succ}^{list} \Leftarrow M$.Contacts in decreasing q
    if join is completed **then**
      / stabilize the next finger entry. */*
      $i \Leftarrow \lceil \log N_{succ} \rceil$
      if $i < m$ **then**
        $id \Leftarrow F_{start}^{i+1}$
        if $id \in (N_{id}, N_{succ}]$ **then**
          $node \Leftarrow N_{id}$
        else
          $node \Leftarrow$ closest-preceding-finger(id) */* procedure 6.4.2 */*
        if $node = N_{id}$ **then**
          $node \Leftarrow N_{succ}$
        send REGISTER $node$
            To: sip:id@sippeer.net

At this point, node 7 refreshes the remaining finger table entries beyond 10. For example, it locates the next hop for the next finger table entry $F^3_{start} = 11 > 10$, and sends a SIP REGISTER query for sip:11@sippeer.net to sip:10@128.59.15.55 as shown below. If the next hop node for this key belongs to node 7 itself, then the request is sent to the successor, N_{succ}.

```
REGISTER sip:128.59.15.55 SIP/2.0
To: <sip:11@sippeer.net>
From: <sip:7@128.59.15.56>
```

When node 7 receives a response for this query for key 11, it continues to refresh remaining finger table entries (procedure 6.4.5) by sending more REGISTER requests. Fig. 6.10 shows the stable Chord network after node 7 has joined.

6.4.5 Node Shutdown (Graceful Termination)

Suppose node 7 wants to gracefully leave the network. It unregisters with its successor 10 and predecessor 1 (procedure 6.4.9). Once the node 10 and 1 know that node 7 has left, other nodes will eventually know using the stabilization algorithm.

Procedure 6.4.9: N.Leave

if N_{pred} is valid **and** $N_{pred} \neq N_{id}$ **then**
  send REGISTER N_{pred}
    To: N_{id}
    Contact: $N_{id} \cup N^{list}_{succ}$
    Expires: 0
if N_{succ} is valid **and** $N_{succ} \neq N_{id}$ **then**
  send REGISTER N_{succ}
    To: N_{id}
    Contact: $N_{id} \cup N^{list}_{succ}$
    Expires: 0

To unregister, node 7 sends a SIP REGISTER request with Expires header with value 0 as follows. The Contact headers are also present in the request indicating the successor list.

```
REGISTER sip:1@128.59.15.60 SIP/2.0
```

```
To: <sip:7@128.59.15.56>
From: <sip:7@128.59.15.56>
Expires: 0
Contact: <sip:7@128.59.15.56> ;q=1.0 ;predecessor=sip:1@128.59.15.60
Contact: <sip:10@128.59.15.55> ;q=.8
...

REGISTER sip:10@128.59.15.55 SIP/2.0
To: <sip:7@128.59.15.56>
From: <sip:7@128.59.15.56>
Expires: 0
Contact: <sip:7@128.59.15.56> ;q=1.0 ;predecessor=sip:1@128.59.15.60
```

Procedure 6.4.10: N.OnRegister (R:registration object, M:request message)

/ This is appended to procedure 6.4.7 */*
if M.Contact is present **then**
  $to \Leftarrow M$.To
  if M is unregister, i.e., expires = 0 **then**
    if $to = N_{pred}$ **then**
      $N_{pred} \Leftarrow M$.Contact.predecessor
    if $to = N_{succ}$ **then**
      $N_{succ}^{list} \Leftarrow M$.Contacts in decreasing q
      set-node-as-inactive(to) */* procedure 6.4.11 */*
  else
    / See procedure 6.4.7 */*

When node 10 receives the unregistration from node 7, it realizes that its predecessor is leaving, so it updates its predecessor location using the predecessor value, 1, in the Contact header: $10_{pred} := 1$ (procedure 6.4.10). Similarly, when node 1 receives the unregistration from node 7, it realizes that its successor is leaving, so it updates its successor to be the next active successor in 1_{succ}^{list}. When a node location is made invalid, it is removed from 1_{succ}^{list} and $1 :: F$. Any inactive finger table node location is changed to the next alive entry in the finger table (procedure 6.4.11).

Procedure 6.4.11: N.set-node-as-inactive ($node$)

```
/* Set the node as inactive in F and N_succ^list */
for all n in F and N_succ^list do
   if node = n then
      n.alive ⇐ false
/* Update N_succ^list to replace dead nodes */
previous ⇐ N_id
for k ← m down to 1 do
   if F_node^k is not alive then
      F_node^k ⇐ previous
   else
      previous ⇐ F_node^k
```

Node 7 should wait for confirmation responses (until a reasonable timeout) from 10 and 1 before shutting down.

6.4.6 Node Failure and Failover

Node failure, unlike graceful shutdown, needs to be detected automatically by other nodes when the SIP REGISTER message fails. If node 7 fails due to some reason, the neighbors 10 and 1 detect the failure and update their states to reflect it. Our SIP library, libsip++, generates OnRegisterFailed event when the outgoing REGISTER message gets a failure SIP response or times out. Registration failures can happen due to many reasons and at different stages (procedure 6.4.12):

Global failure or SIP 600 response may be received if a duplicate node identifier is detected. For example, this occurs if the hash function generates the same node identifier value for two nodes, $H(A) = H(B)$, but A and B have different transport addresses. If A is already in the network, then REGISTER from B will be rejected with a SIP 600 global failure response. The response contains N_A, so node B can use this address, N_A, as the outbound proxy instead of joining the Chord network directly. This is highly unlikely if a large hash space such as 160 bits of SHA-1 is used.

Discovery failure may happen if there is no other P2P-SIP node in the multicast domain. In this case, other means of discovery should be used, e.g., service location for P2P-SIP server

using SLP, or using the bootstrap nodes to join the network. Alternatively, the node can assume singleton node in the Chord network, and adjust its states accordingly. This is useful within a single LAN environment such as P2P VoIP within an organization.

Besides duplicate node detection and discovery failure, we want to address the case of node failures so that the network can failover automatically. When a node detects that another node has failed, the first node deactivates the failed node's location from its finger table and successor list. There are following cases: (1) if the destination key (To header) and the next hop node (request-URI) were same, that node location is deactivated, (2) if the request was not sent to the successor ($uri \neq N_{succ}$), the next hop node location is deactivated, but not the destination key, (3) if the request was a query (no Contact header), the next hop node location is deactivated, but not the destination key, otherwise (4) the node represented by the destination key (To header) is deactivated. If the next hop node is the predecessor, then the predecessor variable is reset ($N_{pred} := \phi$).

The next step is to re-send the original query request to the new failover hop. If the successor node failed, then the next successor is chosen and the request is sent again to the new successor, if one is found. Otherwise, the query is sent to the next closest preceding finger to the destination key. Only the REGISTER query and not updates, are re-sent, because updates are refreshed anyway in the next stabilization interval. If the new next hop is after the destination key in the Chord ring, then the query is not re-sent, and is considered a failed query.

The node checks for duplicate identifiers when the initial discovery returns a a duplicate successor node identifier (procedure 6.4.13). For example, if node sip:7@128.59.15.56 discovers the successor as sip:7@128.59.15.45, it uses 128.59.15.45 as the IP address of its outbound proxy and does not join the Chord network directly.

6.5 User Registration

The DHT module maintains the underlying P2P overlay network, whereas the user location module takes care of user profile and registrations. Both the modules use the SIP REGISTER messages. We describe the user account maintenance in this section.

Procedure 6.4.12: N.OnRegisterFailed (R:registration object, M:response message)

$to \Leftarrow M$.To
$uri \Leftarrow M$.uri
if $to = N_{id}$ **and** R is a 600-class global failure **then**
  / Discovery failed. Probably duplicate node identifier. */*
  if M.Contact is present **then**
    set-fingers(1, M.Contact) */* procedure 6.4.1 */*
    trigger join failed event
else if uri is multicast discovery address, 224.0.1.75 **then**
  / Discovery failed. Assuming singleton node in Chord. */*
  for $i \leftarrow 1tom$ **do**
    $F^i_{node} \Leftarrow N_{id}$
  $N_{pred} \Leftarrow N_{id}$
  trigger join complete event
else
  $succ \Leftarrow N_{succ}$
  if $to = uri$ **or** $uri \neq N_{succ}$ **or** R.Contact is empty **then**
    set-node-as-inactive(uri) */* procedure 6.4.11 */*
    if $uri = succ$ **then**
      $successor\_failed \Leftarrow true$
  else
    set-node-as-inactive(to) */* procedure 6.4.11 */*
  if $uri = N_{pred}$ **then**
    $N_{pred} \Leftarrow empty$
  $node \Leftarrow N_{id}$
  if $successor\_failed$ is $true$ **then**
    / select the next successor. */*
    if N_{succ} is empty **or** $N_{succ} = N_{id}$ **then**
      / Successor not found. Ignore. */*
    else if $N_{succ} \in (N_{id}, to]$ **then**
      $node \Leftarrow N_{succ}$
    else
      / Do not know where to send. */*
  else
    $node \Leftarrow$ closest-preceding-finger(uri) */* procedure 6.4.2 */*
    if $node = N_{id}$ **then**
      / No more addresses left for successor. */*
  / Now resend only query messages if possible. */*
  if R.Contact is absent **and** $node \neq N_{id}$ **and** $node \in (N_{id}, to]$ **then**
    R.uri $\Leftarrow node$
    re-register using R

Procedure 6.4.13: N.OnRegisterSuccess (R:registration object, M:response message)
/ This is appended to procedure 6.4.5 and 6.4.8 */*
if R was a query, i.e., R.Contact is absent **then**
  $to \Leftarrow M$.To.user
  $succ \Leftarrow M$.Contact
  $pred \Leftarrow M$.Contact.predecessor
  if $to = N$ **and** $succ$ equals N_{id} but has different address **then**
    / duplicate node identifier found */*
    set-fingers(1, $succ$) */* procedure 6.4.1 */*
    trigger join failed event
  else if M.To = N_{id} **and** N_{succ} is empty **then**
    / See procedure 6.4.5 */*
  else
    / See procedure 6.4.5 */*

Suppose the table of user registrations in the node is represented as A such that $A[k]$ is the user registration for user identifier k. Suppose the list of local user registrations is represented as L such that L^i is the i^{th} local user registration.

Procedure 6.5.1: RegisterUser (k:user account or identifier)
$L.append(k)$
$node \Leftarrow N.Find(k_{id})$ */* procedure 6.5.2 */*
if $node = N_{id}$ **then**
  $A[k_{id}] \Leftarrow k$
else
  send REGISTER $node$
    To: k_{id}, From: k_{id}, Contact: $k_{contact}$

Procedure 6.5.2: N.Find (key:identifier to find)
/ Find the next hop node for key */*
if $key \in (N_{id}, N_{succ}]$ **then**
  $node \Leftarrow N_{id}$
else
  $node \Leftarrow$ closest-preceding-finger(key) */* procedure 6.4.2 */*
if $node = N_{id}$ **then**
  $node \Leftarrow N_{succ}$
return $node$

6.5.1 Registration Handling

When a user registers her identifier, say k=alice@example.com, a new local user registration object is created to represent this user. The next step is to transfer this registration on to the P2P network to the responsible Chord node (procedure 6.5.1). Suppose the user identifier key is H(k)=1, then this user registration will be stored in the DHT on the node which is responsible for this key, 1. The DHT's Find method is invoked to get the next hop location and the request is forwarded (see procedure 6.5.2). If the local node is responsible for this key, then the registration is stored locally. For example, when user Alice registers from node 7 (Fig. 6.10), the next hop is 1_{id} so the following REGISTER request is sent:

```
REGISTER sip:128.59.15.60 SIP/2.0
To: <sip:alice@example.com>
From: <sip:alice@example.com>
Contact: <sip:alice@128.59.15.56>
```

The Request-URI may contain the domain part of the user identifier instead of the IP address. The receiving node should authenticate any registrations (Section 6.9). The registration is replicated at all the nodes in the successor-list of the responsible node, 1^{list}_{succ}, by sending new REGISTER requests to the nodes in the successor list. The replication can be done either by the responsible node or the registering user.

When node 1 receives the message, it recognizes that the destination key in To header belongs to a user rather than a node. As shown in procedure 6.5.3, there are following cases: (1) if $key \in (N_{pred}, N_{id}]$, then this node is the responsible node, (2) if $N_{succ} = N_{id}$, then there is only this node in the Chord ring, so obviously this node is the responsible node, otherwise (3) find the closest preceding finger for this key and proxy the SIP request to that node location. In this example, node 1 uses itself as the responsible node and stores the registration for alice@example.com.

Now, node 1 replicates the registration to other nodes in 1^{list}_{succ} (procedure 6.5.4). For example, it sends the following REGISTER message to node 10_{addr}, with To header containing the destination key alice@example.com, From header containing 1_{id} and Contact header

Procedure 6.5.3: N.OnRegister (R:registration object, M:request message)

$to \Leftarrow M$.To
if to is **not** a node identifier **then**
  if $to \in (N_{pred}, N_{id}]$ **or** $N_{succ} = N_{id}$ **then**
    / Register the user locally. */*
    $A(to) \Leftarrow M$
  else
    if $to \notin (N_{id}, N_{succ}]$ **then**
      $node \Leftarrow$ closest-preceding-finger(to) */* procedure 6.4.2 */*
    else
      $node \Leftarrow N_{id}$
    if $node = N_{id}$ **then**
      proxy M **to** N_{succ}
    else
      proxy M **to** $node$
else
  / to is a node identifier. see procedure 6.4.10 */*

containing original contact location of Alice.

```
REGISTER sip:128.59.15.55 SIP/2.0
To: <sip:alice@example.com>
From: <sip:1@128.59.15.60>
Contact: <sip:alice@128.59.15.56>
```

Procedure 6.5.4: $A[k] \Leftarrow M$

$A[k] := M$
if $M.To = M.From$ **or** $M.Reason =$ "$leaving$" **then**
  / This node is responsible for k */*
  for all S in N_{succ}^{list} **do**
    send REGISTER S_{addr}
      To: k_{id}, From: N_{id}, Contact: $A[k]_{contact}$

The receiving node 10 recognizes this to be a registration transfer from one node to another, since the To header and From header are different. It stores the registration without routing it further. It should authenticate the sending node 1 before storing the registration. If the From header is also a user identifier, then the REGISTER request is a third-party registration (e.g.,

secretary registering on behalf of her boss), and should be routed using the P2P-SIP routing algorithm based on the To header. Third-party and transferred registrations should be authenticated at each proxy.

6.5.2 Node Shutdown (Graceful Termination)

When a node gracefully leaves the network, it should transfer all stored registrations to the new responsible node, which is its immediate successor (procedure 6.5.5). For example, when node 1 leaves, it sends the following REGISTER request to $1_{succ} = 7_{addr}$.

```
REGISTER sip:128.59.15.56 SIP/2.0
To: <sip:alice@example.com>
From: <sip:1@128.59.15.60>
Reason: SIP ;cause=480; text="leaving"
Contact: <sip:alice@128.59.15.56>
```

Procedure 6.5.5: N.Leave

/ unregister this node using procedure 6.4.9 */*
/ unregister local accounts in L */*
for all u in L **do**
  $R \Leftarrow N.Find(u_{id})$
  if $R \neq N_{id}$ **then**
    send REGISTER R_{addr}
      To: u_{id}, From: u_{id}, Expires: 0
/ transfer local registrations */*
for all k in keys(A) **do**
  send REGISTER N_{succ}
    To: k_{id}, From: N_{id}, Expires: 0, Reason: leaving

When node 7 receives the registration transfer with the Reason field indicating leaving, it can decide to assume the responsibility for this registration. Node 7 can also conclude that node 1 is leaving based on the node unregistration message, and assume responsibility for all the keys that were transfered from node 1 before. The decision is local to node 7 since assuming responsibility for registration is an extra load. Even if node 7 does not take the responsibility

for the transferred registration, when Alice's user agent refreshes the registration, the appropriate responsible node (which may be 7) will get the new registration. Suppose node 7 decides to accept the responsibility for this destination key, it replicates the registration to all the nodes in 7^{list}_{succ}. That means it sends a SIP REGISTER to its successor 10_{addr} as follows:

```
REGISTER sip:128.59.15.56 SIP/2.0
To: <sip:alice@example.com>
From: <sip:7@128.59.15.56>
Contact: <sip:alice@128.59.15.56>
```

Node 10 had earlier received the replicated registration for alice@example.com from node 1. When node 10 receives the new REGISTER from node 7, it concludes that the responsibility for key alice@example.com has been transferred from node 1 to node 7.

6.5.3 Node Failure and Failover

Node failure is similar to node shutdown, except that the failed node does not transfer registrations. The immediate successor detects that its predecessor has failed and owns the responsibility for the keys from its immediate predecessor. For example, if node 7 fails, node 10 detects the failure, and can decide to assume responsibility for the destinations keys sent by node 7. If node 10 decides to not assume the responsibility, it will get the next registration refresh from Alice's user agent, at that time it can authenticate Alice and assume responsibility.

6.6 Call Setup and Message Proxy

So far we have described only the registration request routing. A SIP request such as REGISTER or INVITE belongs to either an user or a node, based on the destination being the user identifier or the node identifier, respectively. For REGISTER request, the SIP To header is used for computing the key for routing decision. For all other requests (e.g., INVITE, MESSAGE), the request-URI is used to make the routing decision. However, this means that SIPPEER must not modify the request-URI on proxy for non-REGISTER requests.

Message Proxy

When an incoming non-REGISTER request is received, and the request-URI is a user identifier (i.e., not a node identifier), and the request does not belong to an existing dialog or local user on this node, then SIPPEER looks up for the user key in its registered user map, A, as shown in procedure 6.6.1. If no registration is found, then a 404 response is returned if the key belongs to this node, otherwise the request is proxied to the next hop node. If valid registrations are found, the request is proxied to those registered contact locations. Alternatively, a 302 redirect response can be used.

Procedure 6.6.1: N.OnReceiveRequest(T:transaction, M:message)
if M.method == REGISTER **then**
  / user or node registration: procedure 6.5.3 */*
else if M.uri is some node identifier **then**
  / this is for the DHT module */*
else if M belongs to existing dialog **then**
  / let the dialog state-machine handle it */*
else if M.uri is in L^i **then**
  / M is for local user on this node */*
else if M.uri is **not** in $A[k]$ **then**
  / no registration found */*
  if M.uri $\in (N.prev, N]$ **or** N.Find(M.uri) failed **then**
    send response 404 User not found on P2P/SIP
  else
    $next$:= N.Find(M.uri)
    proxy M **to** $next$ without modifying uri
else
  / registration found */*
  $contacts$:= $A[M.uri]$.contacts
  proxy M **to** $contacts$ using parallel forking without modifying uri

6.6.1 Multimedia Call Setup and Instant Messages

SIPPEER allows initiating or terminating a SIP call using the command line interface. When the node initiates a request, or acts as an outbound proxy for an existing SIP client, it tries both traditional DNS lookup for the user domain and P2P lookup for next hop in Chord for the user identifier. When one branch gets a final response, the other branch is cancelled. Alternatively,

some P2P-SIP node can try DNS first and fallback to P2P lookup when it fails to get DNS NAPTR or SRV records.

If the node initiates a call or acts as an outbound proxy, it does both DNS and P2P lookup, otherwise it does only P2P lookup. To detect that this node is acting as an outbound proxy for a third party SIP client, SIPPEER uses the Reason header field. All requests initiated or proxied by SIPPEER has a Reason header field indicating that the DNS lookup was already done. When a SIPPEER node receives a message with this Reason it does not invoke another DNS lookup, but uses only P2P lookup. This is not a standard SIP behavior, though it works for our initial prototype. Eventually, a new SIP header or parameter needs to be defined to convey this information.

Usually, the BYE message is sent directly between the two endpoints to terminate the call, without involving P2P lookup. Other messages such as MESSAGE for instant messaging follow similar lookup mechanism as INVITE. The SUBSCRIBE message handling for locating users in the friends list on startup is described in Section 5.8.

6.7 Advanced Services

Basic call setup is not enough to be competitive in Internet telephony. This section describes some of the advanced services such as NAT and firewall traversal, presence, offline message storage and multi-party conferencing.

Many advanced services can be specified using SIP URIs. For example, sip:staff-meet@conferencing.net can indicate the pre-scheduled conferencing service by conferencing.net domain, or sip:dialog.voicexml@ivr.net can reach the generic interactive voice response service. Such services can be built transparently in the basic implementation. For example, a SIP conference server can register all the pre-scheduled conferences in the P2P network, an answering machine module can register to receive incoming calls on behalf of all the registered users, and a VoiceXML browser can register the specific voice dialog service such as voice mail access.

6.7.1 Offline Messages

This section describes problems with offline messaging. When Alice calls Bob or leaves an instant message for Bob, and Bob is not online, the message should be stored reliably by the system and delivered to Bob when he comes online.

There are three places where we can store the offline messages: the source, the destination or some intermediate node in the P2P overlay. The classical PSTN voice mails are stored in the destination answering machine attached to the callee's phone, or in some cases in centralized voice mail server attached to the destination PBX. Similarly, the P2P-SIP client running on the destination user's machine can store the message if the destination user did not pick up the phone. The problem comes when the destination phone itself is not active or the user has not started her client.

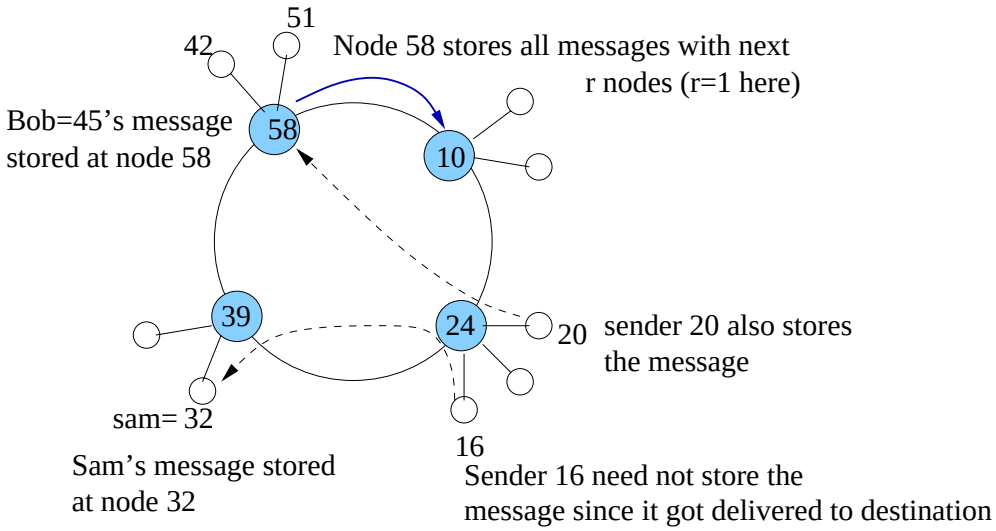


Figure 6.11: Offline message storage

One way to achieve this is by having the DHT peer that is responsible for storing location of Bob, also store the offline multimedia messages for Bob as shown in Fig. 6.11. In the case of super-node failure, the offline messages become unavailable until the storage node becomes online again. To solve this, the node can store the message in multiple places and keep them consistent similar to the Oceanstore architecture [133]. A P2P file storage system with message waiting indication is sufficient to implement offline message storage. POST [134] is a P2P

messaging system that can also be used for offline messages.

Another option is for the caller node to cache the message locally and deliver it to the destination node when the destination becomes available.

The message delivery notification is reliably sent back to the caller. If the message is not delivered or the storage node fails, then the caller node finds the new storage node and records the message again without any user intervention. When a node starts up, it checks for any undelivered message from past boot cycle, and tries to re-send them upon bandwidth and CPU availability. This has certain security issues if the same machine is used by many users as in an Internet kiosk.

Unlike email system, where the intermediate Mail Transfer Agents (MTAs) are reliable and delivery confirmation from an MTA is sufficient, in P2P-SIP an end-to-end confirmation is desirable. Alternatively a third party storage server can take the ownership of the message for the subscribed user, relieving the sender from keeping a copy.

Some nodes may just cache a summary of undelivered messages (such as subject, date, headers) instead of the complete multimedia content to save on bandwidth and disk space. Some nodes may attempt to send the message by alternative means such as email if the email identity can be cryptographically verified to belong to the destination user.

To receive the offline message, the destination node subscribes to the message waiting indication (MWI) event with the P2P network and gets notified on startup when a new offline message is available. The node can then fetch the message using file transfer or real-time multimedia call to a special URI such as *sip:bob-vmail@server*. Alternatively, the user can buy MWI service from some centralized service provider that registers with the P2P-SIP network on behalf of the user to receive her calls.

An alternative approach is to have the user buy the MWI service from some service providers that register with the system with user identifier *bob@yahoo.com* for example. When the caller cannot reach Bob within some time, the server automatically picks up the calls and stores the offline message. One problem with this approach is that it tends to become centralized.

6.7.2 Multi-party Conferencing

In classical telephony, multi-party conferencing is done via pre-arranged dial-in conference bridges (or conference servers). These conference servers can register the intended conference addresses such as “staff-meet@columbia.edu” with the P2P overlay. However, the mixing is done by a centralized server which can become the potential bottleneck for large conferences.

For small scale ad hoc conferencing among the participants, one of the participant who has good capacity (CPU, memory, bandwidth) can become the mixer and mix audio from other participants. Since audio mixing requires access to the un-encrypted audio samples from all the speakers, one cannot pick an untrusted peer as the mixer. One viable alternative is to pick an existing conference participant as the mixer.

Completely decentralized conferencing [135] can be used to establish a full-mesh signaling and media relationship among the participating members. The protocol works for concurrent join and leave of members in the conference. This prevents dependency on a single peer node that does mixing.

Instead of a full-mesh media, a multicast media distribution tree can be used. It assumes that a small number of members (say one or two) will be speaking at any instant, and the receiving node can select or mix the audio samples from multiple streams in the session. Several P2P application layer multicast schemes have been proposed [136, 137], some of which can use the proximity information available in the underlying DHT [102, 138]. The application level multicast seems to be the best option for large scale conferencing in P2P-SIP because of the scalability.

6.7.3 Device Independence

So far we assumed that a user logs in from a particular node and all the user profile information such as friends list or privacy policy are stored in the local node. However, similar to file storage systems or storing offline messages in P2P-SIP, the node can store the encrypted user profile information also in the P2P overlay network [133]. On startup when the user signs in her identifier, the node fetches the profile information reliably and uses that.

6.7.4 Presence and Event Notification

Presence is an important service in Internet telephony. SIP has methods such as SUBSCRIBE and NOTIFY to watch the presence status of a user and to notify the watchers when the presence status changes, respectively.

The basic idea is not different from the call setup and registration, where the responsible node becomes the server for the user identifier. When a watcher subscribes to a user identifier, the responsible node maintains this subscription state. The responsible node detects any change in the user's presence status, either on receipt of incoming registration or explicit publication of presence information by the user using the SIP PUBLISH message. When the presence status of this user changes, the responsible node sends the notification to all the watchers. The responsible node can also perform presence composition in this service model.

Alternatively, to simplify the implementation, the responsible node can use event subscription migration without actually implementing presence. The responsible node receives the subscription request from the watcher, but terminates the subscription when it detects a change in status of the user on incoming registration. Thus, the watcher sends another SUBSCRIBE message, which gets proxied to the current location of the user if the user is available. Thus, this facilitates end-to-end event notification, without having to implement individual events such as presence in P2P-SIP.

Our SIPPEER has only a rudimentary support for event subscription and notification [139, 140] such that the node can store and transfer generic event subscriptions without processing any event package, e.g., presence. Other SIP users agents that support presence or other events can work in conjunction with SIPPEER. In particular, SIPPEER facilitates subscription migration from a P2P-SIP node to subscribee's endpoint and vice versa.

Suppose a subscriber, Alice (alice@example.com), subscribes to the presence status of Bob by sending a SIP SUBSCRIBE message to bob@yahoo.com. Note that the P2P-SIP node may not be able to authenticate the subscription since the subscriber Alice may not be registered with the P2P-SIP network at all. In this case Alice may provide more information about her certificate or public key which Bob can be used to verify the identity.

```
SUBSCRIBE sip:bob@yahoo.com SIP/2.0
```

```
To: <sip:bob@yahoo.com>
From: <sip:alice@example.com>
```

If Bob does not have a valid registration in the P2P-SIP network, the responsible node for Bob's user identifier keeps the subscription information. It responds with a SIP 202 pending response, and a SIP NOTIFY message with Subscription-State of pending (procedure 6.7.1). If SIPPEER understands the event-package (e.g., presence package may be implemented in some P2P-SIP nodes), then it can put appropriate message body in NOTIFY to indicate offline status.

```
SIP/2.0 200 Pending
```

```
NOTIFY <sip:alice@example.com> SIP/2.0
Subscription-State: pending
```

Procedure 6.7.1: N.OnSubscribe (S:subscription object,M:request message)

if no A such that $A.to = S.to$ **then**
  / No valid registrations found */*
  send response 202 Pending
  send NOTIFY $S.from$
    Subscription-State: pending
else if S was active **then**
  / Terminate existing subscription first */*
  send NOTIFY $S.from$
    Subscription-State: terminated; reason=deactivated
  delete S
else
  for all C in $A.contacts$ **do**
    proxy M **to** C
  if a valid 2xx, 401, or 407 response is received **then**
    delete S
    / proxy the response upstream */*
  else
    / do not migrate. respond locally. */*
    send response 202 Pending
    send NOTIFY $S.from$
      Subscription-State: pending

When Bob registers, the subscription is terminated with reason as "deactivated" so that

Alice can subscribe again (procedure 6.7.2).

```
NOTIFY <sip:alice@example.com> SIP/2.0
Subscription-State: terminated ;reason=deactivated
```

Procedure 6.7.2: N.OnRegister (R:registration object,M:request message)
/ This is appended to procedure 6.5.3 */*
for all S such that $S.to = R.to$ **do**
  if S.event is **not** *reg* **then**
    send NOTIFY $S.from$
      Subscription-State: terminated; reason=deactivated
    delete S

If Bob has a valid registered contact, then SIPPEER proxies the SUBSCRIBE message to the contact. If there are multiple registered locations, then the request is forked to all the locations. Once the request is proxied, the SIPPEER node steps out of the subscription path.

When Bob unregisters with P2P-SIP, he sends a NOTIFY message to Alice terminating the subscription with reason "deactivated". Alice subscribes again, and the subscription gets migrated to the responsible P2P-SIP node.

When the responsible P2P-SIP node gracefully leaves the system, it also sends NOTIFY to terminate all the subscriptions for keys stored on that node. Alternatively, the node can send the SUBSCRIBE message to the new responsible node. However, this approach requires additional logic for the node authenticating on behalf of the subscriber to the subscribee, hence not recommended.

Each user identifier, A, is associated with zero or more contact locations, C^i, and zero of more subscriptions, S^j. The algorithm for handing incoming SUBSCRIBE by the responsible node is shown in procedure 6.7.1, and incoming REGISTER for subscription migration is shown in procedure 6.7.2.

One potential problem could be as follows. Suppose Bob registers with his user agent which does not support events. So the SUBSCRIBE request will be rejected, e.g., by "501 not implemented" error code. This terminates the subscription attempt by Alice, who may not retry subscribing. To work around this problem, SIPPEER may use the OPTIONS message to Bob to

find out if Bob's user agent supports SUBSCRIBE or not. It also intercepts the SUBSCRIBE response from Bob. If Bob's user agent fails without notifying Alice, there may be delay before Alice detects and retries.

The P2P-SIP node should implement the registration event package [140] since it acts as registrar for some users. The subscription for event reg [140] is handled locally by the node that is responsible for storing user registrations. This subscription does not get migrated when the user registers or unregisters. When the node storing the subscription is leaving the network, it terminates the subscription so that the subscriber re-subscribes to the new responsible node for the user key.

6.7.5 Adaptor for Existing SIP Phones

A SIP user agent can use the P2P-SIP node as an outbound proxy and take part in the P2P-SIP network. We have tested our P2P-SIP adaptor, SIPPEER, with various SIP user agents such as the Columbia University's sipc, the Cisco IP phone 7960, the Pingtel IP phone, Xten Networks' X-Lite client v2.0 and Microsoft Windows Messenger.

Some phones do not implement outbound proxy as per the SIP specification [3], which says that the outbound proxy should be treated as a pre-loaded route set. In particular, if the outbound proxy does *not* record route the initial INVITE request, then the subsequent request in the dialog such as BYE should not be sent to the proxy. Suppose the sipc user, alice@example.com, INVITEs the Cisco phone user, bob@example.com, using P2P-SIP. After the call, Bob hangs up. The Cisco phone sends the BYE request to the outbound proxy (P2P-SIP node) but the request-URI contains alice@pc2.example.com:5060. The P2P-SIP node may not be able to proxy the request because this URI may not be registered in the P2P-SIP network causing the DHT lookup to fail. We work around this problem in SIPPEER by proxying the request to the request-URI instead of doing a DHT lookup in this case.

6.7.6 NAT and Firewall Traversal

In an ideal world, ISPs and corporate system administrators should enable their NAT and firewall devices with SIP proxies or application level gateways (ALG). However, in practice, this is rarely

done. This forces the application developers to write customized kludges to work around NAT and firewall [141, 142].

There are two aspects to NAT and firewall traversal: automatic detection of the type of NAT and firewall and tunneling though the NAT and firewall devices for inbound or outbound messages. The detection is done at the application startup when the node connects to a super-node. The node implements the Interactive Connectivity Establishment (ICE) algorithm [141] for NAT traversal. UDP is preferred mode of communication. However, if UDP messages cannot be received (e.g., the firewall blocks UDP), then a persistent TCP tunnel presumably to port 80, initiated from the internal node to the external super-node can be used for both inbound and outbound messages.

We refer to firewall or NAT as a *middlebox*, and the internal network behind the middlebox as a *private* network. If a P2P-SIP node in a private network, it does not join the global DHT, but instead uses an existing global DHT node as an outbound proxy. When an existing client (C) uses a P2P-SIP node (P) as an adaptor (outbound proxy), there are three cases: (1) if both P and C are in public network, it does not involve any middlebox, (2) if P is public and C is private, then C needs to implement various middlebox traversal mechanisms, and (3) if both P and C are in private network, then P does not join the global DHT, but uses an existing DHT node as outbound proxy.

Both signaling and media traffic needs to be traversed through the middlebox. SIP signaling traversal through middlebox is handled using symmetric response routing [125] and connection reuse [124]. Interactive connectivity establishment (ICE [107]) is used in conjunction with STUN [105] and TURN [106] to enable media traversal.

We explain how to interwork between P2P-SIP of a private network with the global P2P-SIP next.

6.8 Inter-domain Operation: Multiple DHTs

In real deployments, it is useful to allow multiple P2P-SIP networks (DHTs) to be interconnected. For example, individual large organizations can have an internal P2P-SIP network which is connected to the global P2P-SIP network.

In this section, we propose a two level network: the global (public) DHT represented by sippeer.net and a local DHT, which may be behind a firewall or NAT. Note that the inter-domain operation proposed here is preliminary and needs more experiments as the P2P-SIP work gets matured in the IETF.

Overview

Our hybrid architecture allows both the P2P-SIP network clouds and server-based SIP infrastructure to coexist. There are two approaches: cross register all the users of one network with all the other networks, or locate the user in the other network during call setup. The former method works for small number of known P2P-SIP networks. The latter approach can be implemented using a global naming service such as DNS, or an hierarchy of P2P-SIP networks. In the first case, every P2P-SIP network is represented by a domain name. This is no different from a server-based SIP network where the domain name resolves to one or more bootstrap nodes in that network [28]. In the second case, P2P-SIP is used instead of DNS to resolve the domain name. For example, an individual large organization can have local P2P-SIP network which is connected to the global (public) P2P-SIP network as shown in Fig. 6.12. The local domain-specific DHT has representative server nodes that are also reachable in the global DHT. For example, key private.com maps to nodes A and C in the global DHT. Any node in the domain-specific DHT can reach the global DHT, and any node in the global DHT can reach the domain-specific DHT via the representative server nodes in the domain. The global DHT computes the index based on user identifier of the form user@domain, and if not found then just domain. The local one computes the index based on user for intra-domain calls.

The hybrid architecture allows the user to register with her provider's SIP server, if available, as well as the P2P-SIP network. Call setup is sent to the SIP destination, if resolved via DNS, as well as to the P2P-SIP network.

Registration

Consider the architecture shown in Fig. 6.12 with one global DHT (nodes P, Q, R, S) and two domain specific DHTs. Domain private.com's DHT has nodes A, B, C, D and example.com

has nodes X, Y, Z, where nodes C, D and X are representative server nodes.

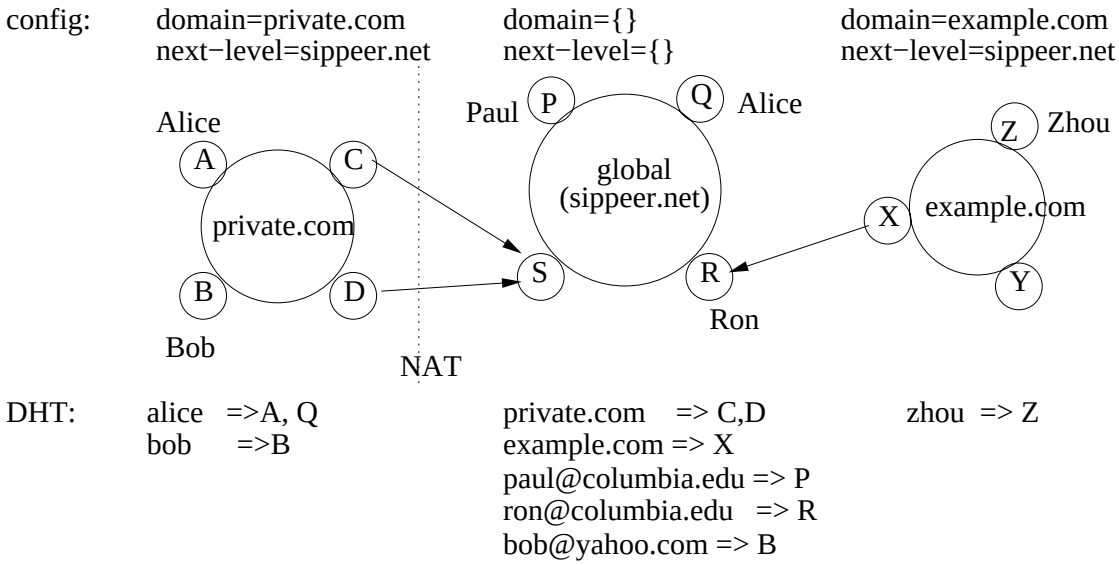


Figure 6.12: Inter-domain P2P-SIP

Every DHT has some bootstrap nodes identified in the DNS entry of the domain. For example, the bootstrap nodes for global DHT are identified by DNS record of sippeer.net, and those for local DHTs of private.com and example.com by their respective DNS records. When a node starts up, it uses its configured domain name and performs DNS NAPTR and SRV lookup for this domain. If no domain name is configured, it assumes global sippeer.net domain. If any IP address matches any of the local interface, the node assumes it is one of the bootstrap nodes for the domain. For example, private.com resolves to IP addresses of nodes C and D, where as example.com to node X.

There are two configuration properties for each node: domain and next-level. The former indicates the domain for the node, where sippeer.net indicates the global DHT, whereas the latter indicates the next level DHT's domain. Bootstrap nodes in global DHT are configured with domain and next-level as empty. When the node starts up it does DNS query and detects that it should be a bootstrap node for the global DHT. Representative server nodes, C and D in the private.com domain are started with domain as private.com and next-level as sippeer.net. When node C starts up, it detects that it is a bootstrap node for its domain. Since C is a bootstrap node and the next-level is not empty, it registers its domain private.com in the next-level DHT

via the bootstrap nodes in sippeer.net domain. The registration gets stored at appropriate global P2P node based on the key private.com. Similarly, nodes D and X register their domains in the next-level global DHT.

```
REGISTER sip:sippeer.net SIP/2.0
To: <sip:private.com>
From: <sip:C@private.com>
Contact: <sip:C_ip_address>
```

The global DHT stores the mapping that private.com is found at node C and D, whereas example.com is at node X.

When a domain-specific node, A, starts up, it discovers node C, e.g., using multicast discovery. Node A gets to know its domain and next-level parameters in the REGISTER response from node C in new SIP headers. It then joins the domain-specific DHT in private.com domain. It also knows that it is not the bootstrap node, so it does not register its domain to the next level DHT. Existing clients such as X-lite do not need to understand domain and next-level parameters, because they will typically be connected to a P2P-SIP node (outbound proxy), and do not take part in DHT directly. Internal DHT nodes maintain their next-level and domain properties, and send that information to other new joining nodes in that DHT.

The domain-specific bootstrap nodes use the P2P-SIP nodes of next-level DHT as outbound proxy. If a domain-specific bootstrap node is in public network, it can directly join the global DHT, in addition to the domain-specific DHT.

Domain administrators may install multiple domain-specific bootstrap nodes to share load. The next-level may be configured as empty so that the domain specific bootstrap nodes do not connect to the global DHT. This allows restricting P2P-SIP calls to within a domain. Nodes may still use DNS [28] to reach outside networks directly without going through the global DHT. Alternatively, administrators may install only bootstrap nodes in the domain as a replacement for SIP proxy and registrar of the domain. In this case, the internal SIP phones use server-based SIP architecture but the domain is connected to global DHT via P2P-SIP.

Call Setup

When a user alice@private.com in a domain using node A, wants to call another user bob@private.com, it discovers that the domain portion of the destination is same as the domain property, so it calls Find(bob) in the domain-specific DHT. The domain-specific P2P-SIP nodes identify the domain, and build the lookup key using only the user part.

When alice@private.com wants to call paul@columbia.edu, the domains do not match, so it proxies the INVITE request to the domain's bootstrap node (C or D) resolved via DNS. Nodes C and D act as proxy to the global DHT, and perform lookup on the global DHT.

When a user paul@columbia.edu using node P in global DHT, wants to call ron@columbia.edu, the domains do not match. This is because node P is configured with domain as empty. In this case it looks up for both keys ron@columbia.edu and columbia.edu. Suppose, ron@columbia.edu is registered from node R in the global DHT, then the call is proxied to node R.

Using similar procedure, suppose paul@columbia.edu wants to call alice@private.com, then it first looks up for both alice@private.com and private.com keys in global DHT. The latter is found to be registered as nodes C and D, so the request gets proxied to C or D or both, which further proxies the request to internal node A which registered as alice@private.com. If such user identifier is not registered, the domain-specific DHT node sends back appropriate failure response, 480 or 404, to the caller.

Suppose alice@private.com on node A, wants to call zhou@example.com. The INVITE request is proxied to C, which in turn proxies to X, which then proxies to internal node Z which registered as this user.

Cross-Domain

The system allows a user in private.com domain to register with user identifier containing another domain. For example, if user on node B registers as bob@yahoo.com, the registration should be propagated to the global DHT. Similarly, a user visiting another network should be allowed to register with her home domain's DHT. We assume such cross registrations are limited in volume and are supported with appropriate authentication.

When a user on node B in domain private.com registers as identifier bob@yahoo.com, the node compares the domain part, similar to the call setup procedure. Since the domain does not match, the REGISTER message is proxied to the domain-specific bootstrap nodes C or D, which in turn proxies it to the global DHT.

When a user on node P in global DHT, registers as identifier alice@private.com, the REGISTER message is first sent with key as the domain private.com. If this fails, then the user key alice@private.com is used for routing. Alternatively, both can be tried in parallel, but will result in duplicate registrations. Since only a few users are expected to cross register, this is not bad.

The OPTIONS request to sip:private.com can be used before sending REGISTER for alice@private.com to detect if the domain-specific servers exist for private.com or not.

When user on node Q calls alice@private.com, it needs to send two INVITE requests, one to sip:alice@private.com and other to sip:private.com. The latter URI is not right since the nodes C or D cannot tell where to proxy the request. There are two alternatives: use URI as sip:user@private.com?p2p-key=private.com or use OPTIONS method to sip:private.com to discover nodes C and D, and then send INVITE to one of those nodes with URI as sip:alice@private.com. Using p2p-key parameter reduces the call setup round-trips but looks like a hack. The problem with Q sending INVITE directly to C or D is that C or D may be behind NAT or firewall and reachable only via P or S, respectively.

6.9 Security

A distributed P2P architecture makes the system more prone to *security* issues such as trust (privacy and confidentiality), malicious node behavior (e.g., call dropping) and DoS attacks [143]. Security is one of the most important problem to be solved for any structured P2P system because of the potentially untrusted peers [144]. The problems include: (1) authentication (to prevent unauthorized calls from spammers), (2) encryption (to prevent others not in the call setup path knowing about the call information), (3) privacy and confidentiality (to prevent sending information to untrusted entity and to prevent misuse of information) and (4) dealing with malicious nodes (what if a peer node happily accepts the call requests but drops them without forwarding

to the appropriate node). The first two problems (authentication and encryption) can be solved using mechanisms similar to those proposed for SIP telephony. For example, end-to-end digest authentication, hop-by-hop transport layer security (TLS) or end-to-end S/MIME can be used.

We do not solve the security issues for P2P-over-SIP architecture, but highlight some of the existing work and potential directions for future work. In particular, the P2P trust and malicious node in a DHT such as Chord is not yet solved, but certain simplifications and assumptions can be made to reduce the problem.

P2P-SIP applications expose existing security threats such as virus and worms to more networked users, even to corporate networks behind firewalls if the firewall allows SIP traffic. Unlike the traditional client-server model where the server is more prone to attacks and most users run only clients, a P2P application acts as a server listening for incoming message on the user machine. In the context of P2P-SIP, there are a number of different types of threats, some of which exist in server-based SIP, whereas others in P2P. In this section we summarize various threats.

Threats: untrusted peers

A number of "untrusted" peers may be involved in user location lookup for a call, unlike the "trusted" servers in the classical SIP telephony. In the classical server based telephony, as long as both caller and callee can trust the server for privacy and confidentiality of the call information, there is no problem. Secondly, the peers may be acting correctly but secretly logging all the call requests which may later be misused.

Freenet [12] solves this problem by hop-by-hop routing of request and responses where each hop (peer) changes the source identifier. This prevents any peer in the request path to know the original sender of the request. Similar techniques can be used in P2P-SIP architecture assuming absence of collusion (i.e., multiple malicious peers collaborating to know the call information), but are difficult if a DHT is used as the underlying P2P network.

Detection and control of misbehaving peers in Chord-based DHT is yet an unsolved problem. There are guidelines that can help reduce the risk. In particular, it is hard to detect a misbehaving node that routes some calls correctly, but drops others. Secondly, the node may secretly

log the call information for later misuse.

The proprietary protocol of Skype makes it difficult for other people to build software that communicates with the Skype clients. Hence, a Skype client can trust the validity of another Skype client (this is not impossible, as Kazaa-Lite and more recently Skype security analysis [145] shown). On the other hand, P2P-SIP based on open protocols cannot trust the validity of another peer. Redundant lookup paths can be used to reduce the risk in structured P2P networks. It will be interesting to answer questions like "how many independent lookups are needed for 99.99% success rate, if at most 5% of the randomly distributed peers are malicious?"

A number of reputation systems have been proposed for P2P [146, 147, 148, 149]. However, they focus on file sharing systems (not real-time), have centralized components, assume co-operating peers or have problems of collusion and multiple identities. Further study is needed to detect the peers who are known to drop calls or do other malicious behavior so that they are not used in the call routing path and not allowed to become part of the underlying DHT.

Besides untrusted trust peers, some of the other security threats are summarized below.

Malicious program: A malicious P2P-SIP application can allow various forms of attacks, break-in, or spread virus, spy-ware or worms. Software developed by trusted entities or open source community can reduce this risk. Even software bugs such as buffer overflow can be exploited by hackers. Running the application as a regular user instead of an administrator (on Windows) or super-user (on Unix) can reduce the risk to some extent.

Copyright violation: P2P-SIP architecture can be easily extended to support file transfer. For example, SIP INVITE can initiate a ftp session using appropriate SDP message body. The problem is similar to other P2P file sharing applications. P2P-SIP does not have an efficient search method, i.e., search for files using regular expression pattern matching. This also reduces the threat, since not many people will use this for sharing music files if the files cannot be efficiently searched.

Stolen identity: The system should prevent a malicious user from stealing the identify of another user. This threat is sometimes also known as authenticity: when you make a call to a user identifier, are you sure that you are reaching the correct user? In P2P-SIP, we reduce the

risk by requiring that the user identify must be a valid email address. (In future this can be extended to a valid telephone number.) The system generates a password for the user identify and sends it to the email address. We describe this mechanism later in this section. The system should be able to authenticate and securely determine whether the user is who he claims to be.

Privacy: Certain user information needs to be conveyed to the other peers to allow call routing. The system should ensure that no sensitive data is conveyed which can be misused later. In particular all signaling and media communication should be encrypted. Privacy and confidentiality in a pure P2P system is difficult. Some parts of the problem is addresses in this section using public key mechanism.

Free riding: There is another kind of threat to P2P systems, called "free riding" [150]. Some nodes may want to use the P2P services for making and receiving calls but refuse to serve in the user location lookup process by becoming a super-node. The system should enforce some policy to discourage such peers. For example, peers can earn some credit for doing services which can later be used for using the services. Every peer can start with an initial amount of credit. Peers behind a NAT and firewall may have to pay for the service if they cannot serve by becoming a super-node. Nodes that run out of credits and refuse to pay are declined the service. A BitTorrent-like approach is useful: if a peer can be a super-node, then it can connect to other nodes only if it also routes calls.

If the user identity is easy to obtain (e.g., yahoo.com email addresses), then people can always acquire new identities to make outbound calls. However, they won't be able to advertise their identity for incoming calls for long period, if they do not serve in the P2P overlay or pay credits to other serving peers. Moreover, making outbound calls does not entitle them to free gateway access or free PSTN calls.

Besides the above threats there are more threats in P2P systems such as anonymity [12] and accounting. Caller anonymity can be provided by having the SIP outbound proxy hide the identity of the caller. Call accounting is needed for PSTN calls, and can be provided by the gateway.

Accounting within P2P-SIP nodes is not required. Finally, if automatic software updates are incorporated in P2P-SIP nodes, then it must be done in a reliable, secure and decentralized manner.

Some of the security problems in P2P are hard to solve. There is a tradeoff between security risks and convenience of server-less systems. We divide the problem into multiple stages and analyze each of them below.

Identity Protection

User identifiers can be randomly assigned by the system, chosen by the user as a screen name (e.g., alice172@sippeer.net) or chosen by the user as her valid email address (e.g., alice@example.com). The first two approaches allow the user to choose her password, but it is not clear how the P2P node can get the password from the user to verify. We use the last approach as it allows the system to generate a random password and email it to the user for authentication. In the first two approaches, if a password is randomly generated by the system, it can be mailed to the user if the Contact header in the SIP REGISTER request has an email address.

When a user signs up with the P2P-SIP network for the first time, we need to verify that the user identifier is valid and indeed belongs to the user. In the absence of public key infrastructure (PKI), the system can generate a new password and send it in an email to the user as mentioned earlier. This requires that the user identifier be same as her email address. For example, when Alice signs up with identifier alice@example.com by sending a SIP REGISTER message, the responsible node generates a random password for Alice and sends it in an email to alice@example.com. It then challenges Alice with digest authentication [127]. We use the domain part of the user identifier as the realm for authentication. The responsible node maintains the authentication information (user identifier, realm and MD5 hash of "user:realm:password") on the DHT. The information is indexed by the user identifier. This information is required and sufficient for future authentication of any user signing up with the same identifier. A reasonable time-to-live, say one week, can be used. The information is refreshed when the user subsequently signs up. So if the user identifier is unused for a week, subsequent sign-in generates new password sent to the user's email address.

The email sent to Alice contains the user identifier, realm and password. It also contains

the IP address (or other identifying information) of the original sender of the REGISTER request, so that Alice can report abuse if she was not the one trying to sign up with P2P-SIP. When Alice receives the password, she signs up again with the appropriate credentials. Subsequent sign-up follow the same procedure.

When a registrar node (A) shuts down, the registration is transferred to another DHT node (B). If node B trusts node A, it just needs to authenticate A, otherwise it re-generates a new password and sends it to the user's email address. We believe that once we have a P2P reputation system, only the more trusted nodes will be present in the DHT. The problem is still there if the registrar node is malicious, and can cause denial of service (DoS).

Sending in email is just one of the ways. Alternatively, if a group of users already have user certificates from other trusted entity such as VeriSign, they don't need to do email based certificates. Another possibility is to also allow telephone number identity (tel URL) if the user can call from that telephone number (with caller id) to a interactive voice system that verifies that the user owns this telephone number and issues a new certificate. This way other friends who know his phone number instead of email address, can also reach him on p2p-sip. Making an outbound call to a telephone number (similar to sending an outbound email) for identity verification is not a good idea, unless user pays for the call.

Misbehaving Nodes

Certain guidelines can be followed to detect and avoid misbehaving nodes [151]. For example, the caller can prefer the redirect (iterate) mode of operation, so that it can monitor at each step whether the routing is as per the DHT specification. There should be no single point of routing decision. In our current implementation, the responsible node also does replication. So a misbehaving responsible node can make the user unavailable.

Generally speaking there are known three models to prevent misbehaving nodes in P2P: (1) hide the security algorithms and protocols, so that only the single vendor implementation will be running on the node (e.g., Skype), (2) form a social network of peers in unstructured P2P system, or (3) keep only trusted nodes in the structured P2P network (e.g., OpenDHT is a managed P2P network). Since identifying untrusted nodes is difficult, we may want keep both

trusted and untrusted nodes in a structured P2P, but nodes should be able to selectively trust other nodes during call routing and registrations. Thus, the requirement does not fit any of the known security mechanisms for handling untrusted nodes.

One option is to use redundancy in lookups. However, if at each of the $\log(N)$ steps in the lookup path, the request is sent to two nodes, then the request can traverse $O(N)$ nodes which is inefficient. Alternative is to build multiple independent DHTs (e.g., Chord using different hash functions) using the same the set of nodes, and perform lookup and update on all the DHTs. For example, if a fraction, f, of N nodes are malicious independent of each other, then the probability of successful lookup is $(1-f)^{\log(N)}$. With k independent Chord-based DHTs, the probability of failed lookup on all the DHTs is $(1-(1-f)^{\log(N)})^k$. For example, if only 1% of the nodes are malicious, then in a DHT with one million nodes ($N = 2^{20}$), 19% of the lookups will fail. But with two independent DHTs this reduces to about 3% failures. This approach increases the DHT maintenance and state overhead by a factor of k, and does not work well with higher f or N.

Malicious nodes cause two kinds of problems visible to the user: (1) denial of service (DoS), i.e., the user identifier becomes unavailable, and (2) intercept, i.e., call goes to the wrong person. The latter can be detected using end-to-end authentication assuming a previous communication has happened, or using a chain of certificates assuming both the caller and callee trust at least one certifying authority (CA). The former (DoS) is difficult to eliminate without a P2P reputation system. "Node calling itself" mechanism can be used for detection to some extent. For example, multiple identities can be created per user including the test identities, and the system periodically makes calls or sends instant messages from one to the other to check correctness. The nodes can periodically verify the routing correctness, e.g., by making calls to itself through some other node. Such a probe-based approach assumes that the nodes are not able to distinguish between a normal request and a probe request.

To build a reputation system one approach is to have separate systems (DHTs) for user lookup, and reputation. This is similar to the judiciary or press system in a real world, which keeps a tab on misbehaving people. The nodes in this reputation system can be selected (or elected) from the original P2P network. Thus, the node can serve in the reputation system along with the lookup system. This service should be temporary (i.e., limited in time) to give chance

to every node instead of having a few nodes hog the reputation board. The election may be based on some criteria such as past records of service, or random. Using the past history may be abused if the nodes provide good service to build the reputation, and finally misbehave when they serve in the reputation board. However, a random selection is always prone to some fraction of misbehaving nodes getting elected.

In such a democratic system, it should be emphasized that a single malicious node should not be able to invite many other malicious nodes in the network. Formally, if only a fraction, f, of the nodes in the P2P network are malicious, then probabilistically at most f fraction of the nodes in the reputation board can be malicious.

The next question is how to detect whether a node misbehaved or not? When a node (A) detects that another node (B) did not forward its request, or forwarded it incorrectly, it can report this to the reputation system. The reputation system, which has rough idea about the P2P network (i.e., which nodes are responsible for what key ranges?) can update the reputation of B, possibly after consulting and querying B. Such a report from A to B needs to cryptographically verify the message exchanges as seen by A.

The size of the reputation board is much smaller than the original P2P network. The exact size of the reputation board will depend on the particular DHT and the election algorithm.

Data Privacy

In addition to the misbehaving nodes which can disrupt the DHT lookups, users also need to store some information on the DHT nodes, which may be untrusted. There are three types of information about the user that can be stored on another node.

public: P2P node should be able to see the information for message routing, authentication, or other processing. For example, user's encrypted password, contact locations (SIP Contact header including preference value, expiration time and URI), voicemail options (such as timeout to go to voicemail, maximum message size, etc). Note that this information is not public to everyone, but only to the P2P nodes that help in lookups.

private: Only the user should be able to see and modify this information. Private data must be

encrypted by the user before storing on the node. For example, user's address book, groups, calendar appointments, watcher and watchee list, programmable scripts (e.g., LESS, CPL, SIP CGI or servlet) and other profile information.

protected: User should be able to see and modify the information, but some other user should be able to create the information. The storing node should not be able to see or modify the information. For example, voice/video mail, and offline messages. Protected data is encrypted by the sender using the recipients public key, and decrypted by the recipient.

Programmable Call Routing

The responsible node cannot trust the registered user except that it can store her information and route her calls. For example, untrusted programmable call routing scripts such as SIP-CGI and SIP Servlet will not be run by the responsible node on the user's behalf. On the other hand, trusted and secure CPL scripts can be run by the responsible node. However, this is purely a local decision by the responsible node.

User Aliases

User can have alias names or other names. For example alice@example.com may also have alias as webmaster@example.com and other names as Alice.Smith@example.com. These are treated as user identifiers and all profile information must be duplicated. Sharing the profile information among the aliases causes complicated trust requirements. On the other hand, user will typically have provisions in her user agent to register with multiple user identifiers or line presence, so that does not require support from P2P-SIP.

Alternatively, a user can maintain a primary identifier such as alice@example.com and point all other identifiers such as Alice.Wonderland@yahoo.com and aw76@columbia.edu by registering them with contact as the primary identifier. This avoids duplicating the profile information for secondary identifiers, but increases the call setup latency when someone wants to reach the user by her secondary identifier. To avoid going into a search loop, the responsible node for the secondary identifier will typically redirect the call request to the primary identifier. The caller's phone then retries search for the primary identifier on the P2P-SIP network.

To simplify the implementation, aliases follow the same procedure for first time log-in, i.e., alias must be a valid email address and the password is sent to the email address represented by the alias identifier.

To summarize the security discussion of P2P-over-SIP, the security threats such as stolen identity and privacy can be solved, but the malicious nodes that do not forward the lookup requests or secretly log the communication are hard to solve without a centralized reputation system.

6.10 Performance Evaluation

In this section we evaluate the P2P-over-SIP architecture in terms of scalability, reliability and call setup latency.

Scalability

Scalability of the P2P-SIP network depends on the capacity (bandwidth, CPU, memory) of the individual participating super-nodes. Suppose there are N super-nodes in the Chord ring, identifier space is m-bit long (i.e., the identifier range is 0 to $2^m - 1$), number of registered users in the system is n (such that number of keys stored per node is approximately k=$\frac{n}{N}$), REGISTER refresh rate to successor and predecessor to keep the Chord ring correct is r_s, refresh rate for finger table entry is r_f, call arrival is Poisson distributed with mean c per node, user registration is uniformly distributed with mean interval t per user, and node joining and leaving are Poisson distributed with mean λ. Because average lookup in Chord travels through $O(\log(N))$ nodes [22], the finger refresh messages, call arrival messages and user registration refresh messages travel $O(\log(N))$ hops. There are $O(\log(N))$ finger table entries per node. Node join and leave generate $O((\log(N))^2)$ messages. The average message rate per node is sum of the message rates due to refresh, call arrival, user registration and node join or leave, which can be given as:

$$M = \{r_s + r_f(\log(N))^2\} + c.\log(N) + \frac{k}{t}\log(N) + \frac{\lambda(\log(N))^2}{N}$$

The message rate in the node determines the bandwidth and CPU utilization for the node. If each node can handle C requests per second, then the equation $C = M$ gives the maximum possible number of nodes, N_{max}, in the system, which roughly translates to $N_{max} = 2^{\frac{C}{r+c}}$ for

large N, where r is the refresh rate and c is the call rate. Note that λ is low because nodes which often join and leave are not made super-nodes.

Suppose the node supports 10 requests per second (which is much less than the typical capacity of hundreds of requests per second as mentioned in Chapter 3) with minimum refresh interval of one minute ($r = \frac{1}{60}$) and mean call rate of one call per minute per node, then the maximum number of nodes in the system can be 2^{10*30}. Our SIPPEER implementation can support about 800 outgoing registrations per second, for example. If more nodes join the system, the super-nodes become overloaded and may deny some incoming call, registration or proxy requests. However, large values of N also increases the call setup latency as we describe below.

Reliability

When a node fails the user registrations stored on that node are lost. To achieve reliability, the refresh rate can be increased (so that node failure detection happens quickly), the user registration refresh rate can be increased (so the the user record is unavailable only for a brief period of time) or the user registration record can be replicated at multiple nodes (e.g., store the user registrations at $\log(N)$ successive nodes in Chord).

Chord provides reliability against node failure by storing $\log(N)$ successor addresses and replicating keys at some constant (K) number of successive nodes. In P2P-SIP, the node update response contains all the $\log(N)$ successor addresses, and user registrations are replicated at K successive nodes. The equation for average message rate does not change if λ includes failure rate along with node join and leave rates.

When a node gracefully leaves the network, it unregisters with its successor and predecessor so that they can update their Chord data structures. It also transfers all the registrations to the successor. When a node fails abnormally, its successor and predecessor detect the failure and update their data structures. The stabilization algorithm ensures that the information gets propagated to other relevant nodes in Chord over a period of time.

The P2P-SIP node that stores the user registration, also proxies the call request to that user. Once the call setup is complete, the P2P-SIP node is not needed in the call path.

Call Setup Latency

The P2P advantages come at the cost of increased call setup latency. For example, with 10,000 nodes in Chord, the average lookup path length is six hops [22], so P2P call setup will take about six times more than traditional client-server call setup in SIP. With good network condition, single lookup (INVITE response) in SIP is expected to take less than 200 ms. So one or two seconds delay before the phone rings in P2P-SIP is tolerable given that on an average the phone will ring for much longer before the callee picks up.

Due to P2P synchronization latency which depends on refresh rate and node join, leave and failure rates, there may be delay in updating the user records. In this case, it may take multiple retransmissions before call setup is complete. This further increases the call setup latency. Successful user location in Skype takes about three to eight seconds [104].

Some kind of hybrid system may be implemented that takes the advantages of many different structured and unstructured P2P algorithms to further reduce the latency and maintenance cost. For example, there has been recent proposal on one hop lookups for P2P [152] assuming large storage space in the peer nodes.

6.11 Conclusions

We have described a pure P2P architecture for SIP telephony. The architecture provides zero configuration, robustness and scalability inherent in P2P systems, in addition to interoperability with existing SIP infrastructure. The advantages come at the cost of increased call setup latency. Note that the media is sent directly between the two parties without going through the SIP proxies in both the client-server and P2P arcitectures and hence, media delay is uneffected.

For SIP-using-P2P, we have presented an example architecture using OpenDHT as an externally managed peer-to-peer network. We explained various P2P deployment components such as clients, proxies and adaptors using pseudo-code and examples. We also presented some of the design issues based on our implementation. The architecture can be used for other DHTs with similar interfaces. Based on our analysis, we recommend using P2P clients instead of the P2P proxies or adaptors as much as possible, and the planned authenticated interfaces [27] when

implemented in OpenDHT. This reduces the number of lookup and updates in the P2P network and, hence, is more scalable. The design and data format presented in this paper can be used by other P2P-SIP implementations to build an interoperable network of P2P-SIP nodes for contact management, key storage, NAT and firewall traversal, presence and offline message storage.

For P2P-over-SIP, we analyze various design alternatives, propose a P2P-SIP architecture using Chord as the underlying DHT, and describe various user location and registration steps in detail. We also present an overview of various advanced services such as offline messaging, conferencing, NAT and firewall traversal and security issues.

We have implemented P2P-SIP node in both P2P-over-SIP and SIP-using-P2P architectures for multimedia communication using our SIP C++ library. The SIP-using-P2P architecture is also implemented in the Columbia SIP user agent, sipc.

We notice that the classical client-server architecture of SIP and the P2P-SIP architecture are two extremes. For example, in the former case, the per-domain SIP is used to locate the user in the domain, and DNS is used to locate the per-domain server. In the latter case, P2P overlay is used to locate the node holding the user location. There can be an intermediate architecture that can use DNS to locate the server but the servers can dynamically join and leave the system using dynamic DNS. This gives rise to the service provider model where the provider sells the SIP service by becoming part of another provider's SIP server pool. DotSlash [92] explores this option in the context of web "hot spots" and uses service location protocol (SLP) to locate the backup servers. Such approaches need explicit synchronization of registration records among the participating servers similar to join and leave maintenance in P2P.

More work is needed in advanced services such as large scale application level multicast conferencing using P2P, distributed reputation system for peers, and PSTN interworking related issues such as authentication and accounting. There should be a reasonable incentive to become a super-node to provide services to other peers.

There are a few open issues: how to turn a node behind firewall or NAT into a super-node in the DHT. This reduces the load on public super-nodes, since most of the residential and corporate users typically will be behind some firewall and NAT. Alternatively, the private nodes in a domain can form a secondary P2P overlay connected to the public DHT via a few external

connections to reduce the port utilization on the NAT device.

Some of the P2P open questions described in [153] are relevant to P2P-SIP architecture also. Some kind of hybrid system may be implemented that takes the advantages of many different structured P2P algorithms to further reduce the latency and maintenance cost. For example, there has been proposal on one hop lookups for P2P [152] assuming large storage space. Applying this in P2P-SIP is transparent to our architecture.

Finally, we conclude on a note that unless the SIP servers (proxies, registrars) are widely deployed, we will need P2P based IP telephony tools so that everyone can use the system. Such P2P-SIP architecture can be extended to other protocols such as H.323.

Part III

Enterprise IP Telephony

This part describes the components in our multimedia collaboration architecture for enterprise IP telephony and large scale conferencing. The goal is to build a multi-platform collaboration architecture using standard protocols that can be accessed from different devices and tools such as IP phone, regular telephone, email, instant message and web. We also describe our SIP-H.323 translation mechanism.

Chapter 7

Background: Conferencing, Streaming and Voice Dialogs

Multimedia collaboration consists of a number of components such as multimedia conferencing, real-time media streaming and interactive voice dialogs. Before we describe our multi-platform collaboration architecture, we provide background on these components in this chapter.

7.1 Multi-party Conferencing

Multi-party conferencing is an important telephony service, provided in the PSTN by conference bridges. Many PSTN carriers offer conference bridges which allow users to take part in a voice conference by dialing a telephone number and conference access code. We can further enhance conferencing for Internet telephony by adding video and collaboration. The conference can be identified by a destination address, and participants can join the conference by making a call to that address, thus requiring no modifications in end systems. There are currently two Internet telephony signaling protocols, IETF's SIP [3] and ITU-T's H.323 [37]. SIP identifies the destination via a SIP URI of the form sip:user@domain, while H.323 uses AliasAddress data structures, which can assume many forms, including URLs.

There are two different aspects of Internet based conferencing, signaling and media. Either SIP or H.323 can be used as a signaling protocol for taking part in a conference. Both SIP

and H.323 use the Real-time Transport Protocol (RTP [1, 2]) for carrying real-time media traffic, such as audio and video. H.323 defines a multi-point control unit (MCU) for handling multi-party conferences. An MCU consists of a multi-point controller (MC), which can also be part of a terminal, to handle signaling and control exchanges with every participant in the conference. An optional component, the multi-point processor (MP), handles mixing and filtering of different media streams. SIP does not define any conferencing entity as such, as these entities are easily implemented as SIP user agents. The core SIP specification supports a variety of conferencing models [50]. In the server-based models, RTP media streams are mixed or filtered by the server and distributed to the participants. There is a standard point-to-point signaling relationship between each participant and the conferencing server.

The conference is identified by the SIP URI, e.g., sip:discuss@server.com. The standard user location and routing mechanisms in SIP forward all calls to the appropriate conference server at server.com without requiring any extension to the protocol. The SIP message routing entities (SIP proxies) need not be aware that the request URI corresponds to a conference and not to an individual person.

The Session Description Protocol (SDP [46]) is used to indicate media capabilities and media transport addresses. The participant sends the information about his media capabilities and the transport address where he wishes to receive RTP packets. In the message body of the 200 success response, the server sends the transport address to which the participant should send his RTP packets. More advanced scenarios can be accomplished using the SIP REFER method. For example, an existing participant can invite another user to join the conference. These conferencing models can be found in [50].

SIP-based authentication can be used to prevent unauthorized participants from joining a conference. The server can support both pre-arranged conferences as well as ad-hoc conferences by assigning special meaning to the user field in the request URI. For example, participants who wish to join sip:ietf.arranged@office.com will need to set up the conference before hand, while those who wish to join sip:library-discuss.adhoc@office.com do not need to setup the conference in advance. In both the cases, the participants have to know the unique conference URI. The conference state is maintained as long as at least one participant is part of the conference.

Participants find out about the conference URL via external means, such as email or a web page.

7.1.1 Conferencing Models

SIP can support many different conferencing architectures. SIP supports various multi-party conferencing models [50], ranging from mixing in end systems to multicast conferences. When multicast is not available, centralized mixing, transcoding and filtering of media can be used to create multi-party conferences.

Conference models can be distinguished based on the topology of signaling and media relationships. Conferences with a central server are easier to handle for end systems and simplify keeping track of the conference participants. On the other hand, network-layer multicast is more scalable for large-scale media distribution and allows a "loose" model of conference membership [154], where each member has only an approximate view of the group roster.

Table 7.1 summarizes the different types of *media distribution models* in multimedia conferencing. The table compares the scaling properties, depending on the the number of active senders, M, and the total number of participants, N. Given that M is almost always one for typical audio conferences, most of these models scale similarly in terms of processing and bandwidth requirements. Note that the centralized model performs better with higher M if inputs are summed.

Centralized Conferencing

In the centralized model, a server receives media streams from all participants, mixes them if needed, and redistributes the appropriate media stream back to the participants (See Fig. 7.1). If the speaker's audio is received in the mixed stream by the speaker, he will hear echo of his own voice. Since senders would have difficulty subtracting out their own contribution due to expensive audio analysis, the server needs to create a customized stream for each of the currently active M senders and a common stream for all $N-M$ listeners, assuming that they can all support the same media format. The server needs to decode audio streams before mixing, as mixing is generally performed only on uncompressed audio. Decoding M and encoding $M+1$ streams limits the amount of active sources or conferences. The available outbound network bandwidth at the server

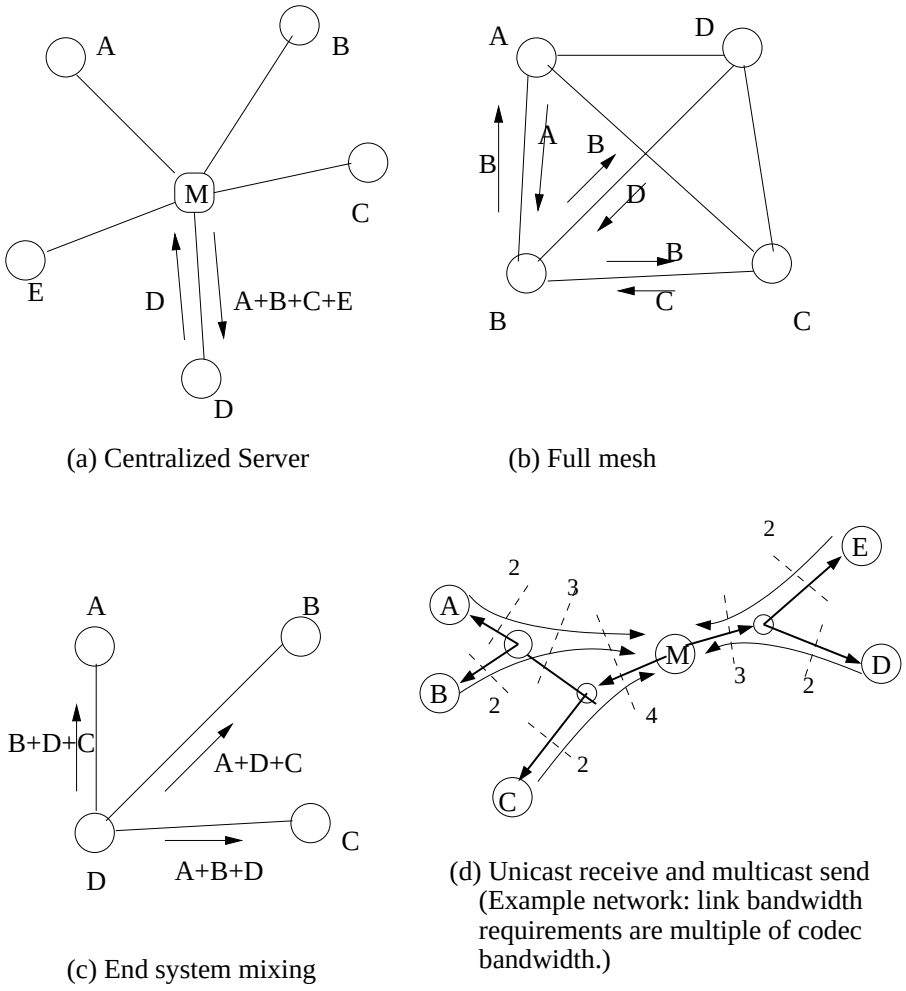
Properties	centralized	full mesh	multicast	unicast rx, multicast tx,	end system mixing
Topology	Star	full mesh	m-cast tree	star and m-cast	ad-hoc
Server processing	O(M+N)	n/a	n/a	O(M+N)	n/a
Endpoint processing	O(1)	O(M)	O(M)	O(1)	variable
Server bandwidth	O(M+N)	n/a	n/a	O(M) using m-cast	n/a
Endpoint bandwidth	O(1)	O(M)	O(1)	O(1)	variable
Scaling	medium	medium	large	large	medium
Heterogeneous endpoints	yes	yes	no	no	yes (partially)
Get back your media	no	no	no	yes	no

Table 7.1: Types of conferences; M is the number of active senders and N the total number of participants

limits the number of participants in the total conference.

The central server model has the advantage that clients do not need to be modified and do not have to perform media summing. In addition, it is relatively easy to support heterogeneous media clients, with the server performing the transcoding. For example, this allows a conference consisting of participants connected through high-bandwidth networks as well as wireless networks, each receiving the best possible quality. At the cost of increased inbound bandwidth, silence detection can be delegated from clients to the server. This is helpful if the phones of the participants do not support silence suppression.

Also, the server can enforce floor control policies and can control the distribution of video based on audio activity. Compared to a distributed model, a central server can readily provide a consistent view of the complete conference membership.



(a) Centralized Server

(b) Full mesh

(c) End system mixing

(d) Unicast receive and multicast send (Example network: link bandwidth requirements are multiple of codec bandwidth.)

Figure 7.1: Types of media distribution model

Full mesh

In a full mesh, each active participant sends a copy of its media stream to all participants via unicast, without a central server. End systems sum the incoming audio streams; since most of the time, only one speaker will be active, the CPU overhead is modest as long as silence suppression is implemented everywhere, but it fails if the access bandwidth of some participants is just large enough for a single stream. For video, full mesh does not scale unless, for example, only currently active speakers send video. In a full mesh, each pair of participants must share a common codec.

Multicast

Network-layer any-source multicast is ideally suited for large-scale conferences. A multicast address is allocated for each media stream, and every participant sends to that address. As in the full mesh, participants receive packets on the same address from all other participants, and need to sum or select streams. While the incoming bandwidth is the same as in a full mesh, each system only needs to generate one copy of the media stream.

Unfortunately, native multicast is not widely available outside network testbeds such as Internet2. Also, all participants must share a common set of codecs.

Unicast receive and multicast send

This scheme combines some of the benefits of the server and multicast models. Participants send their media streams using unicast to the conferencing server, which sums them and sends them out on a pre-established multicast address. Thus, unlike pure multicast, end systems do not have to filter or mix media streams. Every participant receives the mixed stream, which includes his own stream. Unless a sender maintains a buffer of the data sent and there is a means of aligning time scales, it will have difficulty removing its own audio content from the mixed stream, because of expensive audio correlation analysis. If the sender does not remove his own audio, he will hear echo. The gain in bandwidth efficiency is largest if the number of simultaneous senders is small compared to the total group size. This approach lends itself well to single-source multicast [155, 156].

Endpoint mixing

Instead of in a server, mixing can take place in one of the participating end systems. For example, if A and B are in a call, A can also invite C. A sends the sum of the media from A and B to C, and the sum of A and C to B. B and C do not need to be aware of the service performed by A, but can in turn mix other participants.

Cascading mixers increases the delay on some of the media paths. Another problem is that the conference dissolves when the participant who is acting as a mixer leaves the conference. This model is likely to be suitable only for small conferences of three or four parties.

Replication

Besides these, one can imagine a replication model, where the server sends a copy of each incoming media stream to all the participants using unicast. The mixing is done at each end system. This might be useful for media path authentication as every end system exchanges media packets only with the server's IP address. The CPU overhead is modest as long as silence suppression is implemented. The server however is less loaded than in the case of the centralized conference since it is now freed from the task of mixing audio streams. This is the model used in the case of video and text based conferences, since there is inherently no mixing required.

Media vs. Signaling

Media and signaling can use different models in the same conference. For example, one could combine centralized signaling with multicast media distribution, where the server maintains a one-to-one signaling relationship with each of the participants. Unfortunately, this requires cooperation from the end system. The server can indicate a multicast address in its SIP success response, causing the end system to send media streams via multicast, but the end system will still expect to receive media via unicast. More sophisticated session description formats may address this issue.

Also, different media streams can use different models. For example, audio could be mixed by a central server and redistributed, while video can be sent point-to-point between every pair of participants as in full mesh.

Thus, as long as multicast is not widely available, server-based conferences will continue to be the only viable model for mid-size conferences of tens to hundreds of participants.

7.1.2 Requirements for Centralized Conferencing

The main functions of a conference server is the mixing and redistribution of media streams. Typically, Internet audio streams are added or mixed, while video streams and other media are simply replicated. However, a video mixer can also create a new composite video image [157]. For audio, the server needs to ensure that a participant does not receive a copy of his own media in the mixed stream. RTP [158] allows a sender to indicate which sources have been combined in

a single media packet. When summing, the server should absorb the jitter in packet arrival times while introducing minimum delay.

For replication, the server should not need to be aware of the media formats. The RTP SSRC indication [1, 2] ensures that the receiver can distinguish different sources addressed to the same network destination.

For either summing or replication, it is desirable if each participant can use different media types and packetization intervals, to accommodate heterogeneity of end systems and access bandwidths. Implementations need to scale to large number of conferences as well as large numbers of participants per conference.

A media mixing module with a SIP interface can act as a conferencing server component in the distributed application server component architecture. Advanced system can bundle this functionality with other services, such as interactive voice response (IVR) and a web-based user interface.

7.2 VoiceXML: Interactive Voice Response

VoiceXML [35] is an XML-based language developed by the W3C to facilitate interactive voice response (IVR) that feature synthesized speech, digitized audio, recognition of spoken and DTMF key input and recording of audio for telephony applications. It converts the traditionally proprietary and closed IVR systems into an open programmable architecture.

A VoiceXML interpreter, also known as VoiceXML browser, can fetch VoiceXML pages from a web server, allow user input via spoken audio or touch-tone keys, and submit filled forms to the server-side scripts to generate another VoiceXML page for subsequent dialogues. The back-end programmable web CGI scripts can perform the application logic, such as voice-mail access or email access by phone.

The following example VoiceXML page prompts the caller with spoken audio: “Enter the ZIP code ...”. When the user presses a sequence of digits, say 10027#, the variable zipcode gets the value “10027” that gets passed to the URL http://myserver.com/weather.cgi?zipcode=10027. It is up to the script weather.cgi to process the input and generate further VoiceXML pages. If there is some error or user doesn’t press anything, then the prompt is repeated.

```
<?xml version="1.0"?>
<vxml version="1.0">
<form>
 <field name="zipcode">
  <prompt>Enter the ZIP code of the location for which you
   want weather information.</prompt>
 </field>
 <catch event="noinput error help">
 Enter the ZIP code again followed by the pound key.
 </catch>
 <block>
  <submit next="http://myserver.com/weather.cgi" namelist="zipcode"/>
 </block>
</form>
</vxml>
```

User input, either DTMF or spoken audio, can be specified using a set of rules called as grammar. A simple DTMF grammar can be used to receive only the DTMF input. A typical explicit dtmf tag in the VoiceXML page looks like:

```
 <dtmf type="application/x-dtmf">
   1 | 2 | 3 | 4 | *
 </dtmf>
```

The MIME type for this grammar is “application/x-dtmf”. Input is either a fixed length string or terminated by a “#”. A default implicit timeout of five seconds is implemented so that the input is automatically accepted if the user does not press the terminating “#” key within five seconds. If no grammar is specified, then the interpreter will accept any input. Special key sequence such as “**#” may be defined to signal the help event.

A VoiceXML browser needs a call control engine to handle or initiate telephony events such as incoming calls or call transfer. The browser fetches the VoiceXML pages or pre-recorded media files from a web server and presents an interactive dialog to the telephone user. Fig. 7.2 shows an example scenario where the browser, with SIP-based call control engine, is accessed from SIP phones as well as a regular telephone. The VoiceXML pages can either be statically

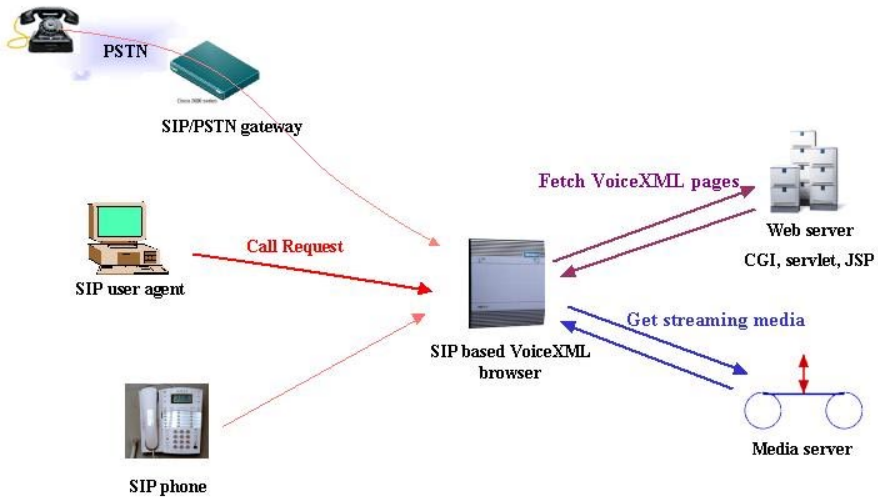


Figure 7.2: Example sipvxml scenario

stored on the web server or dynamically generated based on some server side programming logic like HTTP-CGI (Common Gateway Interface), Java servlet or Java server pages. The media files can either be stored on the web server or can be streamed in real-time from a media server, such as our rtspd, directly to the SIP caller using RTP [1, 2].

7.3 RTSP: Media Streaming

The Real-Time Streaming Protocol (RTSP) [159] allows to control multimedia streams delivered on the Internet. It is similar to HTTP [7] in syntax and semantics, and defines new methods such as SETUP, TEARDOWN, PLAY, RECORD and PAUSE, to start and terminate a stream, perform time-positioned playback, recording and pausing of a stream, respectively. DESCRIBE and ANNOUNCE requests are used to learn or specify the session description of a stream for playback and recording, respectively. They use the Session Description Protocol (SDP) [46] in the message body to describe the session. Unlike HTTP, which downloads the whole media file in the response, RTSP uses the Real-time Transport Protocol (RTP) [1, 2] to deliver multimedia streams in real-time. An example SETUP request is shown below:

```
SETUP rtsp://example.net/bob/movie.rm RTSP/1.0
CSeq: 102
Transport: RTP/AVP;unicast;client_port=8000-8001;mode="PLAY"
```

RTSP clients such as Apple's QuickTime [160] and RealPlayer [161] are well known among Internet users for playing stored audio or video content. However, RTSP can also be used for playing live content. The SDP contains a multicast address to play a live multicast radio, or to record a multicast program. In that case, the RTP packets are streamed to or received on that multicast IP address.

Chapter 8

Related Work: Internet Telephony and Multimedia Collaboration

Internet telephony has been an active area of research and development in the past decade, with a number of companies such as Vonage, Skype, AT&T, Net2Phone, DialPad and MediaRing providing PC-to-PC and PC-to-phone calls. Their objective is mainly to provide low-cost call service to PSTN from the public Internet, whereas our architecture, which is called Columbia InterNet Extensible Multimedia Architecture (CINEMA), is well-suited for Internet telephony infrastructure within an organization with many more additional services such as collaboration and platform independence. CINEMA emphasizes use of existing standards for interoperability and an open and distributed component architecture instead of closed server box implementing all the features.

A number of efforts started about the same time as ours in enterprise IP telephony systems, including but not limited to Cisco Call Manager [162] and Nortel Multimedia Communication Server [163]. All the systems have evolved over time to contain more or less equivalent set of features. However, there are certain design issues and tradeoff such as those affecting the reliability and scalability of the distributed architecture that differ. Our goal is to provide a fully distributed architecture that allows interoperability among different components from different vendors. This means that any standard complaint tool can be used in place of existing tools in the architecture. Secondly, proprietary or single vendor protocols such as Skype [21] and Cisco's

Skinny [164] are not considered for building enterprise IP telephony systems in this thesis.

CINEMA-based Internet telephony can be used to minimize telephone infrastructure and service costs within an organization. We can configure CINEMA to carry calls between campuses or branch offices over IP with virtually no added cost.

Several multimedia conferencing products use SIP or H.323 for signaling, e.g., MeetingPoint from CUseeMe Networks [165], Sametime from Lotus [166], and GnomeMeeting [167] from the Linux community. Our system can provide services beyond standard video conferencing and can actually incorporate these tools as long as they are standards-compliant.

Computer-supported collaborative work (CSCW) has been studied even before the web [168, 169, 170, 171]. ACM's special interest group on supporting group work, SIGGROUP, explores topics related to computer-based systems that affects team or group in workplace settings. However, the focus remained mostly on web-based document sharing and concurrent editing in systems such as BSCW [172], Lotus Domino [173], Hyperwave [174] or Livelink [175]. Many researchers have explored specific types of collaboration such as collaborative software development [176] or electronic class rooms [177].

Multimedia conferencing using audio, video, and data communication using instant messaging and email, have independently evolved and become popular over the years [178, 179, 180, 181, 182]. Using audio and video for collaborative work is not new [183, 184]. There are a number of audio/video collaboration systems such as MBone tools [185, 186], MeetingPlace [187] and GnomeMeeting [167]. The ITU-T's H.323 protocol suite [99, 188] provides video conferencing systems along with T.120 for data conferencing and T.128 for application sharing [189].

Most of the technologies used in our architecture, such as shared web-browsing [190], conference floor control [191, 192, 193, 194], application sharing [195, 196, 197, 198] and web-based collaboration [199] have been investigated extensively elsewhere. A number of web portals such as Yahoo! and MSN provide online calendaring, and sharing of information to some extent. However, the concept of groups has only recently started gaining attention. Our work was the first demonstration of a SIP-based comprehensive and extensible collaboration system combining synchronous and asynchronous collaboration mechanisms. Although our approach comes from a multimedia communication background, it integrates the conferencing and collaborative

computing approaches.

Next, we describe related work specific to components of CINEMA, such as SIP-H.323 translation, unified messaging and centralized conferencing.

8.1 Interworking Between SIP and H.323

The problem of interworking between SIP and H.323 had started to attract attention when we first proposed and demonstrated a translation scheme. Agboh [200] and Kausar and Crowcroft [201] had addressed the problem of interworking, but had not solved the issues of registration and media capability translation. Moreover, the translation of call setup from multi-stage H.323 to single stage SIP was not available until our work. Since the newer versions of H.323 have proposed a number of enhancements including the single stage FastConnect call setup procedure, thus, further simplifying the translation. However, newer versions are supposed to interoperate with older versions, hence our translation scheme is still valid and useful.

A informal work group was formed later within the IETF to investigate the translation with the newer versions of H.323. The group developed the requirements for the translation [202]. A number of products are now available that perform SIP-H.323 translation such as from Vocal-Tec, NexTone and SIPquest.

More recently, vendors such as Cisco and Microsoft have moved to SIP. Thus, with the gradual disappearance of H.323 systems, especially after Microsoft discontinued the H.323-based NetMeeting software, the research interest in SIP-H.323 translation is fading. There are lots of deployed H.323 conferencing systems such as from Polycom and Radvision, and many carriers have made huge investments in H.323-based infrastructure. Therefore, SIP-H.323 translation is still needed.

8.2 Unified Messaging using SIP and RTSP

There is a fair amount of early messaging work, in particular, the Etherphone work done at Xerox PARC [203, 204, 205], but none of it addressed the integration of Internet telephony with voice messaging. Profiles have been defined for Internet messaging to support voice. In particular,

VPIM [206] supports the interchange of voice messages between voice mail systems, unified messaging systems, email servers and desktop client applications. The basic architecture is to carry the voice attachments in the electronic mail. None of these addressed the integration of Internet telephony with the voice messaging system. Moreover, carrying the voice bits across low-bandwidth links while forwarding the messages is not desirable. It also requires special-purpose client applications which can understand the profiles.

We proposed and implemented the first voice mail and answering machine system for SIP-based Internet telephony that combined the power of media streaming and worked without modifying the existing SIP servers or clients. Subsequently, various schemes have been proposed to forward a call to a voice mail server in SIP-based Internet telephony systems. The Common Gateway Interface for SIP [24] or the Call Processing Language [49, 48] can be used to configure the SIP server to use an external voice mail service. Campbell and Sparks [207] suggest the use of SIP Request-URI to carry service control information related to voice mail.

Voicemail and answering machine are now common features in SIP-based Internet telephony systems.

8.3 Centralized Conferencing using SIP

Before we started working on our SIP conference server, most of the then existing conference servers in the market were based on H.323. These included MeetingPoint from CUseeMe Networks, Sametime from Lotus and Microsoft Exchange 2000 Conferencing Server. These supported T.120 for application sharing and whiteboards. MeetingPoint has mechanisms to link servers together so that conferences can be shared and load-balancing can be done. VideoTalks [208] by AT&T Labs is a comprehensive multimedia conferencing system intended to provide a variety of Internet services such as video conferencing and low cost video-on-demand. It is not based on SIP.

A number of software (e.g., RAT and NeVoT) support multicast “light-weight” conferencing, without explicit signaling support [154]. Etherphone [209] is probably one of the earliest systems supporting multimedia conferencing.

Our sipconf was one of the first centralized conference server implementation based on

SIP. We further extended it to form the core of our synchronous collaboration platform by adding video, screen sharing, instant messaging and recording. SIP-based conferencing has now become a common feature in any Internet telephony infrastructure. The IETF's XCON working group is standardizing centralized conference control protocol for operations such as floor control and configuration access[210, 211].

8.4 Integrating VoiceXML with SIP Services

The Voice Browser working group of World Wide Web Consortium (W3C) has developed the VoiceXML [35] specification. VoiceXML applications for interactive voice response are developed by many commercial organizations.

When we developed our SIP-based VoiceXML browser, there were some existing VoiceXML implementations. For example, Plum Voice Portal Technology [212] could present existing websites or intranet applications to a phone user. It could also deliver follow-up information via email or fax. Open VXI [213] was an open source VoiceXML interpreter. IBM's WebSphere provided HTML-to-VoiceXML transcoding that could be converted to speech by a VoiceXML browser. Talking E-Mail [214] allowed users to access emails from various interfaces including voice, i-mode, Web and WAP. None of these applications used SIP for call control. Ours was the first known implementation of a SIP-based VoiceXML browser.

SIP URI for indicating VoiceXML service is specified in [215]. Tellme studio [216] provided the first SIP based VoiceXML development platform that allows users to test custom VoiceXML pages or scripts. We used this for initial testing of our email-by-phone system.

In Section 9.6.2, we describe the design of a SIP-based VoiceXML browser, sipvxml, and its application in our IP telephony test bed. Ours was the first implementation to associate the VoiceXML transfer tag with the SIP REFER message for a conferencing application. Moreover, it can be used as a third-party voice application server like Tellme or an integrated component in CINEMA [89, 217, 88] for campus or enterprise VoIP services. We describe multi-platform collaboration in CINEMA, the first complete IP PBX and collaboration system, in the next chapter.

Chapter 9

Multi-platform Collaboration in CINEMA

9.1 Introduction

In many organizations, e-mail and tele-conferencing are the only means of collaboration. More recently, people have started to use instant messaging (IM) for short interactive communication. Even though these communication means are not designed for collaborative work, the limited set of available options causes them to put all their data such as meeting notes, documents, conference schedules and reminders into the email system.

We need a collaborative environment that seamlessly integrates with the existing communication means of email and phone as well as newer methods like IP telephony and instant messaging. Consider an IP telephony conference with some participants on phone, and some others using desktop audio/video clients. Late-arriving participants can browse through the past meeting proceedings, and non-participating group members can be automatically notified of meeting minutes and other important document locations via email.

Our system is different from earlier conferencing applications in that it integrates the two modes of collaboration: synchronous that requires active real-time participation and asynchronous that are not real-time. We support multimedia conferencing, instant messaging, shared web-browsing, file-sharing, discussion forum, voice and video mails. As an example, same group

of people can be addressed by video conference, instant message and email, with appropriate archival of interactions. Secondly, it provides device-transparency by allowing access and interaction even if participants temporarily have only a phone or email. Although it is not new, we also provide hybrid interaction such that one can use phone for audio and PC for IM and document sharing in the same conference.

Our architecture provides building block tools for any type of multimedia collaboration, instead of focusing on specific types such as collaborative software development. We want to support three kinds of typical interactions: long-lived distributed groups that alternate between synchronous and asynchronous interactions, such as design teams, college classes, committees and work teams, asymmetric events such as lecture and lecture series, where interaction is mostly limited to asking questions to the speaker, and short-lived spontaneous interaction among groups of people.

Our collaboration system is based on standard protocols and tools such as SIP [3] and Real-Time Streaming Protocol (RTSP [32]) for signaling, Real-time Transport Protocol (RTP [1, 2]) for media transport, VoiceXML [35] for voice-based interaction, Call Processing Language (CPL [49]) for network-based service creation, Language for End System Services (LESS [218]) for endpoint-based service creation and a web interface for asynchronous collaboration.

In this chapter, we describe the architecture and implementation of our comprehensive multi-platform collaboration framework. We describe the requirements for comprehensive multimedia communication and collaboration systems in Section 9.2. Section 9.3 provides an overview of the architecture and the user interface. Section 9.4 describes the synchronous collaboration architecture whereas Section 9.5 details the asynchronous collaboration. Additional services such as presence, interactive voice response and integration of phone, IM and email are detailed in Section 9.6. Finally, we present the conclusions in Section 9.7.

9.2 Requirements

The basic requirements for a comprehensive collaboration system consist of a personalized view of the system, real-time or interactive multimedia collaboration (called *synchronous*) and loosely tuned sharing of information (called *asynchronous*). A web-based user interface provides a

portable and personalized way to access the system.

Personalized view

People like to have personalized views of the system such as per-user calendar with appointments and conferences. However, the system should also allow sharing the view with other users or in a group after filtering. For example, Alice may not want to see the events posted by Bob. She can schedule a conference or discussion forum for her project group, and invite members to join.

Synchronous collaboration

The system should allow multi-party audio, video and text conferencing. It may support shared white-board facilities, shared applications and screen sharing. It should allow restricted conferences with only authorized members as well as public unrestricted ones. The conferences may be pre-scheduled or created on the fly. It may support both dial-in and dial-out conferences, floor control, and telephone-based authentication. It should be possible to record, and later, playback the proceedings of a conference. It may allow time-positioned playback (e.g., play after first 30 min of recorded data), sharing files with other participants (e.g., agenda, slides or meeting minutes), playing a media file in the conference, merging two conferences into one, or splitting one into two conferences.

Asynchronous collaboration

The most basic form of sharing information is via various forms of messaging, e.g., email, voice and video mails. The system may allow recording and filtering of IM communications. It may allow storing messages in various folders, accessing remote email clients or servers for multimedia messages and listening to the messages via a telephone. The web-based message board may be accessed via email or telephone. It should be possible to share files and other information within or across groups.

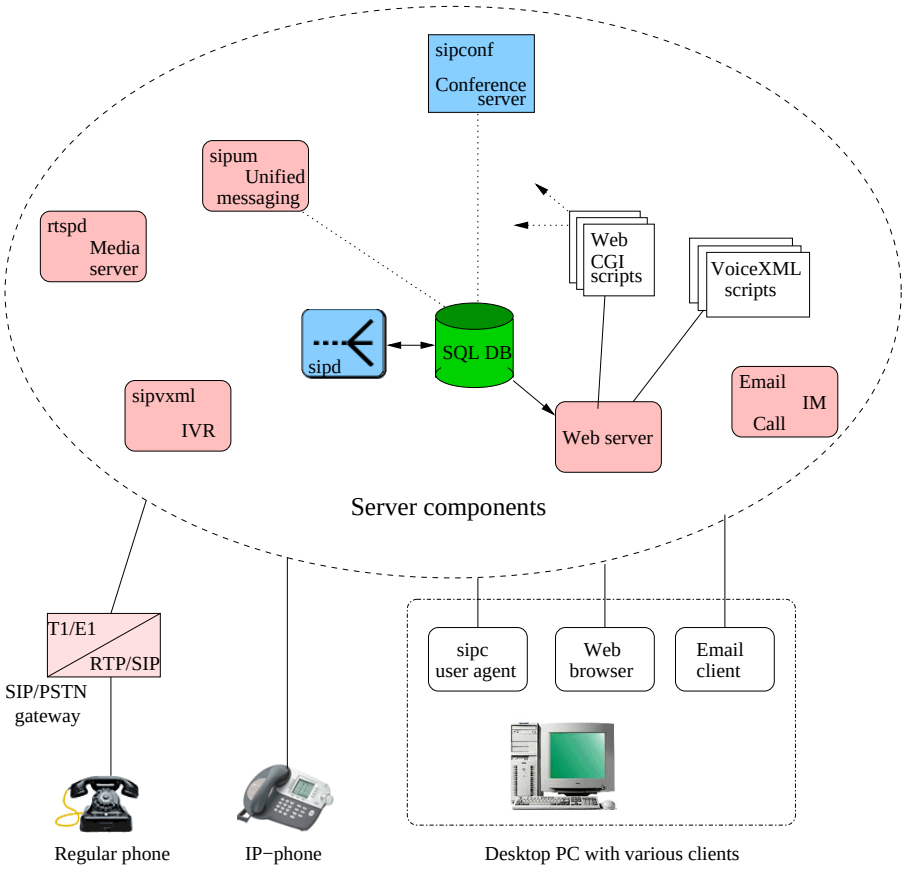


Figure 9.1: SIP-based collaborative work environment

9.3 Architecture Overview

Our CINEMA architecture consists of a set of distributed server components and user agents as shown in Fig. 9.1. The SIP registration and proxy server (sipd) is used for user location and forwarding of signaling messages. The multi-party conference server, sipconf [34], forms the core of the synchronous collaboration infrastructure. The media server, rtspd, allows streaming of multimedia content for playback and recording. The unified messaging server, sipum, provides centralized answering machine, and multimedia mail service [33]. A web-based interface provides asynchronous collaboration support. User agents such as regular PSTN phones via a SIP-PSTN gateway, IP-phones, or desktop based SIP user agents like sipc are used for synchronous collaboration. Interactive voice dialogue via the VoiceXML browser, sipvxml [36],

allows easy access to a telephone user. The SIP server and the SQL database [29] form the core of the infrastructure for basic call flow.

9.3.1 Web Interface

The CINEMA web-based user interface manages user accounts, voicemails and conferences. The web pages are generated using the HTTP CGI [219] scripts that access the SQL database for configuration and profiles. The web pages provide intuitive user interface components and context-sensitive help. There are multiple levels of details in different user expertise levels. For example, a beginner-level user accesses only basic features to get started whereas an advanced-level user can configure and manage detailed information. The interface allows configurable layout of the web pages so that a particular installation of the system can be adapted for each service provider.

There are two types of users: *regular users* and *administrators*. An administrator has additional privileges compared to a regular user. The first user created during installation becomes the administrator. An administrator can add additional users as administrator or regular user, change the user type, or access profiles of other users. New users can also "sign-up" for the service from the web. The web interface functionality can be further classified as follows:

Call-routing profile: The user can manage profile information, current contact locations, alternative names for identification, on-line status of "buddies", access control as to who can call, and programmable call handling, e.g., based on time-of-day or caller-id.

Unified messaging: This includes the integrated interface for voice and video mails, emails and discussion forum on various topics.

Event calendar and conferencing: This provides the personalized calendar for each user. It allows managing various appointments, events and conferences.

Address book and access group management: The user can maintain an address book of his friends' profile such as name, email address, department, birthday and postal address. The user can organize his address book entries into groups with different access privileges. For

example, people in “my family” group can access his personal calendar whereas others cannot.

Administration and accounting: An administrator can manage several server configuration parameters, user privileges, as well as the visual layout of the web pages. He can also assign various tariff rates for the phone calls and configure the gateway locations for the telephone destinations.

The web interface is just a front-end to the user profile and system configuration information stored in the SQL database.

9.3.2 Personal Calendar and Address Book

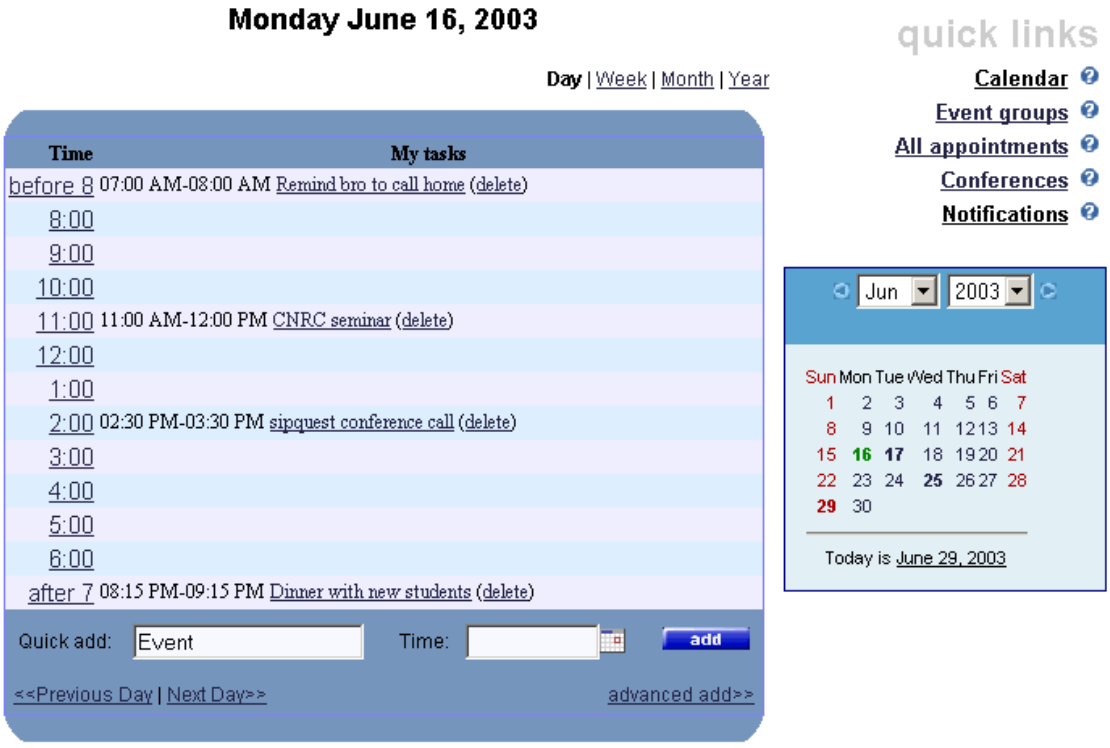


Figure 9.2: Personal calendar

When the user logs in from the web, it shows the most recent appointments and voicemails. A personal calendar shows the various appointments or conferences scheduled for the user

or his group (Fig. 9.2). The user can see the day, week, month or year view for different levels of information.

The per-user address book allows organizing the contacts into local or global access groups. A *local* group is visible only to the owner, e.g., “my friends”, whereas a *global* group is visible to everyone, e.g., “network research group”. An address book entry can belong to zero or more access groups. An event, such as an appointment or a class schedule, can have a group-name with given group-privileges. The read or write access privilege for an event can be owner, group or everyone, similar to Unix file permissions. The read access privilege specifies who can view the description and details of the event. The write access privilege tells who can modify the event attributes. A personal appointment typically has owner privileges for read and write, whereas a seminar series has group read access and owner write access privileges.

9.3.3 Events and Event-groups

An event is an individual event or appointment. An eventgroup is a collection of related events, e.g., an university course for which individual classes, or events, happen weekly. Each event can belong to an eventgroup. An eventgroup can have zero or more events. An eventgroup can optionally have a repeat indicator, e.g., every month, every year. The repeat indicator is useful if one does not want to itemize individual events, e.g., yearly birthday reminders.

An event-group may be associated with an optional conference name, e.g., on-line lecture series. While an eventgroup defines a group of events used in calendar, a conference is strictly a synchronous collaboration with additional attributes like supported media-types, dial-in numbers, recording formats, default audio sampling rate, public or private conference type and public or private participant list. Various SQL tables for storing the information and their relationships are explained in the CINEMA technical reports [88, 220].

9.4 Synchronous Collaboration

A multi-party multimedia conference is the simplest form of synchronous collaboration. In the absence of multicast, centralized conference servers provide an attractive solution for small to

medium scale conferences. Moreover, a centralized control integrates easily with other collaboration requirements such as floor control. For example, the organizer can control who gets to speak at any instant if there are multiple speakers, and enforce the policy at the server.

A conferencing server consists of a signaling and a media module. The signaling module receives SIP requests to join or leave the conference, while the media module receives and sends RTP media streams from and to the participants. The participants dial the conference URL, e.g., *sip:staff-meet@cs.columbia.edu,* to join the dial-in conference. The conferences can be pre-scheduled from the web interface, or created on the fly, e.g., by dialing *sip:letsmeet.adhoc@conference-server*.

The conference can have heterogeneous endpoints used by various participants with different media capabilities. For example, some user agents connected to low bandwidth links can have only low bit-rate audio codec whereas others on high bandwidth links can support high-quality codecs along with video. When a user agent joins a conference, it indicates its capabilities to the server. The server selects a subset of capabilities based on the intersection of user agent capabilities and the server capabilities on per-participant basis.

Our conference server, sipconf, consists of a number of features such as audio mixing, video forwarding, instant messaging, shared web browsing, screen sharing and conference control.

9.4.1 Audio Mixing

When the participants join the conference, the server mixes and redistributes the audio such that a participant hears everyone else except herself from the server. The server decodes the incoming audio from the participant, and puts it in a per-participant queue as shown in Fig. 9.3. On periodic interrupt, the participant audio is mixed, and redistributed back to the participant after encoding. The server acts as an RTP mixer [1, 2] for the audio. However, since each receiver can potentially have different audio stream mixed from audio streams of all the participants except herself, and each call leg in the conference forms an independent RTP session between the server and the participant, the conference represents multiple logical RTP mixers.

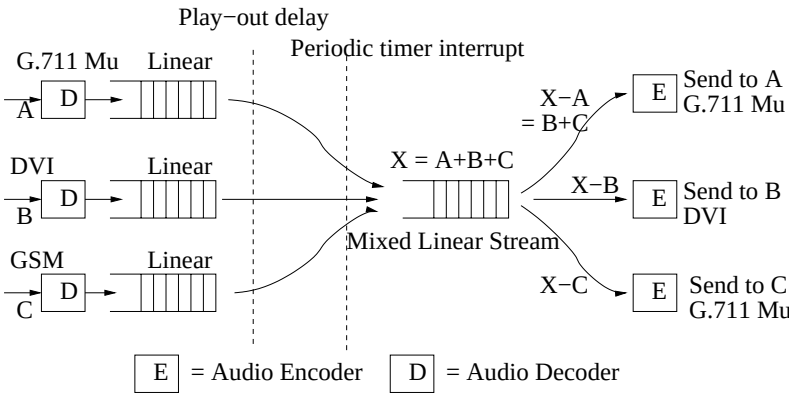


Figure 9.3: Audio mixing

Decode-mix-encode

In Fig. 9.3, participant A supports the G.711 codec, B DVI codec and C both GSM and G.711. Participants list the codecs they support in their SDP component of the SIP INVITE requests. The server selects an intersection of the algorithms supported by the participant as well as by the server. This selection is returned in the success response to the participant. These algorithms are listed in order of preference in the SDP of the INVITE or its response.

The mixing algorithm follows a *decode-mix-encode* sequence. When an audio packet arrives at the mixing module, it is decoded into 16-bit linear samples and appended to the per-participant audio buffer queue. Each buffer is labeled with the corresponding RTP timestamp. The jitter in packet arrivals is absorbed by a play-out delay algorithm. Every outbound packetization interval, a timer triggers a routine that mixes a range of the samples from one of more input buffers from each active participant into a combined packet by simple addition of the sample values. The timer is adjusted to account for processing delay in each interval.

To allow input and output packets to have different packetization intervals, the mixer routine can grab samples from one or more input buffers. Using a linked-list of buffers saves memory compared to a circular buffer of maximum size, and makes it easier to detect when a particular source is silent. For each of the participants, the linear sample values from the per-participant queue (e.g., A) is subtracted from the mixed data (X) and the resulting data ($X - A$) is encoded using the preferred audio compression algorithm of A. The encoded data is packetized

and sent to that participant. If there are M participants, then both mixing and redistribution will take M additions and M subtractions. Note that the receive and transmit audio algorithms need not be same for each participant.

While the *decode-mix-encode* sequence is the most straightforward approach to implementing an audio mixer, there are alternative approaches. For instance, one can build an addition or subtraction table for G.711 samples, so that conversion to linear is not required to do mixing. This only works for G.711, not for codecs with cross-sample dependencies such as G.723.1 or GSM.

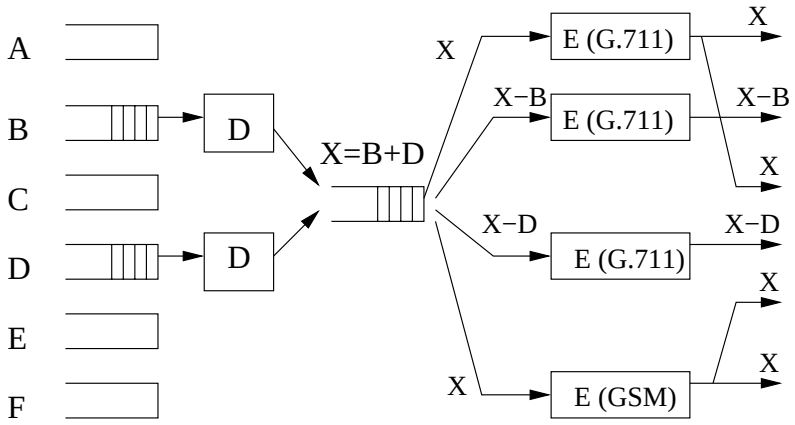
Also, instead of subtraction, one could create $M + 1$ different streams directly, one for each talker and one for the listeners. However, that requires M^2 additions.

Optimizing the Mixing Logic

It is possible to optimize the mixing logic, although we have not implemented any optimizations. One such scheme (Fig. 9.4) combines the encoding step for the output streams that have same mixed audio data and uses the same encoding algorithm. For all the participants who did not speak in the last timer interval and who have a common subset of supported receive audio algorithm, we can call the encoder only once. However, if a stream stops being active, it will receive the general listener packet stream rather than its own version, so that the predictor will be wrong. It is not clear how much this would matter in practice.

Packetization Interval

Although RTP implementations are supposed to handle a wide range of packetization intervals, we found 20 ms to be the only one that worked across a range of media clients such as RAT [52] or Microsoft NetMeeting. End systems permitting, it may be useful to dynamically change the packetization interval for outgoing packets, as smaller packetization intervals decrease delay, but increase network bandwidth and computational effort.



A−D support G.711; E and F support GSM.

Figure 9.4: Possible optimization in decode-mix-encode sequence

Inactivity Detection

The system should be able to detect if a particular participant becomes inactive, e.g., due to user agent failure. Failures can be detected by observing ICMP errors or sudden discontinuation of RTCP reports.

Automatic Gain Control

If the participants use different types of devices, it is possible that some users are heard louder whereas some others are hardly audible due to different speaker and microphone volumes. The server can do automatic gain control for both incoming and outgoing audio. However, this puts additional processing overhead on the server and reduces scalability. Alternatively, the server can indicate the volume level if it is too high or too low to the participant, who can then adjust her microphone and speaker volume, or it can selectively implement automatic gain control for participants who want it. For example, using VoiceXML a participant can press 6 to increase her microphone volume, or press 7 to reduce her speaker volume.

Playout Delay Algorithm

Playout delay algorithms help absorb the jitter in network packet arrival due to network congestion. Adaptive playout delay further allows an application to adapt to changes in the amount of jitter, thus giving minimum delay in the audio stream. Playout delay compensation takes place before mixing, stretching or shrinking silence periods between talkspurts to adjust the time between arrival and mixing [221, 222]. In the absence of silence periods, time stretching or companding can be used, albeit at much greater computational cost. We have used Algorithm 1 from [221], with $\alpha = 0.95$, for our implementation. The algorithm is basically a linear recursive filter. The adapted delay at any instant depends on the measured delay (using RTP timestamps) plus the previous adapted delay, with a weighting factor α. The playout delay depends on both the adapted delay and the variation in the adapted delay.

9.4.2 Video Forwarding

Unlike audio, mixing does not make sense for video. Every participant may want video from everyone else in the conference. The server implements transparent packet forwarding for video. A video packet from a participant is distributed to every other participant in the conference without modification. In this case, the server does not implement the RTP stack for video session. The lip synchronization between the audio and video sessions is done at the participant's user agent on receiving the two streams.

Alternatively, the server can send only one video stream of the *active* speaker to all the participants, or the chair can decide whose video stream needs to be distributed. Organizing the participants' video in a single stream (NxN tile) puts additional processing load on the server, degrades quality, and is undesirable.

If the user agent does not indicate video capability, i.e., no video port, then video is disabled for this call leg. The participant can dynamically change the capability. For instance, she can start with audio, and later, switch to audio and video sessions.

9.4.3 Instant Messaging

The instant message (IM) handling in the conference server is similar to video forwarding. When *alice@office.net* sends an IM to *bob@home.com*, the SIP server at *home.com* domain proxies it to the current location of Bob's phone. An IM sent to the conference URI *sip:staff-meet@servers.com* is intended for all the conference participants. If the conference is not active or there is no other participant, then the server indicates the error to the sender. If the sender is not already in the conference, then the server can either indicate an error to the sender, or still continue to distribute the IM to the participants. In a way, the server provides a group address to send IM to, similar to email-groups.

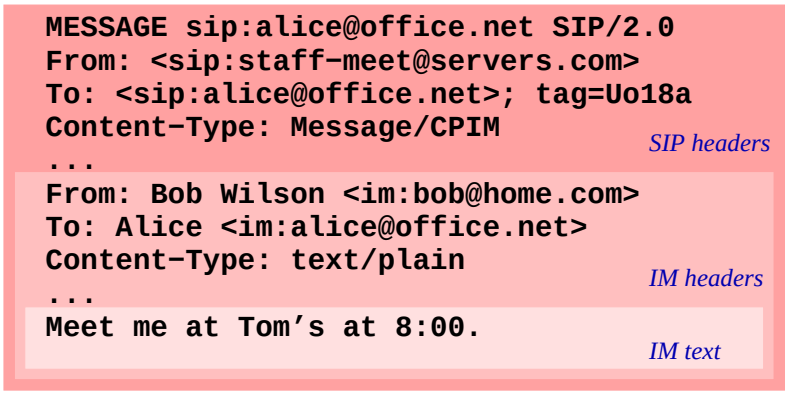


Figure 9.5: Example SIP MESSAGE for instant messaging

An example SIP MESSAGE sent by the server is shown in Fig. 9.5. It indicates that the SIP message is sent from the conference server to the participant, Alice, and the IM is originated by the user Bob. The server can also forward indications [223] that allows Alice's user agent to display status such as "Bob is typing a message".

The server should allow transitioning from an IM session to a full multimedia session, and vice-versa, when the participant changes her media capabilities accordingly.

9.4.4 Shared Web Browsing

The SIP MESSAGE method can be used not only for instant messaging, but also for some additional control. For example, sipc can capture the browser event on navigation and indicate

that HTTP URL to the remote party. The server forwards the message like any other IM, thus, readily supports shared web browsing among multiple participants. The message is similar to Fig. 9.5 except that the IM header Content-Type is `text/uri-list` and the IM text contains the HTTP URL. If the remote party understands this content, it can also invoke the browser pointing to the given HTTP URL.

9.4.5 Screen Sharing

We have added support for the open source Virtual Network Computing (VNC [224])-based screen sharing in sipconf. VNC is a client server protocol, where the server shares its screen to a viewer or client. To avoid authenticating the client, we initiate the session from the VNC server to the listening client. If a participant shares her screen, her user agent invokes the VNC server application whereas all the other participants invoke the VNC client application. The conference server merely forwards packets similar to video forwarding. The data packets containing the screen buffers are forwarded from the VNC server to all the VNC client applications whereas the control packets such as mouse and keyboard input are sent from the VNC client to the VNC server application. The VNC protocol can be tunneled through SSH for secure sessions.

9.4.6 Conference Control

In a hybrid conference using phone for audio and PC for IM, it should be possible to control the conference from either phone or IM client. Simple IM to the server can be used as control commands, e.g., if a participant sends IM text as “list”, the server returns the IM text containing list of all the active participants. Similarly, when a new participant joins or one leaves, all the existing participants are notified by the server via IM.

Conference floor control [210] means controlling who gets the exclusive access of the shared media channels or resources. For example, typically only one participant should speak in a conference. In case of multiple contenders, the conference chair can decide who gets to speak. There are many ways to do advanced floor control such as using Simple Object Access Protocol (SOAP) to run Remote Procedure Calls (RPC) on the server, web interface, and via touch-tone phones. We have implemented SOAP-based floor control in our server.

SIP and SOAP: Conference floor control consists of two parts: notifying the participants about who is holding the floor [139], and allowing the moderator and the participants to remotely control the floor. For example, a moderator can grant or deny a floor request and a participant can claim or release a floor. We use XML-based platform independent SOAP [225, 226]) for encapsulating and exchanging the floor-control commands instead of creating a new RPC (remote procedure call) protocol.

Web interface: The control messages can be sent from the web via CGI scripts or Java applets. The moderator can grant or reject floor to other participants from the web. For the web-based floor control, the web components communicate with the conference server and indicate the appropriate control message.

Interactive voice response: This allows a telephone user to control the conference via limited touch-tone keys. For example, “press 1 to ask for floor”. The DTMF (Dual-tone multiple frequency) digits are typically detected and translated to special RTP packets [227] at the telephony gateway.

9.4.7 Dial-in vs Dial-out Conferences

Although most of our earlier discussion focused on dial-in conferences, dial-out mode is equally important, for example, a participant invites another user in the conference, or the server sends out call invitations to the intended participants at a scheduled time. Usually some form of audio and text announcement indicates the purpose of the call to the user. To avoid the dialed-out call going to answering machine, the server may prompt the user to press certain digits to actually join the conference.

9.5 Asynchronous Collaboration

There are a number of related events during or after the conference that need to be shared with others even when the conference is not active. For example, the recorded conversation or meeting minutes may be needed in subsequent meetings, off-line discussion on the topics covered in the conference needs to be co-ordinated in the same way as the conference was controlled or the notes

may be edited remotely using WebDAV [228]. The primary objectives of these collaboration mechanisms are to avoid duplicating shared data and to provide some form of change control on shared data.

As mentioned earlier, every conference is associated with some eventgroup. An eventgroup can be associated with various forms of asynchronous collaboration mechanisms, such as file sharing and discussion forum. Conference participants can share meeting notes, agenda or other documents via the web.

9.5.1 File Sharing

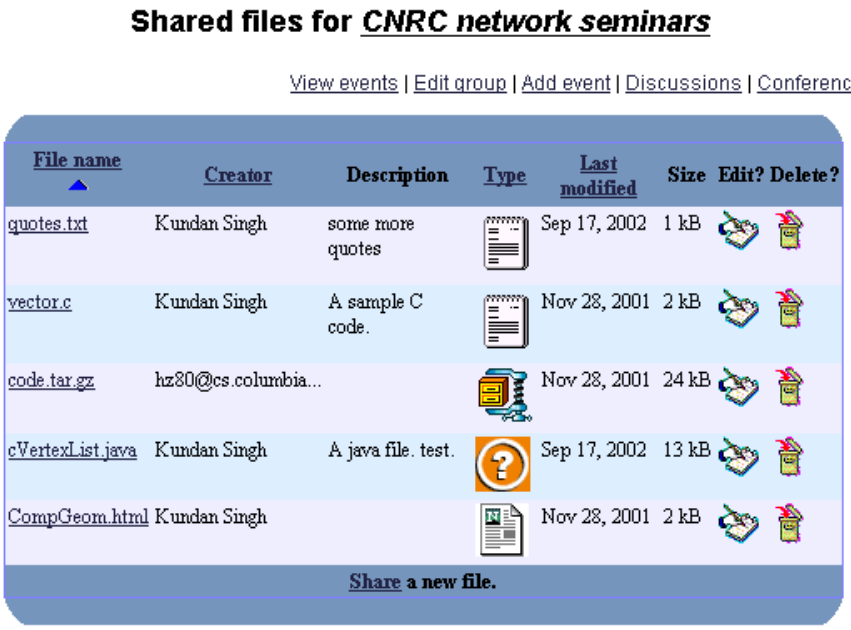


Figure 9.6: File sharing

The web interface allows uploading shared files as shown in Fig. 9.6. The shared file attributes consist of the creator's user identifier, name of the file, MIME-type [229] for display, a brief textual description, date of creation and last modification, and the access privileges for read, write and delete. The read access privilege can be for the group or public, whereas the write and delete access privilege can be for the group or owner. The group name of the file is inherited from the associated eventgroup. The users can register to get notified via email when the shared

file is modified or deleted.

9.5.2 Discussion Forum

Message boards and discussion forums facilitate asynchronous discussion on a particular topic. One advantage over email-based discussion is that it can systematically display the various discussion threads, postings and replies. The message information stored in the SQL table includes the message subject, content, sender identifier, date and time, associated eventgroup identifier, a unique message identifier and the message identifier of the parent message. If there is no parent message, then this message is the start of some thread. If there is a parent message, then this message is a reply to that parent message. The associated eventgroup specifies the read and write access attributes for the message board.

The users can also register to receive new posts or replies in their email. They can use email to post a message or reply to the discussion thread. Fig. 9.7 shows an example web interface. Integrating email with the system is discussed in detail in Section 9.6.3.

9.5.3 Conference Event Recording

CINEMA allows recording of the audio, video and IM communications in a conference. The audio recording at the conference can be done either when the media packets (RTP) are received from the participant or when the mixed stream is created as in Fig. 9.3 (p. 190). In the former case, the recording is done by dumping the raw RTP (and RTCP) packets along with packet size and time-stamp, in a file. This "rtpdump" format can later be played out using our media server, rtspd. The server does not need to understand any specific media file-formats, such as MPEG or "wav", but works as long as the playing client understands the codec used by the recording client. On the other hand, a mixed audio stream can be recorded in standard Sun "snd" or Microsoft "wav" file format. Only rtpdump recording format is needed for video, since the server does not generate any mixed video stream. The system allows recording in a local file or to remote media server using an RTSP URL. A per-conference quota on maximum recorded file size can be imposed. The recorded file path or URI information is stored in the SQL table for each conference instance (or event), whereas recording format preference is indicated for each conference (or event-group).

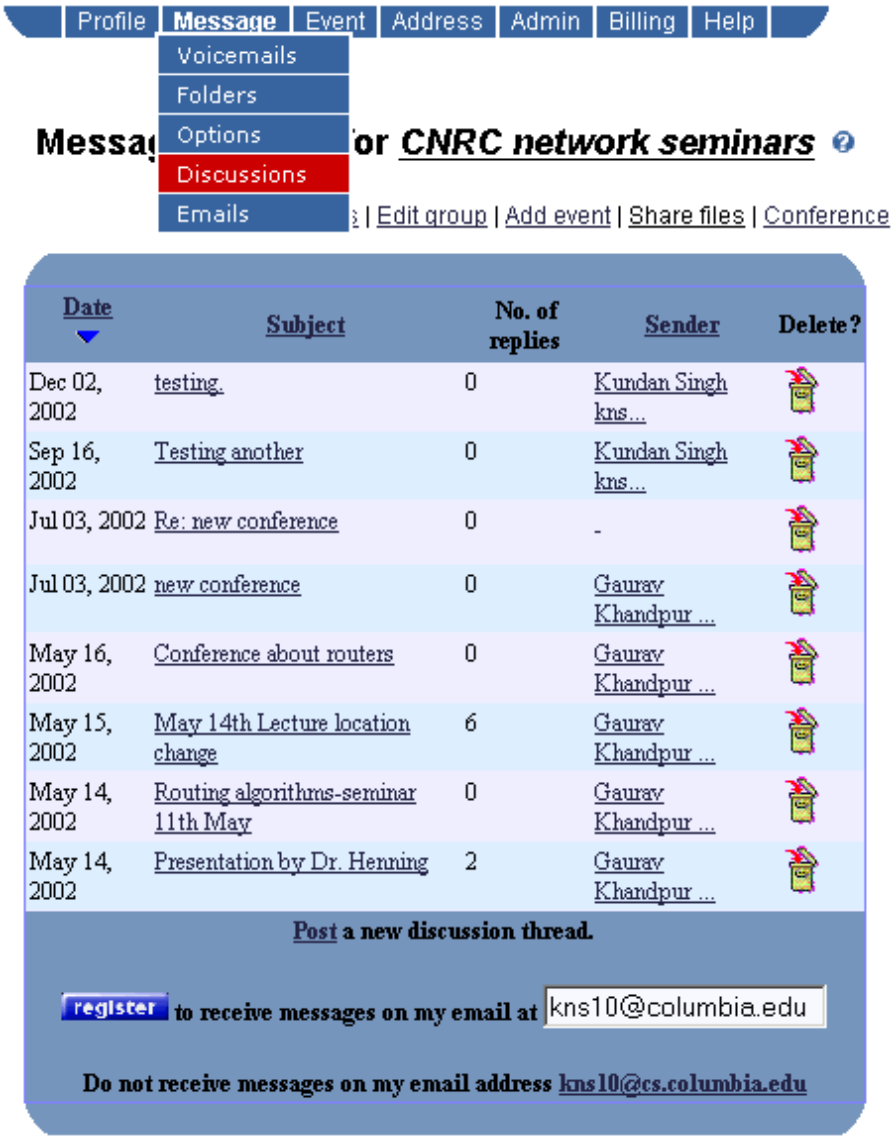


Figure 9.7: Web-based discussion forum

The conference proceedings can be displayed using a time-line on the web interface as shown in Fig. 9.8. The first time-line indicates the complete conference duration with the important events, such as the new user join, leave, file uploads and instant message interaction. The second time-line is the zoom-in view of a part of the conference duration as selected in the first time-line. A user can click on the appropriate icon to playback the recorded media (audio, video), instant message or view the uploaded file. User can click on the time axis to jump to that location.

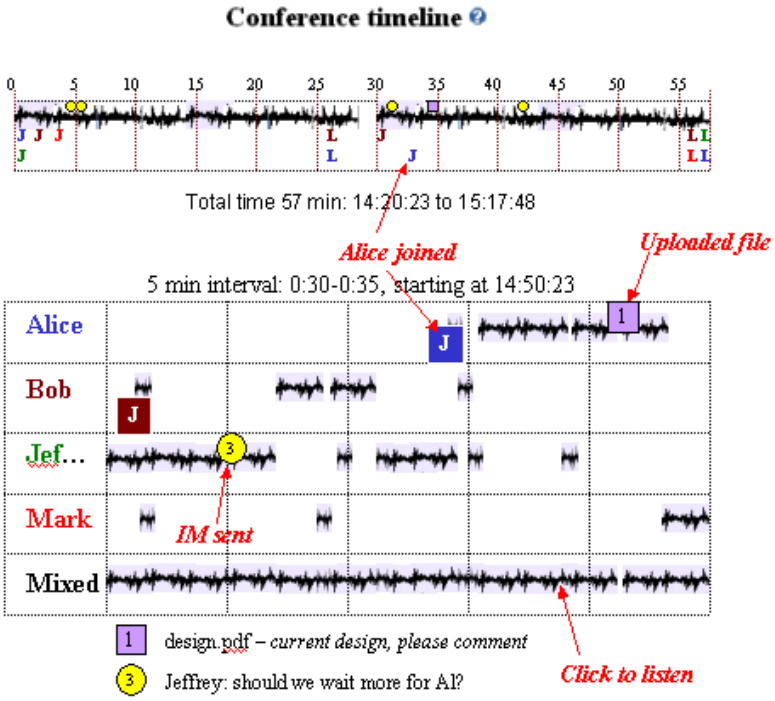


Figure 9.8: Web interface for conference recording

Different colors are used to identify a small number of active participants.

9.5.4 Unified Messaging and Multimedia Mail

The ability to send multimedia messages to other individuals or a group is an important feature of collaboration systems. Registered users can listen to their voice/video messages, recorded conference proceedings or view their emails from the web.

The basic requirements of a voicemail service are secure recording and playback of messages, ease of access and navigation, touch-tone interface, new message notification and call reclaiming by the receiver, i.e., if the receiving user picks up the phone while the caller is recording the message, he should be able to talk to the caller. Additionally, an Internet-based voicemail service should integrate with email, web and instant messaging.

Answering Machine and Message Recording

The voice and video mail is recorded at the media server, rtspd, by the centralized answering machine and voicemail server, sipum. The sipum, directly connects the media path between the caller and the media server, rtspd, hence scales to large user population. Secondly, it uses the standard protocols such as SIP and RTSP, and existing features such as "request-forking", hence does not require any modification to the current infrastructure and can work well even if the voicemail provider is different from the Internet telephony provider.

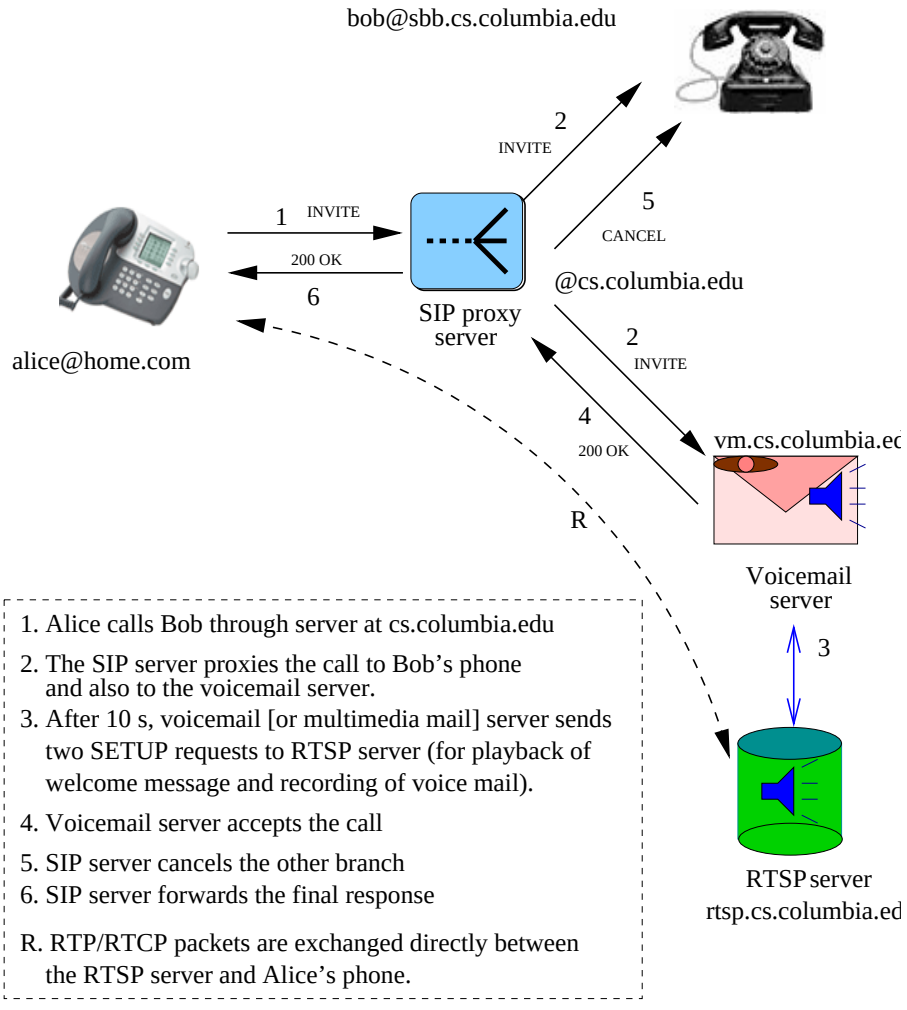


Figure 9.9: Forwarding the call to voicemail

Fig. 9.9 shows an example of recording voicemail. A SIP server handles all the users in

a particular domain, e.g., cs.columbia.edu. Different users register their current location with the SIP server, so that the server can contact the user on receipt of an incoming call. The voicemail server also registers its location on behalf of all the users it is serving. From the SIP server's perspective, there are two active locations for every user, one is his actual SIP based phone and the other is the voicemail server.

When a user Alice, (alice@home.com) calls Bob, bob@cs.columbia.edu, the SIP server proxies the call to both locations. If the user picks up the phone, the branch to the voice mail server is cancelled and a normal SIP call proceeds between Alice and Bob.

The voicemail server is configured to wait for some time, say 10 seconds, before accepting the call. So, if Bob does not pick up the phone in 10 seconds, the voicemail server is going to accept the call on his behalf. Before accepting the call, the voicemail server sets up the media path with the RTSP server. It sends an RTSP SETUP message to the RTSP server to play back the voice prompt to Alice for leaving a voice message. The voice prompt for the outgoing message can be generated using a recorded media file, or, if configured, by converting the text of Bob's vacation message to speech. The voicemail server sends another SETUP message to the RTSP server to record the message.

Once the caller has finished recording, he hangs up and triggers a SIP BYE request. The voicemail server informs the RTSP server to stop recording. Media data for the outgoing and the recorded message is exchanged directly between the caller (Alice) and the RTSP media server using RTP [1, 2].

Having sipum register with the SIP server on behalf of the user is very simple and does not need any intelligence in the SIP user agent or the SIP server. However, there is a race condition, as to whether the user Bob or sipum picks up the call first. If both pick up the call at approximately the same instant, Alice will receive two final responses. It is up to the caller to keep one or both the call legs. The response should indicate whether it is from a multimedia mail system or a human user. This will help the caller's user agent automatically send SIP BYE request to one of the call legs.

This approach does not distinguish between a busy callee and no response from callee. In either case the multimedia mail server will wait before accepting the call. This might be desirable

if the callee's user agent implements call waiting service.

There are several other ways to forward an incoming call to a multimedia mail server: the callee's phone can forward the call to voicemail after few rings, the SIP server can transfer to voicemail if the callee is busy or there is no response. The transfer can be based on either a programmable script or global server configuration. The phone-based forward does not work if the callee's phone is dead or unreachable. Secondly, such intelligence in an user agent is not always possible, particularly in low cost SIP enabled embedded devices. The programmable script such as CPL or sip-cgi allows more precise per-user control over the service. For example, Bob can use the script of Fig. 9.10 to selectively forward the call to his voicemail depending on caller address, time of day, etc. However, this approach requires programmable SIP servers such as sipd.

```
<?xml version="1.0" ?>
<!DOCTYPE cpl SYSTEM "cpl.dtd">

<cpl>
 <subaction id="voicemail">
  <location url= "sip:bob@vm.cs.columbia.edu"><redirect /></location>
 </subaction>
 <incoming>
  <address-switch field="origin" subfield="host">
   <address subdomain-of="cs.columbia.edu">
    <location url= "sip:bob@sbb.cs.columbia.edu">
     <proxy>
      <busy><sub ref="voicemail" /></busy>
      <noanswer><sub ref="voicemail" /></noanswer>
      <failure><sub ref="voicemail" /></failure>
     </proxy>
    </location>
   </address>
   <otherwise><sub ref="voicemail" /></otherwise>
  </address-switch>
 </incoming>
</cpl>
```

Figure 9.10: CPL script for forwarding a call to voicemail

Scalability

The voicemail server has both SIP and RTSP parts. On one side it can receive Internet telephony calls using SIP, and on the other side it behaves as a RTSP client and can perform playback, recording and other control on the multimedia mail residing at the remote RTSP server.

The RTSP server acts as a storage server for the multimedia mails. Separating the voicemail server from the storage server helps in building scalable systems. For example, a single voicemail server can serve all students of an university, while using the departmental RTSP servers for load balancing. Since the voicemail server does not have to handle the media stream, processing speed is not a bottle-neck.

POP3 [230] and IMAP (Internet Message Access Protocol [231, 232]) are not used directly because they do not support media streaming. One can implement a POP3 or IMAP interface to fetch the voice message similar to text based electronic mails.

Notification of New Messages

The server notifies the user of new incoming messages, e.g., using email, and indicates the pointer or URL to listen to the message. It allows sending the media content instead of the pointer in the email, if the user wants that way. That is, forwarding of the voice message as a MIME (Multipurpose Internet Mail Extensions [45]) attachment to electronic mail is supported. It also implements message waiting indication so that the SIP phones can receive notification when a new message arrives.

Retrieving Voicemail

Our system offers five choices for retrieving multimedia mail messages:

1. Existing RTSP based media players can be used to directly play the voice messages from the RTSP server. For instance, the URI rtsp://server.com/bob/inbox/6532.au can be used to retrieve the message number 6532 from the RTSP server, server.com, for user Bob.

2. Bob can also use his SIP phone and call the URI sip:bob-6532-retrive@voicemail.com

to retrieve his message from his voicemail server, voicemail.com. The call is received by the voicemail server which in turn contacts the RTSP media server and retrieves the message. The media data for the message is directly sent from the RTSP server to Bob's SIP phone.

3. Bob can dial the auto-attendant or voicemail number, and navigate through the interactive voice prompts using the touch tone keypad of his telephone.

4. Alternatively, the multimedia mail server can be configured to send the message as an attachment to Bob's email address, as mentioned earlier.

5. The preferred approach is to access the voicemail from a web page using a web browser, as described next.

Unified Display of Messages

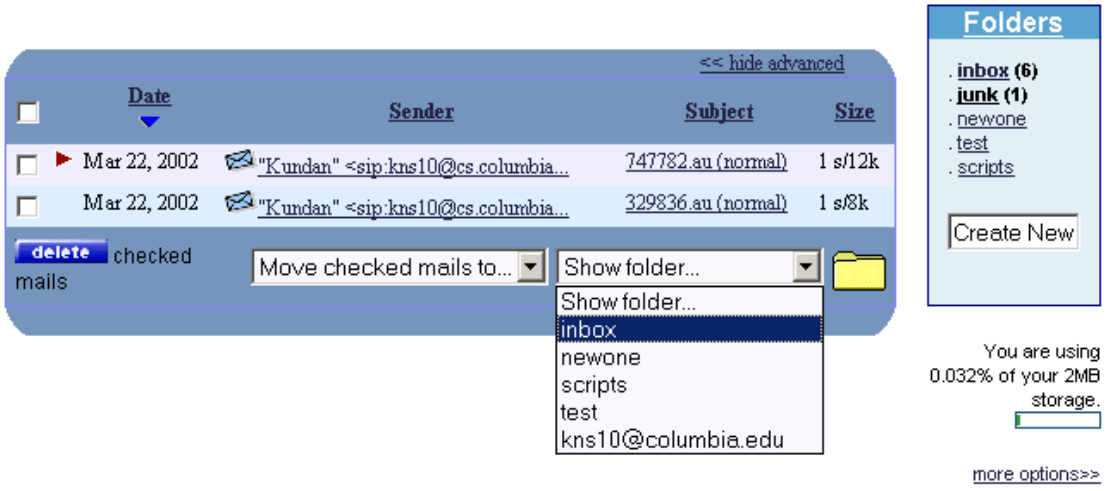


Figure 9.11: Voice messages user interface

An example web page is shown in Fig. 9.11. Basic features like folder management, password change, customizing the voice response, deleting messages and sorting the messages based on different parameters (e.g., date, subject, size) are implemented. The system provides an integrated set of facilities to ease user administration and to share common resources such as

address books, calendar and group messages, and can also be extended to use other email and calendaring tools if the user prefers. The web interface is extended with a simple IMAP-based client to fetch and display the email from user's other email accounts.

One possible enhancement is that the conference invitation sent in an email is automatically added to the user's personal calendar if she accepts the invitation. Alternatively, she can use her own email client to read these messages. She should be able to send multimedia messages to a group of people, such that the message appears in the group member's "inbox" folder. The user can delete or move the message pointer to another folder, and still share the media content across the group.

Call Reclaiming

Another implicit requirement for the voicemail system is to allow reclaiming an already transferred call. If the callee arrives and picks up the phone when the voicemail is being recorded, the system should provide an option for the user to stop the recording and continue talking in a normal call. This is not trivial if the voice mail system is not part of the callee's user agent.

One approach is to use SIP call control to support call reclaiming. In the previous example (in Fig. 9.9), when the call gets transferred to the voicemail server, the voicemail server invites the intended user, bob@sbb.cs.columbia.edu, in the existing call. If Bob picks up the phone while the voicemail is being recorded, he joins the existing call to form a three party conference between the caller (Alice), the voicemail server and himself. The voicemail server then drops out of the conference by sending a SIP BYE. If Bob does not pick up the phone, the voicemail server cancels the call once the message from Alice has been recorded. However, it is not clear how the voicemail server can call Bob without having the SIP server fork a branch back to the voicemail server. One can extend the caller preference [233] to include a description of the user agent picking up the phone.

Another approach is to use third party call control, with the voicemail server as the third party. It simply sends an INVITE to Bob, with Alice's session description. If Bob picks up, it also changes Alice's session description via re-INVITE, so that the two now talk directly to each other media-wise. To avoid any confusion to Bob, the voicemail server may prompt him that

Alice's message is being recorded.

A third approach uses call state notification. Bob subscribes to call events from the voicemail server and can INVITE himself to the call. This requires further study.

It might be desirable to have the user decide whether to stop the recording or not. The caller may not want to repeat the long message if he has already recorded most of it.

It is not clear how essential a call reclaiming feature is in practice, given that most users using the centralized voicemail system of the mobile phone service provider do not currently have this feature and are not complaining. Since implementing the call reclaiming is complicated, it may be desirable to leave it for simplicity.

The voicemail server uses the SIP request-URI to identify the purpose of the call. For instance, if the call is directly made to the voice mail server to leave an announcement or a reminder in user's mail box, the server should not try to contact the intended recipient.

Deletion of Messages

The architecture assumes that the RTSP media server stores the multimedia messages. However, there is no explicit mechanism to delete a resource in RTSP, in its current form.

One option is to define a new method, say DELETE, to delete a resource or a media file on the RTSP server.

The other approach is to pretend as if you are recording the file, but terminate the RTSP connection without actually recording anything. To be more specific, an RTSP SETUP with record mode is sent to the server, immediately followed by an RTSP TEARDOWN, without sending a RECORD message. Our RTSP server interprets this as a command to delete the file. Even otherwise, the recorded file will be empty, and of no use.

While the first method is more direct, it requires modifying RTSP. We have implemented both approaches.

9.5.5 Notifications and Announcements

The system can notify the user of various appointment reminders, conferences schedules or changes in shared files, message board or incoming multimedia message. The notification in-

formation stored in the SQL table and can be associated with an event and an eventgroup. The information also contains the destination for notification such as phone number, SIP URI, email address or IM address, time relative to the event in seconds (e.g., notify 60s before the event), and the identifier of the scheduled notification. The notifications are scheduled using the "at" command on the Unix platform. The user can schedule the same notification to multiple destinations. It supports different kinds of notifications:

- Birthday, appointment or other event reminders for which the notification is sent before the event occurs.
- Scheduling any text or media as a notification (e.g., wake-up call) that automatically creates an associated event. The notification is sent when the event occurs.
- Notification for the eventgroup, in which case the notification is sent for every individual event in that eventgroup.

While an email or IM is an one-time event with no interaction, a phone-based notification can prompt the user with more options via interactive voice response. For example, "press 1 to get notified again after 5 min, or press 2 to listen to the details of the event". The system can allow scheduling the notifications from the web interface or via telephone using the touch-tone input.

It is possible to send a phone announcement to an eventgroup, in which case all the group members get the announcement, or to a set of SIP addresses or phone numbers. For example, an announcement to 1-212-93970?? will be received by all the valid telephone subscribers in the range 1-212-9397000 to 1-212-9397099. The announcement server makes SIP calls to all the numbers specified, and if successful, speaks out the announcement. It attempts multiple times on busy or no-answer. To avoid leaving the announcement to an answering machine, the server can prompt the recipient to press some digit to confirm user presence. Such announcement system will also be useful in the case of an emergency.

9.6 Additional Services

So far we have discussed the synchronous and asynchronous collaboration tools in CINEMA. There are other interesting services that assist both synchronous and asynchronous collaboration. For example, a conference server can dial-out a scheduled meeting only when all the required participants are on-line. An IM user can join a tele-conference and interact via speech-to-text and text-to-speech conversion between the IM text and other participants' audio. The location information published by the user can determine her availability. We describe some of these enabling technologies in this section.

9.6.1 Presence

The presence information gets used quite often in people's daily life. People are used to checking online status before starting a conversation with their IM "buddies". In our system, we base our presence information handling on the SIP event notification architecture [139].

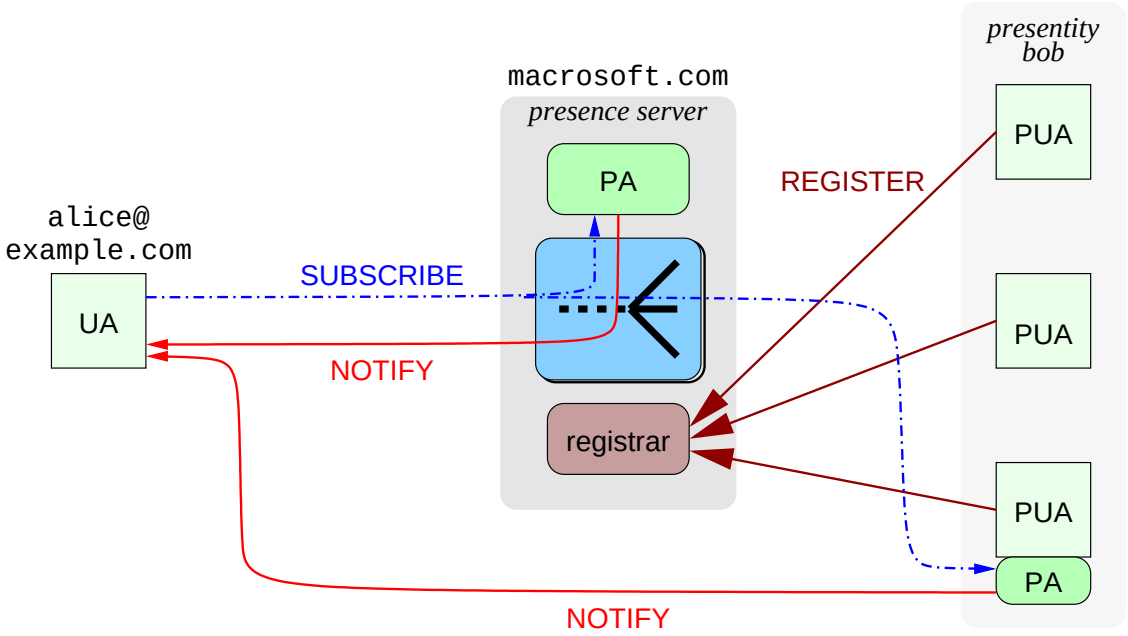


Figure 9.12: SIP-based presence

The presence information is maintained either on the SIP servers residing in networks or on the presence-enabled SIP user agents as shown in Fig. 9.12. If a user Alice is interested in the

presence status of another user Bob, then she subscribes to his address *sip:bob@macrosoft.com* for the event package of "presence". Our SIP server, sipd, proxies the SUBSCRIBE message to the registered user agents with presence capability of Bob. If the user agent wishes to handle the subscription, it sends a 200-class SIP response to sipd. Therefore, sipd disables the internal presence agent for this subscription. On the other hand, if the user rejects (600-class response), then the sipd stores the decision in the SQL database so that future subscription from Alice to Bob are also rejected even if Bob's user agent is off-line. For other responses such as the user agent is not presence-enabled, the sipd enables the built-in presence agent for this subscription. However, before the actual presence information can be conveyed to Alice, the subscribee Bob must approve the subscription from the web.

The web interface displays the list of subscribed users (buddies) as well as all the others who are interested in knowing the presence status of this user as shown in Fig. 9.13. Note that a subscription can be handled only by the server or the user agent but not simultaneously by both [234]. It is possible to transfer the subscription from the user agent to the server and vice-versa.

More recently, our SIP server, sipd, has been simplified by extracting the presence agent part as a separate server.

The network-based presence agent is useful when the end devices are not presence-enabled such as location sensors or magnetic swipe-card reader. For example, Bob can use a passive device, such as a magnetic swipe-card or an iButton [235] and the card or button reader delivers the location information to the server. Alternatively, when Bob's wireless phone REGISTERs with the server, the server can publish his on-line status to the subscriber, Alice.

Pushing the presence information to the end systems also has some benefits. In Internet telephony, end systems are the only entities where signaling and media flows converge whereas intermediate proxies only handle signaling. The means that several services can only be performed in the end system.

The SIP event notification can be applied to non-presence events, e.g., the voicemail server can notify the user's phone of any waiting messages [236].

Figure 9.13: Web-based presence

9.6.2 Interactive Voice Response (IVR)

We have discussed a number of examples involving user interaction via touch-tone input from a telephone. Our sipvxml is a SIP-based VoiceXML browser that allows a SIP phone, or a regular telephone via a gateway, to interact with the back-end application logic [36]. We have implemented only a subset of VoiceXML tags as needed in our application: assign, audio, block, catch, clear, disconnect, dtmf, error, exit, field, filled, form, goto, help, noinput, nomatch, prompt, submit, value, var and vxml. We do not support any client side script (e.g., JavaScript) usually needed for arithmetic or string operations in the browser, as the same effect can be achieved using server side processing.

Operation of the Browser

Fig. 9.14 shows the components of our SIP-VoiceXML browser, sipvxml. We use Apache's XML parser [237] with DOM interface, an HTTP fetcher [238] for getting non-XML pages and CMU's

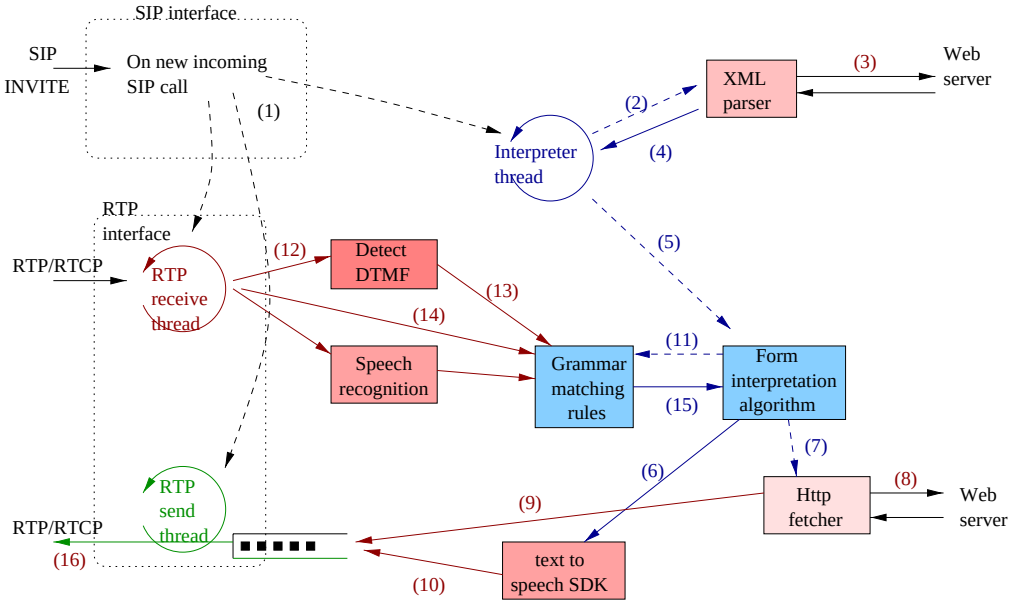


Figure 9.14: Operation of sipvxml

FLite text-To-speech (TTS) for speech synthesis [239].

(1) When the browser receives a new incoming SIP call it creates three different threads: an RTP receive thread, an RTP send thread, and the VoiceXML interpreter thread. The RTP receive thread receives media packets from the caller and invokes the DTMF detection module. The RTP send thread streams out media packets to the caller. A separate send thread helps in maintaining the constant bandwidth (e.g., 64 kb/s for G.711 audio) for outgoing packets and irrespective of the speed of the speech synthesizer. The initial VoiceXML page URL can be preconfigured in the browser or encoded in the SIP request [207]. For example, if the caller dials sip:dialog.vxml.http%3a//dialogs.server.com/ script32.vxml@vxmlservers.com then the call will reach the browser running at *vxmlservers.com* and it will fetch the initial VoiceXML page from http://dialogs.server.com/script32.vxml. On the other hand, if the request-uri is sip:7137@cs.columbia.edu, then the interpreter is invoked with the default pre-configured initial VoiceXML URL, e.g., that of the conferencing service.

(2-5) The interpreter thread calls the XML parser with the initial URL. The XML parser fetches the page from the web server or a local file system (based on the initial URL). It presents the returned XML document into a tree data structure. The interpreter thread invokes the *Form Inter-*

pretation Algorithm (FIA [35]) on the selected form from the VoiceXML document.

(6-8) FIA invokes various other modules based on the content of the VoiceXML document. For example, it can invoke the text-to-speech SDK to synthesize any prompts. The current implementation does not use any speech recognition engine because user input is via touch-tone keys. FIA can also invoke the HTTP fetcher module to fetch an external grammar file or a media file for an audio prompt. XML parser internally has its own HTTP client to fetch VoiceXML pages. The HTTP fetcher implements a simple HTTP GET method to retrieve a document.

(9-10) The media file retrieved from the web server using HTTP fetcher is fragmented into 20 ms packets for interactive telephony, and enqueued for streaming out to the caller by the send thread. The speech synthesizer output is also fragmented and enqueued for sending out to the caller.

(11) The VoiceXML document can specify the grammar rules in various scopes in the document. FIA can set the active grammar for the matching engine based on the current execution scope in the VoiceXML page.

(12-14) The RTP receive thread receives the RTP media packets and invokes the DTMF detector. Any detected DTMF digit is passed to the grammar matching engine. DTMF tones can be transported from the caller to the browser in a number of ways. One approach is to not distinguish them from the spoken voice by encoding them using the same audio codec. However, a low bandwidth audio codec may distort the properties of the in-band DTMF tones making them hard to detect. A second, preferred way is to use “telephone-event” [227] containing the digit codes instead of the encoded audio in RTP packets. In the first case, the browser has to do the DTMF detection, whereas in the second case the caller or the gateway has to do the DTMF detection. The RTP receive module forwards telephone-events directly to the grammar matching engine. We have implemented both these methods. A third method of transporting DTMF in SIP INFO message is not used in our implementation.

(15-16) The grammar matching engine tries to match the received digits with any active grammar, and informs the FIA if a match is found. The RTP send thread periodically sends media packets to the caller. No packets are sent during silence.

We have developed some CGI-based applications for voicemail access and conference participation. Each registered user gets a unique telephone PIN (personal identification num-

ber) for authentication. The voicemail script announces the number of new and old messages, and prompts the caller to listen to the messages. The conferencing application prompts for the conference number and transfers the call to that conference.

Joining a Conference via IVR

Consider a SIP conferencing system where users join the conference by dialing in a conference URI such as sip:staffmeet@conference.com. A regular telephone user with only a touch-tone phone cannot dial such a generic URI. We can assign one phone number per conference for Direct Inward Dialing (DID). However, it is preferred that the user always dials the number of the VoiceXML browser or some auto-attendant that in turn prompts him for the authentication PIN (personal identification number) and conference number. Once the user is authenticated the browser transfers the call to the selected conference. One can also use a single PIN to identify both the participant as well as the conference.

Fig. 9.15 shows how an user, say Alice, interacts with the browser before joining the conference. (1) Alice dials the browser's phone number, 212-9397137, or SIP URI, sip:7137@server.com. (2) The browser accepts the call and prompts the caller to enter PIN for identification. (3) Alice keys in her PIN, 1-2-3-4, followed by a terminating # key. The DTMF digits are sent in RTP. (4) The browser looks up the database and identifies the caller as "Alice". (5) Based on the privileges, the browser prompts her with a list of conferences to choose from. (6) Alice picks up the conference with identifier 23. (7) The browser again checks if Alice is allowed to join the conference identified by number 23, which in this example is sip:staffmeet@conference.com. (8) Once the authentication is done, the browser transfers the call to the actual conference server using the SIP REFER method [77] containing the SIP URI of the conference. (9) Alice's phone accepts the transfer and initiates a new call to the conference server. (10) Alice's phone exchanges audio with the conference server directly, without going through the browser.

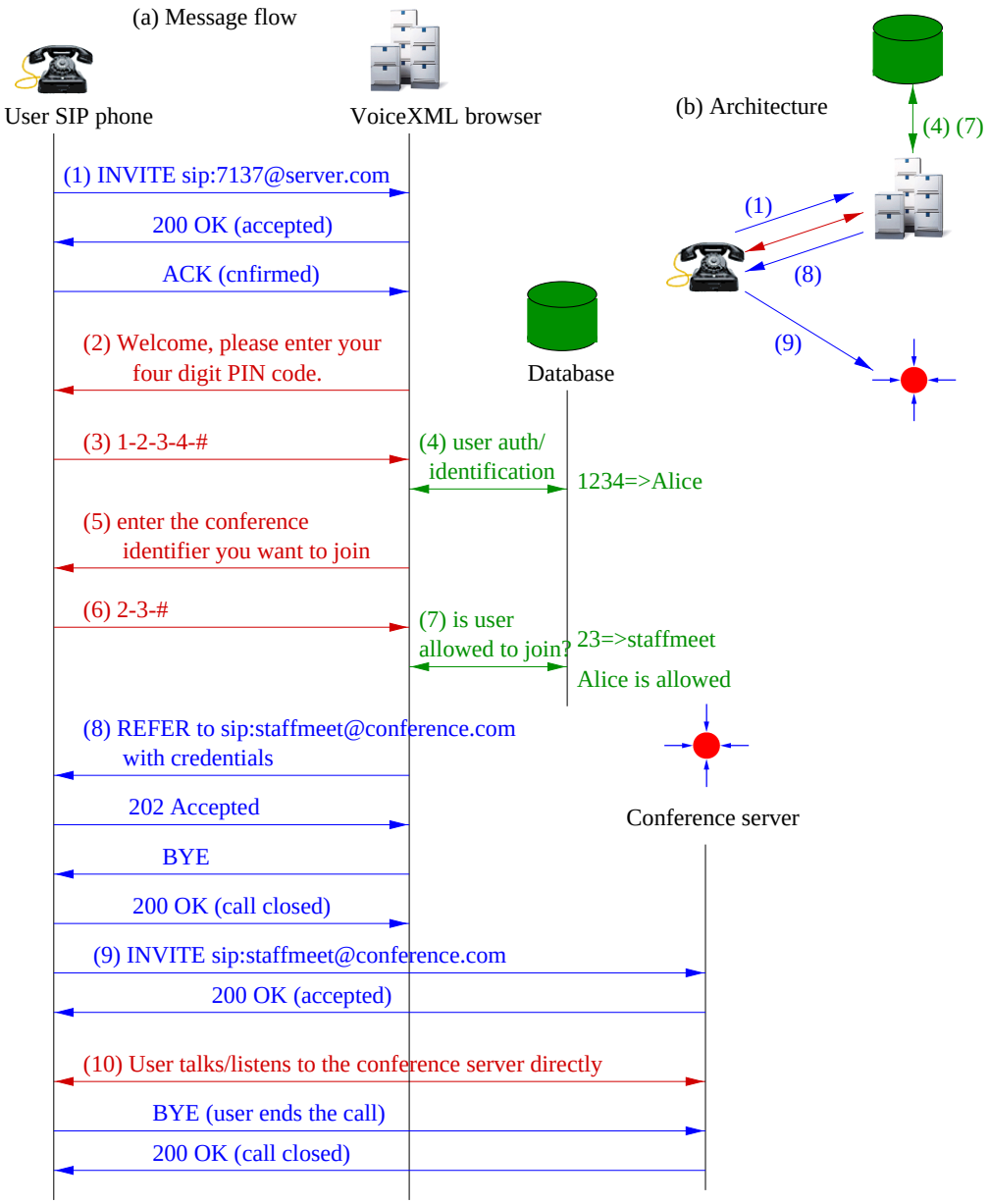


Figure 9.15: Method 1: Joining a conference in blind transfer mode

Call Transfer

Note that the user authentication, conference look up and transfer are actually invoked by the conference service CGI scripts, whereas the browser just interprets the VoiceXML pages generated by the scripts to do the actual transfer or prompt the caller. For instance, the service script may

generate the following transfer tag for the call transfer in step (9).

```
<block>
 <prompt>
  Your call is being transferred, please wait.
 </prompt>
</block>
<transfer dest="sip:staffmeet@conference.com" bridge="false" />
```

There are two ways to transfer a call from the browser to another phone in VoiceXML. The *blind* transfer sends the SIP REFER message, and terminates the original call leg between the caller and the browser. The caller user agent is then responsible for placing another call to the Refer-to location. The *bridged* transfer causes the browser to place another call to the destination, and join the media path between the original caller and the destination.

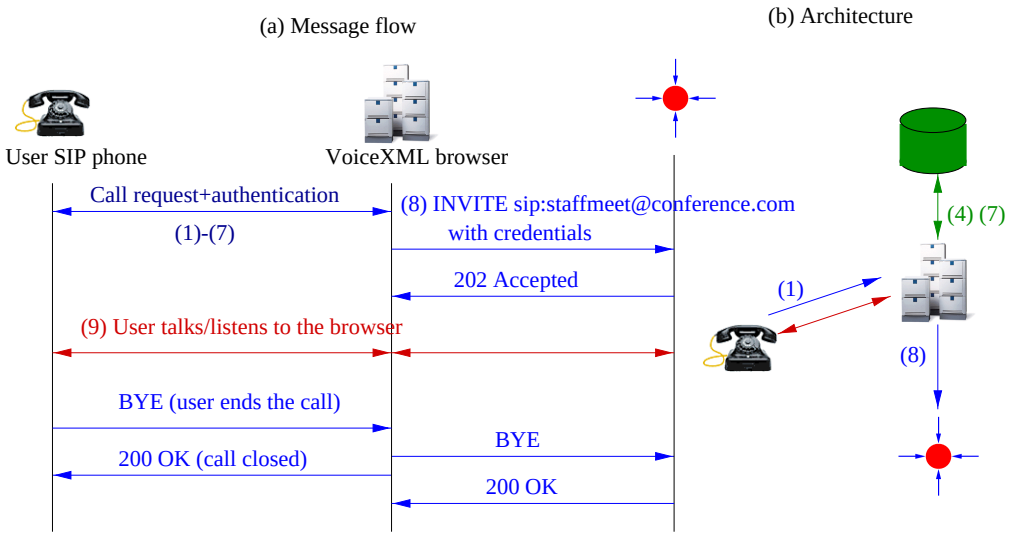


Figure 9.16: Method 2: Joining a conference using bridged mode

Fig. 9.16 shows the bridged transfer case with the browser as a back-to-back-user-agent (B2BUA) bridging the audio path between the user phone and the conference server. Steps 1 to 7 are same as in the blind transfer case. Instead of sending REFER, the browser sends a new call request to the conference server identifying the conference sip:staffmeet@conference.com in the Request-URI of the SIP INVITE message. The browser acts as an application level packet

forwarder in both directions for RTP and RTCP media traffic.

We have implemented both blind and bridged transfer in sipvxml. The advantage of bridged transfer is that the browser remains in the media path and can terminate the call (e.g., if the calling-time exceeds the quota) or accept future control commands (using DTMF) from the user phone. For conferencing, it may be useful to interpret DTMF, e.g., 6-6-# to mute your audio or 6-8-# to join another virtual chat/conference room. Secondly, the browser needs to forward other signaling messages also, e.g., re-INVITE from the caller to the conference server. Moreover, maintaining packet forwarding states for the duration of the conference limits the scalability of the browser on how many simultaneous callers it can handle. The browser may issue re-INVITEs with updated transport addresses for media to both the caller and the conference server such that the media path is direct. However, this still needs to maintain the call signaling state for the duration of the call. On the other hand, a blind transfer does not require any call state in the browser for the duration of the conference. But it expects that the caller's IP phone supports the SIP REFER method.

Instant Message as Input and Prompt

To allow PC-based SIP phones that do not have touch-tone dial-pad, our browser also accepts input via IM text. The prompts can be sent both in audio and IM for such phones.

Distributed Component Architecture

In a distributed component architecture it may be desirable to separate the text-to-speech and speech-to-text functions from a VoiceXML browser, as different modules. Our RTSP server, rtspd, can convert the text supplied in URL to speech and stream to the client. Alternatively, a SIP "text-audio" converter can convert between the text in IM and the audio in the call session. Such external components can be invited in the existing sessions for applications such as email by phone, as we describe next.

9.6.3 Interaction among Email, Telephone and IM

Today, email is the most common form of electronic communication. However, the convenience of email is limited by the necessity of an Internet connected computer. A system that allows interworking of email with other communication means such as telephone or IM, will enhance user experience. Such system can be used to reach those users who only have email access via IM, define certain incoming emails as important and forward them to IM, get a virtual-IM account to interact with other IM users via email, access emails via phone, get notified of any important email on phone, and text-chat with other IM users or in a conference via phone. We describe the SIP-based architecture and on-going implementation of such interactions in CINEMA.

Email by Phone

Assuming that wherever you go there is a telephone, using a telephone to check email is a sensible solution. Our *email-by-phone* system provides a way to check and even send email from a touch-tone telephone [240]. The application runs as Java Servlet on a web server, generating VoiceXML pages for telephony dialogues, and interacting with back-end IMAP or POP3 email-servers as shown in Fig. 9.17. We have also implemented Tcl-based CGI scripts for the email-by-phone service to better integrate with the rest of our system that uses Tcl scripts.

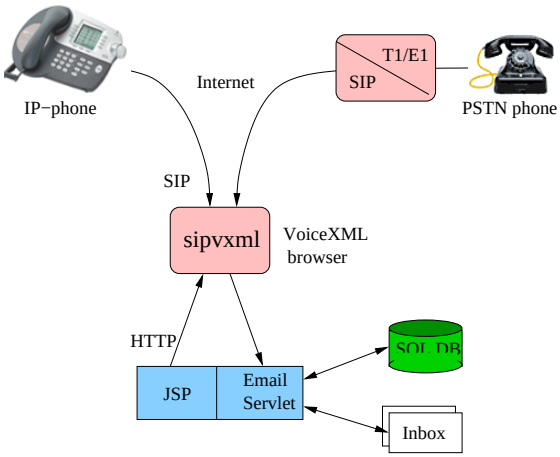


Figure 9.17: Email-by-phone architecture

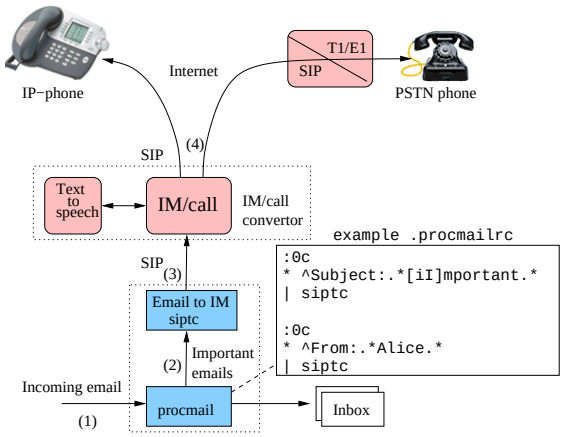


Figure 9.18: Email notification to phone

The caller is prompted to listen to old or new messages, compose a new message, reply

to an existing message, delete a message, forward a message, advance through the messages, and switch between new and old messages for playback.

Email to Phone

Asynchronous event notification is useful when polling for the event is inefficient. For example, the email-by-phone system can be modified to notify the user of any important email by calling user's cell phone. The definition of "important" email can be programmed by the user. The architecture is shown in Fig. 9.18.

Incoming email filtering on Unix is relatively simple using procmail [241]. On other platforms, such as windows, one can periodically poll the IMAP-based email-server for new emails. An example procmail script of Fig. 9.18 treats the subject with "important" or "Important" keyword, or sender as "Alice", as important and forwards to the siptc script.

The siptc script extracts the email body and other information, e.g., subject, priority and sender address, creates a SIP instant message, and sends it to the IM-call converter. The IM text is truncated if it is too big. The forwarded-from, replied-to or signature part in the email are ignored as shown in Fig. 9.19.

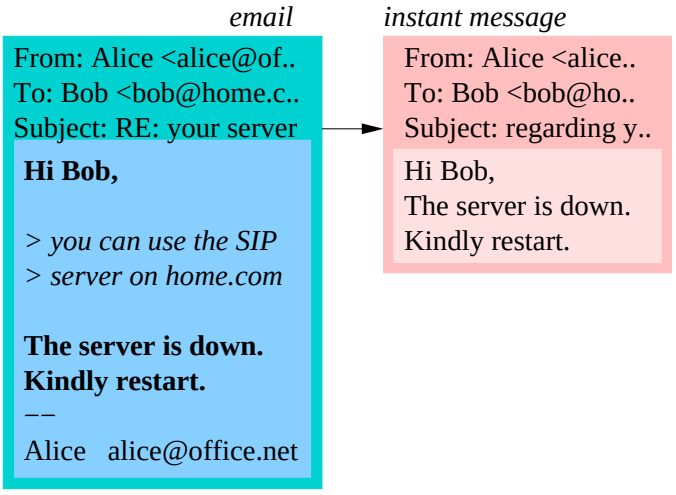


Figure 9.19: Example translation used in email to phone system

The IM-call converter acts as a translator between the SIP-based IM and audio call. In the reverse direction, it can notify the IM over phone. In the absence of session-based IM, it uses

the various headers in the SIP MESSAGE to associate the IM session with the audio call. This avoids making a new SIP call, if IM is received for an existing session. The SIP call destination can be pre-configured or derived from the IM destination address, which can be an IP user or a telephone subscriber via a gateway. Separating the converter from the email to IM translation allows running the email system and speech system on different hosts in the network.

Once the call is established the converted text is spoken out to the destination phone. After that, the system may transfer the phone call to an IVR system, e.g., to prompt the user to repeat the message or connect to the *email-by-phone* system.

IM-call Converter

The IM-call converter described in the previous section can be extended to a generic translator to allow a phone user to initiate an IM conversation. It uses both text-to-speech and speech-to-text. Suppose Bob's IP telephony service provider allocates a telephone extension say 7155 for his IM address. When Alice dials the extension, the service provider maps the destination to *sip:Ym9iQGhvc3RC@serverC* where Ym9iQGhvc3RC is base64-encoding [45] of *bob@hostB*. The translator, simvoice, running on *serverC* receives the call request and sends an initial IM greeting to *sip:bob@hostB*. It maintains the association between the caller and the final IM destination for the duration of the call.

When Alice speaks, the audio is converted to text using CMU Sphinx speech recognition engine [242]. To send an IM, the user can indicate the end of a speech message by pressing a DTMF key. Alternatively, the converter can assume the end of a speech message when it receives some audio followed by a few seconds of silence.

In the reverse direction, when Bob sends an IM text to simvoice, it invokes the Flite text-to-speech engine [243] to convert and send it to Alice as voice.

If Alice hangs up, the association is lost. If the simvoice can not find an associated call for an incoming instant message, it replies with another instant message indicating the error. The IM user should be allowed to initiate the session using session-based IM

In a collaboration environment, the converter allows the users with different system capabilities such as telephone and IM to interact. This helps the deaf, hard of hearing and speech-

impaired individuals to collaborate easily in a multimedia conference [244].

Often we have found that the speech-recognition quality is poor. The converter should provide feedback to the speaker by sending the converted IM text as well to the phone using text-to-speech.

Email to IM

In the email to IM direction, Alice can email to a special address such as *myserver+bob@office.net*[1] which is received by Bob as IM. The procmail script of *myserver@office.net* receives the email, finds the IM destination as bob, translates the email's text content to IM and sends it to *bob@office.net*. Alternatively, Bob can advertise his email address as *bob+im@office.net* to send him an IM. The difference is that the procmail is configured at *myserver* in the former and at Bob's email in the latter case. A third approach is that Bob defines certain emails as important and automatically forwards them to his IM address.

IM to Email

In the reverse direction, Alice can also use the services from *myserver* such that Bob can send IM to *myserver+alice@home.com*. The server should put the appropriate email Reply-to header pointing to the sender via *myserver*, so that the email replies can be sent back to the IM user correctly. Alternatively, Alice can sign-up with her SIP-provider to run a programmable call-routing using SIP-CGI [24] that identifies important IMs and sends her an email with the content when she is not on-line, as shown in Fig. 9.20.

9.7 Conclusions

This chapter describes a SIP-based collaboration framework that integrates with telephony, instant messaging, email and web using existing protocols and tools. We have discussed seamless integration between two types of collaboration modes: synchronous and asynchronous. The conference server and user agent in our CINEMA infrastructure allow synchronous multi-party mul-

[1]Note that some email servers allow sending email to user+something@domain, which will be delivered to the inbox of user@domain.

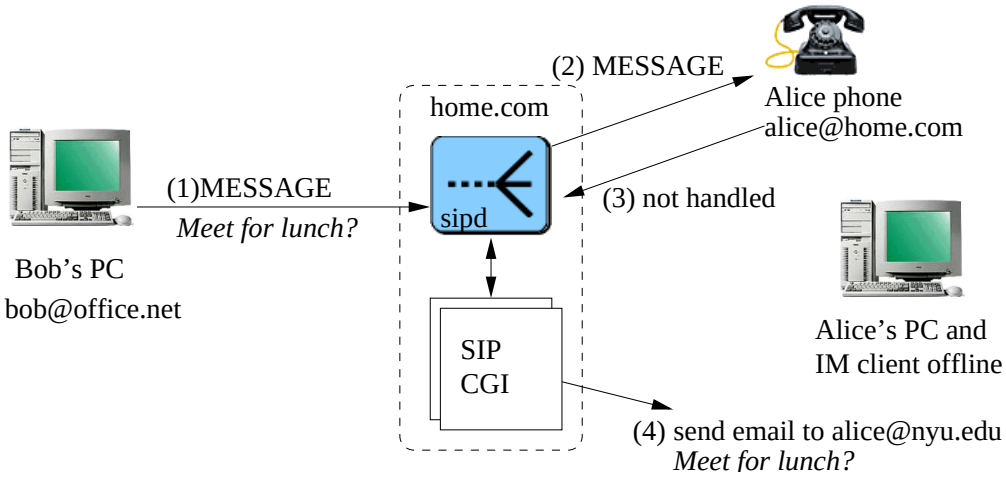


Figure 9.20: SIP-CGI for IM to email translation

timedia collaboration via audio, video, instant message, screen sharing and shared web-browsing. The personalized user profile, calendaring, address book management, event and conference management, and system configuration can be done from the web interface. It also facilitates document sharing and asynchronous discussions among the group members. Moderators can monitor and control various synchronous and asynchronous activities. The messaging and notifications are used to reach the users when they are off-line.

We have described the architecture of our Internet telephony installation consisting of the SIP server, SIP-PSTN gateway, RTSP media server, unified messaging server, conferencing server, interactive voice response server and SIP-H.323 translator. It provides enterprise IP telephony architecture for corporates and campuses. We have used CINEMA in our department as an example of real world deployment. We have also built various components addressing a commercial deployment such as security, billing, and interworking with the corporate firewalls and Network Address Translators (NATs). A similar architecture can be deployed at other campus and organization networks who want to benefit from the services provided by Internet telephony, in particular SIP.

A SIP-based architecture allows to easily extend the infrastructure with new features, e.g., presence-enabled calls and programmable call routing. Interactive voice response provides easy access to the system from a telephone, whereas various text-to-speech tools allow interaction via

plain email. This facilitates access to the system transparent to the end user device. Hence, we claim CINEMA to be a comprehensive multi-platform collaboration architecture. Moreover, the system allows hybrid interaction, e.g., phone for audio, PC for IM and document sharing in the same conference.

Based on our implementation, SIP provides a suitable multimedia conferencing protocol that allows advanced scenarios and services without requiring that end systems are conferencing-aware. Our conference server supports audio mixing based on various codecs such as G.711 μ-law and A-law, GSM, DVI ADPCM, high quality G.722 and, more recently, Speex. The video codecs are transparent to the server, because the server does not need to decode video and can just do packet forwarding to other participants. In addition to audio and video conferences, various other services are provided at the conference server such as instant messaging and screen sharing. Our SIP-H.323 gateway permits the participation of H.323 clients in the SIP-based conference. Participants can also join conferences from the PSTN via a SIP to PSTN gateways using interactive voice dialogs.

VoiceXML is a powerful technology for interactive voice dialogs that allows telephone users to access services that are typically available to web users. SIP interface to a VoiceXML browser allows such services from a IP telephone as well as a regular telephone, via a gateway, using the call transfer feature in an interoperable manner. We have implemented a SIP VoiceXML browser and used it to allow telephone users to connect to our conferencing server. This along with other services that we have implemented will help users to join a conference, check mails and stay informed from anywhere thus enabling ubiquitous availability.

We have described a multimedia mail architecture for Internet telephony, using SIP and RTSP, and shown how it meets the general requirements for a voicemail service. Various approaches are possible to utilize the voicemail service in the Internet telephony environment. Applicability ranges from a single user subscribed to a voicemail service to a whole university using the campus wide service. Separation of the voicemail server from the signaling and the storage servers helps in building scalable systems. We have also described some of the protocol issues, in particular, reclaiming a transferred call and deleting a mail, based on our implementation.

Our architecture is being enhanced and licensed by SIPquest Inc., to over a hundred com-

mercial sites for evaluation or deployment. We have used a number of components of CINEMA in various other projects in our lab. For example, the NG911 project uses the conference server to bridge the emergency caller, emergency responder and any third party medical or police assistant.

Total physical lines source lines of C/C++ code of various CINEMA components measured using SLOCCount [245] is about 180,000 including some external software. Out of this my contribution is more than 60,000 in C/C++ and an additional 30,000 in Tcl.

We have described the various collaboration tools in CINEMA and how they interact to achieve new services. Collaborative work is a vast research area incorporating numerous technologies such as networks, multimedia, object oriented concepts, virtual reality and artificial intelligence. Our aim is to complement these research innovations by providing a framework over which other collaboration tools can be built or integrated. Although CINEMA's main focus is on real-time synchronous communication, we also correlate the two modes of collaboration for an enhanced end-user experience. CINEMA can be used within an organization as well as in portal mode by application service providers. We have not yet implemented the recording of conference events such as join or leave, and the programmable conference server behavior.

We evaluate the performance of our conference server in the next chapter.

Chapter 10

Scalable Centralized Conferencing

10.1 Introduction

Multimedia conferencing forms the core of synchronous collaboration described in the previous chapter. Section 9.4.1 (page 189) describes our centralized conference server and audio mixing algorithm. In particular, the server implements the *decode-mix-encode* steps to perform audio mixing such that each participant can listen to all the other participants.

Providing reliability using server redundancy is not trivial because the conference server maintains call state unlike the call stateless SIP proxy servers. In this chapter, we describe the techniques for reliability and scalability of conference servers. We also evaluate the performance of our conference server for single server as well as distributed cascaded server architecture.

Vendors have built DSP-based customized hardware that are specialized in processing audio. However, one of our main goals is to provide carrier grade services on commodity hardware such as desktop PCs and workstations. We measure the conference server performance on commodity hardware running Linux.

Scalability and reliability of different media are handled differently. There are two types of media handling: mixing and forwarding. Audio is typically mixed whereas video or instant message is forwarded without any mixing in the conference server. For the purpose of this chapter we focus only on audio mixing. We only examine the media path, and do not explore the performance effect of high signaling churn, i.e., high rate of participants join and leave.

10.2 Scalability

Conference size is measured in terms of number of participants. A small scale conference contains two to ten participants, medium scale ten to few hundreds and large scale over thousands of participants. Broadly speaking there are two metrics to quantify the load on a conference server: (P) the number of simultaneous participants in a single conference, and (C) the number of simultaneous conferences with small number (three or four) of participants each. High load on the conference server requires more processing and bandwidth at the server and can hamper the quality of service metrics such as delay (D), jitter (J) and packet loss (L).

10.2.1 Requirements

Performance is typically limited by three factors: server's network bandwidth, CPU and memory. Which of these becomes the bottleneck in turn depends on other parameters such as media codec and packetization interval. For example, high bandwidth G.711 codec imposes less CPU overhead whereas the low bandwidth GSM or G.723.1 codecs are CPU intensive.

The goal of performance measurement of a conference server is to understand the effect of load (P, C) on the limiting factors (CPU, bandwidth, memory) for a given set of parameters (codec, interval). Thus, we identify the bottleneck for the given hardware and measure the maximum capacity (P_{max},C_{max}) without affecting the quality of service. In particular, more than 5% of packet loss or more than 150 ms of mouth-to-ear delay will hamper smooth bidirectional conversation, and we pick those as indicating that the server has reached its performance limit.

Server bandwidth

On average, only one participant is speaking in a conference at a given time. Otherwise the listeners may not be able to understand the conversation. In large conferences, participants should perform silence suppression to reduce the mixing load on the conference server. If the codec bitrate is B, then P participants in a conference with one active speaker at any instant and the others performing silence suppression, requires an inbound bandwidth of B and an outbound bandwidth of $P \cdot B$ at the server.

CPU

An audio mixer with C conferences and P participants per conference requires $(\alpha P + \beta)C$ instructions per second, where α corresponds to the per-participant processing and β to per-conference processing. α includes encoding of the receiver's audio and any copying of data among buffers, whereas β includes decoding of speaker's audio and any processing of periodic interrupt for the conference. A simple codec such as G.711 requires minimal encoding and decoding effort, but imposes heavier burden on buffer copying due to the large message size.

Memory

The memory at the server is usually not an issue. A real-time conference server needs to keep the delay and hence the buffer size small, thus it does not use lots of memory in audio buffers. Nevertheless, each conference and participant requires some call signaling (SIP) state at the server.

Increase in parameter value	CPU	Bandwidth	Delay
Packetization interval (T)	reduces	reduces	increases
Codec bitrate (B)	increases	increases	N/A
Codec complexity (M)	increases	N/A	N/A
Network jitter (J)	N/A	N/A	increases

Table 10.1: Effect of various parameters on the server performance

Table 10.1 summarizes the effect of various parameters on performance. Note that the delay column indicates the base delay under normal load. At high load, when CPU utilization is close to 100%, the delay and loss increase abnormally.

To derive a mathematical relationship among these parameters, consider a single conference server with C conferences, each with P participants. Each participant is using audio codec with bitrate B kb/s, and complexity of M_e and M_d cycles of CPU for encoding and decoding, respectively. Suppose the packetization interval is T ms and the network jitter in the speaker to server path is J ms. The packet size includes the audio payload size as well as the packet header for IP, UDP and RTP, which is $20+8+12 = 40$ bytes. The packet header contributes a bandwidth

of $\frac{40\times8}{T} = \frac{320}{T}$ kb/s. Thus, under normal load on the conference server, we get the following:

$$\begin{aligned}
&\text{IP packet size} && = \tfrac{BT}{8} + 40\,\text{bytes}\\
&\text{Inbound bandwidth} && = C(B + \tfrac{320}{T})\,\text{kb/s}\\
&\text{Outbound bandwidth} && = C(P-1)(B + \tfrac{320}{T})\,\text{kb/s}\\
&\text{Delay at server} && \leq T + 4J\,ms\\
&\text{CPU usage} && \propto (\alpha P + \beta)C\\
& && = ((M_e + a.B_1 + b)P + (M_d + c.B_1 + d))C\\
& && where\, B_1 = B + \tfrac{320}{T}
\end{aligned}$$

The term $T + 4 \times J$ assumes the adaptive playout algorithm as described in [221], which adjusts the playout delay as four times the variance in packet arrival time. Here, a, b, c, d are constants that can be derived using experiments. For G.711 codec M_e and M_d are negligible. Usually the constant factors b and d are small because most of the processing depends on the packet size, and hence bandwidth. Thus, CPU usage becomes $C(a.P + c)(B + \frac{320}{T})$ where a and c are some constants. For G.711 B is 64 kb/s, and if the packetization interval, T, is changed from 20 ms to 40 ms, the CPU usage reduces by a factor of 0.9, thus giving a better performance.

For CPU-intensive codecs such as GSM and G.723.1, the M_e and M_d components are significantly higher than the other components, thus the CPU usage roughly becomes $C(M_e.P + M_d)$. The packetization interval does not have much effect since the buffer copying and packet processing are masked by more CPU intensive encoding and decoding operations.

10.2.2 Performance Evaluation

To measure the performance of our conference server, sipconf, we built a load generator using our sippeer application in user agent mode and a built-in audio tool, rtpqos. The audio tool generates incremental patterns in the media packet instead of using a real encoded audio. For example, the first packet has payload with all bytes set to 1, second with all bytes set to 2, and so on. The pattern wraps to 0 on overflow. The conference server is configured to send the speaker's audio in the mixed stream back to the speaker for the measurement. On receiving the media packet, the audio tool compares it with the actual sent pattern and infers the quality of service (QoS) characteristics such as delay and loss.

We did not use the QoS statistics from RTCP because the RTP/RTCP session is established between the participant and the server, and does not indicate the QoS of the voice traffic end-to-end from the speaker to the listener. For example, the packet loss and delay introduced by the server is not captured in the RTCP statistics.

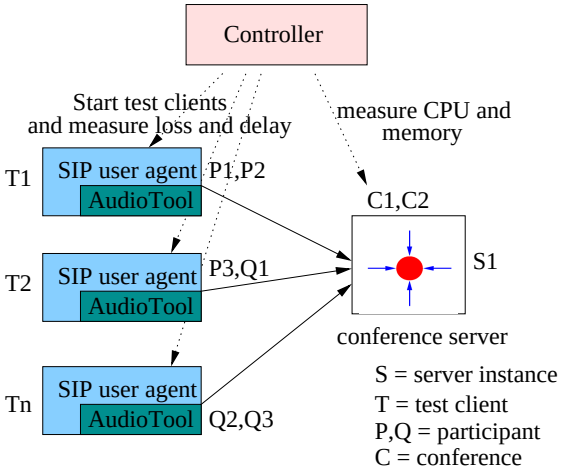


Figure 10.1: Physical configuration

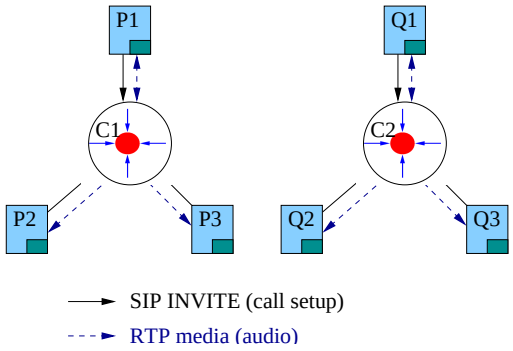


Figure 10.2: Logical configuration

An example test setup is shown in Fig. 10.1. The server, S_1, hosts two conferences, C_1 and C_2. The controller starts three test clients, T_n, with six participants, P_i and Q_j, to join the two conferences as shown in Fig. 10.2. In each conference, the first participant is the speaker and everyone is listening. The first participant gathers the QoS statistics for speaker-to-listener voice path. The controller collects the QoS statistics from the test clients as well as the periodic performance statistics (CPU and memory utilization) from the server host, and summarizes the resulting server performance.

The server and test clients were running on Pentium 4, 3 GHz CPU machines with 1 GB memory running Linux 2.6. All the hosts were connected to the same 100 Mb/s Ethernet switch. All the participants used G.711 μ-law audio codec with 20 ms packetization interval. The conference server also used 20 ms packetization interval and did not implement the optimization (see Section 9.4.1, p. 191) to reduce the number of encode operations. Each test host ran at most two test clients, and each test client emulated at most 40 participants. This ensured that the load on test machines was light (less than 30% CPU usage in our experiments) so that the test machine

never became a bottleneck.

We conducted two tests to measure the number of participants (P) in a single conference, and the number of four party conferences (C), respectively. Each of the tests was repeated three times and we did not see any significant deviation in performance between these repetitions.

In the test, we increased the participant count incrementally in steps. Each step ran for two minutes. For the first test, a single conference was used and the number of participants was increased by 40 in each step. In each step, the participants joined the conference at the rate of one participant per second. Thus, out of 120 s of a step, first 40 s indicates churn, and next 80 s indicates steady state, i.e., the number of participants remains constant. For the second test, each conference contained at most four participants and the total number of participants was increased by 40 in each step, i.e., the number of four-party conferences was increased by 10 in each step. Note that only one participant in each conference was an active sender of media packets.

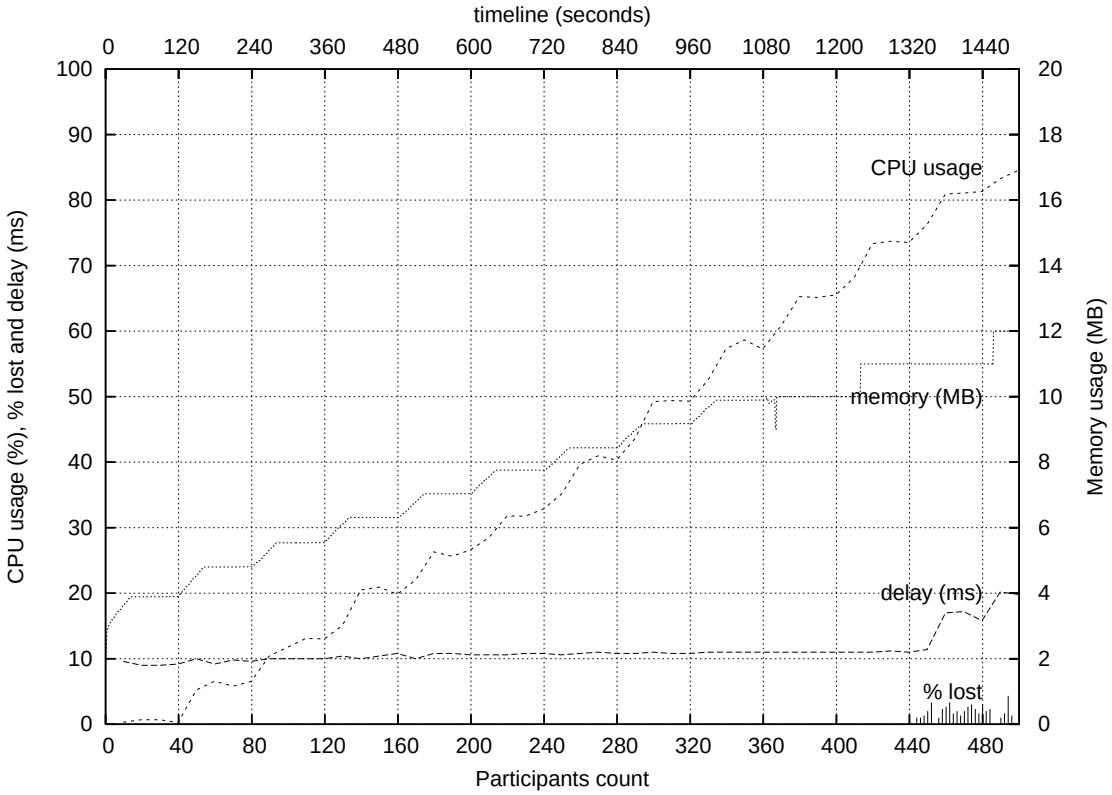


Figure 10.3: Server performance with increasing number of participants in a single conference

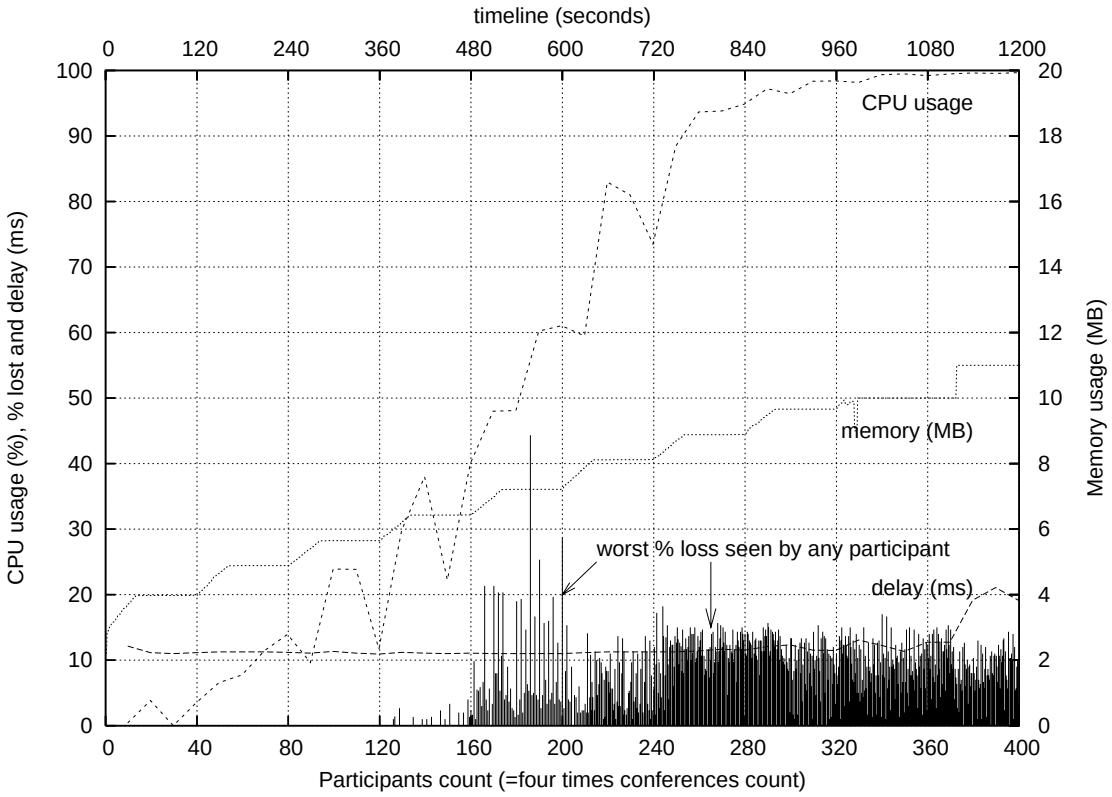


Figure 10.4: Server performance with increasing number of four-party conferences

CPU utilization

Fig. 10.3 and 10.4 present the results for the two tests, respectively. In particular, our hardware (3 GHz, Pentium 4) can support about 440 participants in a single conference and about 80 four-party conferences with G.711 μ-law audio while keeping the server CPU utilization below 80%. At 500 participants in a conference, the server CPU usage reaches close to 100%.

> The timeline (x-axis top) can be co-related to the participant count (x-axis bottom) as follows: 0-120 s corresponds to the first step with 40 participants, 120-240 s corresponds to 80 participants, and so on.

Memory utilization

Memory utilization at the server is about 11 MB for 500 participants. The memory is mostly allocated for call state with about 20 kB per call. The memory graph shows that in each step of

120 s, there is a linear increase in memory for first 40 s and then the memory remains constant for the duration of that step. This is because the fourty participants join during the first 40 s in each step. The later parts in the memory plot are flat because the Unix top command gives MB resolution instead of kB resolution after 9 MB. Thus it does not show the linear increase in each step after 9 MB. As seen here, memory is clearly not the bottleneck.

Audio delay and loss

The speaker-to-listener delay is less than 20 ms, which is the packetization interval. This excludes any wide-area network delay because the experiment is done on the same LAN. The delay is measured from the time the packet leaves the speaker application, to the time it is received by the listener application. Thus, it does not include the playout delay at the receiver application. In the absence of any network jitter on the same LAN, the adaptive playout algorithm at the mixer calculates the average playout delay as half the packetization interval, i.e., 10 ms. Fig. 10.5 shows the difference in delay experienced by the first and the last participants a single conference as the number of participants increase. Because the conference server sends the media packet from the first to the last participant in that order in every packetization interval, the delay for the first participant remains constant whereas that for the last participant gradually increases from 10 ms to 24 ms when the participant count increases from 40 to 440, as long as the CPU is below 80% utilized. When the CPU usage is close to 100% the delay as well as the packet loss suddenly increase. This is expected because as long as all the processing can be done within the packetization interval, the QoS doesn't get degraded. Once the processing takes more than 20 ms and spills over to the next interval, an additional processing keeps accumulating, resulting in packet loss of UDP media packets and additional delay in sending the packets to the participants. A packet loss of less than three packets per five seconds of logging interval is ignored by the listener when counting the number of packet losses.

Audio bandwidth

The bandwidth for G.711 codec payload is 64 kb/s. Adding the headers (RTP, UDP and IP) for every packet in 20 ms gives a packet size of 200 bytes, i.e., 80 kb/s above the link layer. Thus,

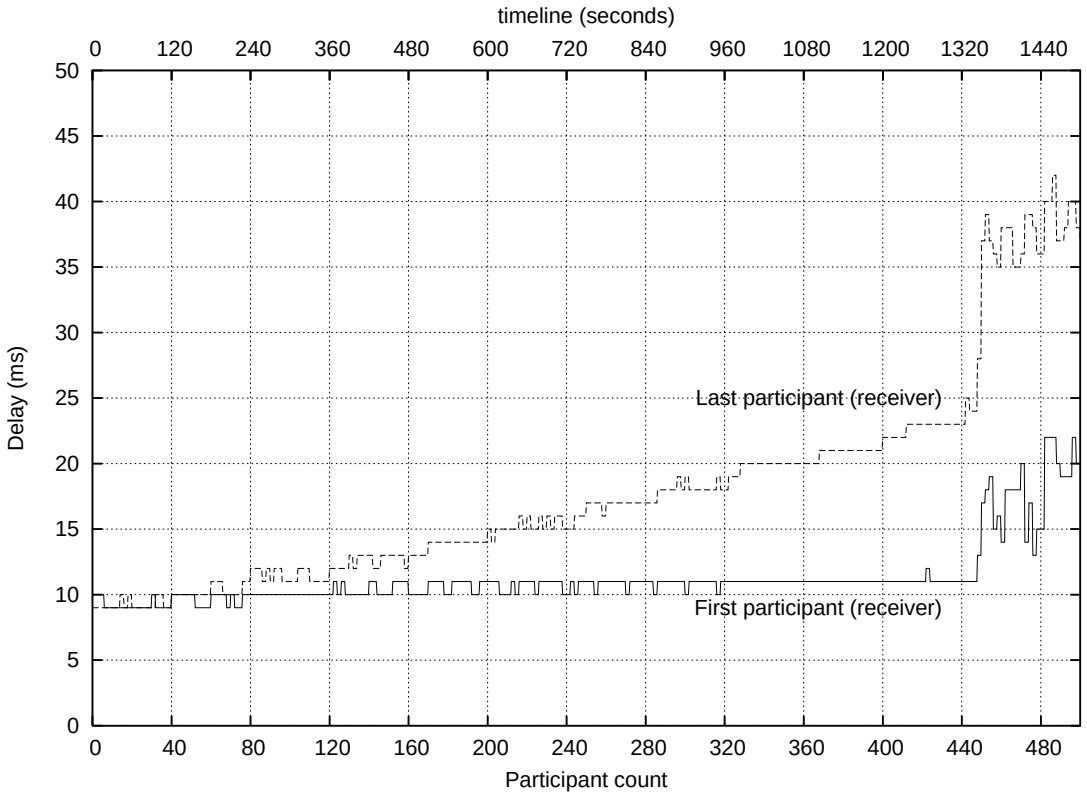


Figure 10.5: Speaker-to-listener delay for first and last participant to receive packets from the mixer

a server on a 100 Mb/s duplex Ethernet can support up to 1,250 outbound media streams to the participants, ignoring any control traffic such as SIP and RTCP. With Gigabit Ethernet, the server bandwidth is never a problem for a single conference server for medium scale conferences. In practice, the access link bandwidth of the server may impose a stricter limit on the conference capacity.

Sockets – open file descriptors

Currently, we use separate RTP and RTCP sockets for each participant. Thus, with P participants, the server opens $2P$ sockets for media. If the operating system imposes a limit of 1024 open file descriptors for ordinary users, the server running as ordinary user can support up to 512 participants. However, this can easily be fixed either by increasing the open file descriptors limit or by modifying the server to reuse sockets among multiple participants.

Effect of packetization interval

A 20 ms interval in the conference server seems to be the most interoperable. For example, Robust Audio Tool (RAT) cannot correctly handle intervals that are not multiples of 20 ms. Moreover, a small packetization interval also guarantees that the maximum delay incurred by the server processing is bounded by this value under normal load.

On the other hand, using a larger packetization interval such as 40 ms causes less overhead in terms of header bandwidth and buffer copying at the server. However, this can increase the delay to at most 40 ms at the server with heavy load. Hence, we recommend using 20 ms if the load is well below the server capacity.

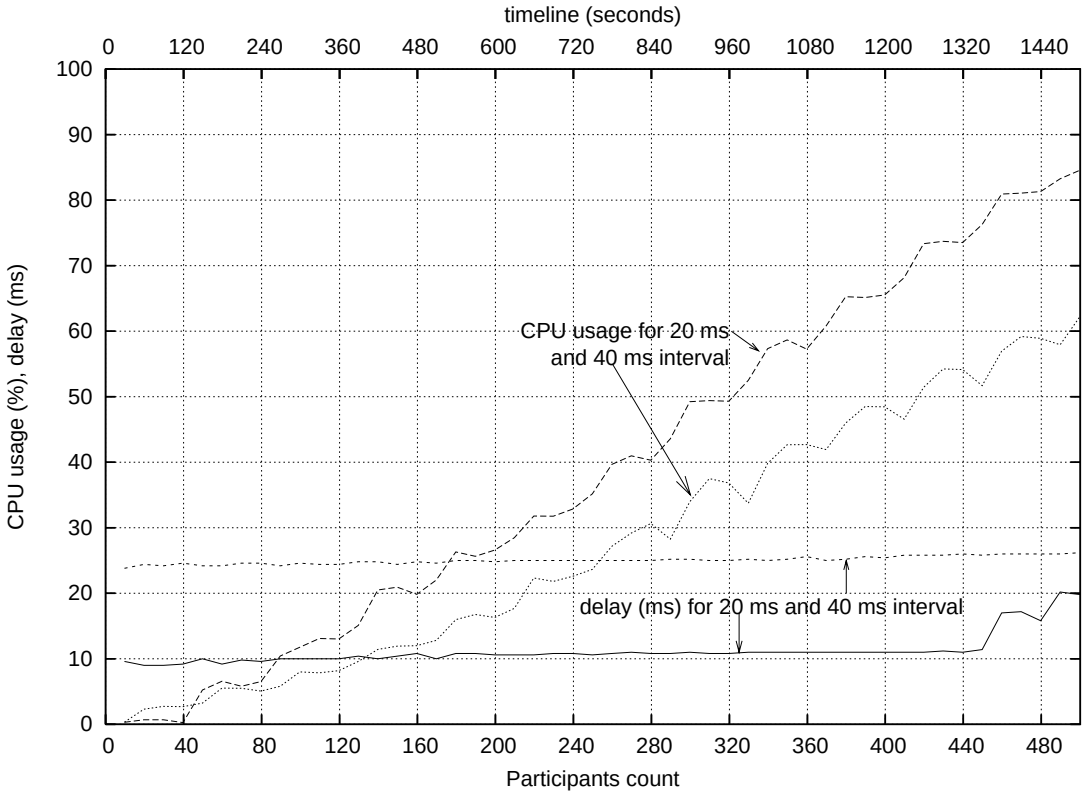


Figure 10.6: Effect of packetization interval on performance

Fig. 10.6 compares the performance using the packetization intervals of 20 and 40 ms. As shown, at 50% CPU utilization we can support about 320 and 400 participants with 20 and 40 ms intervals, respectively. Thus, 40 ms interval improves the performance by a factor of approxi-

mately 1.25 over 20 ms interval. Due to the smaller packet size, the header overhead in 20 ms interval is larger. The IP layer bandwidth required for 20 ms is $\frac{80}{72} = 1.11$ times that for 40 ms. If we include the Ethernet packet overhead of 18 bytes per packet, the factor becomes $\frac{87}{75} = 1.16$. We conclude that the performance benefit in using the larger packetization interval is due to the lower packet size, and hence lower buffer copying and processing.

We also observe an increase in the speaker-to-listener delay from about 12 ms to 24 ms when changing the interval from 20 ms to 40 ms. This is expected, since the average delay caused by the server is half of the packetization interval if there is no jitter. The memory utilization is independent of the packetization interval, but depends on the number of participants.

Extrapolating the performance for 40 ms interval, the server can support about 720 participants in a single conference. However, the current implementation needs to be modified to share socket connections across different sessions to support these many participants, or the operating system's default limit on per-process open file descriptor count needs to be increased to about 1500.

Performance on Sun SPARC vs Pentium

Fig. 10.7 shows the conference server performance on a Sun SPARC Ultra 5/10 with 256 MB memory and a 360 MHz CPU, using a 20 ms packetization interval. One difference with the earlier measurements on Pentium running Linux, is that on Sun machine the quality of service (delay and loss) degrades as the CPU utilization becomes more than 50%. The server can support about 60 participants in a conference without degrading the audio quality.

Compared to the capacity of 480 participants on Pentium 3 GHz CPU, which roughly translates to 6.25 MHz of CPU cycles per participant (or MHz/participant), the Sun machine gives similar performance of 6 MHz/participant. Thus, there is not much difference between Sun SPARC and Pentium 4 in our conference server test. Most of the processing at the server involves buffer copying and network I/O. With 40 ms interval, the performance improves to about 4.2 MHz/participant on Pentium.

Our earlier measurement of the conference server in 2001 [34] on the same Sun hardware gave the server capacity of about 80 participants in a single conference with one active speaker, or

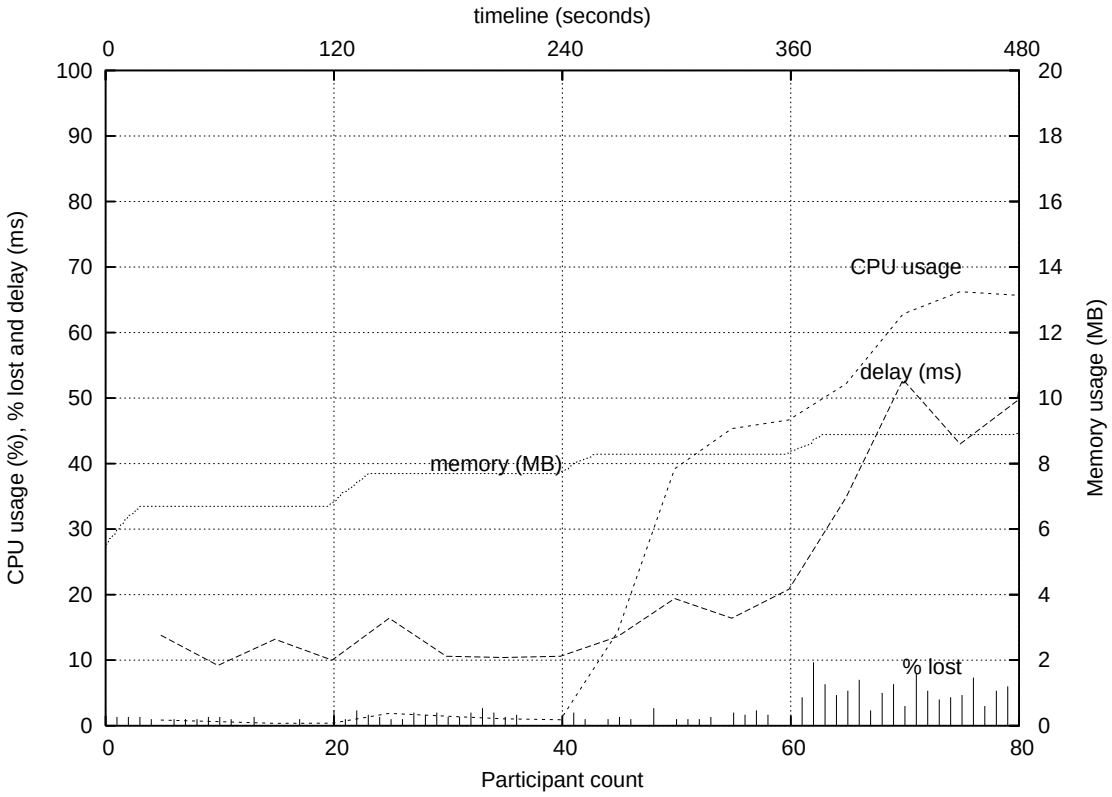


Figure 10.7: Server performance on 360 MHz Sun/SPARC as the number of participants in a single conference increases

about 15 three-party conferences with all participants as active speakers supporting G.711 μ-law audio, without degrading the audio quality. The server sent packets at 40 ms intervals whereas the participants sent at 20 ms intervals. Thus, the performance was roughly 4.5 MHz/participant, which is slightly worse than the current performance of 4.2 MHz/participant on a Pentium processor with 40 ms interval. Even at that time, the CPU was the primary bottleneck. Since then the architecture of our conference server has improved from a thread-per-participant and a thread-per-conference to a single-threaded event-based architecture.

Performance of various audio codecs

The codec performance is an important factor in the overall server performance. For example, if the encoder takes M_e=50 μs to encode T=20 ms worth audio to each participant, then the server cannot encode more than 400 streams. Thus, the encoder and decoder performance can impose

Table 10.2: Comparison of various audio codecs: time taken for encoding and decoding of 20 ms of audio on Pentium 4, 3 GHz CPU running Linux 2.6.9 in our test-bed: E means encoder, and D means decoder. G.711 and G.722 are ITU-T's, and DVI is Intel/IMA's

Codec	bitrate	3 GHz Pentium 4		360 MHz Sun/Sparc		900 MHz Sun/Sparc	
	(kb/s)	E (μs)	D (μs)	E (μs)	D (μs)	E (μs)	D (μs)
G.711 μ-law	64	5.38	1.63	51.21	13.86	20.18	6.22
G.711 A-law	64	5.47	1.77	54.03	14.48	22.56	9.24
DVI (ADPCM)	32	5.08	2.76	50.24	19.66	17.36	10.09
G.722 (wide-band)	64	69.51	49.69	1005.69	613.66	382.29	233.23
GSM 06.10	13	73.98	29.13	718.17	488.49	327.80	206.16

an upper bound on P_{max} and C_{max}.

$$P_{max} \leq \frac{T}{M_e}$$

$$C_{max} \leq \frac{T}{3.M_e + M_d}$$

Table 10.2 shows the time taken for encoding and decoding of 20 ms of audio on different platforms: 3 GHz Pentium 4, 360 MHz UltraSparc, and 900 MHz UltraSparc. Fig. 10.8 compares the relative performance in terms on kilo-cycles (1024 cycles) of CPU on these platforms for G.711 μ-law, G.722 (wide-band) and GSM codecs (low bitrate). The GSM and G.722 codecs are many times more expensive than the G.711 codec.

Effect of multi-processor hardware

Since the current implementation uses a single thread event-based architecture, it does not take advantage of a multi-CPU hardware. This can be easily enhanced to a thread-pool implementation similar to the SIP proxy server architecture (Section 3.6, p. 55). With an N-processor machine, the thread-pool can have N threads, where each thread can process about $\frac{1}{N}$ of the participants. However, for a single conference splitting the participants among the processors does not help. To reduce the implementation complexity, on every packetization interval only a single thread sends packets to all the participants in the conference. Alternatively, another buffer can be used to synchronize the mixing operation with the send operation, thus utilizing multi-processor architecture. On the other hand, for video packet forwarding (i.e., no mixing step) multiple processors

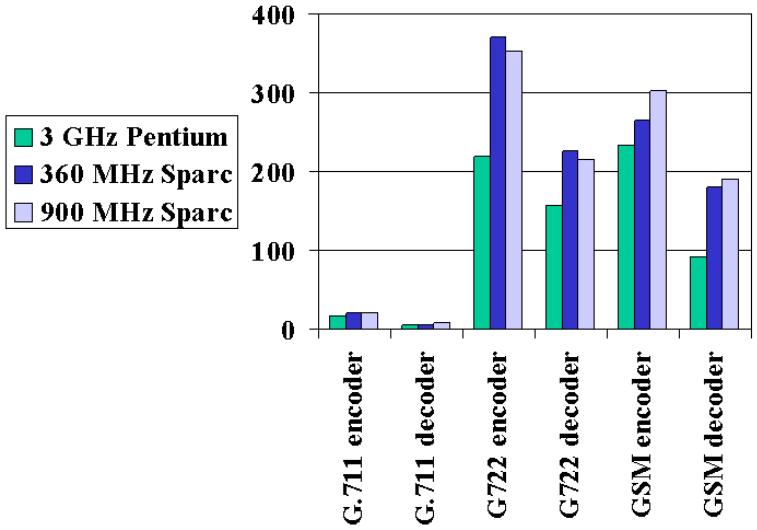


Figure 10.8: Relative audio codec performance in terms of CPU speed on various platforms for processing 20 ms audio. The y-axis provides numbers in Kilo cycles (1024 cycles). For example, GSM encoder took about 300 Kilo cycles on a 900 MHz Sparc, which means $\frac{300\times1024}{900\times1048576}s \approx 325\mu s$.

can help a single large conference since the receiving thread can distribute the packets to all the participants without waiting for it to be mixed.

10.2.3 Cascaded Conference Servers

We have shown that a single conference server can support medium scale conferences with up to a few hundred simultaneous participants. For large conferences, a cascaded server architecture can be deployed. This section describes and evaluates our cascaded server architecture.

For large conferences, it is possible to create a cascaded conference server architecture, where each server appears as a participant to the server at the aggregation level above it (Fig 10.9). Such a tree adds packetization and playout delay, but can approximate the bandwidth scaling benefits of network-layer multicast if participants select the closest server. Since it is common that corporate conferences consist of a large number of participants spread across a relatively small number of facilities, having a server in each LAN is likely to be a common mode of operation.

There are two approaches to cascading the conference servers: tree-based and full mesh,

which we describe below.

Tree-based

In the simple tree based approach (Fig. 10.9), all the conference servers, S_n, are connected in a tree topology. Each server can have a bunch of participants limited by the single server capacity. Each server treats the other server that it is connected to as another participant in the conference. The example shows three servers and nine participants. Server S_1 views S_2 and S_3 as additional participants and uses the decode-mix-encode logic to send packets to each participant. Thus, each link carries the audio packet containing mixed audio from all the participants on the originating side of the link in the tree as shown in Fig 10.9.

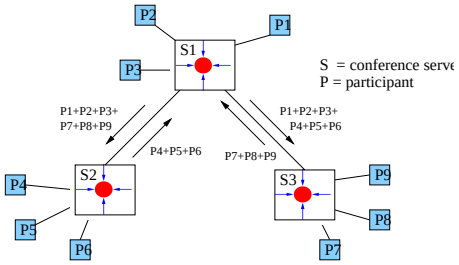


Figure 10.9: Tree-based cascaded servers

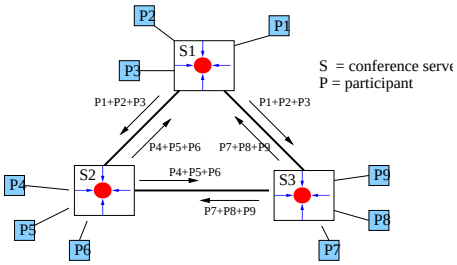


Figure 10.10: Full mesh cascaded servers

Each stage in the conference adds additional delay. For example, audio from participant P_4 to P_9 goes through three servers, S_2, S_1 and S_3. Assuming each server operates at 20 ms packetization interval, the worst server processing delay will be 60 ms. The delay is in addition to the transport delay between the four application-level hops in the path from P_4 to P_9. Hence, a tree with a diameter larger than two is not desirable if the conference requires low delay.

Secondly, each stage in the cascaded architecture causes transcoding, and thus degrades the perceived audio quality. For high quality codecs such as G.711 with Mean Opinion Score (MOS) of 4.5, two steps of transcoding may be acceptable. However, for codecs such as GSM with MOS of 3.5, the two steps is not acceptable. This is because a MOS value smaller than 3 is not suitable for smooth audio communication, requiring considerable effort to comprehend the spoken audio.

If each server supports N participants, then the two stage cascaded tree contains $1 + N$ conference servers, and can support $N \cdot (N - 1)$=$O(N^2)$ participants. On our hardware, this translates to 0.23 million participants with 481 conference servers.

Most large scale conferences are lectures with communication from a single speaker or a small number of panel speakers to a large audience. In this case the delay is not an issue, the tree can have any diameter, and hence the cascaded architecture can scale to any population size.

Full-mesh

To avoid the large delay of four application hops in the tree-based architecture, the conference servers can form a full mesh network for media distribution as shown in Fig. 10.10. In this case, the server treats the attached server different from the attached participant. The media sent to the participant follows the normal decode-mix-encode cycle. However, the media sent to the attached server contains the audio mixed from the participants that are directly attached to this server only, instead of including the media from other servers. For example, when S_1 sends media to S_2 it does not include the media from S_3 in Fig. 10.10.

Since the speaker-to-listener path contains at most two servers and three application-level hops, the delay due to server processing is at most 40 ms, assuming 20 ms packetization interval at each server.

If each server can support N participants, then this full-mesh architecture can contain $N/2$ servers, each attached to $N/2$ participants, giving a total user population of $\frac{N^2}{4}$ for a single conference. On our hardware, this is about 58 thousand participants. Thus, compared to the tree-based architecture, the full-mesh architecture gives four times lower capacity, between $\frac{2}{3}$ (due to number of servers) and $\frac{3}{4}$ (due to number of application level hops) times lower delay and uses half the number of servers.

> The optimal number of servers, $N/2$, is derived as follows. With x servers connected in full mesh such that each server is connected to $x-1$ other servers, each server can have $N - (x - 1)$ participants due to its capacity of N participants. Thus, the total number of participants is $x(N - x + 1)$. This is maximum when integer $x \approx \frac{N}{2}$.

Note that this analysis is only theoretical. In a real deployment other factors such as link bandwidth and heterogeneity of servers will affect the total performance of the cluster.

Performance evaluation

To verify the scaling property of the cascaded architecture, we modified out test setup as follows. Instead of one conference server, two servers are started. Then, the two servers, S_1 and S_2, are connected in cascaded mode for conference C_1 by making an outbound call from S_2 to S_1. In particular, a SIP REFER message is sent to S_2 with Refer-To header containing the conference URI on S_1. Thus, S_1 treats S_2 as a dial-in participant in conference C_1, whereas S_2 treats S_1 as a dial-out participant in its conference C_1. This two-server configuration is the common subset for both tree-based and full-mesh cascaded architecture.

The earlier loopback mode in the server is not enough in this configuration. This is because we want to measure the speaker-to-listener delay that goes through the two servers, instead of looping back from the first server. The audio tool, rtpqos, is modified to work with our SIP user agent such that two instances of the audio tools from the two independent calls can be associated as speaker and listener for measurements.

The results of our experiment confirms linear increase in capacity. In particular, for the two servers we can support close to 1000 participants, which doubles the single server capacity, without degrading the audio quality. The results are summarized in Fig. 10.11.

> We have not measured the packet loss in this case, because the current method of matching the received packet with the sent packet to detect the loss does not work for multiple transcodings. In particular, when the encoded audio is decoded and re-encoded at the first server, it causes some false negatives, i.e., samples that are mismatched even though there is no packet loss. When the transcoding is done again at the second server, the number of false negatives further multiplies, thus causing incorrect inference of packet loss.

The SIP REFER message can be used to connect any number of servers in the tree-based cascaded architecture without modifying the current implementation of the server. On the other hand,

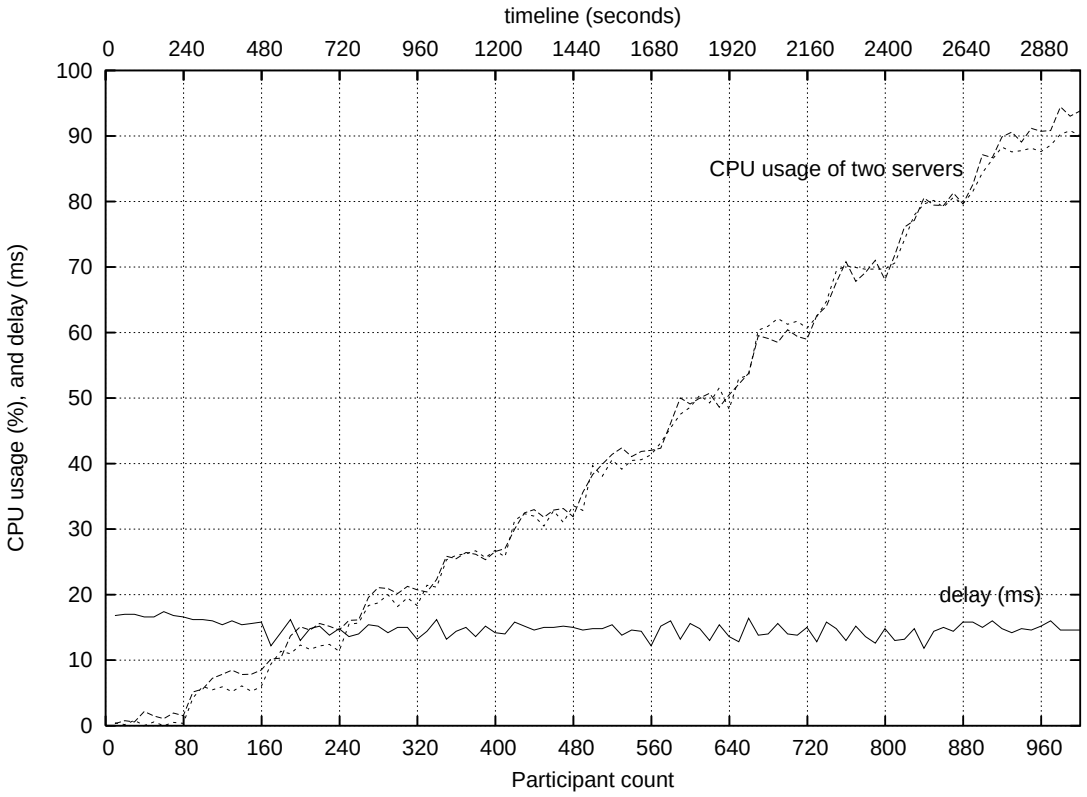


Figure 10.11: Performance of two cascaded conference servers for a single conference

the full-mesh architecture requires modification in the current implementation to treat participants differently from the other conference servers, as described earlier.

10.2.4 Distributing Conferences

To scale to a large number of (small) conferences instead of a single conference with large population, a SIP proxy server can act as a load distribution system and direct incoming requests for new conferences to different servers. Alternatively, the conference server itself can redirect a request to an alternate server. We have implemented a time-based load distribution for pre-scheduled conferences, where an incoming conference request is redirected to a relatively less loaded conference server based on the scheduled conference times and expected number of participants of the conferences.

10.2.5 Handling Overload: Graceful Denial and Admission Control

As we described earlier, when the CPU utilization goes above 80%, the audio quality degrades, i.e., delay and packet loss increase. This affects the existing calls as well as new calls in the current implementation. This is clearly undesirable for the existing calls. Thus, if the server is overloaded, it should reject any new participant or redirect her to the cascaded server.

Since the system load depends on the codec used, the server may also try to restrict the codecs for the new participants or change the codecs of the old participants using mid-call session update via SIP re-INVITE.

A server should detect the bottlenecks in real time. Identifying bottlenecks of CPU and memory are easy. For bandwidth the server can rely on packet loss statistics using RTCP. Usually if participant's access is congested only that participant will see losses. If server's access is congested, many participants will see losses. Since the server sequentially sends RTP to all participants every interval, say 20 ms, the first few participants do not see any loss but subsequent ones will, because of the bursty traffic. To prevent this the server can spread the packets sent throughout the 20 ms interval. This increases the delay for some participants but prevents loss. Randomizing the participant order while sending is not useful as it will increase the jitter, thus increasing the playout buffer and hence the delay. In large group conferences such as lectures, some delay doesn't matter but packet loss is undesirable.

If all else fails, the server should give priority to the existing participants, and gracefully reduce the audio quality of the new participants.

10.3 Reliability

When the conference server fails for some reason during a conference, all the participants stop receiving media. A secondary server can be used to restart the conference. However, the conference state such as the list of participant addresses and their session description is lost. There are two alternatives: the participants reconnect to the secondary server after detecting failure, and secondary server uses the shared state from the failed server to reconnect to all the participants. We explore some of these techniques to improve the reliability of the conference server.

The techniques can be classified into two types: reactive and proactive. The reactive mechanisms do failover after the failure is detected, whereas the proactive mechanisms provide redundancy to continue the conference even when the conference server fails.

10.3.1 Reactive Failover

Existing techniques for web server failover can also be applied to conference server failover where the primary and backup servers share conference state. For example, the backup server can take over the IP address of the primary server if the primary server fails, without affecting the participants. This scheme works only when the primary and backup servers are in the same subnet. Alternatively, IPv6 mobility can be used in which the conference server represents a mobile node with a fixed home address which gets bound to the particular foreign address of the primary or backup server. Multi-homing further enhances the reliability in case of network failures.

SIP allows dynamic updates in a session. Thus, if the primary server fails, the backup server can send a re-INVITE to all the participants and update their sessions to the backup server. This means the primary and backup servers may be geographically distributed and get can located using DNS.

However, the primary and backup servers still need to share state regarding the ongoing conferences and participants. At the minimum, this state consists of the SIP call (dialog) state between the server and every participant. The re-INVITE message sent from the backup server to the participant uses the existing call dialog established between the primary server and the participant. The call state can be stored in the SQL database shared between the primary and the secondary server, e.g., using MySQL replication described in Section 3.3.6.

Connecting to the participants from the backup server does not work if the participant user agent does not support mid-session updates in the media transport addresses. In that case the participant can dial-in again to the conference, which is now hosted by the backup server.

One problem with reactive failover approach is that when a failure happens, there is some delay before the backup server takes over. The delay depends on a number of factors such as keep-alive interval and database replication latency. An alternate approach is to have multiple

redundant conferences at the same time as described next.

10.3.2 Proactive Redundancy

In this approach, the participants are connected to two conference servers via two independent media paths when they join the conference. The participants send the same media streams to both conference servers, and receive the same media stream from both the servers, under normal operations. The player plays only the first stream, as long as the primary server is active. If the first stream stops working, the user agent switches to the second stream.

Having media traverse multiple paths also improves network redundancy if the two servers are distributed in the network causing independent network paths from the participant to the two servers.

Alternatively, IP anycast [246] can be used to send media packet to any of the server in the cascaded architecture. This mechanism does not require two streams, but dynamically picks the appropriate server.

There are two approaches to implement this: client-based and server-based. In the client-based approach the client connects to the two conference servers, say using two lines on his phone. In the server-based approach, when a participant joins the primary server, the primary server informs the secondary server, which in turn calls back the participant's phone. Thus, the participant is in two active calls, one with each server. The second approach doesn't work if the participant needs to be billed or is connected via some complex dialog interaction through a gateway which does not allow call in the reverse direction (e.g., calling cards).

The two servers are cascaded. The second call is usually put on hold either by the participant or the server to save bandwidth. When the primary server fails, all the participants are still connected to the secondary server and can hear each other. This mechanism does not work on most IP phones without manual switching, e.g., the user has to manually unmute the secondary call on failure. This mechanism assumes enough capacity in terms of bandwidth and CPU to support twice the number of calls.

If the call state is easy to create and does not cost anything, then the proactive approach is better than reactive approach because the failover time is less. In the reactive failover case,

the secondary server needs to detect failure and send new call invitations to all the participants, which can take time for large number of participants. However, for small and medium sized conferences, it is desirable to do reactive failover to avoid the complexity and overhead of the proactive mechanism.

10.4 Conclusions

We have shown that it is possible to build medium scale conference servers in software running on commodity PCs that can support a few hundred participants. The performance can be further improved to large scale conferences with tens of thousands of participants using a cascaded server architecture. Furthermore, if delay is not a problem such as for large scale lectures or panel discussions, then the tree-based cascaded architecture can scale to any user population.

We have performed our evaluation with only G.711 audio codecs. Most user agents support G.711 to allow interoperation with PSTN, for example. The performance data will be different for other codecs. Moreover, advanced audio features such as echo cancellation and automatic gain control in the server will further affect the performance. In a complex heterogeneous conferencing environment with various participants using different codecs, the limiting factors such as CPU, memory and bandwidth can be calculated and the bottlenecks identified using the analysis shown in this chapter.

A number of features can be added to our current implementation. For example, the server currently has a limited dial-out facility because the outbound calls to the participants are not authenticated. Conferences could also be bounded in duration. However, since the resource consumption of inactive conferences is very small as long as media streams are muted, it is quite feasible to set up permanent conferences in work groups, for hoot-and-holler applications. The seamless transition from centralized conferences to full-mesh and multicast conferences, as well as hybrid solutions, needs to be supported in the conference server.

As shown in Chapter 9, we can use our SIP-H.323 gateway and commercial SIP-PSTN gateway to provide a multi-protocol distributed conferencing server that can be contacted from any of the SIP, H.323 or PSTN networks. To integrate PSTN users, we have implemented interactive voice response, e.g., to prompt for conference and participant access codes. The server can

further be integrated with a text-to-speech and speech recognition system to allow text-only participants in an audio session. We can use an external streaming client for recording and playback of media (Section 9.5.3). An automatic transcript can be created using speech-to-text. Speaker indication can be done via instant messages.

Chapter 11

Interworking Between SIP/SDP and H.323

There are currently two standards for signaling and control of Internet telephone calls, namely ITU-T Recommendation H.323 [37, 99] and the IETF Session Initiation Protocol (SIP) [3, 4]. We describe how a signaling gateway can allow SIP user agents to call H.323 terminals and vice versa. Our solution addresses user registration, call sequence mapping and session description. We also describe and compare various approaches for multi-party conferencing and call transfer.

Both SIP and H.323 run over IP (Internet Protocol) and use RTP (Real time Transport Protocol [1, 2]) for transferring real-time audio/video data, reducing the task of interworking between these protocols to merely translating the signaling protocols and session description. Since no media data needs to be translated, a single gateway can likely serve thousands of end systems.

Interworking between SIP and H.323 requires transparent support of signaling and session descriptions between the SIP and H.323 entities. We call the server providing this translation a SIP-H.323 *interworking function* (IWF). Note that the earlier version of our paper [42] called this a signaling gateway (SGW). We refer to the set of terminals speaking H.323 and SIP as the H.323 and SIP *clouds* (or *networks*), respectively, even though they are likely to be intermingled on the same IP network. We use the term *native network* to refer to the network used by a particular terminal, while the *foreign network* is the network whose access is mediated by the IWF. For an

H.323 terminal, a SIP terminal is in a foreign network.

When addressing a terminal using another signaling protocol, there are two approaches. First, the user can explicitly identify the protocol as part of the address, for example, by inventing some form of H.323 URL [247] such as h323:alice@columbia.edu. If, for example, an H.323 URL is used by a SIP terminal, it would then be the responsibility of the SIP terminal to find the appropriate IWF.

Alternatively, a terminal using a particular signaling protocol sees all other terminals as being native, and does not know or care that a particular address refers to a terminal in the foreign network. Indeed, an address could well change between being native and foreign, depending on what equipment the owner of the address happens to be using. This approach is preferable, but requires that user registrations are exported into the foreign network. Depending on the type of information sharing between H.323 or SIP elements and the IWF, different architectures are possible to provide the transparent address resolution and call establishment, as we discuss below.

This chapter describes the details of interworking between SIP version 2.0 [3] and H.323 version 2.0 [37], including the translation between H.245 and SDP. However, since an H.323v2 terminal may or may not support FastConnect (Section 11.4.1), solutions without using this feature are also detailed. Only a simple call scenario is presented. We list general requirements for such a translation and a solution which meets those requirements. We describe the call setup via message flows and pseudo code. Issues related to a new enhanced version of SDP (Session Description Protocol [46]) is kept open while discussing the solution, so in the future any change in SDP can be handled easily.

11.1 Background and Requirements

11.1.1 Protocol Overview

H.323 includes various other subprotocols: H.225.0 [248] for connection setup and media transport (RTP), resource access and address translation, H.245 [249] for call control and capability negotiation, H.332 [250] for large conferences, H.235 [251] for security, H.246 [252] for interoperability with the PSTN, H.450 [253, 254] for supplementary services like call transfer.

In H.323, a simple call is established as follows. If a user (say Alice) wants to talk to another user (Bob), Alice first sends an admission request to its gatekeeper. The *gatekeeper* acts as a management entity in H.323, which grants access to resources, controls bandwidth and maps user names to IP addresses, among other things. The gatekeeper finds out the IP addresses at which Bob can be reached and informs Alice. After that, Alice establishes a TCP connection to the IP address of Bob. This is followed by a ISDN-like *call signaling* procedure. Alice sends a Q.931 [255] SETUP message and Bob responds with a Q.931 CONNECT message. Once the first stage of Q.931 signaling is complete, H.245 takes over. H.245 messages are used to negotiate terminal capabilities, i.e., the support for various audio and video algorithms. The H.245 OpenLogicalChannel procedure is used for opening different unidirectional media channels. A *media channel* is defined as a pair of UDP channels, one for RTP and the other for RTCP. Audio and video packets are encapsulated in RTP and sent from one end system to the other. Depending on the version of H.323, Q.931 and H.245 steps can be combined in various ways.

The message flow for normal call connect in H.323 between two terminals registered with different gatekeepers is shown in Fig. 11.1. More details of individual messages and the information conveyed are described later as needed.

SIP sets up calls with an INVITE message and a response from the called party. Both INVITE and the response contain a *session description* indicating terminal capabilities, typically, but not necessarily, encoded using SDP. Proxy and redirect servers are responsible for translating between user names and the called party's IP address.

An endpoint is either a SIP user agent or H.323 terminal. We use the term *signaling* to mean the protocols specified by Q.931 [255], H.245 [249] or SIP [3]. Data traffic refers to RTP and RTCP encapsulated multimedia data.

11.1.2 Translation Requirements

Basic requirements for any SIP-H.323 IWF are summarized below:

Protocol compliance: The IWF should use the components of H.323 and SIP. The IWF should handle all mandatory features of H.323 as well as SIP. Common call scenarios should be simple to implement.

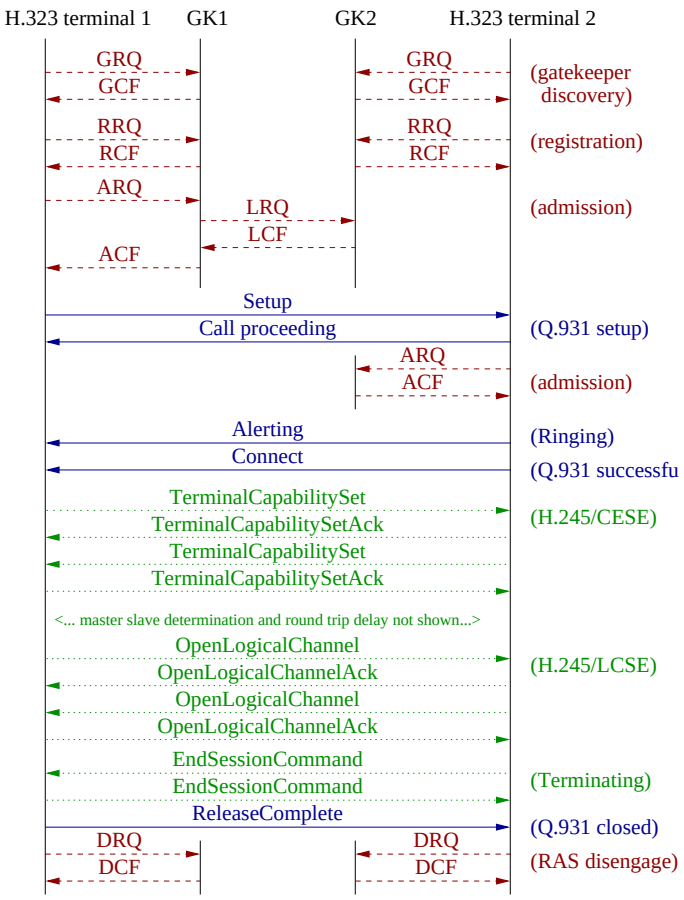


Figure 11.1: H.323 call without fast-connect

User registration: The IWF should use the user registration in both the H.323 and SIP clouds to resolve the user name (alias or URL) to an IP address. In other words, it should provide a framework in which the user may dial any address without actually knowing whether it belongs to the H.323 or the SIP cloud.

Mapping between H.245 and SDP: The IWF should be able to map all the mandatory H.245 messages to appropriate SDP messages and vice-versa, without the endpoint being aware that such conversion is taking place. Other optional features of H.245 and SDP should be mapped as much as possible to facilitate maximum interworking between the two clouds.

Direct data traffic between the endpoints: Where possible, the IWF should route RTP and RTCP traffic directly between the endpoints involved in the conference without going through the

IWF. This reduces the delay for media packets and helps building scalable IWFs.

Transparent support for audio and video algorithms: The IWF should provide transparent support for audio and video algorithms, i.e., the IWF should not restrict the capabilities of the endpoints in terms of audio/video algorithms supported.

Call sequence mapping: The IWF should map the message sequence between H.323 and SIP in such a way that every important decision (accept or reject a call, choose an algorithm for a logical channel, and so on) is taken by the endpoints involved in the conference and not by the IWF itself.

We assume throughout most of this chapter that the session description given by a SIP endpoint refers to both the transmit and the receive capabilities of the SIP endpoint. This may not be true in a particular application. If that is the case then the SIP endpoint is expected to give that information in SDP using recvonly or sendonly media attributes.

The analysis of SIP-H.323 interworking can be split into simple call setup, mapping addresses, finding a subset of capabilities described by H.245 and SDP, conferencing, call services, security and authentication. Section 11.4 describes call setup and teardown; while Section 11.3 describes address mapping and section 11.5 the capabilities calculation. Security and authentication is not covered in this thesis.

Call Setup Translation

Three pieces of information are needed for establishing a call between two endpoints, namely the signaling destination address, local and remote media capabilities, and local and remote media transport addresses at which the endpoint can receive the media packets. In H.323, this information is spread over different stages of the call setup, while SIP conveys it in an INVITE message and its response.

Translating a SIP call to an H.323 call is straightforward. The IWF gets all three pieces of information in the SIP INVITE message and can split them across multiple stages of the H.323 call establishment. However, in the reverse direction, from H.323 to SIP, the different stages of H.323 call establishment have to be merged into a single SIP INVITE message. We describe and

compare various approaches in Section 11.4. The H.323v2 (version 2.0) Fast Connect procedure is a step towards simplifying the multi-stage signaling of H.323. However, it is optional and an H.323v2 entity is required to support the traditional multi-stage signaling. Thus, we describe call setup both with and without Fast Connect.

User Registration

SIP-H.323 translation also has to solve the user registration problem. User registration involves mapping of user names, phone numbers or some other human-understandable identifier such as email addresses to network addresses. By allowing users to be reached by location-independent identifiers, user registration provides personal mobility. For instance, a call destined at *sip:bob@mydomain.com* reaches user Bob no matter what IP address he might currently be using.

In SIP, proxy and redirect servers access a location server, often a registrar that receives user registration information. A server at *mydomain.com* will map all the addresses of the form *sip:xyz@mydomain.com* to the appropriate IP addresses, depending on where *xyz* is currently logged in. In H.323, the same functionality is performed by the H.323 gatekeeper. The IWF should use the user registration information available in both networks to resolve a user name to an IP address. The IWF can contain a SIP registrar server, an H.323 gatekeeper or neither, as discussed in Section 11.2.

Session Description

An IWF also must map session descriptions between the two signaling protocols. H.323 uses H.245 for session description. H.245 can negotiate media capabilities, provide conference floor control, and establish and tear down media channels. In H.245, media capabilities are described as a set of capability descriptors, listed in decreasing order of preference. A *capability descriptor*, also called a simultaneous capability set, is a set of alternative capability sets, where each alternative capability set contains a list of algorithms, only one of which can be used at any given time. For instance, a capability descriptor $\{[a_1, a_2][v_1, v_2][d_1]\}$ has three alternative capability sets: $[a_1, a_2]$, $[v_1, v_2]$, and $[d_1]$. The curly bracket indicates conjunction, i.e., $\{A\,B\}$ means A

and B, and the square barcket indicates disjunction, i.e., $[A, B]$ implies A or B. Thus the example capability descriptor above indicates that the terminal can support audio, video and data simultaneously. Audio can use either codec a_1 or a_2, video codec v_1 or v_2, and data format d_1.

SIP can, in principle, use any session description format. In practice, however, only SDP is used. SDP lists media types and the supported encodings for each. Unlike H.245, SDP cannot express cross-media or inter-media constraints, however. For example, SDP cannot indicate that for a particular media type, the other side can only choose subset A or subset B of the listed codecs, but not codecs from both subsets. Similarly, SDP cannot express that certain audio codecs can only be used in conjunction with certain video codecs.

Thus, a SIP media capability can be easily described in H.245, however the reverse is more complicated. One approach is to carry multiple SDP messages in the message body of SIP INVITE requests and responses, using the "multipart" content type. Each SDP message then represents one capability descriptor of the H.245 capability set. One problem with this is that it does not interoperate with many existing SIP user agents that do not understand multipart body. In Section 11.4 we describe how sending multiple SDP messages can be avoided.

Multi-party Conferencing

Ad-hoc conferencing among SIP and H.323 end systems is not possible without modifying one or both of these protocols. Ad hoc conferencing is defined as the one in which the participants do not know in advance whether the call will be point-to-point (two-party) or multi-party. The participants can switch from a point-to-point call to a multi-party conference or vice-versa during the call. It is possible for the participants to invite a third party in the conference or for the third party to join the conference. Both SIP and H.323 individually support ad hoc conferencing. In SIP, conference topology can be a full mesh with every participants having a signaling relationship with every other participant or a centralized bridged conference (star topology) in which every participant has a signaling relationship with the central conference bridge. It is possible to switch from a mesh to a bridged conference. In H.323, conferences are managed by central entity called a *Multipoint Controller* (MC). An MC can be part of an H.323 terminal, gateway, gatekeeper, or MCU (Multipoint Control Unit). H.323 conferences have inherently a star topology with every

participant having an H.245 control channel with the MC. The MC is responsible for deciding the common media capabilities for the conference, conference floor control, and other conferencing functions. All the participants are required to obey the media capabilities given by the MC. Because of the difference in the topology of the conferences in the SIP and H.323 (star like in H.323 and full mesh or star like in SIP), the transparent support of multi-party conferencing cannot be achieved without modifying the protocols. However, with some simplifying assumptions, basic conferences can be set up, as described in Section 11.6.

Call Services

Advanced call services like call forwarding and call transfer are supported by both SIP and H.323. H.323 uses H.450.x for these supplementary services. SIP has support for call hold, blind transfer, operator assisted transfer, call forwarding, call park and directed call pickup [256]. These services are not yet widely deployed, so that translation is not critical at this moment. Section 11.6 describes some of the related issues.

Security and Quality of Service

Other problems in SIP-H.323 translation include security and quality of service (QoS). Both, SIP and H.323, individually support these. However, translating from the open architecture of SIP, where security and QoS is independent of the connection establishment, to H.323, where security and QoS go hand-in-hand with the call establishment, remains an open issue. For example, an H.323 terminal communicates its QoS capabilities, whether it is able to reserve bandwidth, during registration and call admission to the gatekeeper, which is a signaling entity. On the other hand, QoS is handled end-to-end in SIP without involing proxies and registrars. Thus, an IWF that remains in the signaling path only, cannot translate QoS capabilities.

11.2 Architecture for User Registration

In this section, we describe different architectures for user registration and address resolution. *User registration servers* are the entities in the network which store user registration information.

SIP registrars and H.323 gatekeepers are user registration servers. It simplifies locating users independent of the signaling protocol if the IWF has direct access to user registration servers. The user registration server forwards the registration information from one network, to which it belongs, to the other.

11.2.1 IWF Contains SIP Proxy and Registrar

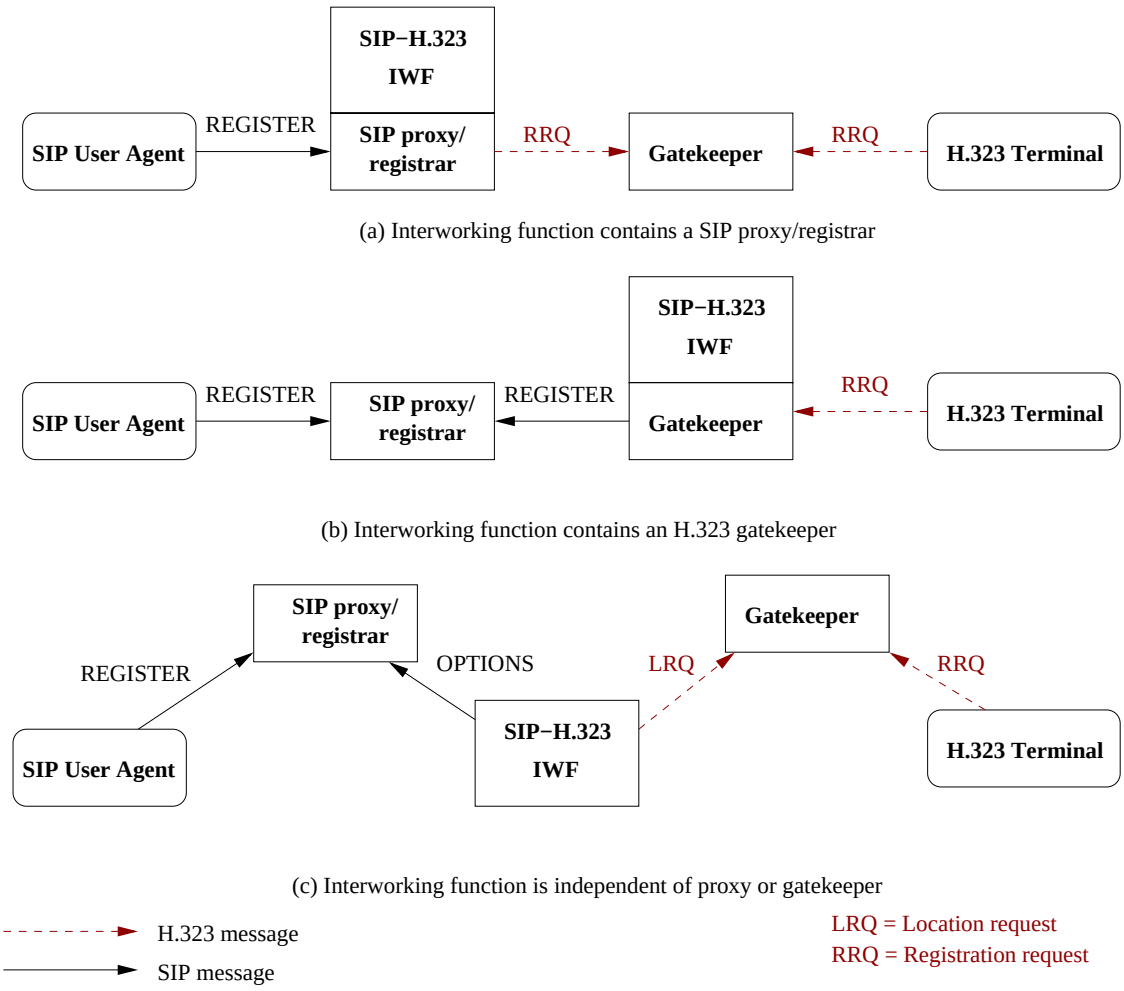


Figure 11.2: Architecture for user registration in SIP-H.323 interworking

Our first approach combines an IWF with a SIP registrar and proxy server, as shown in Fig. 11.2(a). In this approach the registration information is maintained by the H.323 gatekeeper(s). Whenever the SIP registrar receives a SIP REGISTER request, it generates a regis-

tration request (RRQ) to the H.323 gatekeeper, translating a SIP URI into H.323 Alias Address. H.323 users register via the usual H.225.0 procedure. Since the SIP registration information is also available through the H.323 gatekeeper(s), any H.323 entity can resolve the address of SIP entities reachable via the SIP server/IWF. In the other direction, if a SIP user agent wants to talk to another user, who happens to reside in the H.323 network, it sends a SIP INVITE message to the SIP server. The SIP server multicasts H.323 location requests (LRQ) to the H.323 gatekeepers. The gatekeeper to which the H.323 user is registered responds with the IP address of the H.323 user. Once the SIP server knows that the address belongs to the H.323 network, it can route the call to the destination.

One drawback of this approach is that the H.323 gatekeepers are burdened with all the registrations in the SIP network.

This approach only makes those SIP addresses handled by the registrar available to the H.323 zone. Typically, a registrar is responsible for a single domain, e.g., columbia.edu. Thus, each H.323 zone would have to have an IWF. If an H.323 user wants to call a SIP terminal, first the H.323 terminal locates, using DNS TXT records [257, p. 57], the appropriate gatekeeper[1], which in turn uses the registration information conveyed by the IWF to discover that this address is actually located in the SIP network.

Translation specification details

When receiving a SIP REGISTER request, the IWF generates an H.323 RAS RRQ request to its local GKs. The callSignalAddress of the RAS message contains the network address of the IWF; the terminalType is set to "gateway" and the terminalAlias is derived from the SIP To header or the Request-URI, as described in Section 11.3.

Thus, any address resolution request coming from the H.323 cloud to a SIP address can be resolved by H.323 gatekeeper(s) using H.323 RAS requests. Any request coming from the SIP cloud to H.323 is forwarded to the H.323 gatekeeper(s) by the IWF. H.323 gatekeeper(s) resolve this address using RAS/H.323.

During initialization, the IWF registers its own alias address (e.g., `gw1`) with its local

[1] It is not clear how widely implemented this approach is.

H.323 gatekeepers, so that anybody from the H.323 cloud can reach SIP endpoints by directly connecting to the alias address of the IWF and by providing a SIP address in the remote extension address of the SETUP message of H.323.

Fig. 11.3 shows the message flow sequences for successful initialization.

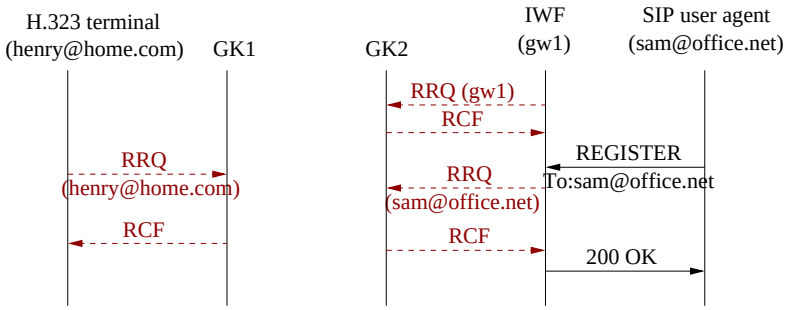


Figure 11.3: Initialization of SIP and H.323 terminals, and the IWF when IWF contains SIP proxy and registrar. The registration may get stored on two independent gatekeepers in the H.323 cloud.

Address resolution from SIP to H.323 is shown in Fig. 11.4, while address resolution from H.323 to SIP is shown in Fig. 11.5.

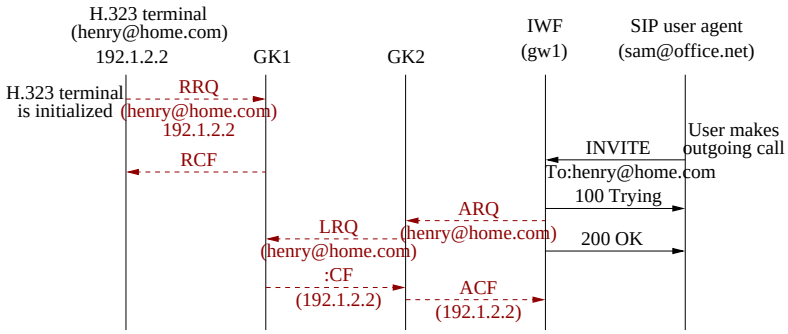


Figure 11.4: Address translation from SIP to H.323

This scheme assumes that the IWF is aware of the client part of the H.323 RAS protocol so that it can talk to the gatekeeper. Each SIP user agent (UA) that registers with the registrar also appears in the gatekeeper's database.

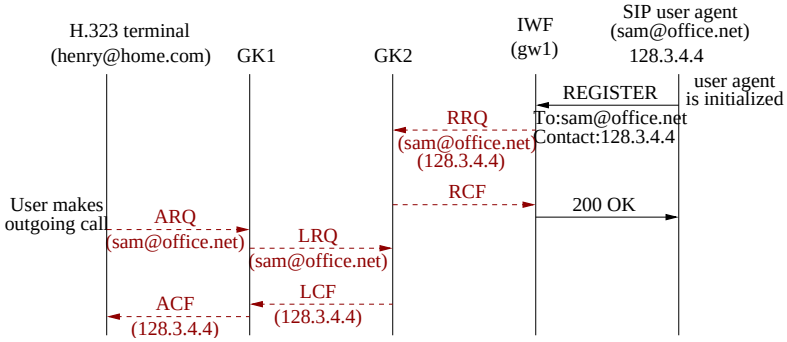


Figure 11.5: Address translation from H.323 to SIP

11.2.2 IWF Contains an H.323 Gatekeeper

This architecture, shown in Fig. 11.2(b) is similar to the previous approach except that the SIP proxy server maintains the user registration information from both networks. Any H.323 registration request received by the H.323 gatekeeper is forwarded to the appropriate SIP registrar, which thus stores the user registration information of both the SIP and H.323 entities.

To the SIP terminal, H.323 terminals simply appear as SIP URLs within the same domain. (See Section 11.3 on how H.323 addresses are translated to SIP URLs.) If an H.323 entity wants to talk to a user who happens to reside in the SIP network, it sends an admission request (ARQ) to its gatekeeper. The gatekeeper multicasts the location request (LRQ) to all the other gatekeepers. The GK-IWF server captures the request and tries to find out if the address belongs to a SIP user. It does so by sending a SIP OPTIONS request, which does not set up any call state. If the address is valid in the SIP network and the user is currently available to be called, the IWF responds with the location confirmation (LCF), letting the H.323 terminal know that the destination is reachable.

This approach has the similar drawback as the previous approach (Section 11.2.1) in that the proxy has to store all H.323 registration information.

However, this approach has the advantage that even if some H.323 gatekeepers are not equipped with a IWF, the address resolution works: If an H.323 gatekeeper cannot resolve a called address, it multicasts a location request (LRQ) to the other gatekeepers in the network. As long as at least one H.323 gatekeeper exists with the SIP-H.323 signaling translation capability, the SIP user can be located from the H.323 network. Note that the previous approach (Section 11.2.1)

required that all the SIP registrars/proxy servers must be equipped with IWFs.

Translation specification details

Address resolution from SIP to H.323 is shown in Fig. 11.6. while address resolution from H.323 to SIP is shown in Fig. 11.7.

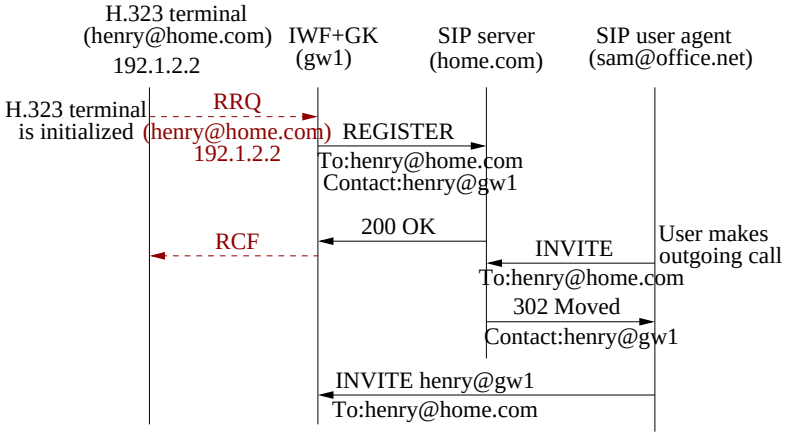


Figure 11.6: Address translation from SIP to H.323 when IWF contains an H.323 GK

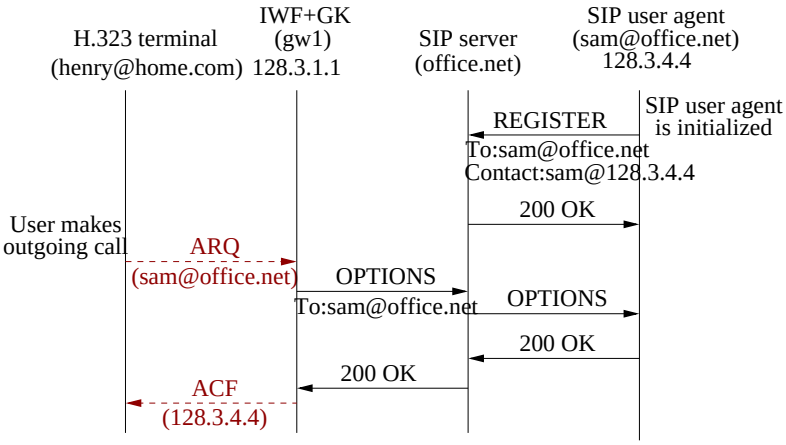


Figure 11.7: Address translation from H.323 to SIP when IWF contains an H.323 GK

11.2.3 IWF is Independent of Proxy or Gatekeeper

In the third approach, shown in Fig. 11.2(c), IWF is not co-located with either an H.323 gatekeeper or a SIP proxy server. User registration is done independently in the SIP and H.323 networks. However, when a call reaches the IWF, the IWF queries the other network for user location. Here, we assume that the IWF is capable of interpreting and responding to the location request (LRQ) from the H.323 network.

The address resolution mechanism works as follows. Suppose the SIP user Sam wants to talk to Henry, an H.323 user. Henry has registered with its own gatekeeper in the H.323 network and the gatekeeper knows Henry's IP address, conveyed via RRQ. When Sam contacts the SIP proxy with Henry's name, the SIP proxy has no registration for Henry, but is configured to contact the IWF in case the called party is in the H.323 network. The IWF, in turn, multicasts the location request (LRQ) for Henry to all gatekeepers. If there is no positive response from the gatekeepers of the H.323 network within a timeout period, the IWF concludes that the address is not valid in the H.323 network and the branch fails.

In the other direction, Henry sends an admission request (ARQ) to its gatekeeper. Since this gatekeeper does not have the address mapping for Sam, it multicasts the location request (LRQ) for Sam to the other gatekeepers in the network. In addition, the IWF is tuned to receive the LRQ. The IWF then uses the SIP OPTIONS request (as in Section 11.2.2) to find out if Sam is available in the SIP network and informs the GK if the request succeeds. This is followed by H.323 call establishment between Henry and the IWF and a SIP call between the IWF and Sam.

Translation specification details

When a call arrives at the IWF from SIP cloud, the IWF sends a RAS ARQ request to the H.323 cloud. If the address cannot be resolved or if the RAS request times out, it sends an appropriate response to the SIP endpoint. Similarly, calls from the H.323 cloud are translated into SIP requests and sent to a proxy or end system.

This approach works well if calls are identified by URLs indicating the signaling scheme, i.e., if an H.323 request is directed to a SIP URL or vice versa. In that case, it is sufficient if the GK or proxy is pre-configured with the address of the IWF.

If the destination address does not indicate the signaling protocol, a SIP proxy server has to forward all incoming requests to a local IWF, just in case the destination happens to be reachable via H.323.

In this architecture, the IWF must implement the RAS LRQ (location request) and LCF (location confirmation) messages. When a call is initiated by an H.323 entity, its gatekeeper will send an LRQ request to other gatekeepers at the well-known GK multicast address. The IWF captures the LRQ message and can use one of two approaches to find out if a SIP end point is available at that address. In the first approach, the IWF sends a REGISTER request without Contact information to the domain identified in the request (see Section 11.3). If the registrar has information about the endpoint, it returns this information in the Contact headers of the response. The IWF then translates this information and responds to the H.323 cloud with a LCF (location confirmation) message. If the registrar returns a negative indication, the IWF responds with a LRJ (location reject) message or remains silent. (Note that it is permitted that a terminal responds to LRQ messages, so that a gatekeeper is not needed as a part of the IWF application.) This approach is equivalent to SIP third-party registration and will not work if the registrar requires authentication. The second approach uses SIP OPTIONS messages, but is otherwise identical.

Direct Connection: No User Registration

The IWF should support direct H.323 connections. For instance, a SIP user (Sam) should be able to call an H.323 user (Henry) through an IWF (say sip323.columbia.edu) by placing a call to sip:henry@sip323.columbia.edu. Similarly, the H.323 user should be able to reach a SIP user (sip:sam@office.net) by establishing a Q.931 TCP connection to IWF and providing the destination address or the remote extension address in the Q.931 SETUP message as sip:user1@office.net. The direct connection does not involve user registration and the caller is expected to know that the destination is reachable via IWF.

If an IWF receives a Q.931 SETUP message, the IWF tries to parse the Q.931 destinationAddress. If the destinationAddress is not of the IWF itself and if it is able to resolve it to a SIP address, then the procedure described in section 11.4 is used to establish the call. (Note that

the user registration steps are not involved in this scenario.) Otherwise, if the destination address is that of the IWF and a remote extension address is present in the SETUP message of Q.931, then the IWF should use the remote extension address to determine the SIP address. The IWF MAY also be configured to forward all requests to a pre-defined SIP proxy.

11.3 Signaling Address Translation

While user registration exports identities into the foreign network, address translation is performed by the IWF to create valid SIP addresses from H.323 addresses and vice versa. In SIP, addresses are typically SIP URIs of the form sip:*user@host*, where *user* names can also be telephone numbers. However, SIP terminals can also support other URI schemes, for example "tel:" URIs for telephone numbers [258] or H.323 URLs [247]. Generally, SIP terminals proxy calls to their local server if they do not understand a particular URL scheme, in the hope that the server can translate it.

In H.323, addresses (ASN.1 AliasAddress) can take many forms, including unstructured identifiers (h323-ID), E.164 (global) telephone numbers, URLs of various types, host names or IP address, and email addresses (email-ID). Local user names and host names appear to be most common. For compatibility with H.323 version 1.0 entities, the h323-ID field of H.323 AliasAddress must be present.

For SIP-H.323 interoperability, there should be a consistent and unique way of mapping a SIP URI to an H.323 address and vice-versa. Translating a SIP URI to an H.323 AliasAddress is easy: We simply copy the SIP URI verbatim into the h323-ID. The user and host parts of SIP-URI are used to generate an email identifier, "*user@host*", which is stored in the email-ID field of AliasAddress. The transport-ID parameter is copied from the host part of SIP-URI if the latter is given numerically. The e164 field is extracted from the user part of SIP address if it is marked as a telephone number.

Translating an H.323 AliasAddress to a SIP address is more difficult since multiple representations (e.g., e164, url-ID, transport-ID) need to be merged into a single SIP address. In the easiest case, the alias contains a url-ID with a SIP URI, in which case it is simply copied into the SIP message. Otherwise, if the h323-ID can be parsed as a valid SIP address (e.g., "Alice

<sip:alice@host>" or "alice@host") it is used. Next, if the transport-ID is present and it does not point to the IWF itself, then it forms the host and port portions of the SIP URI. Finally, if the H.323 alias has an email-ID, it is used in the SIP URI prefixed with "sip:" URI scheme.

Note that the translated address may not necessarily be valid. On the H.323 side, it may be desirable to configure a gatekeeper to route all calls that are not resolvable within the H.323 network to the IWF, which would then attempt a translation to a SIP URI. This would allow H.323 terminals to reach any SIP terminal, even those not cross-registered.

If the IWF is configured to route all calls to a default proxy, then it will forward whatever SIP addresses it can form (from the H.323 Alias Address) to the proxy. This may be needed when the IWF implementation is split into two (physically separate) parts, namely an H.323 terminal and a SIP user agent. The H.323 terminal receives the call, maps the H.323 address to the SIP address and forwards the request to the SIP proxy server.

11.4 Connection Establishment

A point-to-point call from Alice to Bob needs three crucial pieces of information, namely the logical destination address (A) of Bob, the media transport address (T) at which each of the users is ready to receive media packets (RTP/RTCP) and a description of the media capabilities (M) of the parties.

Logical Destination address (A): This is the SIP address in To header or Request-URI, or the destination alias address in the Q.931 SETUP message.

Media Description (M): In SIP, M is the list of supported payload types as given by SDP media description ("m=") lines. In H.245, M is given by the Terminal Capability Set (TCS).

Media Transport Address (T): The media transport address indicates the IP address and port number at which RTP/RTCP packets can be received. This information is available in the "c=" and the "m=" lines of SDP and the Open Logical Channel message of H.245.

Alice should know A, T and M of Bob and Bob needs to know Alice's T and M. The difficulty in translating between SIP and H.323 arises because A, M, and T are all contained in

the SIP INVITE request and its response, while H.323 may spread this information among several messages.

11.4.1 Using H.323v2 Fast Connect

With H.323v2 FastConnect, the protocol translation is simplified because there is a one-to-one mapping between H.323 and SIP call establishment messages. Both the H.323 SETUP message with FastConnect and the SIP INVITE request have all three components (A, M and T). If the call succeeds, both the H.323 CONNECT message with Fast Connect, and the SIP 200 response, including the session description, have the required components (M and T of the call destination). Call scenarios are shown in Fig. 11.8 and 11.9.

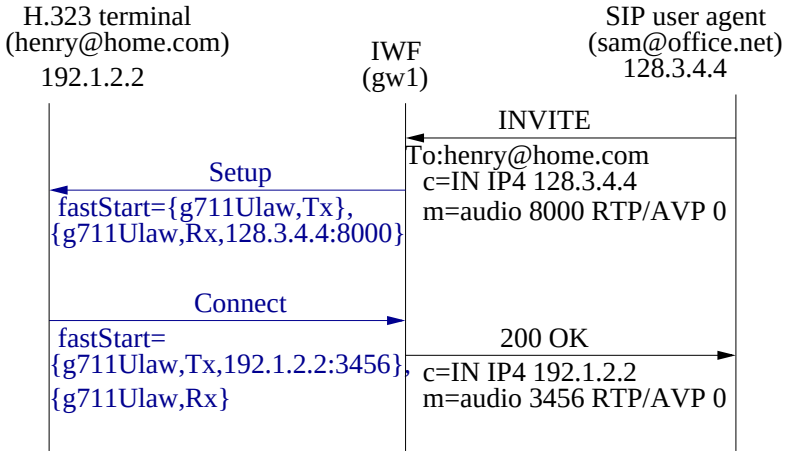


Figure 11.8: Call setup from SIP UA to H.323 terminal with FastConnect

11.4.2 Call Translation Without using Fast Connect

Since Fast Connect is optional in H.323v2, an H.323 entity must be able to handle calls without the Fast Connect feature for backward compatibility. Thus, it is likely that the IWF receives incoming calls from the H.323 network without Fast Connect PDUs. In particular, the IWF must accept a non-Fast Connect call from the H.323 side. In the other direction, the IWF should try to use H.323v2 Fast Connect, but must be prepared to switch to the multi-stage call establishment procedure if the response from the H.323 entity indicates that this is not supported.

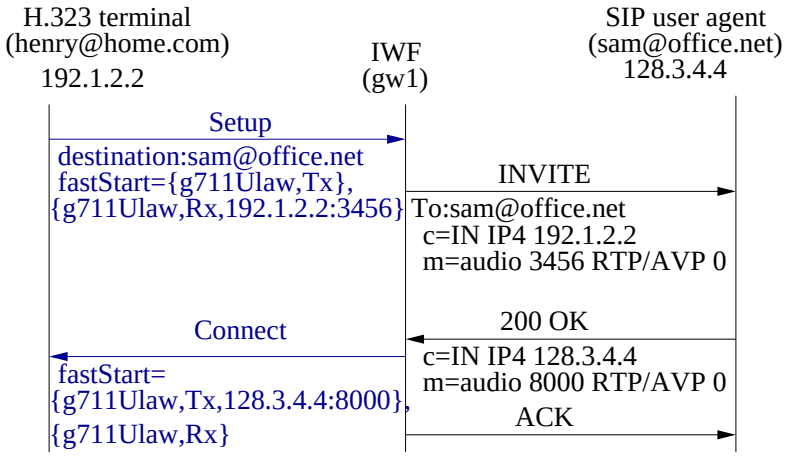


Figure 11.9: Call setup from H.323 terminal to SIP UA with FastConnect

When the call is initiated by a SIP UA all the call information (A, M and T) is present in the SIP INVITE message and can be used to format H.323 messages. The responses from the H.323 side are collated and forwarded to the SIP side, as shown in Fig. 11.10.

But when the call is initiated by an H.323 terminal, A, M and T are present in different messages. In a H.323 call without FastConnect, A is found in the Q.931 SETUP message, the TerminalCapabilitySet of H.245/H.323 contains M and T is present in the H.245 OpenLogicalChannel message. There are different ways in which these can be combined to form a SIP INVITE message.

One obvious approach is to accept the H.323 call without informing the SIP user agent. The H.323 call proceeds between the H.323 terminal and the IWF as if the IWF is just another H.323 terminal. The interworking function may get the media capabilities of the SIP user agent using the SIP OPTIONS message. Media capabilities of the H.323 terminal are obtained via H.245 capability negotiation. Once the logical channels are established from the IWF to the H.323 terminal, the IWF knows M and T and can place a SIP call by sending an INVITE. The media transport address from the 200 response is conveyed to the H.323 terminal while acknowledging the OpenLogicalChannel requests of the H.323 terminal.

While this approach is pretty simple, it has the disadvantage that the IWF accepts the call without even asking the actual destination, leading to caller confusion if the SIP destination is not reachable. This is undesirable if the caller is billed for the call setup.

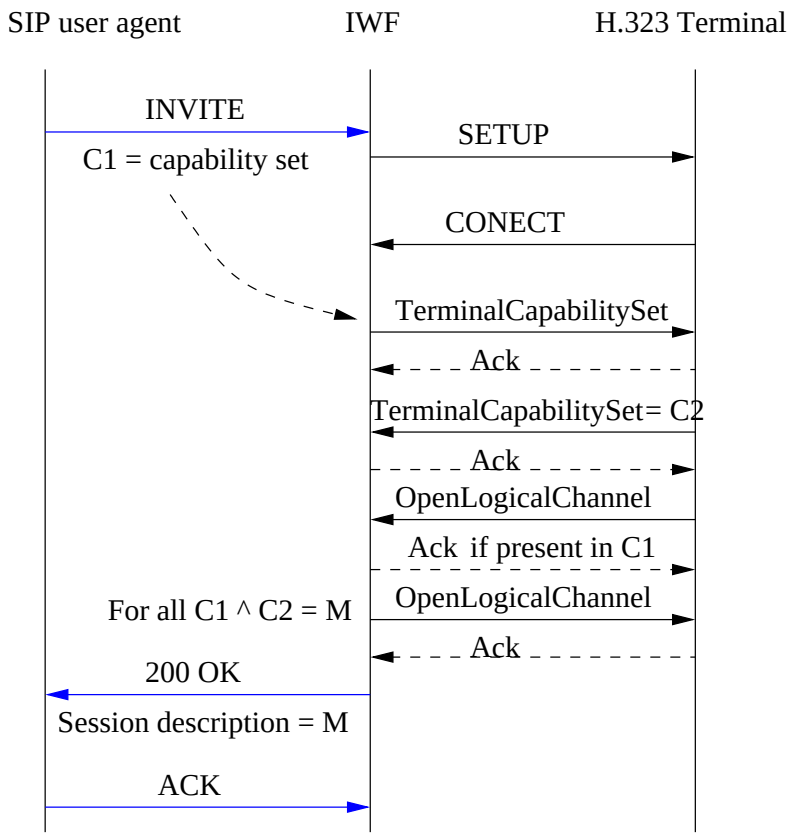


Figure 11.10: Call from SIP terminal to H.323 terminal without Fast Connect

This problem can be solved if the IWF sends a SIP INVITE without session description or a session description without media transport information when receiving the Q.931 SETUP message from the H.323 terminal. Only after the SIP user agent has accepted the call, the IWF forwards the confirmation (Q.931 CONNECT) to the H.323 terminal. The rest of the call establishment proceeds as before, except that the SIP OPTIONS message is not needed because the 200 response from the SIP user agent describes the media capabilities.

The media capabilities of the H.323 terminal are received in the H.245 TerminalCapabilitySet message and are forwarded to the SIP user agent as part of the ACK message or via an additional INVITE. The media capabilities of the SIP user agent are found in the session description of the 200 response to the INVITE request.

The different interpretations of media capabilities by H.245 and SDP potentially cause problems during the call. In SDP, a receive media capability of G.711 and G.723.1 means that the

sender can switch between these algorithms at any time during a call without explicitly informing the receiver. However, in H.245, the sender chooses an algorithm from the capability set of the receiver and explicitly opens a logical channel for that algorithm. The sender cannot switch dynamically to another algorithm without informing the receiver. The sender has to close the previous logical channel and re-open it with new algorithm. Alternatively, the receiver can use a H.245 ModeRequest to request the sender to use a different algorithm.

This problem can be addressed by having the RTP/RTCP packets from SIP to H.323 be intercepted by the IWF. If the IWF detects a change in coding algorithm, it initiates the required H.245 procedures. However, this approach is not advisable, as it scales poorly.

Another approach limits the media description sent to the SIP side to only one algorithm per media (or per alternative capability set). This can be achieved by maintaining a maximal intersection of the SIP and H.323 terminal capability sets (Section 11.5).

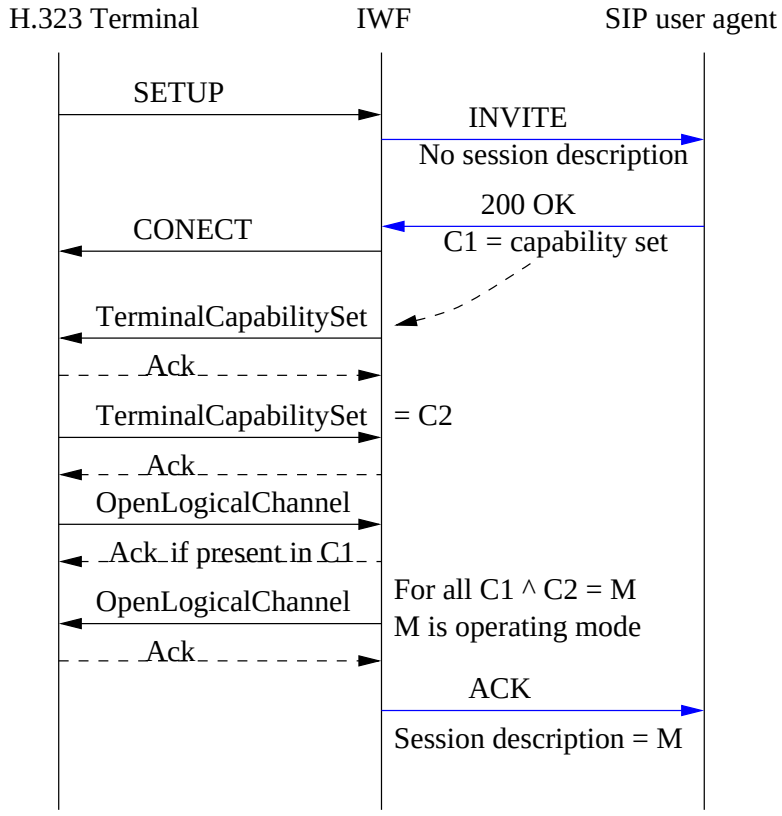


Figure 11.11: Call from H.323 to SIP terminal without Fast Connect

Call from H.323 Cloud to SIP Cloud with H.245 TerminalCapabilitySet (TCS) Mapped to SDP

A first approach has the IWF send a SIP INVITE request when it receives a Q.931 SETUP message. The SDP body of the INVITE request contains a default session description. The default session description must be either empty or contain media description (m=) lines indicating the minimal capabilities of any H.323 terminal handled by the IWF. Currently, these minimal capabilities include only PCMU audio. If the session description is not empty, the IWF has two choices:

1. The IWF controls an RTP translator that can forward RTP packets between two different IP addresses. The SDP "c=" line indicates the address of the translator, with the port indicated in the "m=" line.

2. The connection ("c=") line indicates a zero address and the media ("m=") line a zero port.

When the IWF receives a 200 (OK) response for the INVITE request from the SIP cloud, the IWF transmits a Q.931 CONNECT message to the H.323 endpoint. The IWF initiates the H.245 capability with the TCS (Terminal Capability Set) sent to the H.323 endpoint. On receipt of the TCS from the H.323 end point, which has a list of media supported by the H.323 endpoint, a SIP ACK message is formed with an updated session description reflecting the TCS. However, T is still unknown at this point, so that the SDP "m=" and "c=" lines remain as described above.

When the IWF receives an H.245 Open Logical Channel (OLC) message, the IWF acknowledges it with session information derived from the session description received from the SIP UA in the 200 (OK) response. When the first RTP packet of any media is received by the IWF from the SIP cloud, the IWF knows what payload type is used by the SIP UA for that media type and it can send OLC to the H.323 cloud. RTP packets received until OLC Ack is received are ignored or buffered for future transmission.

The problem with this approach is that RTP packets from the SIP UA cannot directly go to the H.323 terminal, but are instead routed through the RTP translator, violating requirement 4 in Section 11.1. This problem can be solved by having the IWF send a re-INVITE to the SIP

endpoint after the logical channels have been opened. This new INVITE message indicates media transport addresses (T) of the H.323 endpoint and not that of the translator.

A second problem is caused by the different interpretation of dynamic payload type switching in H.323 and SIP. When the TCS is mapped to SDP, the "m=" line is likely to list more than one payload type. This indicates to the SIP-controlled media agent that it may switch dynamically between all the payload types listed, without any H.323 or SIP signaling. However, in H.323, switching payload types requires Open Logical Channel signaling. This problem can be solved by restricting the SDP sent to the SIP endpoint to contain only one payload type per media description line. It is not clear how this payload type should be chosen or how the SIP endpoint can then switch payload types.

A third problem is that mapping a generic TCS to SDP requires enhancing SDP or SIP so that it can indicate multiple H.245 capability descriptors. For example, we could use SIP multipart message bodies, with each body part containing the SDP mapped from a single capability descriptor. Alternatively, the IWF could send a SIP OPTIONS request to the SIP UA and use that to calculate the common subset of capabilities (Section 11.5).

Call from H.323 Cloud to SIP Cloud Mapping H.245 Open Logical Channel (OLC) to SDP

In the second approach, on receipt of a Q.931 SETUP message, the IWF sends a SIP INVITE request as before. The IWF performs the H.323 capability exchange with the H.323 cloud without involving the SIP UA. The IWF then calculates the subset of capabilities from the H.323 TCS and the SDP contained in the 200 (OK) response to the INVITE. The IWF then sends an H.245 OpenLogicalChannel message for each of the media present in this subset. The OpenLogicalChannelAck message received from H.323 terminal will have the media transport addresses (T) of the H.323 terminal. On receipt of OpenLogicalChannelAck for all the OpenLogicalChannel messages, the IWF sends a SIP ACK message with the new transport addresses. This call scenario is shown in Figures 11.12 and 11.13.

Dynamic switching of H.245 Mode or Logical Channels is accomplished using SIP re-INVITE. For example, if video logical channel is opened from H.323 to IWF after initial call setup procedure (i.e., Logical Channels for audio are already opened), then the IWF sends a re-

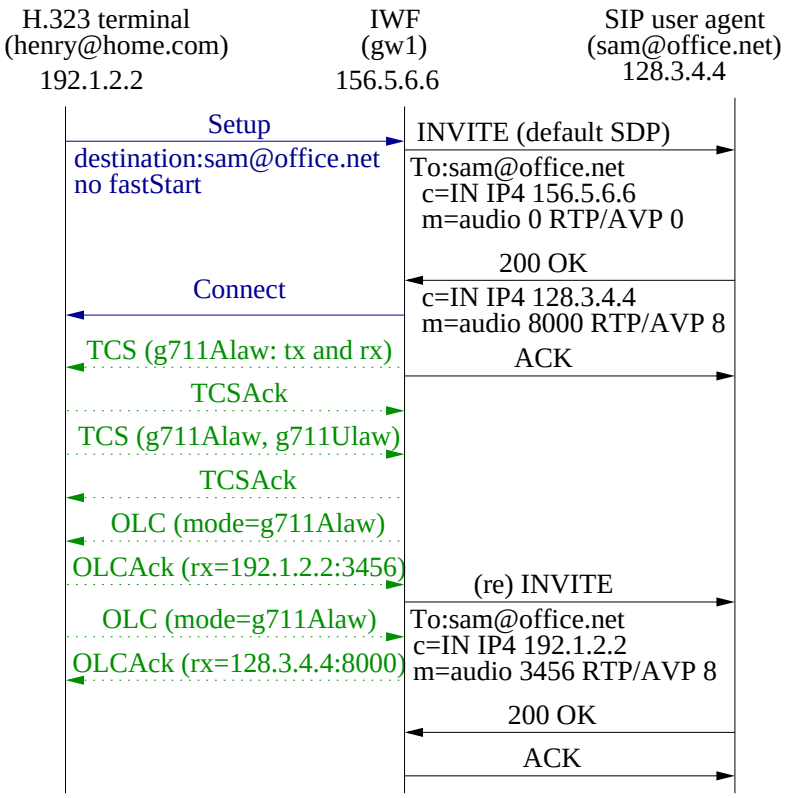


Figure 11.12: Call from H.323 to SIP with conversion between OLC and SDP

INVITE message to the SIP side with new SDP describing the video capability also. When the IWF receives 200 response from the SIP side, it sends OpenLogicalChannelAck to H.323 side with the media transport address as received in SDP in the response. The IWF will also initiate OpenLogicalChannel procedure for the video channel in IWF to H.323 direction.

If the media transport address of SIP UA changes during a call for a particular logical channel, (e.g., as a result of re-INVITE initiated by the SIP side) then the IWF sends RequestChannelClose H.245 message to the H.323 terminal for the logical channel. H.323 terminal will close the logical channel and will re-open it using OpenLogicalChannel. The changed media transport address of SIP UA can then be returned to H.323 terminal in a OpenLogicalChannelAck message.

In this approach, RTP packets can be sent directly between the two endpoints. However, the SIP UA is restricted to algorithms chosen by the IWF. Since these algorithms are derived from

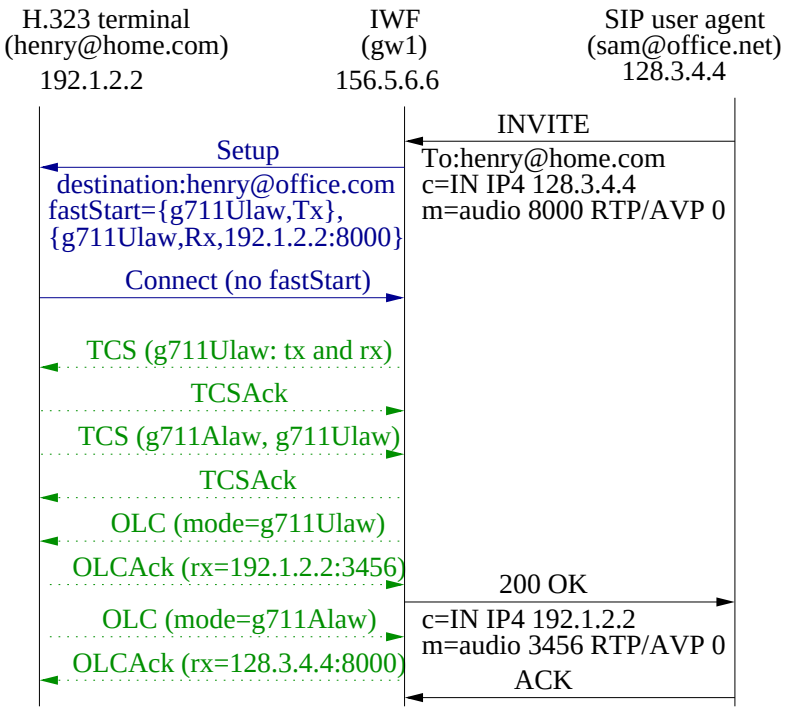


Figure 11.13: Call from SIP to H.323 with conversion between OLC and SDP

the common subset of H.323 and SIP capabilities, communications should still be possible.

A small problem with this message flow sequence is that ACK timeout on the SIP side and OLC timeouts on H.323 side may not match. This may result in lots of retransmissions in the SIP network. To avoid this, the IWF may choose to send an ACK immediately upon receipt of the 200 (OK) response from the SIP UA and then re-INVITE with an updated SDP after all OpenLogicalChannelAcks have been received from the H.323 endpoint.

We prefer the mapping of SDP to and from OpenLogicalChannel because mapping OLC is simpler than mapping TerminalCapabilitySet to SDP, which requires modifications to SIP or SDP, and it avoids the introduction of a temporary RTP translator.

11.5 Calculating a Common Subset of Media Capabilities

The *capability set* of a terminal or a user agent refers to the set of algorithms for audio, video and data that it can support. It also conveys information about constraints in the selection of

algorithms it may have. For example, due to limited bandwidth, a terminal may indicate that it can use either G.711 without video or G.723.1 with H.261 video.

The *operating mode* of a call refers to the algorithms which are used for the actual transfer of media. To determine the operating mode for a call it is necessary to find out the intersection of the capabilities of the endpoints in the conference. This section presents a way to calculate this intersection of the capability sets described by H.245 Terminal Capability Set (TCS) and that by SDP.

A *maximal intersection* of two capability sets is a capability set which is a subset of both the capability sets and no other superset of the maximal intersection is a subset of those capability sets. It can be proven that if M is an operating mode for capability set $C1$ as well as for capability set $C2$, then M will be an operating mode for maximal intersection of $C1$ and $C2$. Thus, we fulfill requirement 5 described in Section 11.1.

H.245 defines *Terminal Capabilities* as a list of capability descriptors, ordered by decreasing preference. Any one of the capability descriptors can be used for selecting operating modes. Each capability descriptor includes a simultaneous capability set. Each element in the simultaneous capability set is an alternative capability set. Each element in the alternative capability set represents an algorithm. Each algorithm has a payload type and can be fully described by the payload type, a profile and some optional attributes.

As mentioned earlier, $\{\}$ represents capability descriptor or simultaneous capability set (conjunction), and [] contains alternative capability set (disjunction).

Let a1, a2, a3, a4, a5 be audio algorithms and v1, v2, v3 be video algorithms. $C1$ represents a capability set with two capability descriptors:

```
C1 = { [a1, a2, a3] [v1, v2] }
     { [a1, a4, a5] [v1] }
```

Operating modes could be (a1, v1), (a1, v2), (a4, v1), (a5), etc. Note that (a4, v2) is not an operating mode since a4 and v2 are drawn from different capability descriptors.

Let C2 be another capability set.

```
C2 = { [a1, a4, a2] [v1, v2, v3] }
```

```
{ [a1, a2, a5] [v1, v3] }
```

The maximal intersection of C1 and C2 is

```
C = { [a1, a2] [v1, v2] }
    { [a1, a4] [v1] }
    { [a1, a5] [v1] }
```

Note that there are other capability sets which are intersections of C1 and C2 (e.g., {[a1,a2][v2]}), but they are subsets of C and hence can be derived from C.

Algorithm for Finding Maximal Intersection of Capability Sets

An algorithm to find the maximal intersection of any two capability sets $C1$ and $C2$ is given below:

1. Set the result C to the empty set.
2. Outer loop: for each pair of capability descriptors ($d1$, $d2$), where $d1$ is from $C1$ and $d2$ is from $C2$, derive the permutations of alternative sets, $s1$ and $s2$.

   Inner loop: for each such permutation, where $s1$ is from $d1$ and $s2$ is from $d2$, intersect $s1$ and $s2$ (written as s=$s1$^$s2$) and add s to C.
3. Remove duplicate entries from C.

Using the example with $C1$ and $C2$ given above, the outer loop runs for four iterations, since $C1$ and $C2$ both have two descriptors.

1. ```
   d1 = {[a1,a2,a3][v1,v2]},
   d2 = {[a1,a4,a2][v1,v2,v3]}
   ```

   The inner loop runs for 2 iterations:

   ```
   1) {[a1,a2,a3]^[a1,a4,a2],[v1,v2]^[v1,v2,v3]}
                = {[a1,a2][v1,v2]}
   2) {[a1,a2,a3]^[v1,v2,v3],[v1,v2]^[a1,a4,a2]}
                = {[][]}  /* Empty set */
   ```
   ```

2. d1 = {[a1,a4,a5][v1]},
   d2 = {[a1,a4,a2][v1,v2,v3]}
   1) {[a1,a4,a5]^[a1,a4,a2], [v1] ^[v1,v2,v3]}
   = {[a1,a4][v1]}
   2) {[a1,a4,a5]^[v1,v2,v3],[v1]^[a1,a4,a2]}
   = {[][]} /* Empty set */

3. d1 = {[a1,a2,a3][v1,v2]},
   d2 = {[a1,a2,a5][v1,v3]}
   1) {[a1,a2,a3]^[a1,a2,a5],[v1,v2]^[v1,v3]}
   = {[a1,a2][v1]}
   2) {[a1,a2,a3]^[v1,v3],[v1,v2]^[a1,a2,a5]}
   = {[][]} /* Empty set */

4. d1 = {[a1,a4,a5][v1]},
   d2 = {[a1,a2,a5][v1,v3]}
   1) {[a1,a4,a5]^[a1,a2,a5],[v1]^[v1,v3]}
   = {[a1,a5][v1]}
   2) {[a1,a4,a5]^[v1,v3],[v1]^[a1,a2,a5]}
   = {[][]} /* Empty set */

After these iterations the intersection set becomes

{ [a1,a2] [v1,v2] } { }
{ [a1,a2] [v1] } { }
{ [a1,a4] [v1] } { }
{ [a1,a5] [v1] } { }

After removing duplicates, the maximal intersection is

{ [a1,a2] [v1,v2] }
{ [a1,a4] [v1] }
{ [a1,a5] [v1] }

Since H.323 does not require that all algorithms listed within a single alternative capability have the same media type, we need the inner loop to find out all the possible combinations.

For example, if C1 = {[a1,a2,a3] [a1,a4,v2,v1]} and C2 = {[a1,a4,v2] [v1,v2,v3]}, then the above algorithm correctly finds the intersection as {[a1] [v1,v2]} {[a1,a4,v2]}

As an example, let the SIP capability set be {[PCMU,PCMA,G.723.1] [H.261]} and H.323 capability set be {[PCMU,PCMA,G.729] [H.261]} {[G.723.1] [H.263]} (i.e., the SIP user can support PCMU, PCMA or G.723.1 audio and H.261 video, whereas the H.323 user can support either one of the PCMU, PCMA, G.729 audio with H.261 video or G.723.1 audio with H.263 video). The maximal intersection as calculated by the IWF is {[PCMU,PCMA] [H.261]} {[G.723.1]}. The IWF derives an operating mode by selecting a capability descriptor from the maximal intersection and selecting one algorithm per alternative capability set (e.g., {PCMU,H.261}). The IWF conveys only the PCMU audio and H.261 video to the SIP user agent. If the SIP side sends additional INVITE with a different capability set ({[G.729,G.723.1][H.261]}, the new maximal intersection becomes {[G.729][H.261]}{[G.723.1]}. The IWF derives a new operating mode ({G.729,H.261}) and initiates the H.245 procedure to change the PCMU audio to G.729.

11.6 Translating Advanced Services

Both SIP and H.323 support advanced services like multi-party conferencing and call transfer. In this section we propose possible approaches for translating these services.

11.6.1 Multi-party Conferencing

A transparent support for multi-party conferencing can be achieved by having the IWF mirror the endpoint(s) in each direction. Fig. 11.14 shows a scenario in which two H.323 terminals (H1 and H2) and two SIP user agents (S1 and S2) are involved in a conference. From the H.323 side, the interworking function (IWF1) looks like a single H.323 terminal. From the SIP side, the interworking function acts as a single SIP user agent.

This approach fails if S1 invites another H.323 user H3 via a different interworking func-

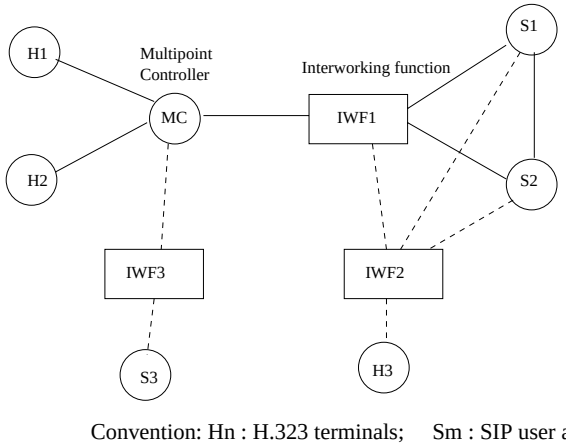


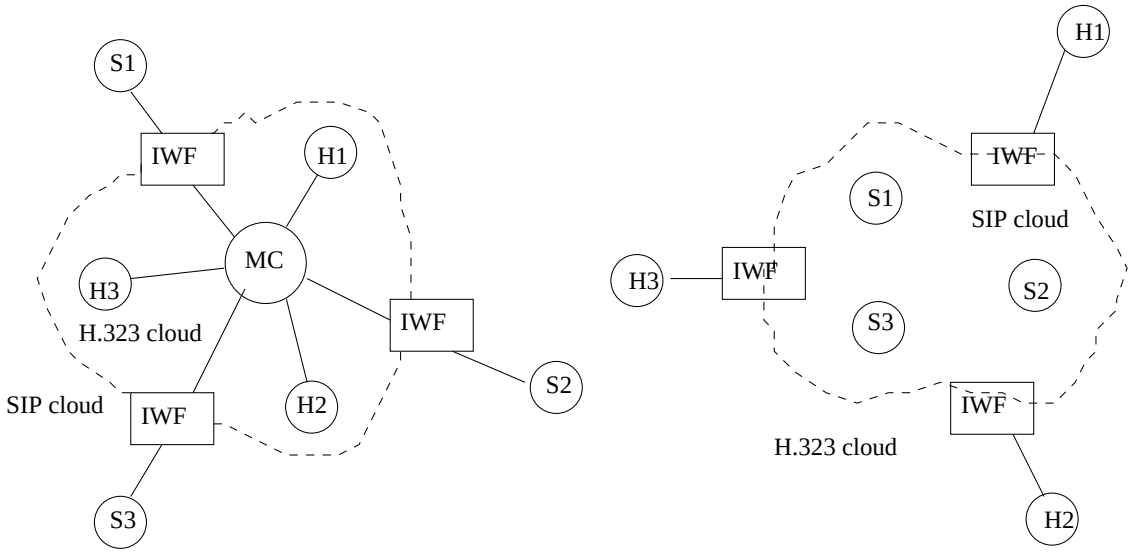
Figure 11.14: Ad-hoc conferencing among SIP and H.323 endpoints

tion (IWF2). For example, the participant H2 cannot know when H3 joins the conference. Alternatively, if H1 invites a SIP user, S3, S2 will not know of the presence of S3. One way for the participants to know about the existence of the other participants is to rely on the RTP/RTCP packets. This goes against the idea of H.323 conferencing where H.245 messages are used to convey the existence of new participants.

We can solve this problem by forcing all invitations to pass through the IWF. Fig. 11.15(a) shows a conference managed by an MC where H.323 terminals are directly connected to the MC and SIP user agents are connected through interworking functions. A SIP user agent is allowed to only invite other SIP UAs through the IWF, so that the IWF can update the MC state. In a SIP-centric architecture, Fig. 11.15(b), the H.323 terminals take part in the conference through the interworking functions.

We recommend a SIP-centered architecture because the SIP conferencing model is more general, allowing full mesh with distributed control or centralized bridged conferences. In general, translating services is greatly simplified if an operator adopts a primary signaling protocol, with services offered only in that protocol. Terminals using another protocol are restricted to making calls through the IWF.

Supporting H.332 loosely coupled conferences is straightforward, since SDP is used by both H.323 and SIP in that context.



(a) H.323 centered conference (b) SIP centered conference

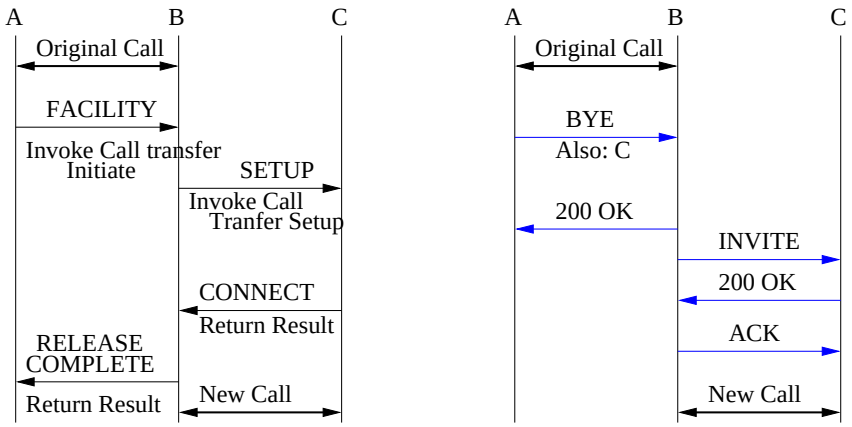
Figure 11.15: Different conferencing architectures

11.6.2 Call Transfer

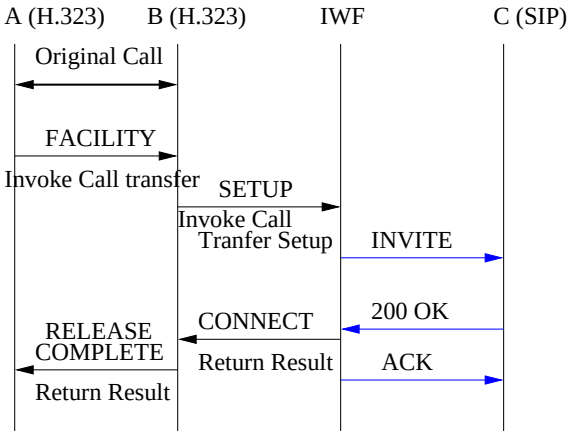
Call transfer is one of the many supplementary services needed for internet telephony. The idea is to convert a call between two entities (say, A and B) to a call between B and C. Fig. 11.16 shows the message sequence in H.323 and SIP and a possible translation when A and B are H.323 terminals and C is a SIP user agent.

A difference between SIP and H.323 arises because of the different philosophies of protocol extension. H.323 designers identify a supplementary service such as call transfer, call forwarding, call hold and define a new set of messages to accomplish it. This results in different procedures for different advanced services (e.g., H.450.2 for call transfer, H.450.3 for call diversion, H.450.4 for call park and call pickup). In SIP, crucial information needed for call services is identified and is encapsulated in new message headers (e.g., Replaces, Requested-By). Different call services are then designed using these building blocks. SIP also defines call transfer using the REFER method. The translation is similar.

A number of open issues remain when translating advanced services, including whether all call parameters can be translated and how security and authentication are to be handled. Since



(a) Call transfer in H.323 (b) Call transfer in SIP



(c) Call transfer in mixed network. A and B are H.323 terr
and C is a SIP user agent.

Figure 11.16: An example of call transfer mapping

the two protocols, H.323 and SIP, have many differences, a complete one-to-one translation is not possible for all advanced services, especially for end-to-end security and authentication.

11.7 Conclusion

We have described a framework for interworking between SIP and H.323. The challenges include call sequence mapping, address translation and mapping session descriptions. The implementa-

tion requirements and detailed interworking function behavior are specified in Appendix D.

Ad-hoc conferencing among SIP and H.323 participants is not possible without modifying one or both of these protocols. The problem can be made tractable by keeping an interworking function aware of all call state changes.

H.323 has picked up a number of features from SIP, such as Fast Connect and UDP-based signaling. It is possible that further convergence may occur, although not without fundamental changes to either SIP or H.323.

We have implemented a basic interworking function using the OpenH323 [259] library and a SIP signaling stack developed by us as part of CINEMA, and demonstrated a simple audio call setup between SIP user agents and Microsoft NetMeeting. My implementation was later adapted and commercialized by SIPquest, Inc., and licensed to a number of people including carriers. SIPquest engineers had also load tested the software for more than 10,000 simultaneously active connections.

The translation described in this chapter is not complete in all respects, but facilitates simple call setup. Overlap sending of dialed digits is not described. Data Application (T.120), encryption, security and authentication are not covered in this chapter. We have not addressed the issue of multistage translation, where two H.323 users communicate via a SIP gateway. It is not yet clear how common such a scenario would be, given direct network connectivity between the two parties.

Chapter 12

Conclusions and Future Directions

We conclude this thesis by emphasizing the need for reliability and scalability in Internet telephony. Although the Internet is perceived as less reliable than the Public Switched Telephone Network (PSTN), people expect PSTN-grade reliability and performance from Internet telephony. There are four high-level areas that must be addressed before Internet telephony can be adopted by the masses: reliability, scalability, quality of service and security. This thesis addresses only the reliability and scalability issues in the Session Initiation Protocol (SIP)-based Internet telephony systems.

12.1 Summary of the Problems and Contributions

We have addressed the following specific questions:

1. Can SIP servers provide carrier-grade reliability and scalability using commodity hardware? What factors affect the SIP server performance?

2. How can we build a server-less self-organizing peer-to-peer Internet telephony system in a standards compliant way?

3. Can SIP-based communication be extended to multi-platform collaboration using existing tools? How well does multi-party conferencing scale on a commodity hardware? How does SIP interoperate with another competing protocol, ITU-T's H.323?

These problems are addressed in this thesis in three parts:

Server redundancy

We used server redundancy to provide failover and load sharing in a server-based SIP infrastructure (Chapter 3). We implemented failover using database replication. We developed the two-stage architecture for SIP load sharing, which scales linearly with the number of servers. We quantitatively verified using real measurements, not just a simulation, that a cluster of six commodity PCs costing a few thousand dollars can support 10 million busy hour call attempts (BHCA), and 10 million users, and thus, exceeds the performance of a typical class-5 PSTN switch costing millions of dollars. We quantitatively compared the performance of various thread and event architectures in a single SIP server software. Our two-stage thread architecture gives the best performance for a stateful SIP server implementation compared to other event and thread models.

Peer-to-peer

To reduce the configuration and maintenance cost of a server-based system, we developed and implemented mechanisms to build a peer-to-peer network for Internet telephony using SIP, while keeping the syntax and semantics of SIP messages (Chapters 4, 5 and 6). Additionally, we built mechanisms to securely use an external P2P network as a SIP location service (Chapter 5). The advantage of using SIP is that it can interoperate with existing SIP-based infrastructure such as gateways and conferencing servers. The hybrid architecture allows a user to be located in both peer-to-peer and server-based infrastructure.

Enterprise IP telephony

We extended our Internet telephony architecture, CINEMA, to a multimedia collaboration system (Chapter 9). We built software pieces for both synchronous collaboration such as highly interactive conferencing as well as asynchronous collaboration that does not require simultaneous active presence of the participants. We evaluated the performance of our conference server and quantitatively verified that it can support large scale audio conferences with thousands of participants

in a cascaded mixer architecture (Chapter 10). We also developed and implemented a translation scheme between SIP and H.323 so that our SIP-based components can interoperate with H.323 infrastructure (Chapter 11).

12.2 Connecting Themes

There are three main themes that connect various parts of this thesis: reliability, scalability and interoperability.

Reliability

There are two aspects of PSTN reliability: equipment reliability and network availability. PSTN switches have a "5 nines" reliability requirement, i.e., are available 99.999% time. PSTN has call success probability of three to four nines. On the other hand, current Internet telephony has 99.5% probability of call success, and 1.5% probability of call abortion due to poor audio quality, giving 98% availability [260]. The SIP server reliability determines the equipment reliability in Internet telephony. The failover architecture described in Section 3.3.6 can be used to achieve "5 nines" reliability of SIP servers. The overall availability depends on a number of other factors such as underlying routing infrastructure and DNS.

Peer-to-peer (P2P) systems are inherently more reliable and robust because there is no central point of failure, and the network self-organizes itself when a node fails. Data stored on the P2P network is replicated to improve the data availability. Our P2P-SIP architecture benefits from the robustness of the underlying P2P network.

Scalability

We have shown that unlike web server redundancy, a simple server farm with identical redundant SIP servers is not scalable, because user registration needs to be propagated to all the servers or databases (Section 3.4.3). We built the two-stage scalable architecture which allows linear scalability with the number of servers (Section 3.4.5). The two-stage architecture can also be applied to a multi-threaded transaction stateful server implementation to reduce lock contention

among multiple threads (Section 3.6).

We also evaluated the performance of our conference server, and showed the performance gain in a two-stage cascaded mixer architecture.

P2P systems are inherently scalable because a new node also serves other nodes in lookup, unlike a server-based system where a new client only adds load on the server. Our P2P-SIP architecture benefits from the scalability of the underlying P2P network.

Interoperability

When extending the SIP-based multimedia communication system to a comprehensive multi-platform collaboration system, a number of new components are included, e.g., unified messaging, document sharing and screen sharing. Instead of using proprietary extensions, we have reused existing protocols and tools as much as possible in our architecture. For example, we use RTSP for voice mail recording and playback, so that existing media tools such as QuickTime can be used to listen to the messages. Shared web browsing uses the SIP MESSAGE request to convey the browsed URL. We use VNC for screen sharing, and SOAP and VoiceXML for conference control. The components interoperate with other implementations based on these open standards.

We also presented interworking between basic user registration and call between SIP and H.323, so that our SIP-based components can interoperate with other H.323 infrastructure.

One of the distinguishing factors between our P2P-SIP and Skype is that Skype uses a proprietary protocol, thus supports only a single vendor and single identity provider model. On the other hand, our P2P-SIP uses SIP for Internet telephony signaling, and can also interoperate with the client-server SIP infrastructure. Using an open standard allows us to have systems from different vendors and service providers, instead of a single vendor Skype system.

12.3 Server-based vs. Peer-to-peer Internet Telephony

We have described two architectures for scalability and robustness of Internet telephony: server redundancy and peer-to-peer. The main problem with a server-based system is that it usually requires a dedicated system administrator to configure and maintain the servers. The system

relies on external dependencies such as DNS. On the other hand, P2P systems automatically configure themselves, and the P2P network self-organizes itself. This reduces the configuration and maintenance cost.

Secondly, server-based systems are prone to catastrophic failures, e.g., if all the servers in the cluster are destroyed in a bomb explosion, the users become unreachable even if the user machines and parts of the underlying IP network are functioning. On the other hand, a P2P network is implicitly fault tolerant.

Peer-to-peer systems have significantly higher lookup latency. For example, unlike a simple request-response message in the server-based system, a lookup in Chord-based P2P network of N nodes can result in $\log N$ application level hops. However, for Internet telephony call setup this delay of a few seconds is not a problem.

The security of a structured P2P network against malicious node behavior is still an open issue. Additional challenges are in building a pure P2P reputation system and working around spy nodes. On the other hand, in a server-based system, the clients can trust the server, and transport security of the messages (e.g., using TLS) can guarantee the system security.

Both P2P and server-based systems are scalable. We showed that our two-stage cluster architecture performs linearly with the number of servers. Therefore, we can achieve any desired performance by adding more servers in the cluster. P2P networks are inherently scalable, because a new node also shares the total cost of service, i.e., user lookup.

PSTN interoperability via a gateway is achieved using additional protocols such as ENUM and TRIP. A P2P system can use the server-based infrastructure for PSTN interoperability similar to Skype, or allow discovery of PSTN gateways co-located with peer nodes.

Given the tradeoffs between P2P and server-based architectures, and wide deployment of server-based Internet telephony infrastructure, we predict that both the systems will exist for quite some time. Thus, we need to interoperate between the two. Our open standards-based P2P-SIP architecture provides a hybrid system where lookup can be done in both P2P or DNS, and allows interworking between the two architectures.

12.4 Implications of this Research

Internet telephony is more rich in features compared to the PSTN, and allows extending the system for new services easily. For example, VoiceXML-based telephony applications can be easily developed using existing web infrastructure. On the other hand, PSTN switches are closed systems. It is hard to add new services. A result of Internet telephony research in the past decade is that many organizations and universities are gradually replacing the local PBX with a SIP-based IP PBX, such as our CINEMA system. SIP-based systems can provide all the PBX features such as voice mail, conferencing and call transfer, albeit at a lower cost and better quality and performance, e.g., wide-band audio codec such as G.722 can be used in Internet telephony instead of only G.711 in PSTN.

Cost is an important factor in determining research directions. For example, Internet telephony saves long distance cost of PSTN calls, and a P2P network saves the configuration and maintenance cost of a server-based system. This translates to zero-cost PC-to-PC calls on the Internet, similar to free emails and instant messaging. The user has to pay only when the call crosses into the PSTN.

Using open platforms with existing protocols calls for a plethora of new services. We have built a number of applications in our CINEMA test-bed. CINEMA is also being used in a number of other experiments and new systems in our lab. For example, the NG911 project uses the centralized conference server, a Verizon sponsored project builds a firewall and NAT controlling proxy using CINEMA, and another project extends the two-stage architecture to presence scalability. A number of students have done projects using CINEMA, e.g., an auto-attendant application using VoiceXML, without having to modify the existing components because CINEMA uses open standards.

We have shown how to build a scalable cluster of SIP servers, and identified which software architecture performs well in terms of threads and event-based implementations. This gives us a better understanding of scalability and system design issues for SIP-based systems.

There are two types of factors affecting the system scalability: server and network. There is a limit on single server scalability even with any number of optimizations. Thus, load distribution on the network of servers is more promising to address the growing performance needs of

Internet telephony systems. We have shown this for both SIP call setup and conference mixer.

Our systems are modular. The logical components are separated, and usually implemented as independent software. This promotes modular architecture, e.g., keeping the P2P layer separate from the SIP layer allows extending the system to the external DHT architecture. Furthermore, this promotes DHT-as-a-service model, to save the cost of building new P2P systems from scratch.

12.5 Future Directions

We have described both server-based and server-less (peer-to-peer) Internet telephony scalability. We also described how P2P can be used in a server farm to reduce the configuration and maintenance cost of servers. This is very promising particularly for carriers with lots of servers, e.g., the SIP proxies in the 3GPP architecture can be extended to automatically configure themselves to serve different roles for different networks.

We have described how to use the shared and managed OpenDHT in P2P-SIP securely. However, handling malicious nodes in an unmanaged distributed hash table is an open problem. Unless this is addressed, a global public DHT for P2P-SIP where the user's phone becomes a DHT node cannot be deployed securely.

Another interesting question to ask is: what is the performance overhead of security and quality of service in Internet telephony? Our performance evaluation used UDP transport. However, in a real deployment on the Internet, TLS is preferred. Although the cluster architecture we presented can potentially scale to other transport protocols, the absolute performance of the individual server may be quite different.

SIP-based Internet telephony allows adding new services such as presence, instant messaging and multi-player gaming. The user model and load for these services are different than the simple user registration and call arrivals that we analyzed. Although our scaling architecture forms the building block for these SIP-based services, the performance evaluation needs to be redone to verify the performance gain.

A number of open issues remain in P2P-SIP. In particular, we need to explore the effect of NAT and firewall traversal on the DHT performance given that many residential users are behind

NAT, interworking with PSTN without using a centralized server-based system, optimization for locating the best media relay in the P2P network to forward media packets to nodes in a private IP address space, and extension of Internet telephony communication to a multimedia collaboration in P2P using standard protocols.

Appendix A

Design and Implementation of the Columbia SIP Library

I have implemented a modular SIP library in C++ and used it as the underlying SIP implementation in all our SIP-based components in CINEMA, such as voice mail server, conference server, peer-to-peer client adaptor, and VoiceXML browser. This chapter describes the SIP library, libsip++ and the implementation overview of various components in CINEMA.

A.1 Background

The libsip++ is derived from the Columbia SIP server, sipd, and reuses the parsing and transaction handling from sipd. This section describes the components needed for understanding libsip++ design.

Call Routing

This section describes how sipd handles an incoming call. This is useful in understanding the design of the SIP library and other components, such as our voice mail server, sipum.

Canonicalization

An incoming call is processed as shown in Fig. A.1. Here, Alice, sip:alice@cs.columbia.edu calls Bob, sip:Bob.Wilson@cs.columbia .edu. Through DNS SRV records, Alice's user agent finds out that the host conductor.cs.columbia.edu serves SIP requests for the domain cs.columbia.edu. We assume that Bob can be reached in many different ways, for example, as bob, Bob.Wilson, bob_wilson, Bob.V.Wilson and webmaster.

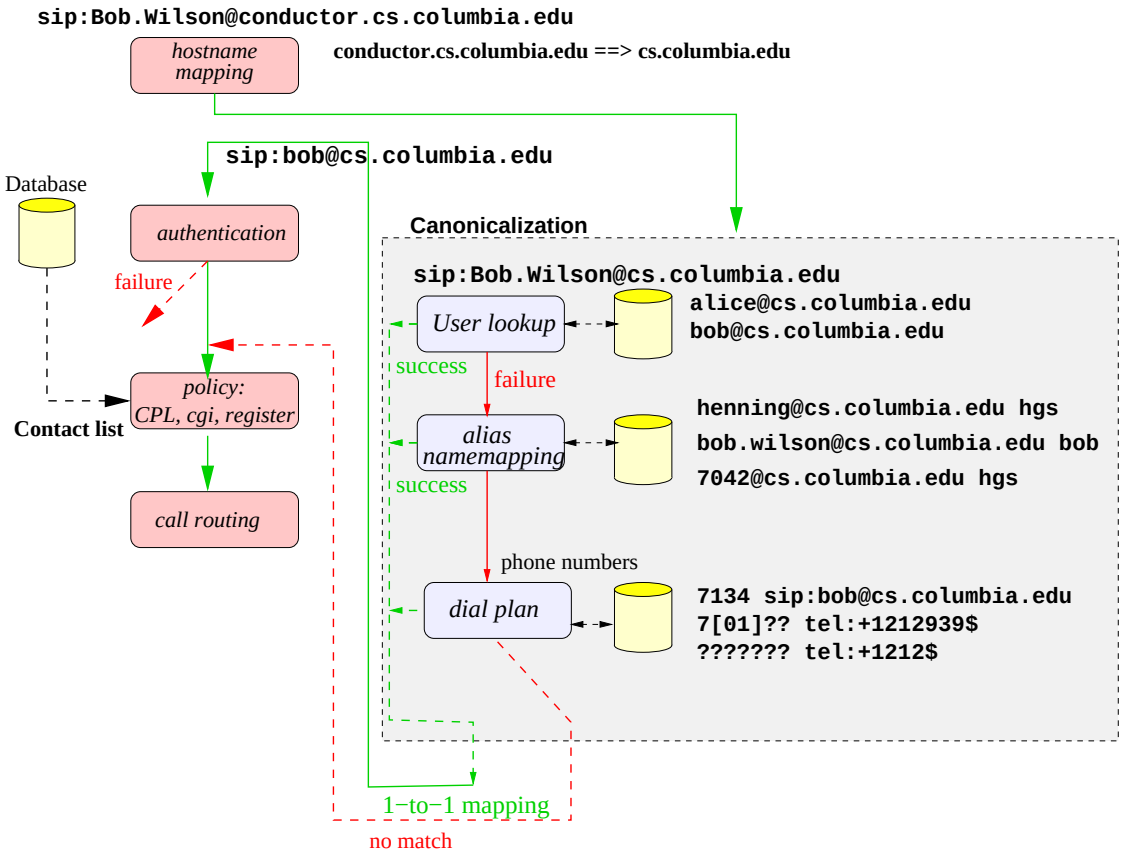


Figure A.1: Canonicalization, authentication and routing for a call

After validating the syntax of the call request, the server transforms the callee address to a canonical user identifier for database lookup, by first transforming the host portion and then the user name portion. For example, the domain portion, conductor.cs.columbia.edu is canonicalized to cs.columbia.edu. This is done by matching the domain portion of the request URI against a list of possible domain names and IP addresses for SIP requests to this proxy server.

In our case, this includes the domain name cs.columbia.edu and the host name and IP address on which sipd is running. If the canonicalized host name does not match, the server is being used as an "outbound proxy server" and just routes the request to the SIP server for the domain, without any processing. Outbound proxy servers are useful for logging and firewall control, for example. Outbound proxies are not needed for "sip" URLs, but SIP requests with "tel" URLs need to designate such a proxy to translate the telephone number into a routable SIP identifier. This SIP identifier can either point to a PSTN gateway or be a regular sip:user@host URL.

The server first checks whether the SIP identifier is present in SQL put table. If it is present, then the username is used unchanged and is the canonical user identifier. If it is not present, then the server tries to translate the username into a canonical form by two transformations. In the first, the SQL aliases table is checked to see whether an alias entry is present for the user. If an alias is present, it is resolved to its canonical identifier user. In the second step, the name mapper function searches the SQL person table to see if it can deduce a username, by comparing the user part of request URI to various combinations of the first, last, and middle names recorded in that table. (In the example, the name mapper determines from the person table that the name "Bob Wilson" corresponds to user bob.)

Finally, the server checks whether the user identifier is a telephone number or not. A request URI for telephone numbers can be of the form: "tel:*number*", "sip:*number@domain*;user=phone", or "sip:*number@domain*". Note that for the "tel" URL, the domain portion does not exist hence there is no need to canonicalize the domain part. The *number* can have an optional prefix of "+" to indicate a globally routable number, e.g., +1-212-9397000. The first and second cases specifically tell the server that the address is a telephone subscriber. A heuristic is used to determine if the address matches the third case. A database lookup is done to compare the address against the available user names and aliases to find a match. This allows to create telephone number as user identifier or to create telephone number aliases for *user@domain*. If the resulting address is still a telephone number, it is canonicalized using a dial plan. If none of the rules match, the user identifier is returned unchanged to the server.

The SIP server then retrieves contact and policy information for the user bob@cs.columbia.edu. The policy information describes how the call is handled, for exam-

ple whether it is to be proxied or redirected. Bob's preferences and policy are then executed. These may, for example, demand that a calling user be authenticated, refuse or redirect calls, or apply preferences about where Bob wants to be reached. If the server determines that Bob's current policy allows Alice's call to reach him, it contacts Bob's list of registered locations. Bob's current SIP phones ring, he picks up the handset and starts talking to Alice. When they are done, either of them can terminate the call.

If the callee's contact location is a telephone number, then the dialplan matching is done on the contact location. The dialplan leads to a gateway to reach the PSTN destination.

If there are multiple contacts found for the user, then all of the contact locations are used. The preference values (q-value) of the contacts are used to order the contact locations. The more preferred value is tried first, and if it fails or times out, the next preferred location is used. If multiple contacts have the same or similar q values, then the server forks the call request to all those locations in proxy mode. In redirect mode, it returns all those contact information back to the caller. For example, if user *sales@company.com*, has locations *rep1@pc1.company.com* (preference 1.0), *rep2@pc2.company.com* (preference 1.0), *rep3@pc3.company.com* (preference 0.8), *senior-rep@company.com* (preference 0.3) and *manager@company.com* (preference 0.3) then a call to *sip:sales@company.com* is first forwarded to both rep1 and rep2. If they do not pick up the phone or the call fails, then rep3 is tried. If rep3 also does not answer the call, then it is forwarded to senior-rep and manager simultaneously. The forking behavior with the configurable priorities for different contact locations can achieve enhanced automatic call distribution (ACD).

Programmable Call Handling

Sipd supports both CPL and SIP-CGI. SIP Servlet [261] implementation is also partially implemented. The piece of software which alters the server behavior, either a SIP-CGI or a CPL script, can be uploaded to the server using a SIP UA such as sipc, or from the web interface.

Database Lookup

Database lookup for locating the contacts of the users constitutes a substantial fraction of the processing power in a SIP proxy server. Higher delay in database lookup (approximately 10

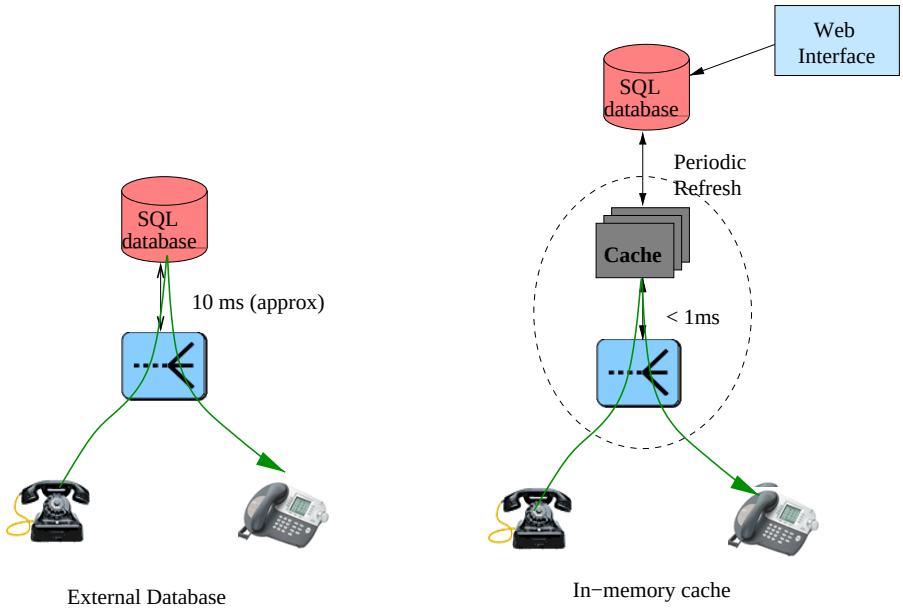


Figure A.2: SQL vs FastSQL

ms per query) increases the response time/delay of the transaction, hence the performance and the scalability. We have implemented an in-memory database scheme to speed-up the database access time in our SIP server (sipd) as shown in Fig. A.2. This involves loading the various database tables (e.g., user information, contact locations, aliases) into the main memory, instead of doing lookup into the database for every transaction. Since each table entry takes less than few hundreds of bytes it is perfectly reasonable to use it in an enterprise environment with only a few thousands of users for improved performance. However this optimization causes another problem related to synchronization of the in-memory and external database. In particular, care must be taken to updated the in-memory database when a new contact is added from the user interface. We define a periodic refresh interval (About two minutes for contacts table and half an hours for user information and aliases tables) to refresh the in-memory database. The contacts table is written out to the external database from in-memory database periodically. We read only those entries that are modified since last read and write only modified entries back to the database during refresh.

CINEMA Libraries

Many of the architectural components of CINEMA described in chapter 9 are implemented in C/C++, e.g., SIP server (sipd), media server (rtspd), conference server (sipconf), SIP-H.323 interworking function (siph323) and voice mail server (sipum). All these pieces of software share the common code base wherever possible. The common part is identified and abstracted as a set of libraries. Then the applications are built on top of these libraries. This section describes the various modules used for implementing the system and discusses the design details of the SIP library.

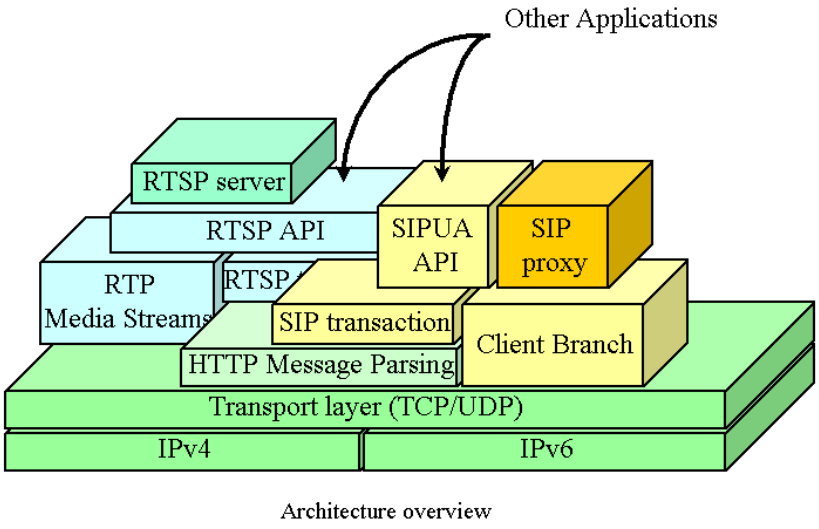


Figure A.3: Software design modules

The layered hierarchy of various sub-modules is shown in Fig. A.3. The lowest transport layer is assumed to be TCP or UDP. We use the standard `socket` interface for this layer. A generic HTTP message parsing layer is used for parsing various HTTP-like messages, SIP and RTSP. RTSP and SIP-specific routines are added above this layer. In particular, the RTSP transaction layer maintains the state for a media session, while the SIP transaction and client branch layers maintain the state for a SIP transaction. The SIP transaction layer is used in implementing the SIP proxy server. The SIP user agent library (libsip++) uses the transaction layer, and imple-

ments the call control state machine above that. Both internal and external libraries are used to build various applications as shown in Fig. A.4.

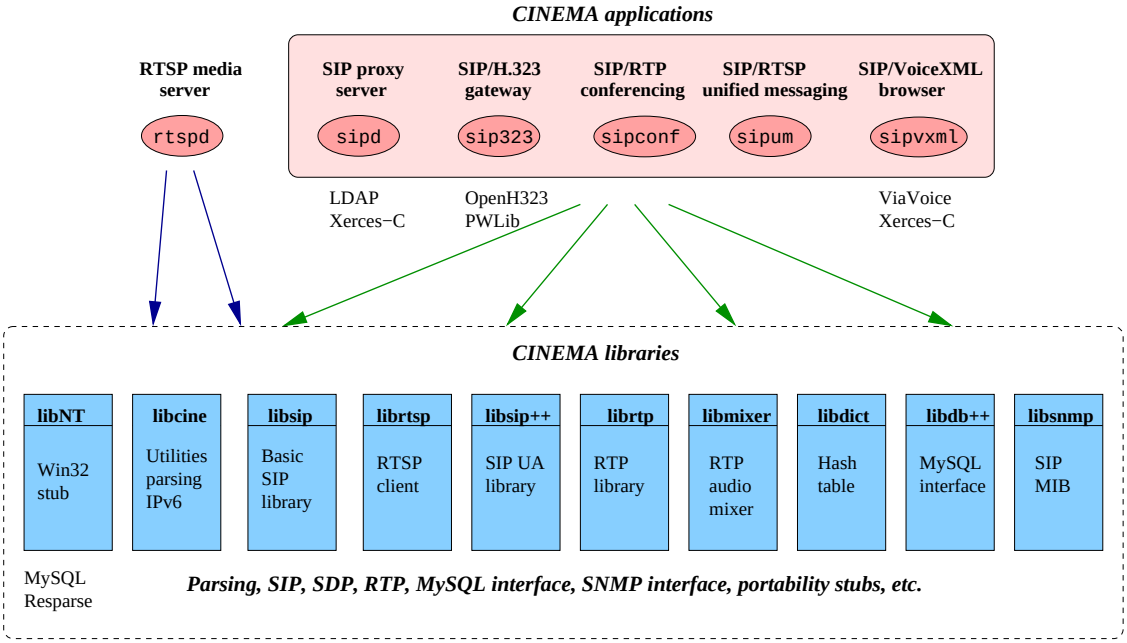


Figure A.4: Software library and applications

The CINEMA libraries are briefly described below:

libcine: libcine is a generic library with general-purpose utility functions for parsing HTTP messages, manipulating URIs, logging requests, MD5 functions, database access, software license check, TCP/UDP wrapper, dynamic string, resolving host names, logging debug information. This library is shared by both SIP and RTSP implementations.

libdict: libdict is a general-purpose library for dictionary or hash-tables in C.

libdb++: We use the MySQL database in our environment to store various user and system configuration. This module is a high-level C++ interface for accessing the database tables built as a wrapper over the libmysqlclient library. It also provides an in-memory database mechanism to speedup database access. It also implements a file based authentication to allow non-database type simple applications like user agent libraries.

libsip: libsip is a SIP library in C that implements the SIP transaction and client branch layers.

It allows different authentication mechanisms used by SIP. It also contains the database interface to libdb++.

libsip++ (or libsipapi): libsip++, which was later renamed to libsipapi after complete upgrade to C++, is a SIP user agent library that implements the call control for establishing, maintaining and terminating a SIP call. It also has SDP parsing routines. It uses libsip for implementation of transaction and client branch layers.

libmixer (or libconf): libmixer, which was later renamed to libconf, is an RTP audio mixing library and video distribution library used by the conference server.

libmedia: This library is used for transcoding between different audio codecs, and writing and reading from audio file for media streaming and recording.

libnat: This library handles NAT and firewall traversal issues using STUN, TURN and ICE protocols.

rtplib++: The RTP library written is written in C++.

NT: NT library implements the basic portability stubs on the Microsoft Windows platform for the commonly used Unix functions. In particular, it contains routines for `aliases`, `crypt`, `hashtable`, `inet`, `regex`, `getopt`, and `pthread`. These stubs allow us to use the same code base for both the Unix and Windows platforms.

I am the primary author of libsipapi, libconf, libnat, and libmedia. Rest of the chapter describes the libsipapi library.

User Agent Policy

A *SIP transaction* is identified by the Call-ID, To, From and CSeq SIP headers and the SIP request URI. A transaction roughly corresponds to a request and all its responses plus their retransmissions.

A transaction can be of two types: proxy transaction and user agent transaction. A proxy transaction is associated with a set of client branches. When the proxy *receives* a request from

an upstream client it creates the transaction object then forwards the request to the downstream server(s) using client branches. The responses received by the client branches from the downstream server(s) are forwarded to the upstream client. A user agent transaction can either receive a request and terminate it or can originate a request and wait for responses. Thus, the user agent transaction can be further classified into incoming transaction (without any client branch) and outgoing transaction (with only one client branch). Contrast this with the proxy transaction which can have one or more client branches. More than one client branches signify the forking proxy behavior. A forking proxy forwards a call to several possible locations simultaneously and completes the call setup by connecting the caller to the first location answering the call. A client branch represents a possible location where the destination can be reached.

Once a request is received it can be processed in a variety of different ways: proxy it (proxy transaction), send a redirect response, inform the user (user agent transaction), or reject it. The decision to choose appropriate behavior can be governed by different *policies* as shown in Fig. A.5.

A.2 User Agent Library

The user agent library, libsip++, is built on top of the user agent policy interface and uses the underlying libsip's transaction and client branch architecture. As mentioned earlier, there are two types of user agent transactions: outgoing and incoming. The state machine for the outgoing transaction is similar to the proxy policy except that it handles only one client branch. Incoming transaction's state machine is more simple as there is no client branch. The user agent library primarily focuses on the following two things:

Call control: A SIP/SDP multimedia call signaling is implemented using the state machine shown in Fig. A.6.

Outgoing registration refresh: The user agent library can perform outgoing registrations to the remote SIP registration servers such as sipd. A single one time registration does not need any state machine, but if one wants to implement an automatic registration refresh mechanism, the state machine shown in Fig. A.7 is useful. The state machine handles the

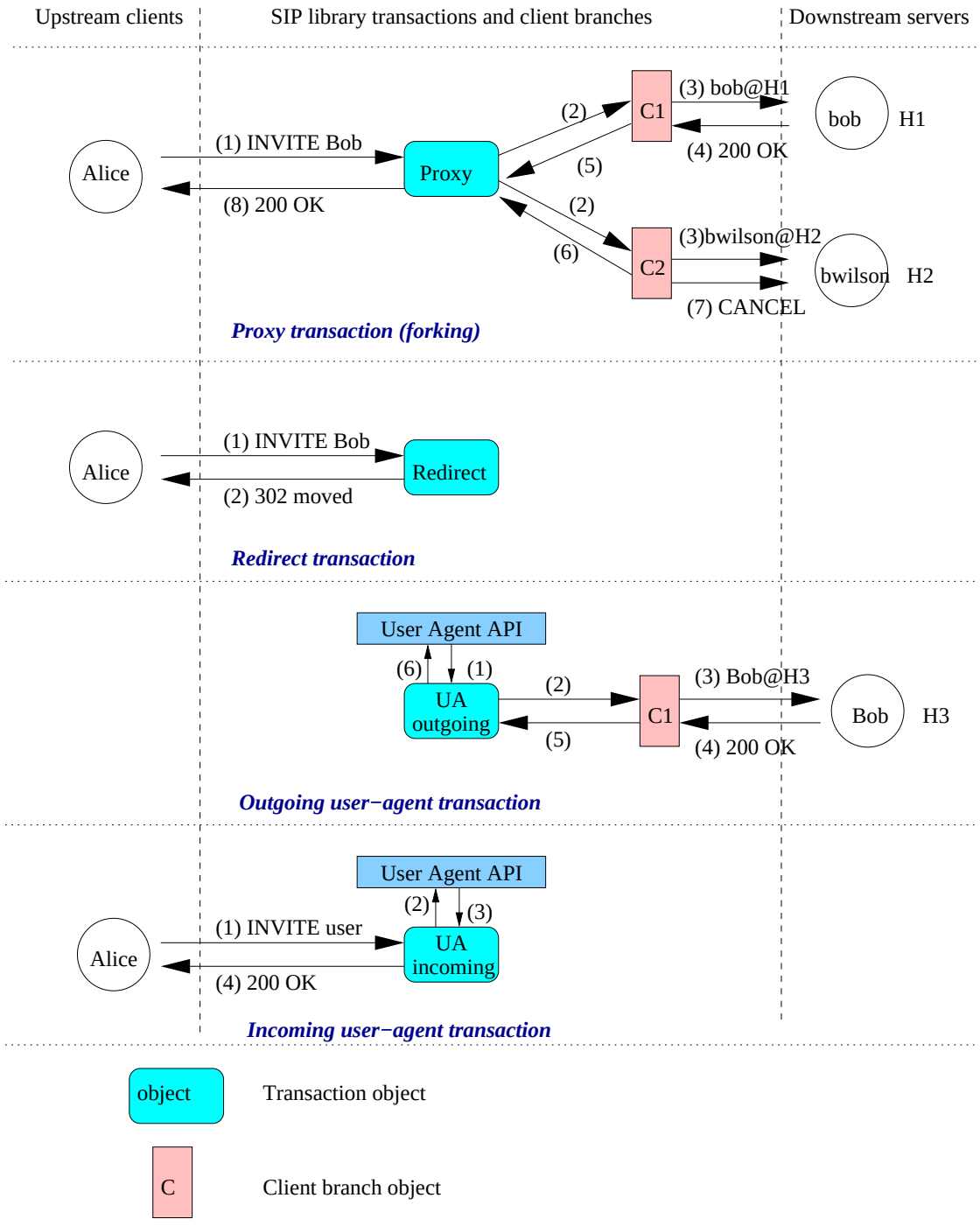


Figure A.5: SIP transaction and client branches

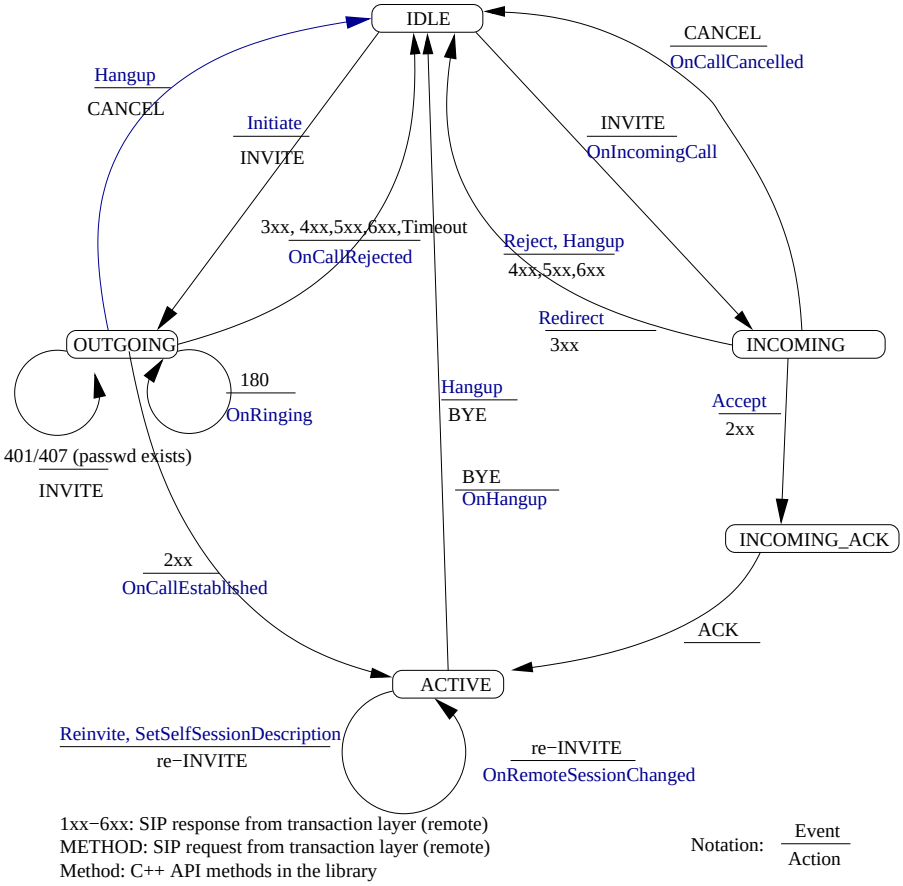


Figure A.6: call control state machine

outgoing SIP REGISTER method for an user agent.

The user agent library also has functions for parsing the SDP message body. However the media transport or encoding and decoding of the multimedia data is outside the scope of this library.

A similar state machine is implemented for other SIP methods like REFER for call transfer, SUBSCRIBE and NOTIFY for presence and event notification. Most of the other methods (e.g., DO and MESSAGE) do not require any state machine and can be used in the application without any libsip++ modification.

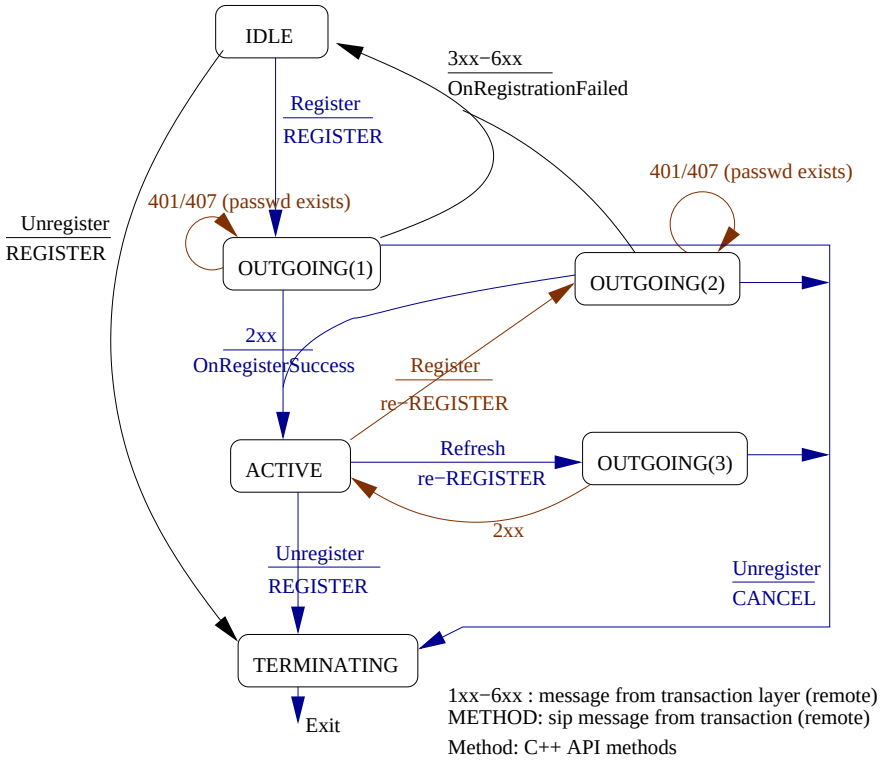


Figure A.7: Outgoing registration state machine

Design Philosophy

There are two popular kinds of API design for signaling protocols such as SIP: polling or events. Polling involves periodically checking if some event such as incoming call has occurred (e.g., Unix `select` call). This puts an unnecessary burden on the application to check for events. On the other hand a event driven system nicely fits in the call-back or event-based API.

In the event model there are various object-oriented designs. The JAIN community proposed a SIP API that is based on the listener and provider concept. For example, there can be a SIP provider object, which implements the SIP stack, and invokes specific methods from the listener object, to signal events. This kind of design is good for lower level SIP library where you define how to send or receive a message. Using this API, the application or higher layer has to to do lots of processing to maintain higher layer states such as transaction state or call state for such designs.

On the other hand, another model is to provide abstract classes for the notion of call or

end-point and let the application create the actual instances derived from these interfaces. Thus, the events are now handled in the object itself, instead of transferring to another listener object. Since the application derives from the abstract classes for call and end-point, it can implement more specific functions such as display alert on an incoming call. We have used this design model for our SIP user agent library (libsip++) as well as the wrapper for the OpenH.323 library.

Applications

Our SIP user agent library is meant for implementation of user agent type of applications. This includes, besides a traditional user agent, a conferencing server, an unified messaging system and a signaling gateway in CINEMA. I have written a SIP user agent, sipua, that is used for testing various SIP features of other applications such as proxy, and conference server. The P2P-SIP implementation, sippeer, also includes the functions of a SIP user agent. I have written a Java-based visualization tool for monitoring various peer nodes. The tool receives parameters such as node identifier and message exchange from various sippeer instances and displays the Chord-based P2P network graphically.

The total physical source lines of C/C++ code of various CINEMA components measured using SLOCCount [245] is about 187000, out of which my contribution is more than 60000 in C/C++, and an additional 30000 in Tcl.

Appendix B

Two-way Replication in MySQL

This section describes the steps needed to setup two-way replication in MySQL. Please refer to Fig. 3.8 (p. 35) for the following steps:

1. Edit `/etc/my.cnf` to set the unique server-id for D_1 and enable binary logging:

   ```
   [mysqld]
   server-id = 1
   log-bin
   ```

   Restart mysqld.

2. Create a replication user on D_1 with appropriate privileges for D_2's IP address.

   ```
   GRANT SELECT,PROCESS,FILE,SUPER,RELOAD,
    REPLICATION CLIENT,REPLICATION SLAVE ON
    *.* TO replication@"sip2.cs.columbia.edu"
    IDENTIFIED BY "somepassword";
   ```

3. Then copy the `data/sip` directory to `snapshot.tar` file

4. Get the master status (file name and position) of the binary log.

   ```
   SHOW MASTER STATUS;
   ```

   Suppose it shows file as `phone-bin.001` and position as 73.

5. Shutdown mysqld and start it again. Make sure no updates are happening in D_1 or D_2 while setting up the replication. Make sure D_2 is dead.

6. Create a replication user on D_2 similar to D_1, but with permissions for IP address of D_1, so that D_1 can access D_2.

7. Copy and uncompress the `snapshot.tar` from D_1 to the D_2 `data` directory. This will ensure the content the sip database of D_2 is same as that of D_1 when for the given master status of D_1. Some fields in the sip database, such as `cinema::sipdhost` should store the actual host name of the machine running the server. These fields can be populated with `sip2` for D_2 using the SQL_SLAVE_SKIP_COUNTER global in MySQL.

8. Edit `/etc/my.cnf` of D_2, similar to D_1, except that the server-id is 2 for D_2. (server-id values are not important as long as they are unique for a given replication setup.)

9. Start mysqld on D_2.

10. Setup D_2 as slave of D_1, by running following command on D_2:

```
CHANGE MASTER TO
 MASTER_HOST='phone.cs.columbia.edu',
 MASTER_USER='replication',
 MASTER_PASSWORD='somepassword',
 MASTER_LOG_FILE='phone-bin.001',
 MASTER_LOG_POS=73;
START SLAVE;
```

    The log file and position are same as that recorded from D_1. At this point we have D_1 to D_2 replication complete.

11. Now record the master status on D_2. Suppose it shows file as `sip2-bin.002` and position as 79.

12. Copy all the `*bin.*` (binary logs) from D_2's `data` directory to D_1's `data` directory.

13. Not set D_1 as slave of D_2 by running following command on D_1:

```
CHANGE MASTER TO
 MASTER_HOST='sip2.cs.columbia.edu',
 MASTER_USER='replication',
 MASTER_PASSWORD='somepassword',
 MASTER_LOG_FILE='sip2-bin.002',
 MASTER_LOG_POS=79;
START SLAVE;
```

At this point D_2 to D_1 replication is also complete. To allow access from other hosts, it may be required to remove the no-authentication line from the MySQL permissions table.

```
USE mysql;
DELETE FROM user WHERE User='';
FLUSH PRIVILEGES;
```

To bring up D_1 after a failover, tables on D_2 should be read-locked to prevent database inconsistency. In case failover messes up for some reason, the whole procedure can be repeated to setup the failover from scratch without losing the data in D_1.

Appendix C

Data Format for SIP-using-P2P

In this section we propose an XML-based data format for storing SIP-related information on the DHT for interoperability among different P2P-SIP implementations. The data format applies to both existing and planned authenticated DHT interfaces [27].

An example user contact of user bob@example.net stored in the DHT at key H(sip:bob@example.net) is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<contacts xmlns="urn:ietf:params:xml:ns:p2p-sip">
 <contact>sip:bob@192.1.2.3:5060</contact>
</contacts>
```

For unauthenticated DHT interface, we need the expires and user attributes as part of the contact information, so that the signature can not be misused as described in Section 5.5. These are not needed for the authenticated DHT interface, since they can be securely derived using other means such as ttl returned by get interface and DHT key, respectively. An example signed contact is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<contacts xmlns="urn:ietf:params:xml:ns:p2p-sip" Id="One"
  user="sip:bob@example.net">
 <contact display-name="Bob Wilson" expires="2006-01-31T18:22:38Z">
```

```
  sip:bob@192.1.2.3:5060
 </contact>
</contacts>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
 <SignedInfo>
  <CanonicalizationMethod
    Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
  <SignatureMethod
    Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
  <Reference URI="#One">
   <Transforms>
    <Transform
     Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
   </Transforms>
   <DigestMethod
     Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
   <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
  </Reference>
 </SignedInfo>
 <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
 <KeyInfo><KeyName>bob@example.net</KeyName></KeyInfo>
</Signature>
```

Any signature is formatted using W3C's Signature element [262]. The URI in Reference tag points to the data signed. The KeyName refers to the user identifier of the signer or the form user@domain.

The user's certificate is stored using the KeyInfo element [262] in the DHT at key H(certificate:bob@example.net) as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
```

```
 <X509Data>
  <X509SubjectName>
    CN=bob@example.net,O=P2P Inc.,ST=New York,C=US
  </X509SubjectName>
  <X509Certificate>MIID5jCCA0+gA...lVN</X509Certificate>
 </X509Data>
</KeyInfo>
```

A user Bob can subscribe for presence status of alice@home.com, by storing the following information in the DHT at key H(subscribe:alice@home.com).

```
<?xml version="1.0" encoding="UTF-8"?>
<watchers xmlns="urn:ietf:params:xml:ns:p2p-sip">
 <watcher event="presence" entity="alice@home.com"
 expires="2006-01-31T18:22:38Z">
   sip:bob@example.net
 </watcher>
 <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  ...
 </Signature>
</watchers>
```

Since this information needs to be encrypted, it gets stored as follows, using the W3C's EncryptedData element [263]:

```
<?xml version="1.0" encoding="UTF-8"?>
<EncryptedData Type="urn:ietf:params:xml:ns:p2p-sip#watchers
 xmlns="http://www.w3.org/2001/04/xmlenc#">
 <EncryptionMethod
   Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
 <ds:KeyInfo xmlns:ds="http://www.w3.org/2001/04/xmldsig#">
  <EncryptedKey CarriedKeyName="TempKey"
```

```
   xmlns="http://www.w3.org/2001/04/xmlenc#">
   <EncryptionMethod
    Algorithm="http://www.w3.org/2001/04/xmlenc#rsa1_5"/>
   <CipherData>
    <CipherValue>xyza21212sdfdsfs7989fsdbc</CipherValue>
   </CipherData>
  </EncryptedKey>
 <ds:KeyInfo>
 <CipherData>
  <CipherValue>A23B45C564587</CipherValue>
 </CipherData>
</EncryptedData>
```

An offline message is also stored as an EncryptedData element. The Type attribute refers to text or audio format for offline text or voice message, respectively.

Complete schema definition

The complete schema definition for urn:ietf:params:xml:ns:p2p-sip is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:ietf:params:xml:ns:p2p-sip"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:p="urn:ietf:params:xml:ns:p2p-sip"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">

 <import namespace="http://www.w3.org/XML/1998/namespace"
   schemaLocation="http://www.w3.org/2001/xml.xsd"/>

 <element name="contacts" type="p:contactsType"/>
```

```
<complexType name="contactsType">
 <sequence>
  <element name="contact" type="p:contactType"
   maxOccurs="unbounded"/>
 </sequence>
 <attribute name="Id" type="ID" use="optional"/>
</complexType>

<complexType name="contactType">
 <simpleContent>
  <extension base="anyURI">
   <attribute name="Id" type="ID" use="optional"/>
   <attribute name="user" type="anyURI" use="optional"/>
   <attribute name="display-name" type="string" use="optional"/>
   <attribute name="expires" type="dateTime" use="optional"/>
   <attribute name="priority" type="p:priority" use="optional"/>
  </extension>
 </simpleContent>
</complexType>

<simpleType name="priority">
 <restriction base="decimal">
  <pattern value="0(.[0-9]{0,3})?"/>
  <pattern value="1(.0{0,3})?"/>
 </restriction>
</simpleType>

<element name="watchers" type="p:watchersType"/>
```

```
 <complexType name="watchersType">
  <sequence>
   <element name="watcher" type="p:watcherType"
     maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Id" type="ID" use="optional"/>
 </complexType>

 <complexType name="watcherType">
  <simpleContent>
   <extension base="anyURI">
    <attribute name="Id" type="ID" use="optional"/>
    <attribute name="entity" type="anyURI" use="optional"/>
    <attribute name="expires" type="dateTime" use="optional"/>
   </extension>
  </simpleContent>
 </complexType>

</schema>
```

Appendix D

Implementation Details of SIP-H.323 Interworking Function

In this chapter, we list the implementation requirements and details of SIP-H.323 interworking function (IWF). This is an appendix to Chapter 11.

In this chapter, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [264].

D.1 Implementation Requirements

This section lists the messages which MUST be supported by the signaling IWF. It also highlights the typical values for parameters for the messages.

H.323 (H.225.0 and H.245)

All the messages which are mandatory in the Q.931 portion of H.225.0 and H.245 MUST be supported. RAS is optional; if used, all messages that are mandatory in RAS MUST be supported. Parameter values (if not specified in this document) MUST be derived from H.225.0 version 2.0 and H.245 version 4.0 for Q.931 and H.245 messages, respectively. This assures that requirement 1 in Section 11.1 is fulfilled.

Handling of Q.931 Messages

The IWF SHOULD support the Q.931 messages listed in Table D.1. An entry of "not applicable" in the table means that it is not visible to the SIP endpoint and is only local to the IWF's H.323 stack.

Message	IWF sends to H.323	H.323 sends to IWF
Alerting	Supported	Supported
Call proceeding	Supported	Supported
Connect	Supported	Supported
Progress	Not applicable	Not applicable
Setup	Supported	Supported
Setup Ack	Not applicable	Not applicable
Release Complete	Supported	Supported
User Information	Not applicable	Not applicable
Information	Not applicable	Not applicable
Notify	Not applicable	Not applicable
Status	Not applicable	Not applicable
Status Inquiry	Not applicable	Not applicable
Facility	Not applicable	Not applicable

Table D.1: Support for Q.931 messages
A "Not applicable" entry in the table means that it is not visible to the SIP endpoint and is only local to the IWF's H.323 stack.

The IWF MUST NOT close the call signaling channel after the call is established. However, if the call is routed through a gatekeeper and the gatekeeper closes the call signaling channel, the IWF MUST comply with H.323 and MUST NOT assume that the call is closed as long as H.245 channel is open. If the Q.931 TCP connection is closed without closing the call signaling channel, then the IWF SHOULD try reopening the TCP connection, as specified by H.323. In case of failure such as TCP connection refused or TCP connection timeout, the IWF SHOULD close the call on the SIP side also by sending a BYE.

Q.931-specific information elements, other than user-user information element (UUIE), do not affect the operation of this IWF, however they are required for interoperation with other H.323 entities. The specific fields of UUIE used in translating to SIP message are given in Appendix D.3.

Bearer Capability: Information transfer capability (octet 3, bits 0–5): Unless some other restrictions apply (e.g., the IWF is connected to a bandwidth-restricted ISDN network), the parameter SHOULD be set to "unrestricted digital information" or "restricted digital information" on outgoing side. If the IWF knows that the call is going to be voice only, it may choose to set it as "speech" or "3.1 kHz Audio". The IWF ignores this field on incoming requests.

Information Transfer Rate and Rate multiplier: If bandwidth information is available from the gatekeeper or some external means (e.g., from bandwidth information in SDP message), then information transfer rate and rate multiplier may be set to values reflecting the bandwidth, else they should be set to some high value as appropriate. This way the bandwidth is not limited to 64 kb/s or 128 kb/s. On the incoming side these values SHOULD be ignored. Note that in a Q.931 message the only possible values are multiples of 64 kb/s.

Layer 1 protocol (octet 5, bits 1–5): For outgoing Q.931 messages, the parameter is set to H.221 ('00101'), indicating an H.323 video phone call, unless the IWF knows that the call is going to be voice only (e.g., if this is hardcoded in the IWF). In that case, it may encode the parameter as G.711 A-law or mu-law to indicate this.

For incoming Q.931 messages, the IWF ignores this field.

Calling or Called party number: For outgoing Q.931 messages, the IWF translates the SIP request-URI into an e164 number, as described in Section 11.3. The calling/called party subaddress is not included in Q.931 messages originating from the IWF.

For incoming Q.931 messages, the IWF relies on user-user information element for addresses (e.g., sourceAddress and destinationAddress fields of UUIE) and ignores the Q.931 parameter. However, if the calling/called party number is present and e164-ID is not present in the H.323 Alias Address then the calling/called party number is used instead of e164-ID while translating address in section 11.3.

> H.323 specifies that the called and calling party Subaddress fields are needed for some circuit switched call scenarios and they SHOULD NOT be used for packet based network side only calls.

Display: For incoming Q.931 messages, the IWF MAY copy the Display IE to the display pa-

SIP	status	releaseCompleteReason
400	Bad Request	undefinedReason
401	Authentication Required	noPermission
402	Payment Required	undefinedReason
403	Forbidden	noPermission
404	Not Found	unreachableDestination
406	Not Acceptable	undefinedReason
407	Proxy Authentication Required	noPermission
409	Conflict	undefinedReason
410	Gone	undefinedReason
413	Request Entity Too Large	undefinedReason
414	Request-URI Too Large	badFormatAddress
415	Unsupported Media Type	undefinedReason
420	Bad Extension	badFormatAddress
480	Temporarily not available	unreachableDestination
483	Too Many Hops	undefinedReason
484	Address Incomplete	badFormatAddress
485	Ambiguous	badFormatAddress
486	Busy Here	destinationRejection
600	Busy Everywhere	destinationRejection
603	Decline	destinationRejection
604	Does not exist anywhere	unreachableDestination

Table D.2: Mapping between SIP status codes and reason fields

rameter of the SIP To header field.

Similarly, for outgoing Q.931 messages, the Display parameter MAY be copied from the display parameter of the SIP To field.

Cause: For incoming Q.931 messages, the Q.931 Cause information element and/or the UUIE reason field are mapped to the appropriate SIP status response code, as described in Table D.2. H.225.0 [248] specifies that either the Cause information element or the releaseCompleteReason MUST be present. It also gives a mapping between the Cause information element and the releaseCompleteReason. Table D.2 gives the mapping between releaseCompleteReason and the appropriate SIP status response.

Similarly, for outgoing Q.931 messages, the Q.931 Cause information element and the UUIE reason field are derived according to Table D.2.

User-User-Information-Element: Below, we detail the fields in UUIE which are relevant to H.323-SIP conversion. Other fields are interpreted as defined by H.225.0.

sourceInfo/destinationInfo: In all messages from the IWF, this field SHOULD be set to indicate that this endpoint is a gateway. However, the sequence of supported protocols in "GatewayInfo" may be empty.

H.245SecurityMode, tokens, cryptoTokens: These fields are interpreted as in H.323. Note that since H.245 is terminated at the IWF, this kind of security information is not relevant to the SIP cloud.

fastStart: FastStart PDUs contain the OpenLogicalChannel (OLC) messages. The IWF converts incoming OLC messages to a SDP message body. One SDP media description line ("m=") is generated for each distinct session-ID. All logical channels with same session-ID appear as payload types in a single SDP media description line. When converting SIP to H.323, the SDP message is converted to a list of OpenLogicalChannel messages, one per payload type. H.323 endpoint will select at most one OLC per session-ID. This selected OLC is returned by the H.323 endpoint in the fastStart field of Q.931 Connect message. When converting H.323 to SIP, each OLC in fastStart corresponds to a payload type of SDP. All the OLC messages with same session-ID form a single media description ("m=") line.

The parameters for the Q.931 SETUP message are listed below.

sourceAddress: Converted to/from SIP header From field as described in section 11.3.

destinationAddress: Converted to/from SIP header To field as described in section 11.3.

destCallSignalAddress: If the To SIP header field contains a numeric host identifier then destCallSignalAddress is set to the IPv4 address represented by the numeric identifier.

conferenceGoal: Set to "create" in outgoing Q.931 messages. (Additional values may be supported in future versions of this specification that support conferencing.)

remoteExtensionAddress: Not present in outgoing Q.931 messages. For incoming Q.931 messages, this parameter is combined with the DestinationAddress parameter to generate the SIP To header field and the request-URI.

mediaWaitForConnect: Set to “false” in outgoing Q.931 messages. Ignored in incoming Q.931 messages, as media transmission is transparent to the IWF.

canOverlapSend: Set to “false” in outgoing Q.931 messages and ignored in incoming Q.931 messages since this version of the specification does not support overlap sending.

Use of the Q.932 facility message for call redirection is for further study.

Handling H.245 Messages

Table D.3 details how an IWF handles H.245 messages. An entry of “not applicable” means that the message does not affect the behavior within the SIP cloud.

The remainder of this subsection lists the possible values of some of the fields of H.245 messages. Refer to H.245 version 4.0 for description and details of the ASN.1 structures for H.245.

MasterSlaveDetermination: The terminalType parameter is set to indicate that this terminal is a gateway. H.323 specifies a set of numerical values of terminalType for different types of terminals. For example, a gateway without a multipoint controller (MC) has a terminalType of 60; A gateway with a MC and no multipoint processor (MP) has a terminalType value of 80. Other values of terminalType are not relevant to this IWF in the case where media traffic is transparent. See H.323 [37] for other possible values of terminalType.

TerminalCapabilitySet: multiplexCapability::h2250Capability: maximumAudioDelayJitter should be set to max possible value as specified by H.323. MultipointCapabilities should reflect minimum capability of Centralized Control/ Audio/ Video/ Data. Other conferencing capabilities are for further study. RTCP videoControlCapability should be set to false because anyway H.245 indications have to be used for this purpose.

Message	REQUIRED or Not applicable
MasterSlaveDetermination/Ack/Rej/Rel	Not Applicable
TerminalCapSet/Ack/Reject/Release	REQUIRED
Send TerminalCapabilitySet	Not Applicable
OpenLogicalChannel/Ack/Reject	REQUIRED
OpenLogicalChannelConfirm	Not Applicable
CloseLogicalChannel/Ack	REQUIRED
RequestChannelClose	OPTIONAL
RequestMode/Ack/Rej/Rel	RECOMMENDED
RoundTripDelayReq/Res	Not applicable
MaintenanceLoopReq/Ack/Reject	Not supported
MaintenanceLoopOffCmd	Not supported
CommunicationModeReq/Res/Cmd	For further study
ConferenceReq/Res/Cmd/Indic	For further study
EndSessionCommand	REQUIRED
FlowControlCommand	For further study
Encryption Command	For further study
Jitter Indication	For further study
User Input	OPTIONAL
H2250MaxSkewIndic	For further study
MClocationIndication	For further study
FunctionNotUnderstood	Not Applicable
FunctionNotSupported	Not Applicable
vendorIdentifier	Not Applicable
MiscCommand/Indication	For further study

Table D.3: Support for H.245 messages.
An entry of “not applicable” means that it is not visible to the SIP endpoint and is only local to the IWF’s H.323 stack.

MediaPacketizationCapability should contain the information about the dynamic payload types used by SIP endpoint. Transport Capability should be absent. redundancyEncodingCapability should be absent as this is not supported in this version. logicalChannelSwitchingCapability may be supported by the IWF’s H.323 stack. This makes mapping SIP re-INVITE easier. t120DynamicPortCapability is set to false because T120 data is not supported in this version.

CapabilityTableEntry and

CapabilityDescriptor are mapped from the session description given by SDP. A single

capability descriptor is used in H.245. All the payload types on a single media description line (m=) are combined to form an alternative capability set in H.245. All such media description lines are combined to form a simultaneous capability set (or a capability descriptor). Mapping multiple SDP received in multipart body of SIP to multiple capability descriptor is for further study.

Capability: H233Encryption is not applicable.

H235Security is not applicable.

DataApplication capability is not supported in this version of the specification.

ConferenceCapability is for further study and is not supported in this version of the specification.

UserInputCapability may be supported by the IWF. This is used to convey DTMF digits. Use of the SIP INFO method is being considered for this purpose.

maxPendingReplacementFor is not applicable.

Audio and Video: A capability in H.323 represents a payload type. Refer to http://www.iana.org/assignments/media-types/media-types for a list of MIME types and http://www.iana.org/assignments/rtp-parameters for a list of static RTP payload types. Use of static RTP payload types in SDP is discouraged. The IWF should maintain a list of all currently available payload types and media formats and the corresponding RFC numbers. (An intelligent IWF MAY periodically download and parse these HTML pages to update its database).

The predefined audio and video capabilities are mapped to appropriate media format and RTP payload type. This mapping is given in this document for ease of reference. This mapping should be used by the IWF to convert the H.323 capability to an SDP media description. When converting from H.323 to SDP, the IWF SHOULD use dynamic payload type. When converting from SDP to H.323, the IWF SHOULD NOT use dynamic payload types because many current implementations do not support these. However, the IWF MUST be able to receive dynamic payload types, in both H2250Capability.MediaPacketizationCapabilty.RTPPayloadType

H.323	IANA	payload type	clock/channels	RFC
g711Alaw64k	PCMA	8	8000/1	RFC1890
g711Ulaw64k	PCMU	0	8000/1	RFC1890
g711Alaw56k	N/A			
g711Ulaw56k	N/A			
g722-64k	G722	9	8000/1	RFC1890
g722-56k	N/A			
g722-48k	N/A			
g7231	G723	4	8000/1	None
g728	G728	15	8000/1	RFC1890
g729	G729?	Dynamic/18?	8000/1	-
g729AnnexA	?	Dynamic	8000/1	?
g729wAnnexB	?			
g729AwB	?			
g7231AnnexC	?			
gsmFullRate	GSM	3	8000/1	RFC1890
gsmHalfRate	GSM-HR	Dynamic	8000/1	-
gsmEnhFullRate	GSM-EFR	Dynamic	8000/1	-

Table D.4: Audio capability mapping

and in H2250LogicalChannelParameters.MediaPacketization. When dynamic RTP payload type are used, H225LogicalChannelParameters.dynamicRTPPayloadType MUST match the payload type description given in mediaPacketization.

AudioCapability: A subset of IANA-registered formats and H.323-supported capabilities are listed in Table D.4.

Note that H.323 only supports a clock rate of 8000 Hz; other values cannot be mapped to H.323. SDP attribute "ptime" gives the maximum length of time in milliseconds represented by media in a packet. This can be used for defining the maximum packet length.

VideoCapability: The mapping of video encodings is shown in Table D.5. The Video MPI (Mean Picture Interval) is mapped to the SDP attribute "framerate" as follows:

mpi = 30 / framerate

It is assumed that 29.97 Hz is rounded to 30 Hz when calculating the framerate. So MPI of 1 become framerate 30.0, similarly MPI of 2 becomes framerate 15. However,

H.323	IANA	Payloadtype	clock	RFC
h261VideoCap	H261	31	90000	RFC2032
h262VideoCap	?			
h263VideoCap	H263/H263+?	34	90000	RFC2190/2429?

Table D.5: Video capability mapping.

the IWF shall do proper rounding error correction on the incoming side. So framerate of 29.97 should also map to MPI of 1. Note that in SDP any possible value for framerate is allowed, but in H.323 only multiples of 1/29.97 are allowed. The IWF should convert the framerate to the next lower value allowed in H.323. For example, a framerate of 12.3 frames per second in SDP is converted to an MPI value of 3 which is equivalent to 10 frames per second.

DataApplicationCapability: Not supported in this version of the specification.

Use of RSVP (Resource reservation protocol) to handle QoS (Quality of service) is for further study.

D.2 Signaling Address Translation

A SIP address can be either a SIP URL or any URI. This document only describes the translation of the SIP (“sip:”), telephone (“tel:”) and H.323 (“h323:”) URL schemes.

The BNF of a SIP address is given below for reference:

```
SIP-Address     =   (name-addr | addr-spec)
name-addr       =   [display-name] "<" addr-spec ">"
addr-spec       =   SIP-URL
SIP-URL         =   "sip:" [ userinfo "@" ] hostport url-parameters
                    [headers]
userinfo        =   user [ ":" password ]
hostport        =   host [ ":" port ]
host            =   hostname | IPv4address | IPv6address
url-parameters  =   *(";" url-parameter)
url-parameter   =   user-param | ...
```

In the url-parameter, only the user-param parameter is relevant. The user name may be a telephone number.

H.323 addresses are typically sequences of Alias Addresses (see H.225.0 [248]). The ASN.1 description of an H.323 Alias Address is:

```
H323-Alias-Address ::= CHOICE
{
  e164    IA5String (SIZE(1..128)) (FROM("0123456789\#*,")),
  h323-ID BMPString (SIZE (1..256)),
  ...,
  url-ID  IA5String ( SIZE(1 .. 512)), -- URL Style address
  transport-ID TransportAddress,       -- IPv4, IPv6, IPX etc.,...
  email-ID IA5String (SIZE(1..512)),   -- rfc822 compliant email address
  partyNumber PartyNumber
}
```

The PartyNumber parameter is not described in this document. Telephone numbers can be conveyed via e164 field of H323-Alias-Address or called/calling party number fields of Q.931 message.

D.2.1 Converting SIP Addresses to H.323 Addresses

h323-ID

The SIP-Address is stored as is in the h323-ID of the Alias Address. If the SIP-Address contains more than 256 characters, only the addr-spec part is copied. If the addr-spec exceeds 256 characters, the IWF generates a SIP response of 414 (Address Too Long). Each BMP[1] character in h323-ID stores the corresponding text character in the SIP Address.

The h323-ID MUST always be generated so that a terminal running version 1.0 of H.323 (which supports only e164 and h323-ID, but does not support transport-ID, url-ID or email-ID) can still decode the address.

[1] BMP stands for basic multilingual plane, i.e., Basic ISO/IEC 10646-1 (unicode) character set.

e164

If the SIP-Address's user is a telephone-subscriber, user-param is set to phone and the user part does not contain a "w", it is converted to the e164 field of Alias-Address. The e164 field only allows characters from the set "0123456789#*,". Thus, any leading "+" is removed from the SIP telephone-subscriber part, as are any visual separators "-" and ".". The pause "p" is replaced with ",".

url-ID

The SIP-URL part of the SIP address is copied verbatim to the url-ID parameter. If the SIP URL exceeds 512 bytes in size, the IWF generates the SIP status 414 (Address too long).

email-ID

The user and host parts are used to generate an email identifier, as in "*user@host*", which is stored in the email-ID field of AliasAddress. If the size exceeds 512 characters, the IWF generates the SIP status 414 (Address Too Long).

transport-ID

If the host part of the SIP-URL is indicated as a dotted quad, e.g., 192.1.2.3, it is translated into a transport-ID. If a port parameter is present in the SIP address, the number is used. Otherwise, the port number depends on the context. For example, for the destination address of H.323 SETUP messages, it is set to 1720, otherwise it is set to 0.

> Although a numeric IP address requires no further address resolution, it is worth noting that other fields (e164, url-ID, h323-ID) are also needed. If the destination is a VoIP gateway, for example, then an Internet telephony gateway destination is mapped from the e164 field or the called party number.

Examples

- The SIP Address “sip:j.doe@big.com” is converted to an H.323 Address sequence with three elements: { h323-ID = “sip:j.doe@big.com”, url-ID = “sip:j.doe@big.com”, email-ID = “j.doe@big.com” }
- The SIP Address “sip:+1-212-555-1212:1234@gateway.com; user=phone” is converted to the H.323 Address: { e164 = “12125551212”, h323-ID = “sip:+1-212-555-1212:1234@gateway.com”, url-ID = “sip:+1-212-555-1212:1234@gateway.com”, email-ID =“+1-212-555-1212:1234@big.com”}
- The SIP Address “sip:alice@10.1.2.3” is converted to H.323 Address: { h323-ID = “sip:alice@10.1.2.3”, url-ID = “sip:alice@10.1.2.3”, transport-ID = IPAddress 10.1.2.3:1720, email-ID = “alice@10.1.2.3” }
- The SIP Address “A. Bell <sip:a.g.bell@bell-tel.com>” is converted to H.323 Address: { h323-ID = “A. Bell <sip:a.g.bell@bell-tel.com>”, url-ID = “sip:a.g.bell@bell-tel.com”, email-ID = “A. Bell <a.g.bell@bell-tel.com>” }

D.2.2 Converting H.323 Addresses to SIP Addresses

In H.323, addresses are typically a sequence of Alias Addresses (referred to as H.323 addresses in this chapter). Since it is not possible to convert all the different representations of the address to a single SIP Address, the IWF will have to drop some of the addresses. However, an IWF MAY try more than one converted addresses either sequentially or in parallel.

The conversion is done in the following order. If the conversion succeeds in one step, the conversion concludes and the remaining steps are ignored. If a url-ID is present and it is a SIP-URL, then it is used as is in the SIP Address. If an h323-ID is present and it can be parsed as a valid SIP-Address, it is used. This is needed when talking to an H.323 terminal running version 1.0. If the transport-ID is present and it does not identify the IWF, then it forms the hostport portion of the SIP URL and the user portion is constructed using h323-ID or e164. If the email-ID is present, then it is used in the SIP-URI. The email-ID is prefixed by the scheme name “sip:”.

If all these efforts fail, then the IWF MAY attempt to construct a legal SIP Address using the information available. For example h323-ID may become the display-name, e164 may become the user and host may be some default domain name.

D.3 Detailed Description of IWF Behavior

This section describes how messages are processed by a SIP–H.323 signaling IWF. The discussion is split into two subsections, with SIP-originated requests discussed in Section D.3.1 and H.323-originated requests in Section D.3.2. Only fields relevant to the conversion are presented here. Other parameters are specific to either H.323 or SIP and can be generated by the respective protocol engine in the IWF without conversion.

The IWF maintains, apart from other call-state information, the capability sets and operating mode for each call. Capability sets are maintained for each H.323 and SIP endpoint, both receive and transmit directions. Operating mode contains the modes in each direction (SIP to H.323 and H.323 to SIP).

D.3.1 SIP-originated Requests

IWF Receives REGISTER

The IWF sends a RAS RRQ message to the H.323 GK, where the callSignalAddress is the address of the IWF, the terminalType is set to “gateway” and the terminalAlias is mapped from the To header of the REGISTER request.

The IWF stores the SIP Contact header field. A “200 OK” SIP status response is sent after receiving a RAS RCF message.

IWF Receives INVITE for a New Call

The IWF MAY respond with a 100 (Trying) response to the SIP entity that sent the INVITE request. It stores the SDP information as the terminal’s SIP capability and converts the capability to H.245 format.

If the IWF is registered with a gatekeeper, sends a RAS ARQ message to the gatekeeper, where the destinationInfo and destCallSignalAddress is derived from the To SIP header, the srcInfo is derived from the From SIP header field and srcCallSignalAddress is the call signaling address of the IWF itself. The gatekeeper assigns an endpointIdentifier during registration. That value of endpointIdentifier is used in the endpointIdentifier field of the ARQ message.

Next, the IWF should receive either a RAS ACF or ARJ message. If an ACF message is received, establish a Q.931 channel as described below. If an ARJ message is received, the behavior depends on the reason parameter:

CalledPartyNotRegistered: The IWF responds with 404 (Not Found).

callerNotRegistered: The IWF MAY register, with a RAS RRQ message, the SIP address with the gatekeeper and then retransmit the RAS request, with the endpointIdentifier returned in RCF. Alternatively, it MAY send a 400 (Caller not registered) response to the SIP entity.

incompleteAddress: Send 484 (Address Incomplete) response to SIP entity.

Other reasons: Send 400 (H.323 translation failure) response to SIP entity.

If the IWF times out waiting for an ARQ response, it sends a SIP 504 (Gateway time-out) response.

If the IWF is not registered with a gatekeeper and it is able to resolve the SIP address to a H.323 address or if the IWF is registered and has received an ACF for the registration request from the gatekeeper, the IWF sends a Q.931 SETUP message to the H.323 entity, where the sourceAddress is derived from the SIP From header, the destinationAddress is derived from the SIP To header or from the RAS ACF response, destCallSignalAddress is derived from the RAS ACF response or from the To SIP header. The remoteExtensionAddress is copied from RAS ACF if present or extracted from To SIP header if possible. sourceCallSignalAddress is the call signaling transport address of the IWF. fastStart PDUs are mapped from the session description in the INVITE message body.

Each SDP payload type entry is converted to an OLC message. All the payload types on the SDP same media description line have the same session id in the OLC messages. This

identifies them as belonging to the same group and the receiving H.323 entity will select one of these.

If the IWF receives a Q.931 CallProceeding message, it sends a 100 (Trying) response to the SIP entity, if not already sent. If fastStart PDUs are present, it stores them.

If the IWF receives a Q.931 Alerting message, it sends a 180 (Alerting) response to the SIP entity, indicating that the final destination is ringing. If fastStart PDUs are present, it stores them.

If the IWF receives a Q.931 Connect message, the behavior depends on whether a FastStart indication is present.

If a FastStart indication is present, the IWF maps the received OLCs to the SDP payload types contained in the original INVITE request. Format a new SDP packet with more constrained media description and correct media transport address of the H.323 entity. Now each media description line will contain a single payload type, depending on which OLC PDUs are present. The operating mode and H.323 capability set are set to this reduced set of payloads.

The SDP message is sent in a 200 (OK) response. The IWF then waits for the ACK request from the SIP entity. If the IWF times out, it declares the call closed and terminates the H.323 call. Once an ACK has been received, the IWF may proceed with other H.245 signaling (CESE, RTDSE and so on).

If the H.323 entity does not support FastStart, the IWF proceeds with H.245 signaling as described below. First, it sends a TCS to the the H.323 entity and uses the stored SIP capability set to generate the H.245 capabilities.

If the IWF receives an H.245 TCS message, it updates the H.323 capability set and calculates maximal intersection of H.323 and SIP capability sets (call this C). Derive a suitable operating mode from C (say, M). For each element in M (for the data from the SIP UA to the H.323 terminal), send an H.245 OLC message to the H.323 entity. Use the transport address of the SIP capability set, derived from the SDP received in the original INVITE message.

If the IWF receives an OLC message and the logical channel is present in the operating mode from the H.323 terminal to the SIP UA, the IWF sends an OLCAck to the H.323 terminal. The OLCAck contains the transport address from the SIP capability set, again derived from the

SDP in the INVITE message body. If the logical channel is not present in that operating mode, the IWF sends an OLCReject.

Once the IWF has received an OLCAck or OLCRej for all outstanding OLC requests, it updates the operating mode and sends a 200 (OK) response to the SIP entity. The session description in that response is formed using the new operating mode and the transport addresses received in the H.245 OLCAcks.

The IWF should wait for the ACK request from the SIP entity. If the IWF times out, it should close the H.323 call. This concludes the description of the non-FastStart handling.

If, at any time, the IWF receives a Q.931 ReleaseComplete message, a H.323 call could not be established. The IWF sends a 400 (Client Failure) with reason phrase “H.323 call failed”.

If the Q.931 SETUP times out, the IWF sends a 504 (Gateway time-out) response.

If the SIP address is not resolved to an H.323 address, send a 501 (Not Implemented) response to SIP entity.

IWF Receives INVITE for Existing Call

- Update the SIP capability set.
- Recalculate the operating mode, minimizing changes. An H.245 Mode Request message is sent if the operating mode has changed. If the Mode Request fails, either close the media channel or the call.

IWF Receives BYE Request

The IWF sends an H.245 Endsession to the H.323 entity. Upon receipt of a response or on timeout, the IWF sends a Q.931 ReleaseComplete to H.323 entity. If the call was admitted by a GK, send a RAS DRQ (Disengage Request) message to the GK.

IWF Receives OPTIONS Request

There is no equivalent message in H.323 to query H.323 capabilities without establishing the call.

D.3.2 H.323-Originated Requests

IWF Receives RAS GRQ

The IWF sends a RAS GCF (Gatekeeper Confirm) response to GRQ (Gatekeeper Request) only if the IWF also contains a gatekeeper implementation.

IWF Receives RAS RRQ

This is possible only if the IWF also contains a gatekeeper implementation. On receipt of RRQ (Registration Request) the IWF sends a SIP REGISTER message to the SIP server where the To SIP header field is derived from the terminalAlias parameter; the Contact SIP header field indicates the IWF's location. The callSignalAddress received in RRQ message is stored internally by the IWF. The IWF may send multiple REGISTER requests if the sequence of terminalAlias can be mapped to multiple SIP addresses

Once the IWF receives a 2xx response to this REGISTER, it sends a RAS RCF (registration confirmation) message to the H.323 entity. If it receives any other status response or the REGISTER request times out, the IWF sends a RRJ (registration reject) to the H.323 entity.

IWF Receives RAS ARQ

This is possible only if the IWF also contains a gatekeeper implementation. Receipt of this message indicates that the H.323 entity knows that the destination is reachable via this IWF. One simple implementation is to accept the admission request giving the callSignalAddress of the IWF itself. Alternatively, a procedure similar to that given for RAS LRQ, below, can be followed.

IWF Receives RAS LRQ

If the IWF receives a RAS LRQ (Location Request) message, the IWF sends an OPTIONS message to the SIP entity, where the SIP entity address is resolved from the H.323 address. The To SIP header field is derived from the destinationAddress. The IWF MAY send multiple forking OPTIONS requests if the sequence of destinationAddresses can be mapped to multiple SIP addresses.

If it receives a 2xx response for the OPTIONS request, it sends a RAS LCF message to the H.323 with the CallSignalAddress of the IWF itself. If any other response is received or the request times out, the IWF MAY choose to remain silent or it may send a RAS LRJ to the H.323 entity.

IWF Receives a Q.931 Setup

The IWF generates an ARQ/ACF sequence if required here as per H.323 standard. However, that is local to the H.323 stack and does not affect translation.

If fastStart is present, convert it to H.323 capability set, else build some default H.323 capability set. The IWF MAY send a Q.931 CallProceeding message to H.323 entity.

The IWF then sends an INVITE, where the To SIP header field is derived from the Q.931 destinationAddress and/or destCallSignalAddress. If destinationAddress is the IWF itself, then use remoteExtensionAddress. The From SIP header field is derived from sourceAddress and/or srcCallSignalAddress. The session description is constructed from the H.323 capability set.

If the IWF receives a 2xx response for the INVITE, it updates the SIP capability set using the session description in the response body. It then sends a Q.931 Connect message to the H.323 entity.

Then, the IWF sends an ACK request to the SIP entity.

Then, it sends an H.245 TCS to the H.323 entity using the SIP capability set.

If it receives a TCS, it updates the H.323 capability set and calculates the maximal intersection of the H.323 and SIP capability sets, called C. From C, the IWF derives a suitable operating mode (say M). For each element in M in the direction from SIP to H.323, send a H.245 OLC to the H.323 entity. The OLC messages use the transport addresses of the SIP capability set, derived from the session description in the 2xx response body.

If the IWF receives an OLC and the logical channel is present in the operating mode from H.323 to SIP, it responds with an OLCAck. The OLCAck uses the transport addresses of the SIP capability set. If the logical channel is not present in the operating mode, the IWF sends an OLCReject

Once the IWF has received OLCAck or OLCRej for all the requests, it updates the operating mode. Then, the IWF sends a re-INVITE. The session description is formed using the new operating mode if it is different from what was sent in the first INVITE message and the transport addresses received in OLCAcks. The IWF should wait for a 2xx response from the SIP entity and respond with an ACK request. If it times out or if it fails, it should close the call.

If the IWF receives a 180 (Alerting) SIP response, it sends a Q.931 Alerting message to the H.323 entity.

If the IWF receives any other 1xx SIP response, it sends a Q.931 CallProceeding message to H.323, but only if one was not already sent for this call.

If no response is received or a failure response is received, the IWF sends a Q.931 ReleaseComplete message to the H.323 entity.

IWF Receives Mode Request or Change in Logical Channels

Update operating modes, Send re-INVITE to SIP entity. If that fails then reject the Mode Request or Open Logical Channel request.

IWF Receives H.245 EndSession

If the IWF receives a H.245 EndSession, it closes the H.245 call. Send H.245 EndSession and Q.931 ReleaseComplete to H.323 entity and send RAS DRQ to gatekeeper if it admitted the call.

IWF Receives Q.931 ReleaseComplete

If the IWF receives a Q.931 ReleaseComplete, the H.323 side of the call is closed. The IWF sends a BYE to the SIP entity if the call has been established.

IWF Receives RAS DRQ

If the call is active, close it. Send RAS DCF (disengage confirm) to H.323 entity.

IWF Receives RAS URQ

If the IWF receives a RAS URQ (unregister request) message, the behavior depends on whether the IWF also acts as a gatekeeper. If the IWF also contains a gatekeeper, unregister the endpoint as specified by RAS. Otherwise the request must have come from a gatekeeper. Close all the associated calls on both SIP and H.323 sides and send a RAS UCF (unregister confirm) to the H.323 entity.

Appendix E

Glossary

3GPP	Third Generation Partnership Project
AAA	Authentication, Authorization and Accounting
ACD	Automatic Call Distribution
ACL	Access Control List
API	Application Program Interface
ARP	Address Resolution Protocol
B2BUA	Back-to-Back User Agent
BHCA	Busy Hour Call Arrivals (or Attempts)
CAN	Content Addressable Network
CGI	Common Gateway Interface
CINEMA	Columbia InterNet Extensible Multimedia Architecture
CPL	Call Processing Language
CPS	Calls Per Second
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DHT	Distributed Hash Table
DID	Direct Inward Dialing
DNS	Domain Name System (or Service or Server)

DoS	Denial of Service (attack)
DTD	(XML) Document Type Definition
DTMF	Dual-Tone Multiple Frequency
ENUM	Telephone Number Mapping
FIA	(VoiceXML) Form Interpretation Algorithm
GMT	Greenwich Mean Time
GSTN	Global (or Global) Switched Telephone Network (same as PSTN)
H.323	ITU-T recommendation for multimedia communication over packet networks
HTML	Hyper-Text Markup Language
HTTP	Hyper-Text Transport Protocol
MD5	Message Digest version 5
IETF	Internet Engineering Task Force
IM	Instant Message (or Messaging)
IMS	(3GPP's) IP Multimedia Subsystem
I/O	(Device) input and output
IOS	(Cisco) Internetwork Operating System
IP	Internet Protocol
IPv6	IP version 6
ISDN	Integrated Services Digital Network
ISP	Internet Service Provider
ITU	International Telecommunications Union
ITU-T	ITU - Telecommunication standardization sector
ITSP	Internet Telephony Service Provider
IVR	Interactive Voice Response
LAN	Local Area Network
LESS	Language for End System Services
LDAP	Lightweight Directory Access Protocol
MAC	Medium (or Media) Access Control (or link layer)
MIB	Management Information Base

MIME	Multipurpose Internet Mail Extension
MOS	Mean Opinion Score
MTBF	Mean Time Between Failures
MTTR	Mean Time To Recover
NAPTR	(DNS) Naming Authority Pointer
NAPT	Network Address and Port Translator (see NAT)
NAT	Network Address Translator
P2P	Peer-to-Peer
PBX	Private Branch eXchange (telephone switch)
PC	Personal Computer
PCM	Pulse Code Modulation
PIN	Personal Identification Number
POSIX	The Portable Operating System Interface
POTS	Plain Old Telephone Service (also PSTN)
PSTN	Public Switched Telephone Network
PUT	Primary User Table
QoS	Quality of Service
RADIUS	Remote Authentication in Dial-In User Service
RAT	Robust Audio Tool
RPC	Remote Procedure Call
RPS	Registrations Per Second
RTP	Real-time Transport Protocol
RTCP	Real-time Transport Control Protocol (also RTP)
RTSP	Real Time Streaming Protocol
SAP	Session Announcement Protocol
SDK	Software Development Kit
SDP	Session Description Protocol
SHA	Secure Hash Algorithm (also SHA1)
SIP	Session Initiation Protocol

SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol (also RPC)
SRV	(DNS) Service resource record
SQL	Structured Query Language
STUN	Simple Traversal of UDP through NAT
Tcl	Tool Command Language
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TRIP	Telephony Routing over IP
TTL	Time-To-Live
TURN	Traversal Using Relay NAT
UA	User Agent
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
URI	Universal Resource Identifier
URL	Universal Resource Locator
VNC	Virtual Network Computing
VoiceXML	Voice eXtensible Markup Language
XML	eXtensible Markup Language

Appendix F

Bibliography

[1] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: a transport protocol for real-time applications,” RFC 1889, Internet Engineering Task Force, Jan. 1996.

[2] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: a transport protocol for real-time applications,” RFC 3550, Internet Engineering Task Force, July 2003.

[3] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, “SIP: session initiation protocol,” RFC 3261, Internet Engineering Task Force, June 2002.

[4] H. Schulzrinne and J. Rosenberg, “Internet telephony: Architecture and protocols – an IETF perspective,” *Computer Networks and ISDN Systems*, Vol. 31, pp. 237–255, Feb. 1999.

[5] International Telecommunication Union, “Network grade of service parameters and target values for circuit-switched services in the evolving ISDN,” Recommendation E.721, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, May 1999.

[6] F. Schmidt, F. G. López, K.-D. Hackbarth, and A. Cuadra, “An analytical cost model for the national core network,” consultative document, Wissenschaftliches Institut für Kommunikationsdienste, Apr. 1999.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol – HTTP/1.1," RFC 2616, Internet Engineering Task Force, June 1999.

[8] H. Schulzrinne and J. Rosenberg, "The session initiation protocol: Internet-centric signaling," *IEEE Communications Magazine*, Vol. 38, Oct. 2000.

[9] H. Schulzrinne, S. Narayanan, J. Lennox, and M. Doyle, "SIPstone - benchmarking SIP server performance," Technical Report CUCS-005-02, Department of Computer Science, Columbia University, New York, New York, Mar. 2002.

[10] G. Patel and S. Dennett, "The 3GPP and 3GPP2 movements toward an All-IP mobile network," *IEEE Personal Communications Magazine*, Vol. 7, Aug. 2000.

[11] M. Martin, "Input 3rd-generation partnership project (3GPP) release 5 requirements on the session initiation protocol (SIP)," RFC 4083, Internet Engineering Task Force, May 2005.

[12] D. Milojicic, V. Kalogeraki, R. M. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, "Peer-to-peer computing," technical report HPL-2002-57 20020315, Technical Publications Department, HP Labs Research Library, Mar. 2002. http://www.hpl.hp.com/techreports/2002/HPL-2002-57.html.

[13] K. Singh and H. Schulzrinne, "Peer-to-peer internet telephony using SIP," in *NOSSDAV 2005*, (Skamania, Washington), June 2005.

[14] K. Singh and H. Schulzrinne, "Peer-to-peer internet telephony using SIP," Tech. Rep. CUCS-044-04, Department of Computer Science, Columbia University, New York, NY, Oct. 2004.

[15] S. Baset, H. Schulzrinne, E. Shim, and K. Dhara, "Requirements for SIP-based Peer-to-Peer Internet Telephony," Internet Draft draft-baset-sipping-p2preq-00, Internet Engineering Task Force, Oct 2005. work in progress.

[16] A. Johnston, "SIP, P2P, and Internet Communications," Internet Draft draft-johnston-sipping-p2p-ipcom-01, Internet Engineering Task Force, Mar 2005. work in progress.

[17] D. Bryan, B. Lowekamp, and C. Jennings, "A P2P Approach to SIP Registration," Internet Draft draft-bryan-sipping-p2p-02, Internet Engineering Task Force, Mar 2006. work in progress.

[18] "Kazaa: peer-to-peer file sharing software application." http://www.kazaa.com.

[19] "Gnutella: peer-to-peer file sharing software application." http://www.gnutella.com.

[20] "Zero configuration networking (zeroconf)." http://www.ietf.org/html.charters/zeroconf-charter.html.

[21] "Skype: Free internet telephony that just works." http://www.skype.com.

[22] I. Stoica, R. Morris, D. R. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (San Diego, CA, USA), ACM, Aug. 2001.

[23] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Transactions on Networking*, Vol. 11, pp. 17–32, Feb. 2003.

[24] J. Lennox, H. Schulzrinne, and J. Rosenberg, "Common gateway interface for SIP," RFC 3050, Internet Engineering Task Force, Jan. 2001.

[25] J. Schwartz, "Collaboration: More hype than reality," *InternetWeek (online newsletter)*, Oct. 1999. http://www.internetweek.com/trans/tr99-bp1.htm.

[26] J. Lennox, "Services for internet telephony," PhD. thesis, Department of Computer Science, Columbia University, New York, New York, Jan. 2004. http://www.cs.columbia.edu/~lennox/thesis.pdf.

[27] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "OpenDHT: a public DHT service and its uses," *SIGCOMM Computer Communication Review*, Vol. 35, no. 4, pp. 73–84, 2005.

[28] J. Rosenberg and H. Schulzrinne, “Session initiation protocol (SIP): locating SIP servers,” RFC 3263, Internet Engineering Task Force, June 2002.

[29] MySQL AB Co., “MySQL home page,” http://www.mysql.com.

[30] X. Wu and H. Schulzrinne, “sipc, a multi-function SIP user agent,” in *7th IFIP/IEEE International Conference, Management of Multimedia Networks and Services (MMNS)*, pp. 269–281, IFIP/IEEE, Springer, Oct. 2004.

[31] X. Wu, “Columbia university sip user agent (sipc).” http://www.cs.columbia.edu/IRT/sipc.

[32] H. Schulzrinne, A. Rao, and R. Lanphier, “Real time streaming protocol (RTSP),” RFC 2326, Internet Engineering Task Force, Apr. 1998.

[33] K. Singh and H. Schulzrinne, “Unified messaging using SIP and RTSP,” in *IP Telecom Services Workshop*, (Atlanta, Georgia), pp. 31–37, Sept. 2000.

[34] K. Singh, G. Nair, and H. Schulzrinne, “Centralized conferencing using SIP,” in *Internet Telephony Workshop*, (New York), Apr. 2001.

[35] S. McGlashan, D. Burnett, J. Carter, S. Tryphonas, J. Ferrans, T. User, B. Lucas, and B. Porter, “Voice extensible markup language (VoiceXML) version 2.0,” tech. rep., World Wide Web Consortium (W3C), Feb. 2003. http://www.w3.org/TR/voicexml20/.

[36] K. Singh, A. Nambi, and H. Schulzrinne, “Integrating voicexml with SIP services.,” in *ICC 2003 - Global Services and Infrastructure for Next Generation Networks*, (Anchorage, Alaska), May 2003.

[37] International Telecommunication Union, “Packet based multimedia communication systems,” Recommendation H.323, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Feb. 1998.

[38] International Telecommunication Union, “Narrow-band visual telephone systems and terminal equipment,” Recommendation H.320, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, May 1999.

[39] International Telecommunication Union, “Terminal for low bit-rate multimedia communication,” Recommendation H.324, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Feb. 1998.

[40] H. Schulzrinne and J. Rosenberg, “A comparison of SIP and H.323 for Internet telephony,” in *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, (Cambridge, England), pp. 83–86, July 1998.

[41] I. Dalgic and H. Fang, “Comparison of H.323 and SIP for IP telephony signaling,” in *Photonics East*, (Boston, Massachusetts), SPIE, Sept. 1999.

[42] K. Singh and H. Schulzrinne, “Interworking between SIP/SDP and H.323,” in *IP-Telephony Workshop (IPtel)*, (Berlin, Germany), Apr. 2000.

[43] H. Schulzrinne and J. Rosenberg, “Signaling for Internet telephony,” in *International Conference on Network Protocols (ICNP)*, (Austin, Texas), Oct. 1998.

[44] T. Berners-Lee, R. Fielding, and L. Masinter, “Uniform resource identifiers (URI): generic syntax,” RFC 2396, Internet Engineering Task Force, Aug. 1998.

[45] N. Freed and N. Borenstein, “Multipurpose internet mail extensions (MIME) part one: Format of internet message bodies,” RFC 2045, Internet Engineering Task Force, Nov. 1996.

[46] M. Handley and V. Jacobson, “SDP: session description protocol,” RFC 2327, Internet Engineering Task Force, Apr. 1998.

[47] J. Rosenberg and H. Schulzrinne, “An offer/answer model with session description protocol (SDP),” RFC 3264, Internet Engineering Task Force, June 2002.

[48] J. Rosenberg, J. Lennox, and H. Schulzrinne, “Programming Internet telephony services,” *IEEE Network*, Vol. 13, pp. 42–49, May/June 1999.

[49] J. Lennox, X. Wu, and H. Schulzrinne, “Call processing language (CPL): a language for user control of internet telephony services,” RFC 3880, Internet Engineering Task Force, Oct. 2004.

[50] J. Rosenberg, “A Framework for Conferencing with the Session Initiation Protocol (SIP),” RFC 4353, Internet Engineering Task Force, feb 2006.

[51] UCB/LBNL, “vic – video conferencing tool.” http://www-nrg.ee.lbl.gov/vic/.

[52] M. A. Sasse, V. J. Hardman, I. Kouvelas, C. E. Perkins, O. Hodson, A. I. Watson, M. Handley, and J. Crowcroft, “RAT (robust-audio tool),” 1995.

[53] J. Highfield and K. Hasler, “Whiteboard tool,” 1995. http://www-mice.cs.ucl.ac.uk/multimedia/software/wbd/.

[54] M. Handley, C. Perkins, and E. Whelan, “Session announcement protocol,” RFC 2974, Internet Engineering Task Force, Oct. 2000.

[55] H. Schulzrinne and K. Arabshian, “Providing emergency services in Internet telephony,” *IEEE Internet Computing*, Vol. 6, pp. 39–47, May 2002.

[56] H. Bryhni, E. Klovning, and Øivind Kure, “A comparison of load balancing techniques for scalable web servers,” *IEEE Networks*, Vol. 14, July 2000.

[57] K. Suryanarayanan and K. J. Christensen, “Performance evaluation of new methods of automatic redirection for load balancing of apache servers distributed in the Internet,” in *IEEE Conference on Local Computer Networks*, (Tampa, Florida, USA), Nov. 2000.

[58] O. Damani, P. Chung, Y. yu Huang, C. M. Kintala, and Y. Wang, “ONE-IP: techniques for hosting a service on a cluster of machines,” *Computer Networks*, Vol. 29, pp. 1019–1027, Sept. 1997.

[59] D. Oppenheimer, A. Ganapathi, and D. Patterson, “Why do internet services fail, and what can be done about it?,” in *4th USENIX Symposium on Internet Technologies and Systems (USITS ’03)*, (Seattle, WA), Mar. 2003.

[60] A. C. Snoeren, D. Andersen, and H. Balakrishnan, “Fine-grained failover using connection migration,” in *USENIX Symposium on Internet Technologies and Systems*, (San Francisco), Mar. 2001.

[61] S. Osman, D. Subhraveti, G. Su, and J. Nieh, “The design and implementation of zap: A system for migrating computing environments,” in *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, (Boston, MA), Dec. 2002. pp. 361-376.

[62] High-Availability Linux Project, http://www.linux-ha.org/.

[63] Cisco Systems, Failover configuration for LocalDirector, http://www.cisco.com/warp/public/cc/pd/cxsr/400/tech/locdf_wp.htm.

[64] G. Hunt, G. Goldszmidt, R. P. King, and R. Mukherjee, “Network dispatcher: a connection router for scalable Internet services,” *Computer Networks*, Vol. 30, pp. 347–357, Apr. 1998.

[65] C.-L. Yang and M.-Y. Luo, “Efficient support for content-based routing in web server clusters,” in *2nd USENIX Symposium on Internet Technologies and Systems*, (Boulder, Colorado, USA), Oct 1999.

[66] Akamai Technologies, Inc. http://www.akamai.com.

[67] A. Gulbrandsen, P. Vixie, and L. Esibov, “A DNS RR for specifying the location of services (DNS SRV),” RFC 2782, Internet Engineering Task Force, Feb. 2000.

[68] M. Mealling and R. Daniel, “The naming authority pointer (NAPTR) DNS resource record,” RFC 2915, Internet Engineering Task Force, Sept. 2000.

[69] V. Cardellini, M. Colajanni, and P. S. Yu, “Dynamic load balancing on web-server systems,” *IEEE Internet Computing*, Vol. 3, no. 3, pp. 28–39, 1999.

[70] N. Ohlmeier, “Design and implementation of a high availability SIP server architecture,” Thesis, Computer Science Department, Technical University of Berlin, Berlin, Germany, July 2003.

[71] P. Conrad, A. Jungmaier, C. Ross, W.-C. Sim, and M. Tüxen, “Reliable IP telephony applications with SIP using RSerPool,” in *World Multiconference on Systemics, Cybernetics and Informatics (SCI)*, (Orlando, USA), July 2002.

[72] M. Tuexen, Q. Xie, R. Stewart, M. Shore, J. Loughney, and A. Silverton, "Architecture for reliable server pooling," Internet Draft draft-ietf-rserpool-arch-10, Internet Engineering Task Force, Jan 2006. work in progress.

[73] M. Tuexen, Q. Xie, R. Stewart, M. Shore, L. Ong, J. Loughney, and M. Stillman, "Requirements for reliable server pooling," RFC 3237, Internet Engineering Task Force, Jan. 2002.

[74] L. G. M. Bozinovski and R. Prasad, "A state-sharing mechanism for providing reliable SIP sessions," in *6th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services*, (Nis, Serbia and Montenegro), Oct. 2003.

[75] H. S. M. Bozinovski and R. Prasad, "Maximum availability server selection policy for session control systems based on 3GPP SIP," in *Seventh International Symposium on Wireless Personal Multimedia Communications*, (Padova, Italy), Sept. 2004.

[76] A. Srinivasan, K. G. Ramakrishnan, K. Kumaran, M. Aravamudan, and S. Naqvi, "Optimal design of signaling networks for Internet telephony," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Tel Aviv, Israel), Mar. 2000.

[77] R. Sparks, "The session initiation protocol (SIP) refer method," RFC 3515, Internet Engineering Task Force, Apr. 2003.

[78] J. Rosenberg, "Requirements for management of overload in the session initiation protocol," Internet Draft draft-rosenberg-sipping-overload-reqs-00, Internet Engineering Task Force, Feb 2006. work in progress.

[79] Emic Cluster for MySQL, http://www.emicnetworks.com.

[80] J. Janak, "SIP proxy server effectiveness," Master's Thesis, Department of Computer Science, Czech Technical University, Prague, Czech, May 2003.

[81] V. Pai, P. Druschel, and W. Zwaenepoel, "Flash: An efficient and portable web server," in *USENIX Annual Technical Conference*, (Montery, California, USA), Jun 1999.

[82] M. Welsh, D. Culler, and E. Brewer, "SEDA: an architecture for well-conditioned, scalable Internet services," in *Symposium on Operating Systems Principles (SOSP)*, (Chateau Lake Louise, Canada), ACM, Oct. 2001.

[83] R. Behren, J. Condit, F. Zhou, G. Necula, and E. Brewer, "Capriccio: scalable threads for internet services," in *ACM Symposium on Operating Systems Principles (SOSP)*, (Bolton Landing, NY), 2003.

[84] S. Mishra and R. Yang, "Thread-based vs event-based implementation of a group communication," in *Proceedings of the 12th IEEE International Parallel Processing Symposium and 9th IEEE Symposium on Parallel and Distributed Processing (IPDPS)*, (Orlando, FL), Apr 1998.

[85] J. Ousterhout, "Why threads are a bad idea (for most purposes)," in *USENIX Technical Conference (Invited Talk)*, (Austin, TX), Jan. 1996.

[86] Cisco IP phone 7960, Release 2.1, http://www.cisco.com.

[87] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, "DNS performance and the effectiveness of caching," in *ACM SIGCOMM Internet Measurement Workshop*, (San Francisco, California), Nov. 2001.

[88] K. Singh, W. Jiang, J. Lennox, S. Narayanan, and H. Schulzrinne, "CINEMA: columbia internet extensible multimedia architecture," technical report CUCS-011-02, Department of Computer Science, Columbia University, New York, New York, May 2002.

[89] W. Jiang, J. Lennox, S. Narayanan, H. Schulzrinne, K. Singh, and X. Wu, "Integrating Internet telephony services," *IEEE Internet Computing*, Vol. 6, pp. 64–72, May 2002.

[90] J. Ousterhout, "Tcl: A universal scripting language," in *USENIX Symposium on Very High Level Languages*, (Santa Fe, New Mexico), Oct. 1994. Invited Talk.

[91] P. Srisuresh and D. Gan, "Load sharing using IP network address translation (LSNAT)," RFC 2391, Internet Engineering Task Force, Aug. 1998.

[92] W. Zhao and H. Schulzrinne, "Dotslash: A self-configuring and scalable rescue system for handling web hotspots effectively," in *International Workshop on Web Caching and Content Distribution (WCW)*, (Beijing, China), Oct. 2004.

[93] B. Jenkins, "Algorithm alley," *Dr. Dobb's Journal*, Sept. 1997. http://burtleburtle.net/bob/hash/doobs.html.

[94] D. R. Karger, A. H. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B.-J. J. Kim, and L. Matkins, "Web caching with consistent hashing," *Computer Networks*, Vol. 31, pp. 1203–1213, May 1999.

[95] I. Jackson, "GNU adns: advanced, easy-to-use, asynchronous-capable DNS client library and utilities." http://www.chiark.greenend.org.uk/˜ian/adns/.

[96] "SIP express router (ser): a high performance free sip server." http://www.iptel.org/ser.

[97] F. P. Duffy and R. A. Mercer, "A study of network performance and customer behavior during-direct-distance-dialing call attempts in the USA," *Bell System Technical Journal*, Vol. 57, no. 1, pp. 1–33, 1978.

[98] D. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, "Resilient overlay networks," in *18th ACM SOSP*, (Banff, Canada), Oct. 2001.

[99] J. Toga and J. Ott, "ITU-T standardization activities for interactive multimedia communications on packet-based networks: H.323 and related recommendations," *Computer Networks and ISDN Systems*, Vol. 31, pp. 205–223, Feb. 1999.

[100] Z. Ge, D. Figueiredo, S. Jaiswal, J. F. Kurose, and D. Towsley, "Modeling peer-peer file sharing systems," in *IEEE Infocom 2003*, Mar. 2003.

[101] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (San Diego, CA, USA), ACM, Aug. 2001.

[102] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, (Heidelberg, Germany), pp. 329–350, Nov. 2001.

[103] D. Liben-Nowell, H. Balakrishnan, and D. Karger, "Analysis of the evolution of peer-to-peer systems," in *ACM Conf. on Principles of Distributed Computing (PODC)*, (Monterey, CA, USA), ACM, July 2002.

[104] S. Baset and H. Schulzrinne, "An analysis of the skype peer-to-peer internet telephony protocol," in *IEEE INFOCOM 2006*, (Barcelona, SPAIN), Apr. 2006.

[105] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy, "STUN - simple traversal of user datagram protocol (UDP) through network address translators (nats)," RFC 3489, Internet Engineering Task Force, Mar. 2003.

[106] J. Rosenberg, R. Mahy, and C. Huitema, "Traversal Using Relay NAT (TURN)," Internet Draft draft-rosenberg-midcom-turn-08, Internet Engineering Task Force, Sep 2005. work in progress.

[107] J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for Offer/Answer Protocols," Internet Draft draft-ietf-mmusic-ice-06, Internet Engineering Task Force, Oct 2005. work in progress.

[108] F. Strauss and S. Schmidt, "P2P CHAT - a peer-to-peer chat protocol," internet draft, Internet Engineering Task Force, June 2003. Work in progress.

[109] "Groove workspace software." http://www.groove.net.

[110] "Magi p2p technology being adopted across vertical industries." http://www.endeavors.com/PressReleases/partners1.htm.

[111] "Apple iChat AV: Videoconferencing for the rest of us." http://www.apple.com/ichat/.

[112] "Nimcat networks." http://www.nimcatnetworks.com/.

[113] "Popular telephony." http://www.populartelephony.com/.

[114] "SIP beyond voice and video." http://www.research.earthlink.net/p2p/.

[115] J. Rosenberg, J. Peterson, H. Schulzrinne, and G. Camarillo, "Best current practices for third party call control (3pcc) in the session initiation protocol (SIP)," RFC 3725, Internet Engineering Task Force, Apr. 2004.

[116] K. Singh and H. Schulzrinne, "SIPpeer: a session initiation protocol (SIP)-based peer-to-peer Internet telephony client adaptor," white paper, Computer Science Department, Columbia University, New York, NY, Jan 2005. http://www.cs.columbia.edu/IRT/p2p-sip/papers/sip-p2p-design.pdf.

[117] M. Wahl, T. Howes, and S. Kille, "Lightweight directory access protocol (v3)," RFC 2251, Internet Engineering Task Force, Dec. 1997.

[118] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (New York), Mar. 1999.

[119] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in dynamic structured P2P systems," in *IEEE Infocom 2004*, (Hong Kong), Mar. 2004.

[120] M. Roussopoulos and M. G. Baker, "Practical load balancing for content requests in peer-to-peer networks," technical report cs/0209023, arXiv, Sept. 2002.

[121] S. D. Gribble, E. Brewer, J. Hellerstein, and D. Culler, "Scalable, distributed data structures for Internet service construction," in *Operating Systems Design and Implementation*, (San Diego, CA, USA), Usenix, Oct. 2000.

[122] "OpenDHT: a public distributed hash table service." http://www.opendht.org.

[123] H. Schulzrinne, "Composing Presence Information," Internet Draft draft-schulzrinne-simple-composition-00, Internet Engineering Task Force, Jul 2005. work in progress.

[124] R. Mahy, "Connection Reuse in the Session Initiation Protocol (SIP)," Internet Draft draft-ietf-sip-connect-reuse-04, Internet Engineering Task Force, Jul 2005. work in progress.

[125] J. Rosenberg and H. Schulzrinne, “An extension to the session initiation protocol (SIP) for symmetric response routing,” RFC 3581, Internet Engineering Task Force, Aug. 2003.

[126] “The OpenSSL project.” http://www.openssl.org.

[127] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart, “HTTP authentication: Basic and digest access authentication,” RFC 2617, Internet Engineering Task Force, June 1999.

[128] R. Merkle, “A digital signature based on a conventional encryption function,” *Advances in Cryptology — CRYPTO ’87*, *Lecture Notes in Computer Science*, Vol. 293, pp. 369–378, 1988.

[129] E. Niemi, “Session initiation protocol (SIP) extension for event state publication,” Internet Draft draft-ietf-sip-publish-02, Internet Engineering Task Force, Jan. 2004. Work in progress.

[130] W. Zhao, H. Schulzrinne, and E. Guttman, “Mesh-enhanced service location protocol (mslp),” RFC 3528, Internet Engineering Task Force, Apr. 2003.

[131] E. Guttman, C. Perkins, J. Veizades, and M. Day, “Service location protocol, version 2,” RFC 2608, Internet Engineering Task Force, June 1999.

[132] K. Arabshian and H. Schulzrinne, “Hybrid hierarchical and peer-to-peer ontology-based global service discovery system,” Tech. Rep. CUCS-016-05, Columbia University, Apr. 2005.

[133] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, and H. Weatherspoon, “Oceanstore: An extremely wide-area storage system,” technical report UCB//CSD-00-1102, U.C. Berkeley, CA, USA, May 1999.

[134] A. Mislove, A. Post, C. Reis, P. Willmann, P. Druschel, D. Wallach, X. Bonnaire, P. Sens, J.-M. Busca, and L. Arantes-Benzerra, “Post: A secure, resilient, cooperative messaging system,” in *HotOS IX: The 9th workshop on hot topics in operating systems*, (Lihue, Hawaii, USA), USENIX, May 2003.

[135] J. Lennox and H. Schulzrinne, "A protocol for reliable decentralized conferencing," in *ACM NOSSDAV 2003*, (Monterey, California, USC), June 2003.

[136] M. Castro, M. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman, "An evaluation of scalable application-level multicast built using peer-to-peer overlays," in *IEEE Infocom 2003*, Mar. 2003.

[137] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Pittsburgh,PA), p. 13, Aug. 2002.

[138] M. Castro, P. Druschel, Y. Hu, and A. Rowstron, "Proximity neighbor selection in tree-based structured peer-to-peer overlays," technical report MSR-TR-2003-52, Microsoft Research, 2003.

[139] A. Roach, "Session initiation protocol (SIP)-specific event notification," RFC 3265, Internet Engineering Task Force, June 2002.

[140] J. Rosenberg, "A session initiation protocol (SIP) event package for registrations," RFC 3680, Internet Engineering Task Force, Mar. 2004.

[141] J. Rosenberg, "Interactive connectivity establishment (ICE): a methodology for nettwork address translator (NAT) traversal for the session initiation protocol (SIP)," internet draft, Internet Engineering Task Force, July 2003. Work in progress.

[142] B. Ford, P. Srisuresh, and D. Kegel, "Peer-to-peer communication across middleboxes," Internet Draft draft-ford-midcom-p2p-01, Internet Engineering Task Force, Oct. 2003. Work in progress.

[143] D. Murphy, J. Kelly, K. Curley, J. Vickery, and D. O'Keeffe, "P2p security," Online Report, Networks and Telecommunications Research Group, Computer Science Department, Trinity College, Dublin 2, Ireland, Jan. 2003. http://ntrg.cs.tcd.ie/undergrad/4ba2.02-03/p10.html.

[144] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach, "Security for structured peer-to-peer overlay networks," in *Operating Systems Design and Implementation*, (Boston, MA), Usenix, Dec. 2002.

[145] P. Biondi and F. Desclaux, "Silver Needle in the Skype," Mar. 2006. http://www.blackhat.com/.

[146] M. Gupta, P. Judge, and M. Ammar, "A reputation system for peer-to-peer networks," in *ACM NOSSDAV 2003*, (Monterey, California, USC), June 2003.

[147] S. Lee, R. Sherwood, and S. Bhattacharjee, "Cooperative peer groups in NICE," in *IEEE Infocom 2003*, Mar. 2003.

[148] S. Kamvar, M. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in P2P networks," in *International World Wide Web Conference (WWW)*, (Budapest, Hungary), International World Wide Web Conference Committee, May 2003.

[149] L. Xiong and L. Liu, "Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 16, pp. 843–857, July 2004.

[150] E. Adar and B. A. Huberman, "Free riding on gnutella," *First Monday*, Vol. 5, Oct. 2000.

[151] E. Sit and R. Morris, "Security considerations for peer-to-peer distributed hash tables," in *Electronic Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, (Cambridge, MA, USA), IEEE, Mar 2002.

[152] A. Gupta, B. Liskov, and R. Rodrigues, "One hop lookups for peer-to-peer overlays," in *HotOS IX: The 9th workshop on hot topics in operating systems*, (Lihue, Hawaii, USA), USENIX, May 2003.

[153] S. Ratnasamy, S. Shenker, and I. Stoica, "Routing algorithms for DHTs: some open questions," in *International Workshop on Peer-to-Peer Systems (IPTPS)*, (Cambridge, MA, USA), IEEE, Mar. 2002.

[154] M. Handley, J. Crowcroft, C. Bormann, and J. Ott, "The internet multimedia conferencing architecture," internet draft, Internet Engineering Task Force, July 2000. Work in progress.

[155] S. Bhattacharyya and I. Ed., "An overview of source-specific multicast (SSM)," RFC 3569, Internet Engineering Task Force, July 2003.

[156] H. W. Holbrook and D. R. Cheriton, "IP multicast channels: EXPRESS support for large-scale single-source applications," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Cambridge, Massachusetts), August/September 1999.

[157] Z. Y. Shae and M.-S. Chen, "Mixing and playback of JPEG compressed packet videos," in *Proceedings of the IEEE Conference on Global Communications (GLOBECOM)*, (Orlando, Florida), pp. 245–249 (08B.03), IEEE, Dec. 1992.

[158] H. Schulzrinne, "RTP profile for audio and video conferences with minimal control," RFC 1890, Internet Engineering Task Force, Jan. 1996.

[159] J. Luciani, "Classical IP to NHRP transition," RFC 2336, Internet Engineering Task Force, July 1998.

[160] "Apple's Quicktime real-time streaming media player." http://www.quicktime.com.

[161] "Realplayer media player." http://www.real.com.

[162] "Cisco CallManager." http://www.cisco.com/.

[163] "Nortel multimedia communication server 5100." http://www.nortelnetworks.com/.

[164] "Skinny call control protocol (SCCP)." http://www.cisco.com.

[165] "CUSeeMe: Cornell University's video conferencing tool." http://www.cuseeme.com.

[166] "Lotus Sametime 3.0." http://www.sametime.com.

[167] "GnomeMeeting." http://www.gnomemeeting.org.

[168] M. Stefik, G. Foster, D. G. Bobrow, K. Kahn, S. Lanning, and L. Suchman, "Beyond the chalkboard: computer support for collaboration and problem solving in meetings," *Communications ACM*, Vol. 30, pp. 32–47, Jan. 1987.

[169] J. Conklin, "Hypertext: An introduction and survey," in *Groupware — software for computer-supported cooperative work* (D. Marca and G. Bock, eds.), IEEE Computer Society Press, 1992. IEEE Computer, September 1987.

[170] A. Dix, "Computer-supported cooperative work - a framework," in *Design Issues in CSCW, Eds. D. Rosenburg and C. Hutchison*, Springer Verlag, 1994. http://www.comp.lancs.ac.uk/computing/users/dixa/papers/cscwframework94/.

[171] A. Dix, "Challenges and perspectives for cooperative work on the web," in *An International workshop on CSCW and the Web*, (Sankt Augustin, Germany), ERCIM/W4G, Feb. 1996. http://orgwis.gmd.de/projects/W4G/proceedings/challenges.html.

[172] W. Appelt, "WWW based collaboration with the BSCW system," in *SOFSEM (SOFtware SEMinar)*, (Milovy, Czech Republic), pp. 66–78, Springer-Verlag in the Lecture Notes in Computer Science 1725, Nov. 1999. http://bscw.gmd.de/Papers/SOFSEM99/sofsem.pdf.

[173] "Lotus domino." http://www.lotus.com.

[174] "Hyperwave." http://www.hyperwave.com.

[175] "Opentext corporation." http://www.opentext.com/livelink.

[176] G. Kaiser and S. M. Kaplan, "CSCW and software process. session summary in ninth international software process workshop: The role of humans in the process," in *Ninth International Software Process Workshop*, pp. 9–11, Oct. 1994.

[177] M. Mühlhäuser, "Interdisciplinary development of an electronic class and conference room," *Journal of Universal Computer Science*, Vol. 2, pp. 694–710, Oct. 1996.

[178] P. Saint-Andre, "Extensible messaging and presence protocol (XMPP): core," RFC 3920, IETF, Oct. 2004.

[179] E. Schooler, S. Casner, and J. B. Postel, “Multimedia conferencing: Has it come of age?,” in *24th Hawaii International Conference on System Science*, Vol. 3, (Hawaii), pp. 707–716, IEEE, Jan. 1991.

[180] S. Yang, S. Yu, J. Zhou, and Q. Han, “Multipoint communications with speech mixing over IP network,” *Computer Communications*, Vol. 25, pp. 46–55, Jan. 2001.

[181] M. Handel and J. Herbsleb, “What is chat doing in the workplace,” in *Proceedings of ACM Conference on computer supported cooperative work(CSCW)*, (New Orleans, Louisiana, USA), Nov. 2002.

[182] “VidMid: The video working group of the internet2 middleware initiative.” http://middleware.internet2.edu/video.

[183] H. Schulzrinne, “Conferencing and collaborative computing,” in *Dagstuhl Seminar on Fundamentals and Perspectives of Multimedia Systems*, (Dagstuhl Castle, Germany), July 1994.

[184] E. A. Isaacs and J. C. Tang, “What video can and can’t do for collaboration: a case study,” in *ACM Multimedia*, (Anaheim, California), pp. 199–206, Aug. 1993.

[185] S. McCanne and V. Jacobson, “vic: A flexible framework for packet video,” in *ACM Multimedia*, Nov. 1995.

[186] V. Kumar, *MBone: Interactive Multimedia On The Internet*. Macmillan Publishing (Simon & Schuster), 1995.

[187] “MeetingPlace.” http://www.meetingplace.net/.

[188] J. Ott, “Teleconferencing in the ITU-T,” in *IETF*, (San Jose, California), Dec. 1994. Multiparty Multimedia Session Control WG (MMusic), Talk (c).

[189] P. Balaouras, I. Stavrakakis, and L. Merakos, “Potential and limitations of a teleteaching environment based on H.323 audio-visual communication systems,” *Computer Networks*, Vol. 34, pp. 945–958, Dec. 2000.

[190] S. Greenberg and M. Roseman, "Groupweb: A web browser as real-time groupware," in *Conference on human factors in computing systems, companion, proceedings*, (Vancouver, Canada), pp. 271–272, ACM SIGCHI'96, Apr. 1996.

[191] H.-P. Dommel and J. J. Garcia-Luna-Aceves, "Floor control for multimedia conferencing and collaboration," *Multimedia Systems*, Vol. 5, no. 1, pp. 23–38, 1997.

[192] P. Koskelainen, H. Schulzrinne, and X. Wu, "A SIP-based conference control framework," in *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, (Miami Beach, Florida), pp. 53–61, May 2002.

[193] D. Trossen, *Scalable Group Communications in Tightly Coupled Environments*. PhD thesis, University of Technology, Aachen, Germany, Sept. 2000.

[194] F. DePaoli and F. Tisato, "Coordinator: a basic building block for multimedia conferencing systems," in *Proceedings of the IEEE Conference on Global Communications (GLOBECOM)*, (Phoenix, Arizona), pp. 2049–2053 (58.1), IEEE, Dec. 1991.

[195] "pcAnywhere by Symantec, Inc.." http://www.symantec.com/pcanywhere.

[196] "GoToMyPC by Expert City, Inc.." http://www.gotomypc.com/.

[197] International Telecommunication Union, "Multipoint application sharing," Recommendation T.128, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Feb. 1998.

[198] T. Ohmori, K. Maeno, S. Sakata, H. Fukuoka, and K. Watabe, "Distributed cooperative control for application sharing based on multiparty and multimedia desktop conferencing system: MERMAID," *ACM Computer Communication Review*, Vol. 22, pp. 39–40, Mar. 1992.

[199] "VirtualPlaces." http://www.vplaces.com/vpnet/index.html.

[200] C. Agboh, "A study of two main IP telephony signaling protocols: H.323 signaling and SIP; a comparison and a signaling gateway specification," Master's thesis, Unversite Libre

de Bruxelles (ULB), Facutés des Science, Départment Informatique, Brussels, Belgium, 1999. supervised by Eric Manie.

[201] N. Kausar and J. Crowcroft, "An architecture of conference control functions," in *Photonics East*, (Boston, Massachusetts), SPIE, Sept. 1999.

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

[203] D. B. Terry and D. C. Swinehart, "Managing stored voice in the etherphone system," *ACM Transactions on Computer Systems*, Vol. 6, pp. 3–27, Feb. 1988.

[204] P. T. Zellweger, D. B. Terry, and D. C. Swinehart, "An overview of the etherphone system and its applications," in *2nd IEEE Conference on Computer Workstations*, (Santa Clara, California), pp. 160–168, Mar. 1988.

[205] P. V. Rangan and D. C. Swinehart, "Software architecture for integration of video services in the etherphone environment," *IEEE Journal on Selected Areas in Communications*, Vol. 9, pp. 1395–1404, Dec. 1991.

[206] G. Vaudreuil and G. Parsons, "Voice profile for internet mail - version 2 (vpimv2)," RFC 3801, Internet Engineering Task Force, June 2004.

[207] B. Campbell and R. Sparks, "Control of service context using SIP request-uri," RFC 3087, Internet Engineering Task Force, Apr. 2001.

[208] M. R. Civanlar, G. L. Cash, R. V. Kollarits, B.-B. Paul, C. T. Swain, B. G. Haskell, and D. A. Kapilow, "Videotalks: A comprehensive multimedia conferencing system," in *Packet Video*, (Sardinia, Italy), May 2000.

[209] H. Vin, P. T. Zellweger, D. C. Swinehart, and P. V. Rangan, "Multimedia conferencing in the etherphone environment," *IEEE Computer*, Vol. 24, pp. 69–79, Aug. 1991.

[210] P. Koskelainen, J. Ott, H. Schulzrinne, and X. Wu, "Requirements for Floor Control Protocols," RFC 4376, Internet Engineering Task Force, jan 2006.

[211] O. Novo, G. Camarillo, D. Morgan, and R. Even, "A common conference information data model for centralized conferencing (XCON)," Internet Draft draft-ietf-xcon-common-data-model-00, Internet Engineering Task Force, Apr 2006. work in progress.

[212] "Plum voice portals: automated telephony solutions." http://www.plumvoiceportals.com.

[213] "Open Source VoiceXML Interpreter." http://www.openvxi.com.

[214] "Talking EMail: voice-enabled email." http://www.voice3g.com/appblocks.htm.

[215] E. Burger, J. Dyke, and A. Spitzer, "Basic network media services with SIP," RFC 4240, Internet Engineering Task Force, Dec 2005.

[216] "TellMe studio." http://www.tellme.com.

[217] W. Jiang, J. Lennox, H. Schulzrinne, and K. Singh, "Towards junking the PBX: deploying IP telephony," in *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, (Port Jefferson, New York), June 2001.

[218] X. Wu and H. Schulzrinne, "Programmable end system services using SIP," in *International Conference on Communications*, (Anchorage, Alaska), pp. 789–793, May 2003.

[219] D. Robinson and K. Coar, "The common gateway interface (CGI) version 1.1," Internet Draft draft-coar-cgi-v11-04.txt,.ps,, Internet Engineering Task Force, Oct. 2003. Work in progress.

[220] K. Singh, X. Wu, J. Lennox, and H. Schulzrinne, "Comprehensive multi-platform collaboration," Tech. Rep. CUCS-027-03, Dept. of Computer Science, Columbia University, New York, New York, Dec. 2003.

[221] R. Ramjee, J. F. Kurose, D. F. Towsley, and H. Schulzrinne, "Adaptive playout mechanisms for packetized audio applications in wide-area networks," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Toronto, Canada), pp. 680–688, IEEE Computer Society Press, Los Alamitos, California, June 1994.

[222] J. Rosenberg, L. Qiu, and H. Schulzrinne, "Integrating packet FEC into adaptive voice playout buffer algorithms on the Internet," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Tel Aviv, Israel), Mar. 2000.

[223] H. Schulzrinne, "Indication of message composition for instant messaging," RFC 3994, Internet Engineering Task Force, Jan. 2005.

[224] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper, "Virtual network computing," *IEEE Internet Computing*, Vol. 2, pp. 33–38, January/February 1998.

[225] X. Wu, P. Koskelainen, and H. Schulzrinne, "Conference floor control protocol," internet drafts, Internet Engineering Task Force, 2003.

[226] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple object access protocol (SOAP) 1.1," tech. rep., World Wide Web Consortium, W3C, May 2000.

[227] H. Schulzrinne and S. Petrack, "RTP payload for DTMF digits, telephony tones and telephony signals," RFC 2833, Internet Engineering Task Force, May 2000.

[228] Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen, "HTTP extensions for distributed authoring – WEBDAV," RFC 2518, Internet Engineering Task Force, Feb. 1999.

[229] N. Freed and N. Borenstein, "Multipurpose internet mail extensions (MIME) part two: Media types," RFC 2046, Internet Engineering Task Force, Nov. 1996.

[230] J. Myers and M. Rose, "Post office protocol - version 3," RFC 1939, Internet Engineering Task Force, May 1996.

[231] M. Crispin, "Internet message access protocol - version 4," RFC 1730, Internet Engineering Task Force, Dec. 1994.

[232] M. Crispin, "Internet message access protocol - version 4rev1," RFC 2060, Internet Engineering Task Force, Dec. 1996.

[233] J. Rosenberg, H. Schulzrinne, and P. Kyzivat, “Caller preferences for the session initiation protocol (SIP),” RFC 3841, Internet Engineering Task Force, Aug. 2004.

[234] J. Rosenberg, “A presence event package for the session initiation protocol (SIP),” RFC 3856, Internet Engineering Task Force, Aug. 2004.

[235] Dallas Semiconductor Corp., “ibutton,” 2002. http://www.ibutton.com.

[236] R. Mahy, “A message summary and message waiting indication event package for the session initiation protocol (SIP),” RFC 3842, Internet Engineering Task Force, Aug. 2004.

[237] “Xerces C++ Parser.” http://xml.apache.org/xerces-c/.

[238] L. Hanson, “Simple HTTP fetcher in C.” GNU LGPL software.

[239] A. Black and K. Lenzo, “Flite: a small, fast run time synthesis engine.” http://fife.speech.cs.cmu.edu/flite/.

[240] D. Liu and N. Ogasawara, *Email by phone using VoiceXML*. Columbia University, New York, May 2001.

[241] “Procmail home-page.” http://www.procmail.org/.

[242] CMU Sphinx group, “CMU sphinx open source speech recognition engines,” 2000. http://www.speech.cs.cmu.edu/sphinx/index.html.

[243] A. Black and K. Lenzo, *Flite: a small, fast run time synthesis engine*. Speech Group at Carnegie Mellon University, 1.0 ed., Aug. 2001. http://fife.speech.cs.cmu.edu/flite/.

[244] N. Charlton, M. Gasson, G. Gybels, M. Spanner, and A. Wijk, “User requirements for the session initiation protocol (SIP) in support of deaf, hard of hearing and speech-impaired individuals,” RFC 3351, Internet Engineering Task Force, Aug. 2002.

[245] D. Wheeler, “SLOCCount: A tool to measure source lines of code.” http://www.dwheeler.com/sloccount.

[246] C. Partridge, T. Mendez, and W. Milliken, “Host anycasting service,” RFC 1546, Internet Engineering Task Force, Nov. 1993.

[247] O. Levin, “H.323 uniform resource locator (URL) scheme registration,” RFC 3508, Internet Engineering Task Force, Apr. 2003.

[248] International Telecommunication Union, “Media stream packetization and synchronization on non-guaranteed quality of service LANs,” Recommendation H.225.0, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Nov. 1996.

[249] International Telecommunication Union, “Control protocol for multimedia communication,” Recommendation H.245, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Feb. 1998.

[250] International Telecommunication Union, “H.323 extended for loosely coupled conferences,” Recommendation H.332, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Sept. 1998.

[251] International Telecommunication Union, “Security and encryption for H-series (H.323 and other H.245-based) multimedia terminals,” Recommendation H.235, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Feb. 1998.

[252] International Telecommunication Union, “Interworking of H-series multimedia terminals with H-series multimedia terminals and voice/voiceband terminals on GSTN and ISDN,” Recommendation H.246, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Feb. 1998.

[253] International Telecommunication Union, “Generic functional protocol for the support of supplementary services in H.323,” Recommendation H.450.1, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Feb. 1998.

[254] International Telecommunication Union, “Call diversion supplementary service for H.323,” Recommendation H.450.3, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Sept. 1997.

[255] International Telecommunication Union, “Digital subscriber signalling system no. 1 (DSS 1) - ISDN user-network interface layer 3 specification for basic call control,” Recommendation Q.931, International Telecommunication Union, Geneva, Switzerland, Mar. 1993.

[256] A. Johnston, R. Sparks, C. Cunningham, S. Donovan, and K. Summers, “Session initiation protocol service examples,” Internet Draft draft-ietf-sipping-service-examples-10, Internet Engineering Task Force, Mar 2006. work in progress.

[257] O. Hersent, D. Gurle, and J.-P. Petit, *IP telephony*. Reading, Massachusetts: Addison Wesley, 2000.

[258] H. Schulzrinne, “The tel URI for telephone numbers,” RFC 3966, Internet Engineering Task Force, Dec. 2004.

[259] “The OpenH323 project.” http://www.openh323.org.

[260] W. Jiang and H. Schulzrinne, “Assessment of voip service availability in the current internet,” in *Passive & Active Measurement Workshop*, (San Diego, CA), Apr. 2003.

[261] A. Kristensen, “SIP Servlet API Specification,” Java Specification Request (JSR) JSR-000116, Java Community Process, mar 2002. Review Draft.

[262] D. Eastlake, J. Reagle, and D. Solo, “XML-Signature Syntax and Processing,” W3C Recommendation TR/2002/REC-xmldsig-core-20020212/, World Wide Web (W3C) Consortium, Feb 2002. http://www.w3.org/TR/xmldsig-core/.

[263] D. Eastlate and J. Reagle, “XML Encryption Syntax and Processing,” W3C Recommendation TR/2002/REC-xmlenc-core-20021210/, World Wide Web (W3C) Consortium, Dec 2002. http://www.w3.org/TR/xmlenc-core/.

[264] S. Bradner, “Key words for use in rfcs to indicate requirement levels,” RFC 2119, Internet Engineering Task Force, Mar. 1997.